Technische Universität Graz

Institut für Elektronik

Entwicklung einer Elektronik zum Synchronbetrieb von BLDC-Motoren

Clemens Treichler, BSc

Masterarbeit

MA 711

Institutsleiter: Univ.-Prof. Dipl.-Ing. Dr.techn. Wolfgang Bösch
Begutachter: Ass.-Prof. Dipl.-Ing. Dr.techn. Gunter Winkler
Externer Betreuer: Dipl.-Ing. Stefan Lukas

Graz, September 2012

Abstract

This master thesis deals with the development of an electronic circuit to control brushless direct current (BLDC) motors. The circuits layout is routed on a printed circuit board with the size of $37mm$ diameter. Due to the circuit board has to be mounted directly on the top of the motor, it is necessary to keep it small. This concept allows the heat of the power parts to be absorbed by the motor chassis which ends up in a much shorter propulsion system. The power section is able to supply an electrical power of $P_{el} \approx 120W$ at a voltage of $24V$. The control of the motor is done by the so called "field oriented control". This algorithm achieves best running performance and high efficiency of the motor. The electronic control unit, motor and gearbox are working together as an actuator to build height adjustable tables. Therefore a position regulator is combined with the field oriented control algorithm. This position regulator makes it possible to synchronize several actuators. CAN-interface is integrated in the circuit and used to interconnect the actuators. The whole system consists of generally available electronic standard parts. This approach allows an adaption of the circuit to different motor power configurations without the need of redesigning the printed circuit board.

Keywords: BLDC motor, motor performance, efficiency, field oriented control, position regulator, standard parts

Kurzfassung

Im Rahmen dieser Masterarbeit wurde eine Elektronik zur Ansteuerung bürstenloser (BLDC) Motoren entwickelt. Die gesamte Schaltung findet dabei auf einer kreisförmigen Leiterplatte mit $37mm$ Durchmesser Platz. Die kleine runde Bauweise ergibt sich aus der Notwendigkeit, die Leiterplatte direkt am Motor montieren zu können. Dadurch bleibt die Gesamtlänge gering, und gleichzeitig erlaubt diese Montage eine Kühlung der Leistungselektronik am Motorgehäuse. Der Leistungsteil ist so dimensioniert, dass dieser bei einer Betriebsspannung von $24V$ eine elektrische Leistung von zirka $P_{el} \approx 120W$ zur Ansteuerung des Motors bereitstellen kann. Angesteuert wird der Motor mittels Vektorregelung. Diese, auch als „feldorientierte Regelung“ bekannte Ansteuerung, ermöglicht ein optimales Laufverhalten bei geringer Geräuschentwicklung und bestmöglichem Wirkungsgrad des Motors. Ansteuerelektronik, Motor sowie ein nachgeschaltetes Getriebe bilden einen Aktuator, mit dem beispielsweise höhenverstellbare Tische realisiert werden können. Um die Vektorregelung ist deshalb ein Positionsregler implementiert, der unter anderem einen Synchronbetrieb mit weiteren Antrieben ermöglicht. Die Kommunikation der Aktuatoren untereinander erfolgt über eine in der Schaltung inkludierten CAN-Schnittstelle. Bei der entwickelten Schaltung wurde darauf Wert gelegt, größtenteils leicht verfügbare Standardbauteile einzusetzen. Dadurch ist unter anderem eine Anpassung an abweichende Motorleistungen möglich, ohne dass das Layout der Leiterplatten geändert werden muss.

Stichwörter: BLDC-Motor, Laufverhalten, Wirkungsgrad, Vektorregelung, Positionsregler, Standardbauteile

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, _____ _____

Date Signature

Eidesstattliche Erklärung[1]

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am _____ _____

Datum Unterschrift

[1] Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008; Genehmigung des Senates am 1.12.2008

Danksagung

Damit es mir überhaupt möglich war, diese Arbeit erfolgreich zu einem Ende zu führen, war ich auf die Hilfsbereitschaft vieler Menschen angewiesen, die mich auf dem Weg bis zum Abschluss meiner Masterarbeit begleitet und unterstützt haben. Es ist mir daher ein Anliegen, mich an dieser Stelle bei jenen Menschen zu bedanken:

Herr Walter Koch, Geschäftsführer der Firma LogicData Deutschlandsberg, hat die Idee zu dieser Masterarbeit ins Leben gerufen. Herr Gunter Winkler, Assistenzprofessor am Institut für Elektronik der Technischen Universität Graz, sowie Herr Stefan Lukas seitens der Firma LogicData Deutschlandsberg, haben mich bei dieser Masterarbeit perfekt betreut, und hatten stets ein offenes Ohr für Probleme, die im Zusammenhang mit dieser Arbeit aufgetreten sind. Herr Peter Söser, Assistenzprofessor am Institut für Elektronik der Technischen Universität Graz, hat für mich die Aufgabe des Mentors übernommen und dabei organisatorisches zur Masterarbeit erledigt. Herr Eduard Dorner und Kevin Tomaser, seitens der Werkstätte des Instituts für Elektronik der Technischen Universität Graz, hatten stets eine Lösung parat, für scheinbar unmögliche mechanische Konstruktionen. Bettina Lamprecht, Ralf Wießpeiner, Daniel Kollreider und Rupert Rohrmoser, seitens der Firma LogicData Deutschlandsberg, waren mir behilflich in organisatorischen, softwaretechnischen und mechatronischen Angelegenheiten. Anna Mikl, Klaus Meyer und Oliver Meyer haben diese Masterarbeit durchkämmt, und mit größter Sorgfalt auf Tippfehler untersucht. Ein großes Dankeschön geht an meine Eltern, die mich nicht nur während der Masterarbeit, sondern die gesamte Studienzeit hindurch in vielerlei Hinsicht unterstützt haben.

Graz, 18. September 2012 Clemens Treichler

Inhaltsverzeichnis

1. Einleitung

Diese Masterarbeit, mit dem Titel „Entwicklung einer Elektronik zum Synchronbetrieb von BLDC-Motoren“, ist in Zusammenarbeit mit der Firma LogicData entstanden. LogicData ist ein international tätiges Unternehmen im Bereich Elektronik, das Motorsteuerungen und Bedienelemente für ergonomische Lösungen in der Möbelindustrie realisiert. Die aktuelle Steuerungslinie von LogicData verwendet gewöhnliche bürstenbehaftete $24V$ Gleichstrommotoren. Um den notwendigen Bauraum zu verringern und den Wirkungsgrad zu erhöhen wurde im Rahmen dieser Masterarbeit ein System mit bürstenlosen Gleichstrommotoren entwickelt. Die Ansteuerelektronik wird dabei direkt am Motor angebracht, woraus sich eine kompakte Bauweise der Schaltung ergibt.

Das akustische Verhalten des Motors steht im Zusammenhang mit dessen Laufverhalten, welches sich mit dem Ansteuerverfahren beeinflussen lässt. Die entwickelte Schaltung wurde daher so ausgelegt, dass die Hardware alle notwendigen Ressourcen abdeckt, die zur Implementation einer Vektorregelung notwendig sind. Die Vektorregelung, auch bekannt unter der Bezeichnung „feldorientierte Regelung“, ist ein Verfahren zur Ansteuerung bürstenloser Motoren, das bestmögliches Laufverhalten und einen großen Dynamikbereich garantiert.

Üblicherweise werden bei höhenverstellbaren Schreibtischen mehrere Motoren synchron betrieben. Es wurde daher unter anderem der CAN-Bus realisiert, um den Synchronbetrieb mehrerer Antriebe zu ermöglichen.

Im Zuge der Masterarbeit wurde ein marktüblicher Standardmotor mit einer mechanischen Leistung von $P_{mec} = 77{,}5W$ verwendet, um die Funktionalität des entwickelten Schaltungsprototypen zu zeigen.

Diese Masterarbeit gliedert sich folgendermaßen auf:

2 BLDC-Motor & Vektorregelung (ab Seite 3) In diesem einleitenden Teil wird auf die Eigenschaften bürstenloser Motoren näher eingegangen, und auf die daraus resultierenden Anforderungen an deren Ansteuerung. Des Weiteren wird die feldorientierte Regelung näher beschrieben, welche aus der Notwendigkeit eines optimalen Laufverhaltens, als Ansteuerverfahren gewählt wurde.

3 Details zur Hardware (ab Seite 11) In diesem Teil werden Einzelheiten der Schaltung bezüglich ihrer Funktion detailliert beschrieben. Weiters werden die Schaltungsteile hinsichtlich ihrer Belastung und der daraus resultierenden Zuverlässigkeit untersucht.

4 Implementierung der Software (ab Seite 49) Dieser Teil widmet sich der Firmware. Die Software ist dabei auf einer Bibliothek zur Ansteuerung bürstenloser Motoren

aufgebaut, welche an die Hardware angepasst, und um den notwendigen funktionalen Umfang erweitert wurde.

5 Ausblick (ab Seite 65) Dieser Teil behandelt potentielle Möglichkeiten zur Verbesserung der Hardware und der Software hinsichtlich einer Weiterentwicklung zur Serienreife.

Anhang (ab Seite 69) Im Anhang befinden sich Auszüge der Software, Skripte zum Konvertieren der Daten aus dem Layout-Editor, sowie alle Schaltpläne.

2. BLDC-Motor & Vektorregelung

2.1. Der bürstenlose Gleichstrommotor

Die Bezeichnungen „bürstenlos“ und „Gleichstrommotor“ im selben Kontext erscheinen auf den ersten Blick etwas widersprüchlich. Einerseits benötigt ein Gleichstrommotor stets Bürsten, um über den Kommutator die Ankerwicklung anspeisen zu können, andererseits handelt es sich bei bürstenlosen Motoren um Maschinen, die zum Betrieb ein elektrisches Drehfeld benötigen.

Tatsächlich handelt es sich beim bürstenlosen Gleichstrommotor (BLDC) im Prinzip um eine gewöhnliche ungedämpfte Synchronmaschine, bei der der Rotor mit Permanentmagneten erregt ist. Die wesentlichen Komponenten einer derartigen Maschine sind zum Einen der festsitzende Stator, und zum Anderen der bewegliche Rotor, über den letztendlich die Kraft auf die Welle übertragen wird. Die Kommutierung erfolgt über elektronische Schalter, welche die Wicklung so ansteuern, dass ein rotierendes Feld resultiert. Abbildung 2.1 zeigt schemenhaft den Aufbau eines dreiphasigen Synchronmotors mit einem Polpaar. Die Vektoren a, b und c entsprechen den Strängen der Statorwicklung, während die Komponenten d und q das auf den Rotor bezogene Koordinatensystem abbilden. Der Winkel Θ_r ist dabei der Versatz zwischen Statorwicklung und Rotorfeld.

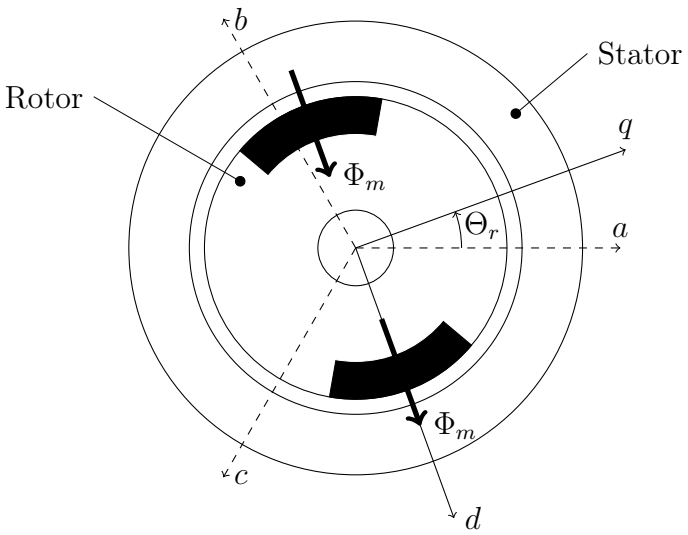

Abbildung 2.1.: Feldkomponenten eines permanent erregten Synchronmotors

Büstenlose Gleichstommotoren lassen sich anhand des Aufbaus des Rotors und der daraus resultierenden Feldverteilung bezüglich Drehmomentbildung unterscheiden. Dar-

aus leiten sich für den Betrieb des Motors relevante Unterscheidungsmerkmale ab, auf welche im Folgenden näher eingegangen wird.

2.1.1. Oberflächig montierte Rotormagnete

[10] Bei oberflächig montierten Magneten liegt auf Grund dieser Anordnung an der Oberfläche des Rotors in den Hauptrichtungen der Magnete die maximale Feldstärke vor. Das Drehmoment kommt bei dieser Art von Rotoren ausschließlich durch Wechselwirkung der Rotormagnete mit dem Statorfeld zustande. Die Herstellung dieser Rotoren ist relativ einfach, da die Magnete lediglich auf den Rotor aufgeklebt werden. Das hat allerdings den Nachteil dass die mechanische Belastung der Klebestelle direkt mit der Motordrehzahl zusammenhängt, und eine zu hohe Oberflächengeschwindigkeit zur Zerstörung des Rotors führt. Da die an der Motorwelle abgegebene Leistung dem Produkt aus Drehmoment und Winkelgeschwindigkeit entspricht, und dabei das Drehmoment von der Größe des Motors bestimmt wird, muss bei gleicher Leistung ein Motor mit derartigem Rotor größer gebaut sein, als ein Motor dessen Rotor viel höheren Drehzahlen standhält.

2.1.2. Eingebettete Rotormagnete

[10] Eine höhere mechanische Festigkeit des Rotors ergibt sich, wenn die Magnete in den Rotor eingebettet sind. für diesen Aufbau ist es notwendig, dass das magnetische Feld mit gut leitenden Materialien an die Oberfläche des Rotors geführt wird. Aus diesem Grund ist der Rotor aus weichmagnetischem Material, zum Beispiel Eisen, gefertigt. BLDC-Motoren mit einem derartig aufgebauten Rotor weisen auf Grund des Eisens ein zusätzliches Drehmoment auf, welches auf die Reluktanz zurückzuführen ist. Es handelt sich dabei um die Eigenschaft, dass ein magnetischer Kreis stets bestrebt ist, den Zustand des geringsten magnetischen Widerstandes zu erreichen. Eisen, das den magnetischen Widerstand verringert, liegt bei diesem Rotor nicht nur in der Hauptrichtung des Feldes der Permanentmagneten vor, sondern auch dazwischen. Für die Ansteuerung des Motors bedeutet das, dass zusätzlich zur Quadratur-Komponente q auch die direkte Komponente d zur Drehmomentbildung beiträgt. Der daraus resultierende Winkelversatz ist jedoch nicht konstant, sondern hängt vom geforderten Drehmoment ab.

2.2. Methoden zur Ansteuerung bürstenloser Gleichstrommotoren

Zur Ansteuerung bürstenloser Gleichstrommotoren gibt es unterschiedliche Methoden. In Anwendungen, wo es hauptsächlich um präzise Drehzahl und Positionierung geht, und weniger um Effizienz oder Geräuschentwicklung, findet man häufig einen gesteuerten Betrieb des Motors vor. Dabei nutzt man die Tatsache, dass die Motordrehzahl synchron zur Frequenz des Ansteuersignals ist. Als Beispiel sind hier Computerfestplatten zu nennen. Im Gegensatz dazu steht der geregelte Betrieb eines bürstenlosen Motors, welcher ein

besseres Lastverhalten ermöglicht. Zwei Vertreter geregeler Ansteuerverfahren, nämlich „Blockkommutierung“ und „Sinuskommutierung“ werden im Anschluss näher beschrieben.

2.2.1. Blockkommutierung

Die einfachste Methode zur Ansteuerung eines bürstenlosen Gleichstrommotors ist die sogenannte Blockkommutierung. Im Falle eines dreiphasigen Motors ergeben sich, auf Grund der möglichen Schalterstellungen für jede Halbbrücke, sechs Sektoren für eine vollständige elektrische Umdrehung. Die Kommutierung erfolgt basierend auf der Kenntnis darüber, in welchem Sektor sich der Rotor gerade befindet. In der Praxis wird die Position des Rotors meist mit drei Hallsensoren erfasst. Dabei ergeben sich, wegen des überlappenden Schaltens der Sensoren, genau jene sechs Sektoren, welche die für die Kommutierung notwendige Schalterstellung der Halbbrücken bestimmen. Die Regelung der Motorleistung erfolgt mittels Pulsweitenmodulation, wobei es ausreichend ist, wenn eines der beiden Schaltelemente jeder Halbbrücke mit diesem Signal angesteuert wird. Der schaltungstechnische Aufwand für den blockkommutierten Betrieb ist relativ gering.

Die Blockkommutierung hat allerdings den Nachteil, dass sich das Ansteuersignal nicht an die Feldverteilung des Motors anpassen lässt. Daraus resultiert ein nicht optimales Laufverhalten und ein schlechter Wirkungsgrad des Motors.

2.2.2. Sinuskommutierung

Den Nachteilen der Blockkommutierung kann man mittels Sinuskommutierung entgegenwirken, wobei die Bezeichnung für dieses Ansteuerverfahren lediglich auf den speziellen Fall einer sinusförmigen Feldverteilung des Motors zutrifft. Grundsätzlich lässt sich jedoch jede Signalform mit dieser Ansteuerung realisieren, woraus sich ein optimales Laufverhalten des Motors bei bestmöglichem Wirkungsgrad ergibt. Amplitude und Phase wird dabei für jeden Strang der Statorwicklung mittels Pulsweitenmodulation gebildet. Jede Halbbrücke muss dabei so ausgelegt sein, dass High-Side und Low-Side synchron geschaltet werden, wobei eine Überschneidung der beiden Schaltvorgänge vermieden werden sollte. Um die Phase des Ansteuersignals bestimmen zu können, ist die genaue Kenntnis des Rotorwinkels notwendig. Im Gegensatz zur Blockkommutierung fordern die häufigeren und steileren Schaltflanken hinsichtlich elektromagnetischer Verträglichkeit aufwändigere Entstörungsmaßnahmen.

Die Sinuskommutierung bildet die Grundlage für die feldorientierte Regelung. Diese ermöglicht eine abstrahierte Ansteuerung des Motors aus Sicht des Rotors, welche als Einstellgröße lediglich den auf den Rotor bezogenen Stromvektor benötigt. Eine detaillierte Beschreibung dazu folgt im nächsten Abschnitt.

2.3. Feldorientierte Regelung

[10] Die feldorientierte Regelung eines Synchronmotors wird über die transformierten Rotorströme durchgeführt, wobei die Quadratur-Komponente i_q hauptverantwortlich für die Drehmomentbildung ist.

Abbildung 2.2 zeigt den geschlossenen Regelkreis der feldorientierten Regelung.

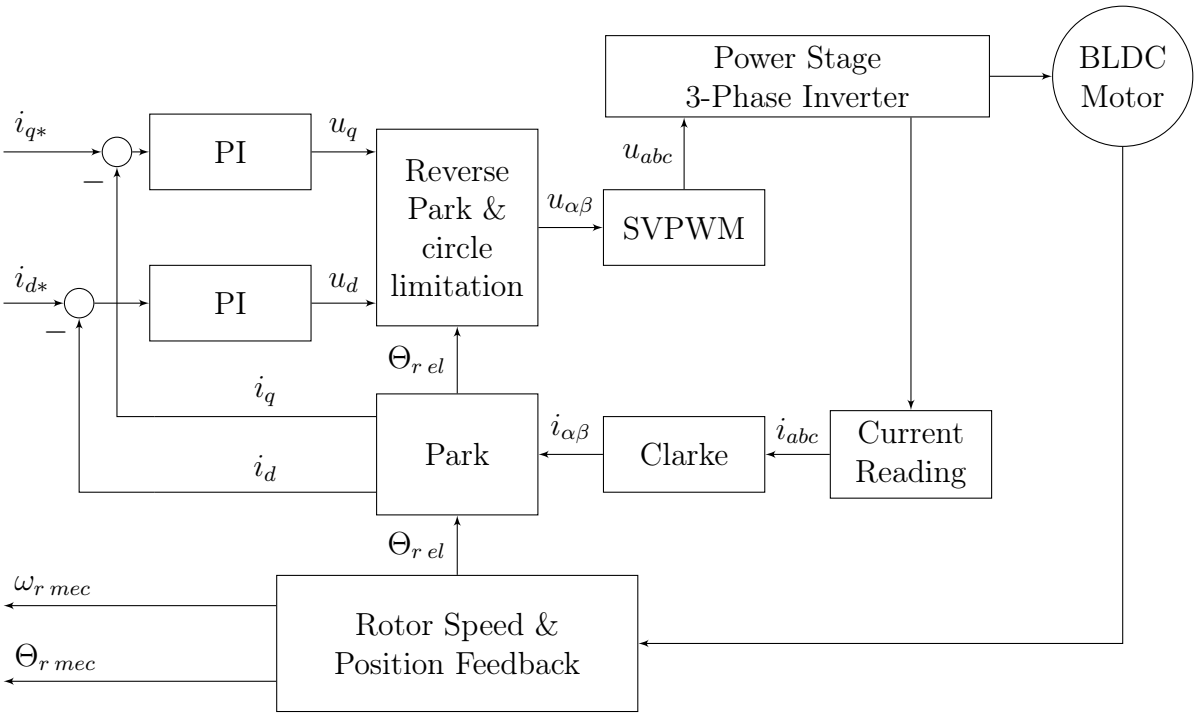

Abbildung 2.2.: Feldorientierte Regelung

2.3.1. Komponenten der feldorientierten Regelung

Im Folgenden werden die Einzelnen Blöcke *Rotor Speed & Position Feedback*, *Current Reading*, *Clarke*, *Park*, *Reverse Park & circle limitation*, sowie *SVPWM* aus Abbildung 2.2 bezüglich ihrer Funktionen näher beschrieben.

Rotorwinkel

Rotor Speed & Position Feedback Über diesen Block wird der Rotorwinkel $\Theta_{r\,mec}$ erfasst. Es gibt unterschiedliche Möglichkeiten, wie dies erfolgen kann, beispielsweise mit Hallsensoren oder optischen Drehencodern. An dieser Stelle sei auf den Abschnitt 4.3 (Erfassung des Rotorwinkels, Seite 59) verwiesen, wo unterschiedliche Strategien zur Erfassung des Rotorwinkels sowie deren Vor- und Nachteile näher behandelt werden.

Eine weitere Funktion dieses Blocks besteht darin, dass über die Anzahl der Polpaare des Motors der für die Park'sche Transformation notwendige elektrische Winkel $\Theta_{r\,el}$

berechnet wird. Zudem wird die Rotordrehzahl $\omega_{r\,mec}$ ermittelt.

Statorstrom

Die Erfassung des Statorstroms erfolgt über die Blöcke *Current Reading* sowie *Clarke*.

Current Reading dient dazu, alle Ströme des Stators zu erfassen. Es gibt hier unterschiedliche Strategien, wie dies bewerkstelligt werden kann, beispielsweise mit einem Stromwandler oder auch über den Spannungsabfall an einem Shunt-Widerstand. An dieser Stelle sei auf den Abschnitt 3.2 (Messung des Motorstroms, Seite 22) verwiesen, wo unterschiedliche Messmethoden sowie deren Vor- und Nachteile genauer behandelt werden.

Clarke Auf Grund des Aufbaus des Stators und der Anordnung der Wicklungen bewegt sich der resultierende Flussvektor auf einer Ebene, wobei die Anzahl an Phasen keine Rolle spielt. Die Clarke'sche Transformation nutzt diese Tatsache und reduziert alle Phasenströme auf einen zweidimensionalen Stromvektor, bestehend aus den Komponenten i_α und i_β. Die nachstehende Gleichung zeigt, wie diese Transformation im Falle einer 3-phasigen Statorwicklung durchzuführen ist.

$$\begin{pmatrix} i_\alpha \\ i_\beta \\ i_0 \end{pmatrix} = \frac{2}{3} \cdot \begin{pmatrix} \cos(\theta) & \cos\left(\theta - \frac{2\pi}{3}\right) & \cos\left(\theta + \frac{2\pi}{3}\right) \\ \sin(\theta) & \sin\left(\theta - \frac{2\pi}{3}\right) & \sin\left(\theta + \frac{2\pi}{3}\right) \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{pmatrix} \cdot \begin{pmatrix} i_a \\ i_b \\ i_c \end{pmatrix}$$

Der zusätzlich eingeführte Strom i_0 entspricht dem Summenstrom des Sternpunktes. Da dieser nicht extra beschaltet ist, muss die Summe der Ströme Null sein. Wenn man $\theta = 0$ setzt, fallen die Achsen i_a und i_α zusammen und es ergeben sich folgende, für die Transformation relevante Gleichungen:

$$\begin{aligned} i_\alpha &= i_a \\ i_\beta &= -\frac{i_a + 2i_b}{\sqrt{3}} \\ i_0 &= i_a + i_b + i_c = 0 \end{aligned}$$

Transformationen

Park Entsprechend der nachfolgenden Gleichungen wird über die Park'sche Transformation aus dem vereinfachten Statorstromvektor $i_{\alpha\beta}$ und dem elektrischen Rotorwinkel $\Theta_{r\,el}$ der auf den Rotor bezogene Stromvektor i_{dq} berechnet.

$$\begin{aligned} i_q &= i_\alpha \cos(\Theta_{r\,el}) - i_\beta \sin(\Theta_{r\,el}) \\ i_d &= i_\alpha \sin(\Theta_{r\,el}) + i_\beta \cos(\Theta_{r\,el}) \end{aligned}$$

Der mittels Park'scher Transformation berechnete Stromvektor stellt die Ist-Größe der Feldorientierten Regelung dar.

Reverse Park & circle limitation Die vom Regler berechnete Stellgröße ist die Spannung, die am beweglichen Rotor anliegen soll, damit der als Soll-Größe eingestellte Rotor-Strom fließt. Da jedoch die Spannung an der festsitzenden Stator-Wicklung anliegt, muss die Spannung auf einen Stator-bezogenen Vektor transformiert werden. Im Wesentlichen handelt es sich dabei um die Inverse der zuvor gezeigten Park'schen Transformation:

$$u_\alpha = u_q \cos(\Theta_{r\,el}) + u_d \sin(\Theta_{r\,el})$$
$$u_\beta = -u_q \sin(\Theta_{r\,el}) + u_d \cos(\Theta_{r\,el})$$

Die zweite Funktion dieses Blocks (*circle limitation*) begrenzt den Spannungsvektor $U_{\alpha\beta}$, sodass der Betrag des Vektors höchstens die maximal zulässige Spannung annimmt.

SVPWM Diese Abkürzung steht für **S**pace **V**ector **P**ulse **W**idth **M**odulation. Wie aus dieser Bezeichnung hervorgeht, wird in diesem Block entsprechend des Stator-bezogenen Spannungsvektors ein in der Pulsweite moduliertes Signal zur Ansteuerung des Motors generiert. Die Hüllkurve des Modulationssignals sollte dabei an die Feldverteilung des Motors angepasst sein, um optimales Laufverhalten bei bestmöglichem Wirkungsgrad zu erzielen.

2.3.2. Regelkreis der feldorientierten Regelung

Die Stellgröße der feldorientierten Strecke ist der Spanungsvektor u_{dq}, welcher über die inverse Park'sche Transformation sowie der Pulsweitenmodulation die Phasenspannungen für die Statorwicklung vorgibt. Diese Spannungen haben Ströme in der Wicklung und weiters eine Kraft auf den Rotor zufolge. Die Ströme werden mit der Rotorposition über die Clark'sche und Park'sche Transformation letztendlich in den auf den Rotor bezogenen Stromvektor i_{dq} umgeformt. An diesem Punkt wird die Strecke über PI-Regler geschlossen, wobei von außen über den Stromvektor i_{dq*} die Sollgröße der feldorientierten Regelung vorgegeben wird. Abbildung 2.3 zeigt eine vereinfachte Darstellung des Regelkreises.

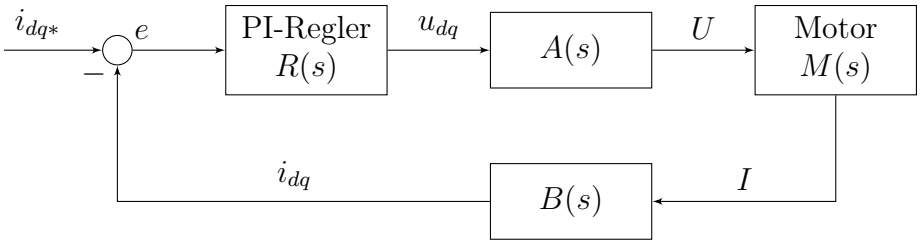

Abbildung 2.3.: Geschlossene Schleife der feldorientierten Regelung

Der PI-Regler bildet die Summe aus einem proportionalen sowie einem integralen Term, mit den zugehörigen Koeffizienten k_p und k_i.

$$R(s) = k_p + \frac{k_i}{s} = \frac{k_i}{s}\left(1 + s\frac{k_p}{k_i}\right)$$

Die Übertragungsfunktion des Motors $M(s)$ ergibt sich aus der Serienschaltung des resistiven (R_s) und des induktiven (L_s) Belags der Statorwicklung, wobei die Eingangsgröße die Spannung U und die Ausgangsgröße der Strom I sind.

$$M(s) = \frac{I}{U} = \frac{1}{R_s + s \cdot L_s} = \frac{1}{R_s\left(1 + s\frac{L_s}{R_s}\right)}$$

Die beiden Blöcke $A(s)$ und $B(s)$ stellen jeweils die Umrechnung von der digitalen in die analoge Domäne und vice versa dar. Sie können innerhalb eines für den Motor relevanten Frequenzbereichs als konstant angenommen werden.

$$A(s) = k_A \qquad\qquad B(s) = k_B$$

Stabilität des Regelkreises

Für die offene Schleife gilt entsprechend der Funktionen jedes Blocks folgende Übertragungsfunktion:

$$L(s) = R(s) \cdot A(s) \cdot M(s) \cdot B(s) = \frac{k_i}{s}\left(1 + s\frac{k_p}{k_i}\right) \cdot k_A \cdot \frac{1}{R_s\left(1 + s\frac{L_s}{R_s}\right)} \cdot k_B$$

[4] Sofern Polstellen und Nullstellen in der linken Z-Ebene liegen, können diese gekürzt werden. Bei geschickter Wahl der Koeffizienten k_p und k_i des PI-Reglers ist dies der Fall. Für die offene Regelschleife bedeutet das, dass lediglich Konstanten und ein Integralterm übrig bleiben.

$$L(s) = \frac{1}{s} \cdot V \qquad \frac{k_p}{k_i} = \frac{L_s}{R_s} \qquad V = \frac{k_i k_a k_B}{R_s} \tag{2.1}$$

Auf Grund des integralen Terms liegt im betrachteten Frequenzbereich eine Phasenverschiebung von $-90°$ vor. Für diese Übertragungsfunktion gilt weiters, dass die Durchtrittsfrequenz genau bei der Verstärkung V liegt. Diese Verstärkung V muss lediglich so eingestellt werden, dass dadurch die Durchtrittsfrequenz innerhalb der betrachteten Bandbreite liegt. In diesem Fall hat die offene Schleife eine Phasenreserve von $\varphi_r = 90°$. Der geschlossene Regelkreis ist demnach entsprechend des vereinfachten Nyquist-Kriteriums auf jeden Fall stabil.

Bandbreite des geschlossenen Regelkreises

Die zuvor getroffenen Annahmen, die zur Kürzung der Pol- und Nullstelle geführt haben und zudem einen stabilen Regelkreis gewährleisten, ergeben folgende Übertragungsfunktion für die geschlossene Schleife:

$$T(s) = \frac{R(s) \cdot A(s) \cdot M(s)}{1 + L(s)} = \frac{1}{k_B} \cdot \frac{1}{1 + s \cdot \frac{R_s}{k_i k_A k_B}}$$

Diese Funktion weist Tiefpasscharakter 1. Ordnung auf, dementsprechend ist die Bandbreite der geregelten Strecke beschränkt. Die $-3dB$ Grenzfrequenz liegt hier bei:

$$\omega_g = \frac{k_i k_A k_B}{R_s}$$

Dies entspricht der in Gleichung (2.1) gezeigten Verstärkung der offenen Schleife. [10] Die Bandbreite sollte im Bereich $\omega_g \approx 1500 s^{-1}$ liegen, wobei das einen Kompromiss einerseits aus Regelgeschwindigkeit und andererseits aus Störunterdrückung der Messwerte darstellt.

3. Hardware

3.1. Motortreiber

Es gibt unterschiedliche Möglichkeiten, wie der Leistungsteil für einen bürstenlosen, permanenterregten Motor realisiert sein kann. In diesem Abschnitt sollen die wichtigsten Topologien hinsichtlich des schaltungstechnischen Aufwands und des daraus resultierenden Flächenbedarfs näher betrachtet werden. Folglich ergibt sich daraus jene Variante, die auf dem Prototyp verwirklicht wurde.

Voll-integrierter Motortreiber Diese Variante scheint auf den ersten Blick die einfachste und komfortabelste zu sein. Hier befinden sich auf einem Chip alle notwendigen Schaltungsteile, beginnend bei einer ausgeklügelten Ansteuerung, welche Schalt- und Verzögerungszeiten berücksichtigt. Aber auch die Leistungstransistoren selbst, bis hin zu zusätzlichen Funktionen wie Temperaturüberwachung und Abschaltung bei Überschreiten des zulässigen Stromes finden hier Platz. Es gibt mehrere Halbleiterhersteller, die im Bereich integrierter Motortreiber für diese konkrete Applikation brauchbare Chips herstellen. Das Produktportfolio an Treiber-ICs für BLDC-Motoren ist aber bei weitem nicht so umfangreich wie beispielsweise jenes für Schritt- und DC-Motoren.

Betreffend der Packages integrierter Leistungsbauteile hat es den Anschein, als ob jeder Halbleiterhersteller diesbezüglich eigene Vorstellungen hätte: Keines der in Frage kommenden Bauteile unterschiedlicher Hersteller ist im selben Gehäuse untergebracht. Es gibt sowohl solche mit Kühlfläche auf der Oberseite, als auch andere, die an der Unterseite zur Kühlung mit der Leiterplatte kontaktiert werden müssen. Selbst ähnliche Bauteile desselben Herstellers unterscheiden sich in der Pinbelegung.

Bei integrierten Motortreibern ist man letztendlich gezwungen zu nehmen, was seitens der Hersteller angeboten wird. Erschwerend kommt noch hinzu, dass man im Falle einer Abkündigung des Chips keine Garantie dafür hat, einen passenden Ersatz zu finden.

Integrierter MOSFET-Treiber Den Nachteilen eines voll-integrierten Motortreibers kann man entgegenwirken, indem man versucht das Problem mit diskreten Leistungstransistoren zu lösen. MOSFETs für Leistungsanwendungen stellen aber auch große Anforderungen an ihre Treiber, daher liegt es nahe, dafür auf integrierte Schaltungen zurück zu greifen, die für diese Aufgabe ausgelegt sind. So genannte Integrierte Gate-Treiber gibt es in den unterschiedlichsten Ausführungen, wobei viele dieser ICs Treiberstufen für High-Side MOSFETs integriert haben. Damit ist es möglich, die gesamte Leistungsstufe mit derselben Type an N-Kanal MOSFETs aufzubauen. Die Freiheit, die man bei der

Wahl der Transistoren hat, ermöglicht eine genaue Anpassung des Leistungsteils an den Motor.

Leistungstransistoren, die der geforderten Strombelastbarkeit standhalten sind zu je zwei Stück in einem Gehäuse der Größenordnung *SO8* untergebracht. Im Vergleich dazu benötigen integrierte Gate-Treiber mit den notwendigen peripheren Bauteilen wesentlich mehr Platz. Dies liegt vor allem daran, dass verfügbare Gate-Treiber hauptsächlich zur Ansteuerung größerer MOSFETs gedacht sind. Selbst mit kleinsten integrierten Gate-Treibern ist die Ansteuerschaltung eigentlich überdimensioniert.

Diskreter Aufbau Diese Variante ist im Vergleich zu den vorigen wohl die kniffligste. Zum Einen gibt es unzählige Möglichkeiten die Treiberstufe zu gestalten, zum Anderen gilt es dann auch die Bauteile optimal zu dimensionieren. Genau genommen lässt sich von vornherein nicht einmal die Frage beantworten, ob mit einem diskreten Aufbau der Platz auf dem Print besser genutzt wird als mit integrierten Schaltungen. Diesen offensichtlichen Nachteilen stehen aber auch Vorteile gegenüber: Ein diskreter Aufbau ermöglicht, dass dieser genau an die Leistungstransistoren angepasst werden kann. Des Weiteren ist es im Falle einer Abkündigung einfacher für Standard-Transistoren einen Ersatz zu finden, als für spezielle integrierte Schaltungen.

Abbildung 3.1 zeigt die Schaltung einer der insgesamt drei identischen Kanäle des diskret aufgebauten Motortreibers.

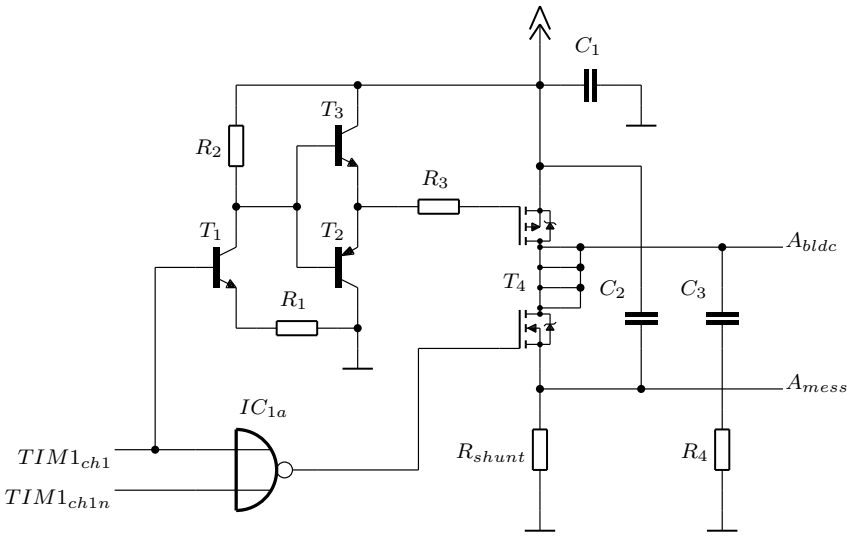

Abbildung 3.1.: Schaltung des Motortreibers – eine Halbbrücke

Zu erwähnen sei hier noch, dass am Mikrocontroller zu den PWM-Ausgängen auch deren komplementäre ausgeführt sind.

Für jeden dieser Ausgänge kann gegenüber den regulären Ausgängen eine Verzögerungszeit eingestellt werden, sodass Schalt- und Verzögerungszeiten kompensiert werden, und somit die Leistungstransistoren ohne Überlappung geschaltet werden können.

3.1.1. Halbbrücke

Jede Halbbrücke wurde mit einem komplementären MOSFET-Paar realisiert, das heißt, dass die positive Spannung mit einem P-Kanal Transistor geschaltet wird. Grundsätzlich sollten P-Kanal MOSFETs nicht die erste Wahl sein, wenn es darum geht, große Ströme zu schalten. Diese Transistoren haben bedingt durch ihre Technologie einige Nachteile gegenüber N-Kanal Typen: [8] P-Schichten weisen eine geringere Mobilität der Ladungsträger auf als N-Schichten. Die Fläche eines P-Kanal MOSFETs muss demnach bei selber Stromtragfähigkeit zirka 2-3 mal so groß sein wie die eines N-Kanal Transistors. Daraus ergibt sich auch eine größere Fläche des Gates. Zur Ansteuerung muss demnach auch eine 2-3 mal größere Ladung aufgebracht werden.

Im Gegensatz dazu spricht für diese Variante der geringe schaltungstechnische Aufwand zur Ansteuerung dieser Transistoren, welcher auf Grund des Platzmangels auf der Leiterplatte durchaus seine Berechtigung hat. Sämtliche für die Schaltung geeigneten komplementären MOSFET-Paare für Leistungsanwendungen weisen ähnliche Eigenschaften auf: Die Gate-Kapazität des P-Kanal Transistors ist ungefähr doppelt so groß wie die des N-Kanal Transistors. Damit beide Transistoren zirka den selben On-Widerstand aufweisen, muss am Gate des P-Kanal Transistors eine um etwa die Hälfte größere Spannung anliegen. Der N-Kanal MOSFET weist bereits bei einer Gatespannung von $U_{GS} = 4V$ ein für die Anwendung akzeptables Verhalten der Source-Drain-Strecke auf.

In den Nachfolgenden Punkten finden sich Überlegungen zur konkreten Ansteuerung der Leistungstransistoren. Diese basieren auf dem Datenblatt [18] des Transistors *si4564dy*. Es handelt sich hierbei um einen Transistor, der in seiner Bauform den größten Drain-Strom sowie die größte Gateladung aufweist. Bei selber Ansteuerung ergibt sich für diesen Transistor, verglichen mit den schwächeren gleicher Bauform, das schlechteste dynamische Verhalten. Er eignet sich daher recht gut für eine Worst-Case-Abschätzung.

3.1.2. Ansteuerung des P-Kanal MOSFETs

Die Schaltung zur Ansteuerung des P-Kanal MOSFETs (Abbildung 3.1) mag zwar recht einfach aussehen, im Folgenden soll sie trotzdem genauer untersucht werden. Nicht nur aus dem Grund, weil dieser Schaltungsteil essenziell für eine ordentliche Ansteuerung des Leistungstransistors ist, sondern auch weil bei einigen Bauteilen die thermische Belastbarkeit ziemlich ausgereizt wird.

Levelshifter

Der P-Kanal MOSFET benötigt zur Ansteuerung am Gate eine Spannung die etwa um $7{,}5V$ kleiner ist als an Source. Die Spannungspegel an den Ausgängen des Mikrocontrollers liegen bei $0V$ für *low* und $V_{DD} = 3{,}3V$ für *high*. Mit den Widerständen R_1 und R_2 sowie dem Transistor T_1 ist eine Schaltung realisiert, um die Ausgangspegel des Mikrocontrollers an die Schaltpegel des MOSFETs anzupassen. T_1 und R_1 bilden quasi eine spannungsgesteuerte Stromsenke, bei der sich der Kollektorstrom bei high-Pegel am

Ausgang des Mikrocontrollers folgendermaßen einstellt:

$$I_C = \frac{U_B - U_{BE}}{R_1} \cdot \frac{h_{FE}}{h_{FE}+1} = \frac{3{,}3V - 0{,}6V}{270\Omega} = 10mA$$

Die Stromverstärkung h_{FE} des Transistors *BC847* beträgt laut Datenblatt [22] mindestens 110, der Term, der die Stromverstärkung berücksichtigt ist somit näherungsweise 1, und kann daher vernachlässigt werden. Aus diesem Grund wird bei folgenden Berechnungen dieser Term nicht weiter berücksichtigt.

Am Widerstand R_2 fällt entsprechend des Kollektorstroms folgende Spannung ab:

$$U_{R_2} = I_C \cdot R_2 = 10mA \cdot 820\Omega = 8{,}2V$$

Wenn der Ausgang des Mikrocontrollers auf low-Pegel und somit unter der Spannung von $U_{BE} = 0{,}6V$ liegt, dann fließt kein Basisstrom in den Transistor, und in weiterer Folge stellt sich auch kein Kollektorstrom ein, die Spannung an R_2 liegt dann bei $U_{R_2} = 0V$.

Entsprechend der logischen Pegel am Ausgang des Mikrocontrollers stellen sich am Widerstand R_2 die Spannungen $0V$ und $8{,}2V$ ein. Diese Spannungen sind auf die Versorgung und somit auf den Source-Anschluss des P-Kanal Transistors bezogen. Die Schaltung, bestehend aus R_1, R_2 und T_1, kann somit als Levelshifter für den MOSFET verstanden werden.

Grundsätzlich könnte der P-Kanal MOSFET aus dieser Schaltung angesteuert werden, sofern die Zeit des Schaltvorgangs eine untergeordnete Rolle spielt. Bei einer maximalen Gatekapazität von $C_G = 6{,}3nF$ und einer Thresholdspannung von $U_{th} = 1V$ ergeben sich folgende Schaltzeiten, wobei die Zeitkonstante τ das Produkt aus R_2 und der Gatekapazität ist und weiters eine Gatespannung von $U_{on\,min} = 7V$ angenommen wird, bei der der Transistor sicher als eingeschaltet betrachtet werden kann:

$$\begin{aligned} t_{on} &= -\ln 1 - \frac{U_{on\,min}}{U_{on}} \cdot \tau = -\ln 1 - \frac{7V}{8{,}2V} \cdot 820\Omega \cdot 6{,}3nF = 9{,}93\mu s \\ t_{off} &= \ln \frac{U_{on}}{U_{th}} \cdot \tau = \ln \frac{8{,}2V}{1V} \cdot 820\Omega \cdot 6{,}3nF = 10{,}87\mu s \end{aligned} \tag{3.1}$$

Wird der Motor mittels Blockkommutierung angesteuert, wäre es ausreichend, wenn die Pulsweitenmodulation mit nur einem der beiden Halbbrückentransistoren realisiert wird, zum Beispiel mit dem N-Kanal MOSFET. In diesem Fall darf der Schaltvorgang des P-Kanal Transistors langsam sein. Man sollte jedoch dafür sorgen, dass die PWM während der Schaltvorgänge der P-Kanal MOSFETs deaktiviert ist, und die N-Kanal Transistoren ausgeschaltet sind.

Gatetreiber

Aus den in Abschnitt 2.2 (Seite 4) gezeigten Überlegungen zur Ansteuerung bürstenloser Motoren geht hervor, dass zwecks optimalen Laufverhaltens des Motors und daraus resultierender geringer Geräuschentwicklung, die Sinuskommutierung gegenüber anderen

Methoden zu bevorzugen ist. Für die Ansteuerung des P-Kanal Transistors bedeutet das, dass im Gegensatz zu den in Gleichung (3.1) gezeigten Schaltzeiten wesentlich kürzere anzustreben sind, nämlich Zeiten im Bereich von $100ns$ bis maximal $300ns$. Bei diesen Schaltzeiten handelt es sich um Erfahrungswerte: Noch kürzere Zeiten haben zwar den Vorteil, dass die Schaltverluste der Leistungstransistoren reduziert werden, hinsichtlich elektromagnetischer Verträglichkeit vergrößert sich jedoch das Störspektrum. Längere Schaltzeiten haben zwar den Vorteil geringerer hochfrequenter Störaussendung, allerdings führen diese zu größeren Schaltverlusten. Die hier angeführten Schaltzeiten sind somit ein guter Kompromiss zwischen Erwärmung der Transistoren und Störaussendung der Schaltung.

Die beiden Transistoren T_2 und T_3 bilden einen Gegentaktemitterfolger, der zwischen dem zuvor beschriebenen Levelshifter und dem Gate des P-Kanal MOSFETs geschaltet ist. [3] Wie der Name dieser Schaltung schon sagt, folgt die Spannung am Ausgang, nämlich am Emitter, jener Spannung am Eingang. Die Spannungsverstärkung ist näherungsweise 1, genau genommen ist bei dieser Schaltung der Spannungshub in beide Richtungen um jeweils die Basis-Emitterspannung von $U_{BE} = 0{,}6V$ kleiner als die Eingangsspannung. Das heißt, wenn an R_2 keine Spannung abfällt, dann stellt sich am Gate eine Spannung von $U_{GS} = -0{,}6V$ ein. Hingegen wenn an R_2 eine Spannung von $U_{R_2} = 8{,}2V$ anliegt, dann stellt sich am Gate eine Spannung von $U_{GS} = -7{,}6V$ ein. [1]

Insofern weist diese Schaltung in der Übertragung ein Loch auf, sprich, eine Änderung am Ausgang in Gegenrichtung erfordert eingangsseitig einen Sprung um eine Spannung von $\Delta U_{R_2} = 2 \cdot U_{BE} = 1{,}2V$. Es gibt durchaus Anwendungsfälle, wo ein derartiges Verhalten unerwünscht ist, jedoch hier geht es darum die Ladung im Gate möglichst schnell zu ändern, und diesen Zweck erfüllt diese Schaltung: Die Stromverstärkung des Emitterfolgers, und das ist der springende Punkt, entspricht der Stromverstärkung des Transistors.

[2] Beim Gegentaktemitterfolger handelt es sich durchaus um eine populäre Schaltung, wenn es darum geht, mit geringem Aufwand einen Gatetreiber diskret aufzubauen. Wie für die meisten Gate-Treiber gilt auch hier, dass der Abstand zum MOSFET so klein wie möglich sein soll, damit durch Schleifen bedingte parasitäre Induktivitäten ebenfalls klein gehalten werden. Auf Grund der kurzzeitig auftretenden hohen Ströme während des Schaltvorgangs sollte der Emitterfolger mit einem Blockkondesator gestützt werden. In der Schaltung aus Abbildung 3.1 sowie im Layout fällt dieser mit dem Blockkondensator C_1 der Halbbrücke zusammen.

Messungen an der Schaltung haben gezeigt, dass für brauchbare Schaltzeiten der Widerstand R_3 überbrückt werden kann. Der Spannungsverlauf am Gate während der Schaltvorgänge entspricht dann einer Rampe, wobei die Einschaltzeit bei $t_{on} = 250ns$ und die Ausschaltzeit bei $t_{off} = 200ns$ liegt.

[1] Die Gate-Sourcespannung weist hier ein negatives Vorzeichen auf, würde man die Spannung entsprechend des Schaltplanes von oben nach unten ablesen, dann wäre von einer Source-Gatespannung die Rede, diese hätte in diesem Fall ein positives Vorzeichen.

Verlustleistung des Levelshifters

Zu guter Letzt soll für die Ansteuerung des P-Kanal MOSFETs die Verlustleistung der Bauteile bestimmt werden, wobei die Bauteilwerte entsprechend ihrer Toleranzen so gewählt werden, das sich für das jeweils zu untersuchende Bauteil der ungünstigste Fall ergibt. Für die Widerstände wird eine Toleranz von 10% angenommen.

Am Kollektor des Transistors T_1 stellt sich entsprechend der folgenden Gleichung ein maximaler Strom von 11,19mA ein:

$$I_{C\,T_1\,max} = \frac{U_B - U_{BE\,min}}{R_{1\,min}} = \frac{3{,}3V - 0{,}58V}{270\Omega \cdot 0{,}9} = 11{,}19mA$$

Dieser Strom führt an R_2 zu folgender Verlustleistung:

$$P_{V\,R_2} = I^2_{C\,T_1\,max} \cdot R_{2\,max} = (11{,}19mA)^2 \cdot 820\Omega \cdot 1{,}1 = 113mW \tag{3.2}$$

An R_1 entsteht nachstehende Verlustleistung:

$$P_{V\,R_1} = \frac{U_B - U^2_{BE\,min}}{R_{1\,min}} = \frac{(3{,}3V - 0{,}58V)^2}{270\Omega \cdot 0{,}9} = 30{,}4mW \tag{3.3}$$

Da der Transistor T_1 als Stromsenke betrieben wird, stellt sich am Widerstand R_2 eine konstante Spannung ein. Am Transistor selbst liegt eine Spannung an, die sehr stark mit der Versorgungsspannung zusammenhängt. Es wird daher für die Nennspannung von $U_{Supply} = 24V$ eine Toleranz von 25% angenommen:

$$\begin{aligned} P_{V\,T_1} &= U_{CE} \cdot I_{C\,T_1\,max} = (U_{Supply\,max} - U_{R_1\,min} - U_{R_2\,min}) \cdot I_{C\,T_1\,max} \\ &= (U_{Supply\,max} - R_{1\,min} \cdot I_{C\,T_1\,max} - R_{1\,min} \cdot I_{C\,T_1\,max}) \cdot I_{C\,T_1\,max} \\ &= (24V \cdot 1{,}25 - 270\Omega \cdot 0{,}9 \cdot 11{,}19mA - 820\Omega \cdot 0{,}9 \cdot 11{,}19mA) \cdot 11{,}19mA \\ &= 213mW \end{aligned} \tag{3.4}$$

An dieser Stelle bedarf es nun einer genaueren Untersuchung der berechneten Verlustleistungen. Die für die Bauteile R_1, R_2 und T_1 relevanten Daten bezüglich Verlustleistung und thermischer Beanspruchung sind den jeweiligen Datenblättern [24], [27] bzw. [22] entnommen. Eines ist diesen Bauteilen gemeinsam: Die gezeigten Verlustleistungen gelten für den Fall eines statischen Betriebs. Tatsächlich verringern sich diese Leistungen, bedingt durch die Pulsweitenmodulation, im Mittel auf die Hälfte.

Beim Widerstand R_1 handelt es sich um ein Vierfach-Netzwerk in der Bauform 1206, wobei einer der Widerstände ungenutzt ist und die anderen drei jeweils einer Halbbrückenansteuerung zugeordnet sind. Jeder einzelne der Widerstände hält einer Leistung von $P_{V\,max} = 62mW$ bis zu einer Umgebungstemperatur von $\vartheta_a = 70°C$ stand. Für höhere Temperaturen bis $\vartheta_{a\,max} = 125°C$ verringert sich die maximal zulässige Verlustleistung linear. Unter Berücksichtigung der PWM verringert sich die unter (3.3) berechnete Leistung auf $P_{V\,R_1} = 15{,}2mW$. Daraus lässt sich mittels linearer Interpolation folgende zulässige Umgebungstemperatur ermitteln:

$$\begin{aligned}\vartheta_{a\,max\,R_1} &= \vartheta_{max} - \frac{\vartheta_{max} - \vartheta_{max\,P_{max}}}{P_{V\,max}} \cdot P_V \\ &= 125°C - \frac{125°C - 70°C}{62mW} \cdot 15{,}2mW = 111{,}5°C\end{aligned} \tag{3.5}$$

Für den Widerstand R_2, welcher in der Bauform 0603 gefertigt ist, gilt ein ähnlicher Temperatur-Leisungszusammenhang mit dem Unterschied, dass hier die höchste zulässige Temperatur bei $\vartheta_{a\,max} = 155°C$ liegt, und die Leistung bis zu $P_{V\,max} = 100mW$ betragen darf. Auch hier verringert sich die unter (3.2) berechnete Leistung auf $P_{V\,R_2} = 56{,}5mW$, und ergo dessen ist für diesen Widerstand folgende Umgebungstemperatur zulässig:

$$\begin{aligned}\vartheta_{a\,max\,R_2} &= \vartheta_{max} - \frac{\vartheta_{max} - \vartheta_{max\,P_{max}}}{P_{V\,max}} \cdot P_V \\ &= 155°C - \frac{155°C - 70°C}{100mW} \cdot 56{,}5mW = 107°C\end{aligned} \tag{3.6}$$

Die für den Transistor unter (3.4) berechnete Leistung wird auf $P_{V\,T_2} = 106{,}5mW$ reduziert und es ergibt sich folgende maximal zulässige Umgebungstemperatur:

$$\vartheta_{a\,max\,T1} = \vartheta_{j\,max} - P_{V\,T_1} \cdot R_{th(j-a)} = 150°C - 106{,}5mW \cdot 625^K/_W = 83{,}4°C \tag{3.7}$$

Man erkennt hier, dass der Transistor T_1, auf Grund der geringsten Temperaturreserve, die Schwachstelle im Levelshifter darstellt. Eine genauere Beurteilung dieses Bauteils, vor allem im Zusammenhang mit der restlichen Schaltung, erfolgt im Abschnitt 3.6 auf Seite 47.

Verlustleistung des Gatetreibers

Die Verlustleistung der Transistoren T_2 und T_3 hängt von der Gatekapazität sowie von der Anzahl der Schalthandlungen ab. Während der Einschaltphase liefert T_2 den Strom zum Laden des Gates, hingegen während der Ausschaltphase wird die Gateladung über T_3 abgebaut. Im Gate ist folgende Energie gespeichert:

$$W_{Gate} = \frac{U_{Gate}^2}{2} \cdot C_{Gate} \tag{3.8}$$

Wird ein Kondensator über einen Widerstand ge- bzw. entladen, dann wird im Widerstand dieselbe Energie in Wärme umgewandelt, wie im Kondensator nach der Ladung gespeichert ist, bzw. nach der Entladung vom Kondensator zur Verfügung gestellt wurde. Die Transistoren T_2 und T_3 können als Widerstände betrachtet werden, und dementsprechend berechnet sich die Verlustleistung für jeden Transistor folgendermaßen:

$$U_{R_2\,max} = I_{C\,T_1\,max} \cdot R_{2\,max} = 11{,}19mA \cdot 820\Omega \cdot 1{,}1 = 10{,}1V$$

$$P_{V\,T_2} = P_{V\,T_3} = \frac{(\Delta U_{GS})^2}{2} \cdot C_G \cdot f = \frac{(U_{R_2\,max} - 2 \cdot U_{BE\,min})^2}{2} \cdot C_G \cdot f$$
$$= \frac{(10{,}1V - 2 \cdot 0{,}58V)^2}{2} \cdot 6{,}3nF \cdot 20kHz = 5{,}04mW$$

Entsprechend nachfolgender Berechnung führt diese Verlustleistung dazu, dass die Sperrschichttemperatur um $3{,}15°C$ gegenüber der Umgebungstemperatur ansteigt.

$$\Delta\vartheta = P_{V\,T_2} \cdot R_{th(j-a)} = 5{,}04mW \cdot 625^K/_W = 3{,}15°C$$

Die am Gegentaktemitterfolger auftretende Verlustleistung und deren thermische Auswirkung sind gering, sodass ein sicherer Betrieb gewährleistet ist.

3.1.3. Ansteuerung des N-Kanal MOSFETs

Der hier beschriebene Schaltungsteil erfüllt genau genommen zwei Funktionen: Zum Einen handelt es sich um eine Treiberstufe für den N-Kanal MOSFET, zum Anderen dient die Schaltung dazu, im Falle eines Versagens der PWM des Mikrocontrollers, die beiden MOSFETs elektrisch so gegeneinander zu verriegeln, dass immer nur ein Transistor eingeschaltet sein kann. Somit wird ein Kurzschluss durch die Halbbrücke und in weiterer Folge eine Zerstörung eben dieser vermieden.

Verriegelung der Transistoren

Die elektrische Verriegelung der Halbbrückentransistoren wird mit einem NOR-Gatter realisiert, welches dem N-Kanal MOSFET vorgeschaltet ist. In Abschnitt 3.1.2 wurde beschrieben, dass der P-Kanal MOSFET mit einem *high*-Pegel aus dem Mikrocontroller eingeschaltet wird. Aus dem Schaltplan 3.1 ist ersichtlich, dass die Ansteuerschaltung des P-Kanal MOSFETs mit einem Eingang des NOR-Gatters verbunden ist. Zur Ansteuerung des N-Kanal Transistors ist ein *high*-Signal notwendig. Aus der Wahrheitstabelle 3.1 sieht man für das NOR-Gatter, dass dies nur dann erfüllt ist, wenn die beiden Eingänge des Gatters 0 sind. Der N-Kanal MOSFET kann demnach nur dann eingeschaltet werden, wenn unter anderem der P-Kanal Transistor nicht angesteuert wird.

b	a	q_{or}	q_{nor}	q_{and}	q_{nand}	q_{xor}	q_{xnor}
0	0	0	1	0	1	0	1
0	1	1	0	0	1	1	0
1	0	1	0	0	1	1	0
1	1	1	0	1	0	0	1

Tabelle 3.1.: Wahrheitstabelle für verschiedene Gatter

Gatetreiber

Der N-Kanal MOSFET benötigt zur Ansteuerung eine Spannung von zirka $5V$, welche vom Mikrocontroller auf Grund dessen Versorgungsspannung von $3{,}3V$ nicht bereitgestellt werden kann. Deshalb ist es notwendig, ein Gatter zu wählen, dessen Eingangsspannungsbereich mit den Ausgangspegeln des Mikrocontrollers zusammenpasst. Gleichzeitig muss es mit $5V$ versorgt werden können und eben diese Spannung am Ausgang als *high*-Pegel ausgeben. Diese Anforderungen erfüllen Gatter mit sogenannten TTL-kompatiblen Eingängen, wobei es sich bei den CMOS Gattern konkret um jene der HCT-Familie handelt.

Gewählt wurde aus der Logikfamilie das 4-fach NOR-Gatter *74HCT02*. Aus dem Datenblatt [21] dieser integrierten Schaltung geht hervor, dass jeder Gateausgang mit einem Strom von $I_O \leq 25mA$ belastet werden darf, was aber nicht bedeutet, dass der Ausgang auf diesen Strom begrenzt ist.

Um eine Aussage über die Schaltzeiten des nachgeschalteten Leistungstransistors treffen zu können, bedarf es genauerer Angaben bezüglich kurzzeitigem Schaltvermögen der Ausgänge. In einem Handbuch zur HCT-Familie [6] findet man dazu folgende Formel zur Berechnung der Anstiegszeiten bei kapazitiver Belastung der Gatterausgänge:

$$t_{rise} = t_{fall} = t_0 + t_{C_{load}} \cdot C_{load} = 6{,}6ns + 0{,}12\frac{ns}{pF} \cdot 3000pF = 366{,}6ns \tag{3.9}$$

Messungen an der Schaltung haben im Gegensatz dazu gezeigt, dass die tatsächlichen Schaltzeiten viel kürzer sind, und im Bereich von $t_{on} = t_{off} \approx 80ns$ liegen. Dies liegt daran, dass der Ausgangstreiber des Gatters für sehr kurze Zeit wesentlich mehr Strom treiben kann, als der Spezifikation zu entnehmen ist, und die tatsächliche Gatekapazität des MOSFETs kleiner ist als die in Gleichung (3.9) angenommene worst-case Kapazität. Hinsichtlich elektromagnetischer Verträglichkeit sind die Schaltzeiten in einem tolerierbaren Bereich. Für eine Serienproduktion sollte man jedoch berücksichtigen, dass es während des Produktlebenszyklus durchaus dazu kommen kann, dass Bauteile, aus welchen Gründen auch immer, geändert werden. Im Falle der Leistungstransistoren würde beispielsweise eine niedrigere Gatekapazität zu noch kürzeren Schaltzeiten führen. Im Abschnitt 5.1.2 (Motortreiber) auf Seite 66 findet sich dazu ein Ansatz, wie man diesem speziellen Problem aus dem Weg gehen kann.

Überlegungen zur Verriegelung

Die hier gezeigte Schaltung zur Ansteuerung des N-Kanal MOSFETs weist in ihrer Funktion als elektrische Verriegelung der Leistungstransistoren eine Schwachstelle auf: Es ist nicht sichergestellt, dass ein MOSFET erst dann einschaltet, wenn der andere sicher ausgeschaltet hat. Man kann sich dazu folgendes Szenario durchdenken: Vom Mikrocontroller kommt das Signal, dass beide Transistoren eingeschaltet sind, sprich *high*-Pegel für die Ansteuerung des P-Kanal und *low*-Pegel für den N-Kanal Transistor. In diesem Fall ist der P-Kanal MOSFET tatsächlich eingeschaltet, während das NOR-Gatter für die Signale einen *low*-Pegel evaluiert, und ergo dessen der N-Kanal MOSFET ausgeschaltet bleibt. Wenn nun das Signal des P-Kanal Transistors auf *low*-Pegel wechselt,

um diesen auszuschalten, dann liegen beide Eingänge des NOR-Gatters auf *low*, und der Ausgang des Gatters beginnt nach einer Verzögerungszeit von zirka $9ns$ auf *high* zu wechseln. Der N-Kanal MOSFET beginnt zu leiten, während der P-Kanal MOSFET noch nicht vollständig ausgeschaltet hat. Es fließt nun für die Dauer des Ausschaltvorganges des P-Kanal MOSFETs ein Querstrom durch die Halbbrücke. Abhängig davon, wie sich die Schaltvorgänge der beiden Leistungstransistoren überlagern, kann ein Spitzenstrom im Bereich von $I_{quer} \approx 5A \ldots 35A$ auftreten. Die Leistungstransistoren halten einem Stromimpuls von $I_{DM} = 40A$ für die Dauer von $10\mu s$ stand.

Im ungestörten Betrieb ist durch das PWM-Modul des Mikrocontrollers sichergestellt, dass zwischen den Schaltvorgängen jedes PWM-Ausgangs und dessen Komplement eine Verzögerungszeit eingefügt ist. Eine detaillierte Beschreibung der PWM befindet sich im Abschnitt 3.3.1 „PWM-Modul zur Motoransteuerung“ auf Seite 32

Für den Fall, dass die Firmware im Mikrocontroller aus irgendeinem Grund versagt, ist zumindest sichergestellt, dass ein „Hängenbleiben“ der Ausgänge in einem undefinierten Zustand keine Schädigung oder Zerstörung der Leistungselektronik mit sich bringt.

Verlustleistung

Die Verlustleistung der Ansteuerung für den N-Kanal MOSFET beschränkt sich auf Schaltverluste, welche durch das Umladen des Gates verursacht werden. Zur Ermittlung der Verlustleistung wird die im Gate gespeicherte Energie, entsprechend Gleichung (3.8), sowie die Anzahl der Schalthandlungen herangezogen. Zu beachten ist hier, dass sowohl der Ladevorgang als auch der Entladevorgang zur Erwärmung des Gatters führen. Für die drei Gatter der gesamten Schaltung des Motortreibers ergibt sich folgende Verlustleistung:

$$P_{V\,IC_1} = 3 \cdot (\Delta U_{GS})^2 \cdot C_G \cdot f = 3 \cdot (5V)^2 \cdot 3{,}1nF \cdot 20kHz = 4{,}65mW$$

Die Eingänge des vierten Gatters der integrierten Schaltung liegen auf *high*-Pegel, während der Ausgang nicht beschaltet ist. Somit hat dieses Gatter keinen nennenswerten Anteil an der Verlustleistung, und die hier gezeigte Leistung entspricht den gesamten Verlusten der integrierten Schaltung.

3.1.4. Verlustleistung der Halbbrücke

Um eine Aussage betreffend der thermischen Belastung der Halbbrücke machen zu können, wird zwischen statischen Verlusten und Schaltverlusten unterschieden. Beiden Anteilen ist gemeinsam, dass sie zur Erwärmung der Leistungstransistoren führen. Bedingt durch die Pulsweitenmodulation sind die MOSFETs im Mittel nur für die halbe Zeit eingeschaltet. Dadurch reduziert sich der Anteil der statischen Verluste auf die Hälfte. Die nachstehende Berechnung zeigt die Verlustleistung für eine Halbbrücke.

$$\begin{aligned} P_V &= \frac{P_{V\,N}}{2} + \frac{P_{V\,P}}{2} + P_{V\,N\,on} + P_{V\,N\,off} + P_{V\,P\,on} + P_{V\,P\,off} \\ &= 0{,}5 \cdot (0{,}85W + 0{,}893W) + 0{,}1W + 0{,}1W + 0{,}3W + 0{,}24W = 1{,}61W \end{aligned}$$

Diese Verlustleistung führt zur einer Erwärmung der Sperrschicht um 100,6°C gegenüber der Umgebung.

$$\Delta\vartheta = P_{V\,T_2} \cdot R_{th(j-a)} = 1{,}61W \cdot 62{,}5^K/_W = 100{,}6°C$$

Die Leistungstransistoren sind im Gegensatz zu allen anderen Bauteilen thermisch mit dem Motorgehäuse verbunden. Bei einer maximal zulässigen Sperrschichttemperatur von 150°C darf die Temperatur des Motorgehäuses bis zu 49°C betragen. Es wurde dabei nicht berücksichtigt, dass über die Drain-Anschlüsse eine thermische Kopplung von $R_{th(j-D)} = 40^K/_W$ zur Leiterplatte besteht, welche ebenfalls Wärme vom Transistor abführt. Demnach könnte die Temperatur des Motorgehäuses sogar noch etwas höher liegen, was allerdings bezüglich des Motors nicht wünschenswert ist. Erhöhte Temperaturen führen nämlich zur Schwächung des Feldes der Permanentmagneten des Rotors, wodurch in weiterer Folge der Wirkungsgrad abnimmt.

In den beiden folgenden Abschnitten soll nun näher darauf eingegangen werden, wie die zuvor gezeigten Verluste zustande kommen.

Statische Verluste

Die statischen Verluste lassen sich recht einfach aus dem Drain-Strom sowie dem ON-Widerstand berechnen. Zusätzlich ist noch der positive Temperaturkoeffizient des $R_{DS\,on}$ bei MOSFETs zu beachten. Dieser hat nämlich zur Folge, dass eine steigende Temperatur zu einem Anstieg des ON-Widerstandes führt, und somit die Stromtragfähigkeit abnimmt.

$$\begin{aligned} P_V &= I_D^2 \cdot R_{DS\,on} \cdot R_{DS\,on\,norm@\vartheta} \\ P_{V\,N} &= (5A)^2 \cdot 0{,}02\Omega \cdot 1{,}7 = 0{,}85W \\ P_{V\,P} &= (5A)^2 \cdot 0{,}021\Omega \cdot 1{,}7 = 0{,}893W \end{aligned}$$

Für die Berechnungen wurde die höchst zulässige Sperrschichttemperatur von $\vartheta_j = 150°C$ angenommen.

Dynamische Verluste

Im Abschnitt 3.1.2 wurde bereits erwähnt, dass Schaltvorgänge bei Transistoren zu Verlusten führen, die von der Schaltzeit abhängen. Weiters ist hier auch die Art der zu schaltenden Last zu berücksichtigen: Bei einer ohm'schen Last nimmt der Strom während des Abschaltvorgangs ab, hingegen bei einer induktiven Last bleibt der Strom näherungsweise gleich. Erst wenn die Spannung U_{DS} am abschaltenden Transistor die Versorgungsspannung erreicht, beginnt die Source-Drain-Diode des komplementären MOSFETs zu leiten, und übernimmt ihrerseits den Strom. Sofern die in der Induktivität gespeicherte Energie größer ist als die Energie, die während des Schaltvorgangs im Transistor in Wärme umgewandelt wird, lässt sich für das Ausschalten einer induktiven Last die Verlustenergie an einem MOSFET folgendermaßen berechnen:

$$W_V = \int_0^{t_{switch}} U \cdot I \cdot \frac{t}{t_{switch}} \mathrm{d}t = \frac{1}{2} \cdot U \cdot I \cdot t_{switch} = \frac{1}{2} \cdot 24V \cdot 5A \cdot 200ns = 12\mu Ws$$

Im Vergleich dazu ist in der Induktivität der Motorwicklung eine viel größere Energie gespeichert:

$$W_L = \frac{I^2}{2} \cdot L = \frac{(5A)^2}{2} \cdot 0{,}8mH = 10mWs$$

Der Wert des induktiven Anteils des Motors wurde aus dessen Datenblatt [26] entnommen.

Aus der Verlustenergie des MOSFETs während des Ausschaltvorgangs lässt sich mit der Schaltfrequenz die dynamische Verlustleistung berechnen:

$$\begin{aligned}
P_{V\,switch} &= W_V \cdot f = 0{,}5 \cdot U \cdot I \cdot t_{switch} \cdot f \\
P_{V\,N\,on} &= 0{,}5 \cdot 24V \cdot 5A \cdot 80ns \cdot 20kHz = 0{,}1W \\
P_{V\,N\,off} &= 0{,}5 \cdot 24V \cdot 5A \cdot 80ns \cdot 20kHz = 0{,}1W \\
P_{V\,P\,on} &= 0{,}5 \cdot 24V \cdot 5A \cdot 250ns \cdot 20kHz = 0{,}3W \\
P_{V\,P\,off} &= 0{,}5 \cdot 24V \cdot 5A \cdot 200ns \cdot 20kHz = 0{,}24W
\end{aligned}$$

An dieser Stelle seien noch ein paar Überlegungen zum Einschaltvorgang angeführt: Wenn die Motorwicklung nicht stromdurchflossen, und demnach auch keine Energie in der Wicklung vorhanden war, dann wird der Einschaltvorgang beinahe verlustfrei ablaufen. Das lässt sich damit begründen, dass die Schaltzeit des Leistungstransistors viel kürzer ist, als die Zeitkonstante der Motorwicklung.

$$\tau = \frac{L}{R} = \frac{L_{Motor}}{R_{Motor} + R_{DS\,on}} = \frac{0{,}8mH}{0{,}55\Omega + 21m\Omega \cdot 1{,}7} = 1{,}37ms$$

Wenn die Motorwicklung bereits stromdurchflossen war, dann wird dieser Strom für die Dauer des Schaltvorgangs näherungsweise gleich bleiben. Sobald durch den abschaltenden Transistor kein Strom mehr fließt, wird dieser vorerst über die Bodydiode des komplementären Transistors geleitet. Wenn nun der komplementäre Transistor eingeschaltet wird, übernimmt dieser den Strom der Motorwicklung. Die Abschätzung der Verlustleistung des Ausschaltvorgangs kann daher nach denselben Überlegungen erfolgen wie für den Einschaltvorgang.

3.2. Messung des Motorstroms

Wie bereits in Abschnitt 2.3.1 (Statorstrom) auf Seite 7 ausführlich beschrieben wurde, ist es für die feldorientierte Regelung notwendig, dass alle Phasenströme zur Bestimmung der Raumvektorkomponenten bekannt sind. Es gibt nun unterschiedliche Strategien, wie die 3 Phasenströme des Motors erfasst werden können:

Direkte Messung aller Phasenströme Bei dieser Messvariante werden alle Phasenströme direkt am Motoranschluss gemessen. Dies kann auf unterschiedliche Art und Weise erfolgen, beispielsweise durch Messen der Spannung an einem Shunt oder mittels Stromsensoren. Die erste Variante stellt dabei hohe Anforderungen an den Messverstärker, so muss dessen Gleichtaktaussteuerbarkeit über die gesamte Versorgungsspannung des Motors gewährleistet sein und zudem eine hohe Gleichtaktunterdrückung aufweisen.

Bei den Stromsensoren handelt es sich im Prinzip um Stromwandler. Die Messung des Stroms basiert im wesentlichen auf der Erfassung dessen Magnetfeldes. Dies kann durch Induktion in einer Leiterschleife sowie mittels Hallsensoren erfolgen, aber auch eine Kombination ist möglich, wie dies beim Kompensationswandler der Fall ist. Auf Grund der ausschließlich magnetischen Kopplung ist bei Stromsensoren eine galvanische Trennung vom Strompfad gewährleistet, weshalb diese Messmethode bevorzugt dort eingesetzt wird, wo der zu erfassende Strom auf gefährlichem Potential liegt.

Direkte Messung von 2 Phasenströmen Sofern der Motor keinen Defekt aufweist, darf man dessen Wicklungen hinsichtlich elektrischer Parameter als symmetrisch annehmen. Weiters gilt für Mehrphasensysteme, dass bei symmetrischer Last die Summe der Ströme Null ist. Somit kann man die zuvor beschriebene Methode auf die Messung der Ströme zweier beliebiger Phasen vereinfachen. Der Strom der nicht gemessenen Phase lässt sich mathematisch ermitteln:

$$\mathbf{I_A} + \mathbf{I_B} + \mathbf{I_C} = 0 \Rightarrow \mathbf{I_C} = -\left(\mathbf{I_A} + \mathbf{I_B}\right)$$

Somit sind wieder alle für die Raum-Vektor-Berechnung notwendigen Ströme vorhanden, und zudem ist der schaltungstechnische Aufwand gegenüber der ersten Variante geringer.

Beiden Varianten ist jedoch gemeinsam, dass der Platzbedarf wesentlich größer ist, als für jene Strommessung, die im Folgenden beschrieben wird.

Messung an der Anspeisung der Halbbrücke Wenn keine galvanische Trennung des Leistungsteils von der Ansteuerung notwendig ist, und die Versorgungsanschlüsse der Halbbrücken voneinander getrennt ausgeführt sind, dann kann man den Shunt auch in die Anspeisung jeder Halbbrücke schalten. Der Vorteil dieses Ansatzes liegt darin, dass das Bezugspotential konstant ist, womit die Anforderung der hohen Gleichtaktaussteuerbarkeit des Messverstärkers wegfällt.

Die Messung des Stroms soll mit dem im Mikrocontroller integrierten Analog-Digital-Wandler erfolgen. Das Bezugspotential des Mikrocontrollers liegt auf derselben Masse wie der Leistungsteil, deshalb wird der Strommesswiderstand R_{shunt}, wie bereits in Abbildung 3.1 gezeigt, in die Low-Side der Halbbrücke geschaltet.

Die Spannung am Shunt kann nun mit einem gewöhnlichen nicht-invertierenden Verstärker, wie in Abbildung 3.2, an den Spannungsbereich des Analog-Digital-Wandlers angepasst werden. Des Weiteren sind in dieser Schaltung noch zusätzliche Bauteile vorhanden, die zur Einstellung des Arbeitspunktes und zur Begrenzung der Bandbreite dienen, sowie die Stabilität des Verstärkers sicherstellen.

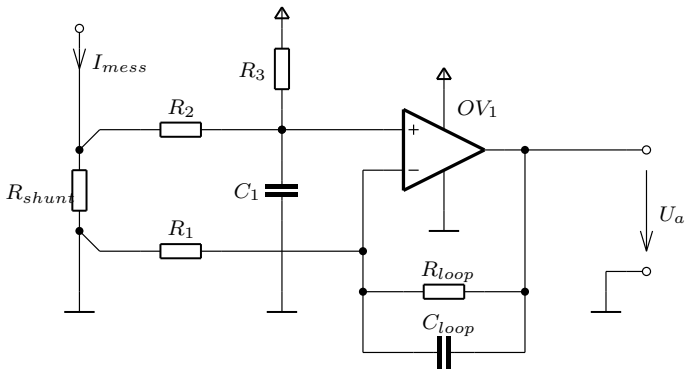

Abbildung 3.2.: Strommessverstärker

3.2.1. Dimensionierung des Messverstärkers

Der Messverstärker soll nun nach folgenden Gesichtspunkten dimensioniert werden:

- Der Verstärker soll mit einer einfachen, massebehafteten Spannung versorgt werden.
- Die am Shunt abfallende Spannung soll soweit verstärkt werden, dass bei den höchsten zu erwartenden Motorströmen der Eingangsbereich des Analog-Digital-Wandlers möglichst voll ausgesteuert wird.
- Der Mikrocontroller misst den Strom synchron zur PWM. Entsprechend des Abtasttheorems nach Shannon sollte das Nutzsignal unter der halben Abtastfrequenz liegen. Der Verstärker sollte im Bereich der halben PWM-Frequenz eine Dämpfung von zirka $20dB$ aufweisen. Dieser Wert stellt einen Kompromiss zwischen verkraftbarem Fehler durch Aliasing und geringer Phasendrehung der höchsten Frequenz des Nutzsignals dar.
- Der Messverstärker soll auf jeden Fall stabil sein.

Dimensionierung des Shunts

Beim Shunt-Widerstand handelt es sich um ein Bauteil im 1206-SMD Gehäuse. Widerstände in dieser Bauform können gewöhnlich eine thermische Leistung von $250mW$ abführen. Der maximale Strom des Motors liegt bei etwa $I_{max} = 5A$, wobei zusätzlich nach oben hin eine Toleranz von zirka 40% berücksichtigt werden soll. Der Shunt soll nun so bemessen werden, dass unter Berücksichtigung dieser Toleranz die zulässige Verlustleistung nicht überschritten wird. Entsprechend dieser Anforderung ergibt sich aus folgender Gleichung (3.10) ein Widerstand von $R_{shunt} = 5{,}1m\Omega$. Der Shunt wird mit $R_{shunt} = 5m\Omega$ gewählt, da dies der nächstliegende erhältliche Widerstandswert ist.

$$P = I^2 \cdot R \Rightarrow R = \frac{P}{I^2} = \frac{250mW}{(7A)^2} = 5{,}1m\Omega \tag{3.10}$$

Einfache, massebehaftete Versorgung

[12] Damit der Verstärker mit einer einfachen, massebehafteten Spannung versorgt werden kann und dabei gleichzeitig in der Lage ist, massebehaftete Signale zu verstärken, muss dieser über eine entsprechende Eingangsstruktur verfügen. Es ist hier die Rede von einem sogenannten *Single Supply Verstärker*. Beim Operationsverstärker (kurz OV) *LM324* handelt es sich um ein derartiges Bauteil. Verstärker dieser Art haben allerdings auch ihre Tücken, so kommt es beispielsweise zu einer Phasenumkehr am Ausgang, wenn am Eingang das Massepotential um zirka $0{,}4V$ unterschritten wird. Die Ruhespannung am Eingang des Verstärkers sollte daher so eingestellt werden, dass ein Unterschreiten der zulässigen Eingangsspannung vermieden wird.

Die Widerstände R_3, R_2 und R_{shunt} bilden einen Spannungsteiler, der am nicht-invertierenden Eingang des OVs den Arbeitspunkt einstellt. R_{shunt} ist im Vergleich zu R_2 mindestens um einen Faktor 10^5 kleiner und kann daher für weitere Betrachtungen vernachlässigt werden. Dem Datenblatt [16] des Operationsverstärkers *LM324* kann man entnehmen, dass die Offsetspannung höchstens $\pm 9mV$ beträgt. Der Motor kann auch als Generator arbeiten, beispielsweise bei Lastwechsel oder beim Entschleunigen. Dementsprechend muss die Ruhespannung am nicht-invertierenden Eingang so eingestellt werden, dass bei einem maximalen negativen Strom von $I = -7A$ sowie einer größtmöglichen Offsetspannung von $U_{os} = \pm 9mV$ auf jeden Fall eine positive Aussteuerung des Verstärkers gewährleistet ist. Entsprechend nachfolgender Gleichung muss die Ruhespannung mindesten $44mV$ betragen.

$$U_{bias\,min} \geq U_{os} + I \cdot R_{shunt} = 9mV + 7A \cdot 5m\Omega = 44mV$$

Werden die Widerstände $R_2 = 5{,}6k\Omega$ und $R_3 = 470k\Omega$ gesetzt, dann liegt entsprechend der Gleichung (3.11), unter Berücksichtigung einer Bauteiltoleranz von 10%, bei einer Versorgungsspannung von $5V$ am nicht-invertierenden Eingang eine Spannung von mindestens $48{,}3mV$ an.

$$\begin{aligned}
U_{+} &= U \cdot \frac{R_2}{R_2 + R_3} = 5V \cdot \frac{5{,}6k\Omega}{5{,}6k\Omega + 470k\Omega} = 58{,}9mV \\
U_{+\,min} &= U \cdot \frac{R_{2\,min}}{R_{2\,min} + R_{3\,max}} = 5V \cdot \frac{5{,}6k\Omega \cdot 0{,}9}{5{,}6k\Omega \cdot 0{,}9 + 470k\Omega \cdot 1{,}1} = 48{,}3mV \\
U_{+\,max} &= U \cdot \frac{R_{2\,max}}{R_{2\,max} + R_{3\,min}} = 5V \cdot \frac{5{,}6k\Omega \cdot 1{,}1}{5{,}6k\Omega \cdot 1{,}1 + 470k\Omega \cdot 0{,}9} = 71{,}8mV
\end{aligned} \tag{3.11}$$

Verstärkung

Die maximal zulässige Verstärkung ergibt sich aus dem Verhältnis der Referenzspannung des ADCs, und der größtmöglichen Spannung, welche am nicht-invertierenden Eingang des Operationsverstärkers anliegt. Diese Spannung setzt sich zusammen aus der Offsetspannung, der Ruhespannung sowie der Spannung am Shunt. Aus Gleichung (3.12) ergibt sich dementsprechend eine Verstärkung von $29{,}1dB$.

$$\begin{aligned} A_{Gain\,max} &= \frac{U_{ref}}{U_{shunt} + \cdot U_{os} + U_{+\,max}} = \frac{U_{ref}}{\Delta I \cdot R_{shunt} + U_{os} + U_{+\,max}} \\ &= \frac{3{,}3V}{7A \cdot 5m\Omega + 9mV + 71{,}8mV} = 28{,}5 \Rightarrow 29{,}1dB \end{aligned} \tag{3.12}$$

Ein Blick in das Datenblatt [16] des Operationsverstärkers *LM324* gibt Auskunft über dessen Frequenzgang. Dieser ist abhängig von der Versorgungsspannung und weist im schlechtesten Fall bei $20kHz$ eine Verstärkung von $30dB$ auf. Der OV *LM324* ist demnach für die Schaltung geeignet.

Die Verstärkung wird mit den beiden Widerständen R_1 und R_{loop} entsprechend Gleichung (3.13) eingestellt. Es handelt sich hierbei um die Gleichspannungsverstärkung.

$$\frac{U_a}{U_{shunt}} = 1 + \frac{R_{loop}}{R_1} \tag{3.13}$$

Wählt man $R_1 = 5{,}6k\Omega$ und $R_{loop} = 120k\Omega$, dann erhält man eine Verstärkung von $27dB$. Die in Gleichung (3.12) gezeigte maximal zulässige Verstärkung wird selbst unter Berücksichtigung der Bauteiltoleranzen in der Größenordnung von 5% nicht überschritten.

$$\begin{aligned} A_{Gain} &= 1 + \frac{120k\Omega}{5{,}6k\Omega} = 22{,}4 \Rightarrow 27dB \\ A_{Gain\,max} &= 1 + \frac{120k\Omega \cdot 1{,}05}{5{,}6k\Omega \cdot 0{,}95} = 24{,}7 \Rightarrow 27{,}8dB \end{aligned}$$

Abweichungen der Bauteile R_1 und R_{loop} von deren nominellen Werten wirken sich direkt auf die Verstärkung aus. Die hier angenommene Toleranz von 5% führt im schlechtesten Fall zu einem Fehler des Messverstärkers von zirka 10%. Zwecks Messgenauigkeit sollten für die Verstärkung Widerstände mit größeren Toleranzen vermieden werden. Die Toleranzen der Widerstände R_2 und R_3 zur Einstellung der Ruhespannung sind dagegen weniger kritisch. Der daraus resultierende Offsetfehler wird in der Software kompensiert, indem die Ausgangsspannung jedes Verstärkers vor Inbetriebnahme des Motors gemessen wird. Dieser Wert wird anschließend im Betrieb nach jeder Messung subtrahiert.

Frequenzgang des Messverstärkers

Um die Dynamik des Strommessverstärkers beschreiben zu können ist es notwendig, die gesamte Strecke des Verstärkers mit allen frequenzabhängigen Bauteilen als Übertragungsfunktion darzustellen. R_{loop} in Gleichung (3.13) wird durch die Parallelschaltung mit C_{loop} ersetzt. Weiters wird die Gleichung um den eingangsseitigen Tiefpassfilter bestehend aus R_2 und C_1 ergänzt. Gleichung (3.14) zeigt die daraus resultierende Übertragungsfunktion. Man muss an dieser Stelle noch beachten, dass die Übertragungsfunktion durch den Frequenzgang des Operationsverstärkers begrenzt wird.

$$\begin{aligned} T(\mathrm{j}\omega) &= \frac{\frac{1}{\mathrm{j}\omega C_1}}{R_2 + \frac{1}{\mathrm{j}\omega C_1}} \cdot \left(1 + \frac{R_{loop} || \frac{1}{\mathrm{j}\omega C_{loop}}}{R_1}\right) \\ &= \frac{1}{\mathrm{j}\omega R_2 C_1 + 1} \cdot \left(1 + \frac{R_{loop}}{R_1}\right) \cdot \frac{\mathrm{j}\omega \frac{C_{loop} R_{loop} R_1}{R_{loop} + R_1} + 1}{\mathrm{j}\omega C_{loop} R_{loop} + 1} \end{aligned} \tag{3.14}$$

Diese Übertragungsfunkion lässt sich auf folgende, für Bode-Diagramme übliche, Form bringen:

$$T(\mathrm{j}\omega) = \frac{1}{\frac{\mathrm{j}\omega}{\Omega_0} + 1} \cdot K \cdot \frac{\frac{\mathrm{j}\omega}{\Omega_1} + 1}{\frac{\mathrm{j}\omega}{\Omega_2} + 1} \tag{3.15}$$

Vergleicht man die beiden Frequenzen Ω_1 und Ω_2 miteinander, dann erkennt man, dass das Verhältnis dieser beiden Frequenzen von den Widerständen R_1 und R_{loop} abhängt.

$$\frac{\Omega_1}{\Omega_2} = \frac{R_{loop} + R_1}{C_{loop} R_{loop} R_1} \cdot R_{loop} C_{loop} = \frac{R_{loop} + R_1}{R_1} = 1 + \frac{R_{loop}}{R_1}$$

Dieses Verhältnis entspricht der Gleichspannungsverstärkung aus Gleichung (3.13) und beträgt somit 22,4. Demnach liegt die Amplitudenanhebung mit der Knickfrequenz Ω_1 etwas mehr als eine Dekade über der Grenzfrequenz Ω_2 des Tiefpasses. Wenn die Frequenzen Ω_0 und Ω_2 näherungsweise gleich gewählt werden, dann weist die Schaltung bis zur Frequenz Ω_2 Tiefpassverhalten 2. Ordnung auf. In diesem Fall liegt eine Dämpfung um $40dB$ pro Dekade vor, und dem Wunsch einer Dämpfung von $20dB$ bei halber PWM-Frequenz wird Rechnung getragen, indem die Frequenzen Ω_0 und Ω_2 bei einem Sechstel der PWM-Frequenz gewählt werden.

$$\Omega_0 \approx \Omega_2 \approx \frac{f_{PWM}}{6} = \frac{18kHz}{6} = 3kHz$$

Dementsprechend werden die Kondensatoren C_1 und C_{loop} so gewählt, dass die beiden Frequenzen Ω_0 und Ω_2 etwas unter $3kHz$ liegen.

$$\begin{aligned} K &= \left(1 + \frac{R_{loop}}{R_1}\right) = 1 + \frac{120k\Omega}{5{,}6k\Omega} = 22{,}4 \Rightarrow 27dB \\ \Omega_0 &= \frac{1}{R_2 C_1} = \frac{1}{5{,}6k\Omega \cdot 10nF} = 17{,}9 \cdot 10^3 1/s \Rightarrow 2{,}84kHz \\ \Omega_1 &= \frac{R_{loop} + R_1}{C_{loop} R_{loop} R_1} = \frac{120k\Omega + 5{,}6k\Omega}{470pF \cdot 120k\Omega \cdot 5{,}6k\Omega} = 398 \cdot 10^3 1/s \Rightarrow 63{,}3kHz \\ \Omega_2 &= \frac{1}{R_{loop} C_{loop}} = \frac{1}{120k\Omega \cdot 470pF} = 17{,}7 \cdot 10^3 1/s \Rightarrow 2{,}82kHz \end{aligned}$$

Der Verstärkungsfaktor K entspricht der Gleichspannungsverstärkung aus Gleichung (3.13).

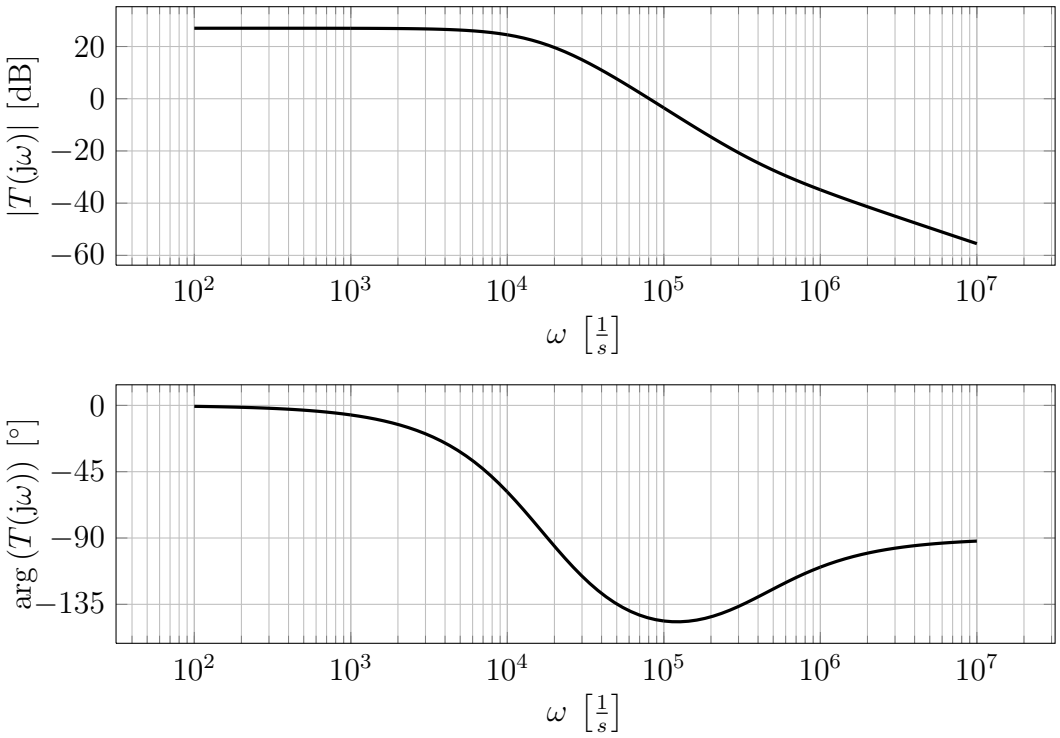

Abbildung 3.3.: Frequenzgang des Strommessverstärkers

Abbildung 3.3 zeigt den Frequenzgang des Messverstärkers. Wie bereits erwähnt ist hier noch zusätzlich die Leerlaufverstärkung des OVs zu berücksichtigen, dessen Frequenzgang den hier gezeigten beschneiden kann. Da die Gerade des Verstärkungs-Bandbreite-Produkts des Operationsverstärkers den gezeigten Frequenzgang einhüllt, ohne diesen zu schneiden, hätte eine Berücksichtigung des Frequenzgangs des OVs keine nennenswerte Änderung der Übertragungsfunktion des Messverstärkers zufolge.

3.2.2. Stabilität des Messverstärkers

Die Stabilität des Strommessverstärkers wird anhand der offenen Schleife untersucht. Abbildung 3.4 zeigt dazu die entsprechende Schaltung. [4] Mit Hilfe des Nyquist-Kriteriums ist es möglich, die Stabilität zu beurteilen. Sofern folgende Punkte erfüllt sind, kann ein vereinfachtes Kriterium angewendet werden:

- Die offene Schleife $L(\mathrm{j}\omega)$ besitzt Tiefpasscharakter
- Die Verstärkung von $L(\mathrm{j}\omega)$ ist positiv
- Alle Pole weisen einen negativen Realteil auf, mit Ausnahme eines eventuell vorhandenen Pols bei Null.
- Der Betrag des Frequenzgangs der offenen Schleife nimmt nur an einer Stelle den Wert 1 an. Die Frequenz an dieser Stelle wird als Durchtrittsfrequenz bezeichnet.

Von Interesse ist nun der Phasenwinkel an der Stelle der Durchtrittsfrequenz. Die Differenz dieses Winkels zu 180° wird Phasenreserve ϕ_r genannt. Die Phasenreserve muss auf jeden Fall positiv sein, damit die Schaltung stabil ist. Weiters hängt das Überschwingen $\ddot{U}$ der geschlossenen Schleife von der Phasenreserve ab, wobei näherungsweise folgender Zusammenhang gilt:

$$\phi_r = 70\% - \ddot{U} \tag{3.16}$$

Für den Fall, dass die Phasenreserve nun größer als 70° ist, gibt es demnach kein Überschwingen.

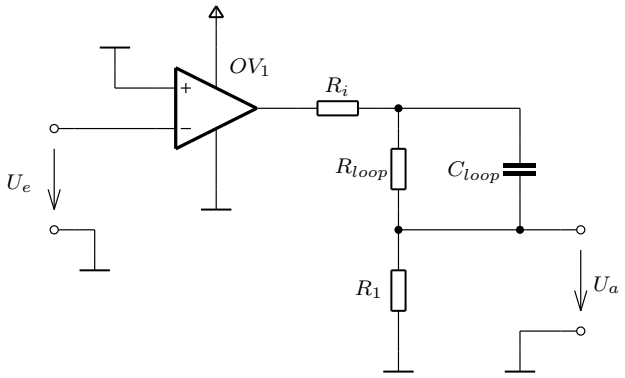

Abbildung 3.4.: Strommessverstärker - offene Schleife

Aus der Schaltung der offenen Schleife (Abbildung 3.4) lässt sich die Übertragungsfunktion $L(\mathrm{j}\omega)$ entsprechend Gleichung (3.17) aufstellen.

$$\begin{aligned} L(\mathrm{j}\omega) &= A_{OV_1} \cdot \frac{1}{\frac{\mathrm{j}\omega}{\Omega_{1\,OV_1}} + 1} \cdot \frac{1}{\frac{\mathrm{j}\omega}{\Omega_{2\,OV_1}} + 1} \cdot \frac{R_1}{R_i + R_1 + \frac{R_{loop} \cdot \frac{1}{\mathrm{j}\omega C_{loop}}}{R_{loop} + \frac{1}{\mathrm{j}\omega C_{loop}}}} \\ &= A_{OV_1} \cdot \frac{1}{\frac{\mathrm{j}\omega}{\Omega_{1\,OV_1}} + 1} \cdot \frac{1}{\frac{\mathrm{j}\omega}{\Omega_{2\,OV_1}} + 1} \cdot \frac{R_1}{R_i + R_1 + R_{Loop}} \cdot \frac{\mathrm{j}\omega C_{loop} R_{loop} + 1}{\mathrm{j}\omega \frac{C_{loop} R_{loop} (R_1 + R_i)}{R_1 + R_{loop} + R_i} + 1} \end{aligned} \tag{3.17}$$

Ähnlich wie beim Frequenzgang des Verstärkers lässt sich auch die Übertragungsfunktion der offenen Schleife auf eine für Bode-Diagramme übliche Form bringen:

$$L(\mathrm{j}\omega) = A_{OV_1} \cdot \frac{1}{\frac{\mathrm{j}\omega}{\Omega_{1\,OV_1}} + 1} \cdot \frac{1}{\frac{\mathrm{j}\omega}{\Omega_{2\,OV_1}} + 1} \cdot K \cdot \frac{\frac{\mathrm{j}\omega}{\Omega_2} + 1}{\frac{\mathrm{j}\omega}{\Omega_3} + 1} \tag{3.18}$$

Es gelten dabei folgende Konstanten und Frequenzen:

$$
\begin{aligned}
A_{OV_1} &= 3{,}5 \cdot 10^5 \\
\Omega_{1\,OV_1} &= 12{,}57 s^{-1} \\
\Omega_{2\,OV_1} &= 6{,}28 \cdot 10^6 s^{-1} \\
K &= \frac{R_1}{R_i + R_1 + R_{Loop}} = \frac{5{,}6k\Omega}{100\Omega + 5{,}6k\Omega + 120k\Omega} = 4{,}46 \cdot 10^{-2} \\
\Omega_2 &= \frac{1}{R_{loop} C_{loop}} = \frac{1}{120k\Omega \cdot 470pF} = 17{,}7 \cdot 10^3 s^{-1} \\
\Omega_3 &= \frac{R_1 + R_{loop} + R_i}{C_{loop} R_{loop} (R_1 + R_i)} = \frac{5{,}6k\Omega + 120k\Omega + 100\Omega}{470pF \cdot 120k\Omega \cdot (5{,}6k\Omega + 100\Omega)} = 391 \cdot 10^3 s^{-1}
\end{aligned}
$$

Die Parameter des Operationsverstärkers sind dem Datenblatt [16] (*LM324*) entnommen, wobei A_{OV_1} die Leerlaufverstärkung und $\Omega_{1\,OV_1}$ sowie $\Omega_{2\,OV_1}$ die beiden Knickfrequenzen des OVs sind. Die zweite Frequenz geht nicht eindeutig aus dem Datenblatt hervor und wird daher knapp oberhalb der Transitfrequenz mit $1MHz$ angenommen. Der Innenwiderstand R_i geht ebenfalls nicht eindeutig aus dem Datenblatt hervor und wurde deshalb einem Spice-Model des Verstärkers entnommen.

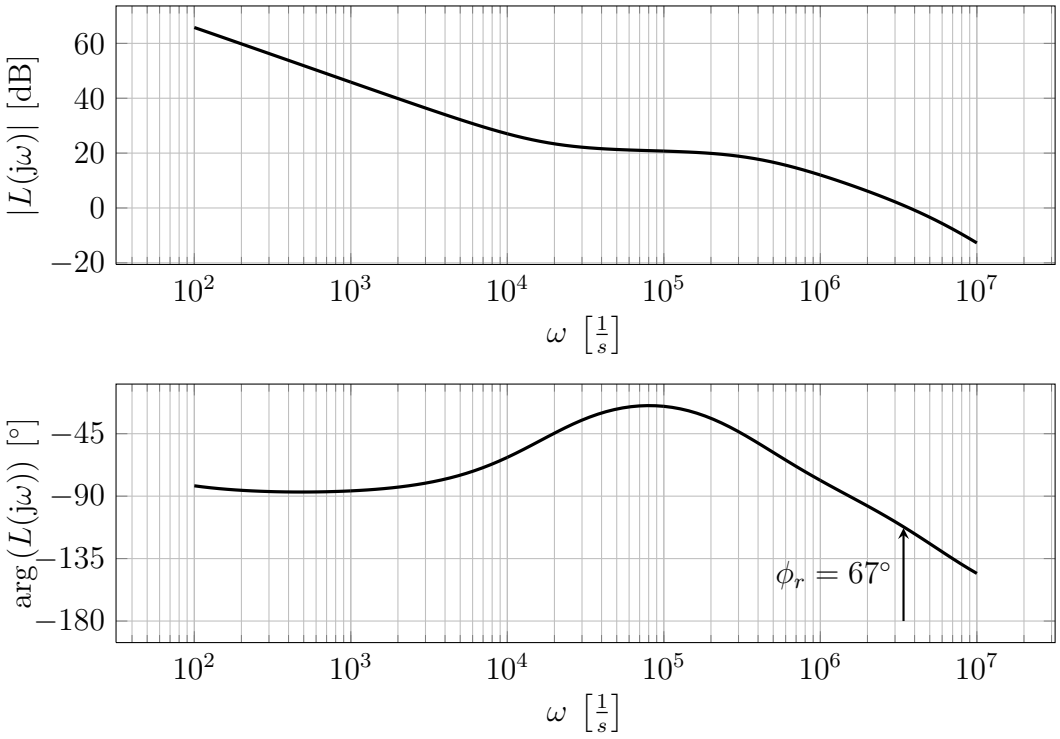

Abbildung 3.5.: Frequenzgang der offene Schleife

Wie man aus Abbildung 3.5 erkennen kann, liegt hier eine Phasenreserve von $\phi_r \approx 65°$ vor. Grundsätzlich ist die Schaltung stabil, jedoch liegt hier entsprechend Gleichung (3.16) ein Überschwingen von 5% der Impulsantwort vor.

Man kann das Überschwingverhalten verbessern, indem der Schleifenkondensator verkleinert wird. Dadurch verschiebt sich der Bereich der Phasenanhebung näher zur Durchtrittsfrequenz, und es bleibt somit eine größere Phasenreserve, bei der letztendlich auch kein Überschwingen auftritt. Allerdings ändert sich damit auch der Frequenzgang der geschlossenen Schleife.

3.3. Mikrocontroller und Peripherie

In den folgenden Abschnitten werden der Mikrocontroller sowie die externen Komponenten Bedienteil, Datenschnittstelle und elektrisch betätigte Bremse behandelt. Eine detaillierte Beschreibung des Mikrocontrollers füllt ganze Bücher, daher beschränken sich die Erläuterungen auf jene Teile, die für die Realisierung der Elektronik zur Ansteuerung bürstenloser Motoren von Relevanz sind. Die externen Komponenten werden im Gegensatz dazu vor allem aus schaltungstechnischer Sicht näher betrachtet.

3.3.1. Mikrocontroller

Den Kern der Schaltung bildet der Mikrocontroller *STM32F103*des Halbleiterherstellers STMicroelectronics. Der Prozessor dieses Controllers ist eine **A**dvanced **R**ISC **M**achine ARM *Cortex-M3*. [1] Aus der Bezeichnung des Prozessors geht hervor, dass es sich um einen **R**educed **I**nstruction **S**et **C**omputer (RISC) handelt. RISC-Architekturen sind dafür bekannt, jeden Befehl ihres Instruktionssatzes in nur einem Taktzyklus auszuführen. Bemerkenswert beim Cortex-M3 ist, dass die Multiplikation von 32-Bit Zahlen zum Befehlssatz gehört und ebenfalls nur einen Takt benötigt. Die Instruktionen des Cortex-M3 sind ausschließlich in 16-Bit codiert. Da die Prozessorarchitektur 32-Bit breit ist, werden jeweils zwei Befehle gleichzeitig geladen. Dies hat den Vorteil, dass im Falle eines Sprunges nur ein Taktzyklus notwendig ist, und nicht die gesamte Prozessor-Pipline neu befüllt werden muss.

Der Prozessorkern wird von der Firma ARM Limited entwickelt und als fertige Lösung den Halbleiterherstellern angeboten. Diese Tatsache ermöglichte letzteren sich bei der Entwicklung des Mikrocontrollers auf dessen periphere Funktionalität zu konzentrieren und führte dazu, dass ARM zum de facto Standard für eingebettete Systeme herangewachsen ist. Speziell in Anwendungen, deren Anforderungen die Leistungsfähigkeit von 8-Bit oder 16-Bit Mikrocontrollern übersteigt, jedoch Energiesparsamkeit trotzdem ein wichtiges Kriterium ist, sind bis auf wenige Ausnahmen ARM-Architekturen im Einsatz. Die Transformationen und Berechnungen, welche für das Funktionieren der in Abschnitt 2.3 ab Seite 6 vorgestellten feldorientierten Regelung notwendig sind, verlangen einiges an Prozessorleistung, sodass der Einsatz eines ARM-basierten Mikrocontrollers durchaus gerechtfertigt erscheint.

[9] [29]der Mikrocontroller *STM32F103*weist hinsichtlich seiner Peripherie einige Besonderheiten auf, die in den folgenden Abschnitten beschrieben werden. Dabei liegt das Hauptaugenmerk auf grundlegenden Konzepten, welche vor allem hinsichtlich der entwickelten Schaltung Vorteile und Vereinfachungen mit sich bringen.

Clock für Peripherie

Der Takt für die Peripherie ist in zwei Gruppen unterteilt, wobei dieser für jede Gruppe vom Systemtakt mit einem einstellbaren Teilerverhältnis von $1 : 2^n$ abgeleitet wird ($n = 0 \ldots 4$). Die Peripherie kann demnach mit jener Taktfrequenz versorgt werden, die tatsächlich notwendig ist. Weiters sind die Taktquellen jeder Peripherie Komponente des Mikrocontrollers extra schaltbar. Aus dieser Kombination „minimal notwendige Taktfrequenz“ und „nur takten was tatsächlich notwendig ist“, lässt sich trotz hoher Performance der Strom auf ein Minimum reduzieren.

Dual ADC-Modul

Zwei miteinander gekoppelte Analog-Digital-Wandler ermöglichen die gleichzeitige Erfassung zweier unterschiedlicher Signalquellen. Dies ist beispielsweise bei der Messung der Motorströme notwendig, da ein zeitlicher Versatz zu Fehlern bezüglich Phase und Amplitude des räumlichen Statorstroms führen. Des Weiteren gibt es zwei Prioritäten für die beiden ADCs. Der Start einer Messung mit hoher Priorität unterbricht die aktuelle Analog-Digital- Umsetzung niederer Priorität, um sofort die sogenannte injizierte Umsetzung zu starten. Danach wird beginnend bei der unterbrochenen Analog-Digital-Umsetzung neu gestartet.

CAN-Modul

Das im Mikrocontroller integrierte Control Area Network Modul genügt den Protokoll-Standards CAN 2.0A und B und unterstützt unter anderem die für sicherheitskritische Anwendungen relevante Time Triggered Option. Ein konfigurierbarer Filter für eingehende Nachrichten leitet nur die für die Anwendung tatsächlich notwendigen Nachrichten an einen FIFO-Speicher weiter, während alle anderen Daten verworfen werden. Im Zwischenspeicher können bis zu drei Nachrichten abgelegt werden. Die Kombination aus FIFO und Nachrichtenfilter, welche Teil der CAN-Hardware sind, ermöglicht eine Entlastung der CPU.

Zur Anbindung des CAN-Moduls an die physikalische Schicht des CAN-Busses ist lediglich ein externer Treiberbaustein notwendig.

PWM-Modul zur Motoransteuerung

Der Mikrocontroller *STM32F103*enthält einen so genannten *Advanced-Control Timer*, welcher gegenüber den restlichen Timer-Modulen einige technische Raffinessen zu bieten hat. Für die Motoransteuerung interessant ist vor allem das an diesen Timer gekoppelte PWM-Modul. [9] Dieses PWM-Modul besteht aus insgesamt vier voneinander unabhängigen PWM-Kanälen. Drei dieser Kanäle verfügen jeweils über zwei Ausgänge, welche so angesteuert werden, dass deren Schaltflanken zeitlich versetzt sind, wobei diese Zeit in einem Register eingestellt werden kann. Somit ist es möglich, die Verzögerungszeiten des unter Abschnitt 3.1 ab Seite 11 beschriebenen Motortreibers zu kompensieren, sodass die Leistungstransistoren ohne Überlappung geschaltet werden.

Der frei laufende Zähler des Advanced-Control Timers, der die Zeitbasis der PWM-Kanäle bildet, ist als auf-ab-Zähler konfiguriert. Der niedrigste und der höchste Zählerstand sind somit genau in der Mitte der high- und der low-Zeit der PWM-Signale. Dies nützt man mit dem vierten PWM-Kanal aus, welcher mit einem um 1 kleineren Wert als der höchste Wert des Zählers geladen ist. Dementsprechend liegt für diesen Kanal die steigende Flanke einen Zähltakt vor der Mitte des high-Pegels der drei anderen PWM-Kanäle am Ausgang an. Diese Flanke startet die Analog-Digital-Umsetzung mit hoher Priorität zur Erfassung des Motorstroms sowie des Winkelencoders. Auf diese Weise ist sichergestellt, dass die Messung des Motorstroms stets zu einem quasi eingeschwungenen Zustand durchgeführt wird, und nicht in der Nähe einer Schaltflanke der Pulsweitenmodulation, welche die Messung negativ beeinflussen könnte.

3.3.2. Bedienteil

Die Anforderungen an das Bedienteil sind nicht besonders groß, so ist es ausreichend, wenn es mit zwei Tasten ausgestattet ist. Die Bedienung gestaltet sich dabei folgendermaßen: Zur Wahl der Drehrichtung gibt es jeweils eine Taste, die den Motor in Bewegung versetzen, solange man eine der beiden Tasten gedrückt hält. Weiters soll auch ausgewertet werden, ob beide Tasten gleichzeitig gedrückt sind, damit beispielsweise ein Reset oder eine Selbstlernfunktion ausgeführt werden kann.

Am Mikrocontroller wären genügend Pins frei, um mehr als lediglich zwei Tasten auszulesen. Das Problem dabei ist, dass die Pins über eine Steckverbindung nach außen gebracht werden müssen, um daran das Bedienteil anschließen zu können. Da die Fläche der Leiterplatte recht wenig Platz bietet, wäre es vorteilhaft, wenn für das Bedienteil so wenig Anschlüsse wie möglich verwendet werden. Führt man nun zwei Pins des Mikrocontrollers sowie ein Bezugspotential aus, dann sind dafür drei Pins eines Steckverbinders notwendig. Gleichzeitig verbaut man sich damit aber auch die Möglichkeit, die Schaltung mit einem Bedienteil verbinden zu können, das mehr als zwei Tasten hat.

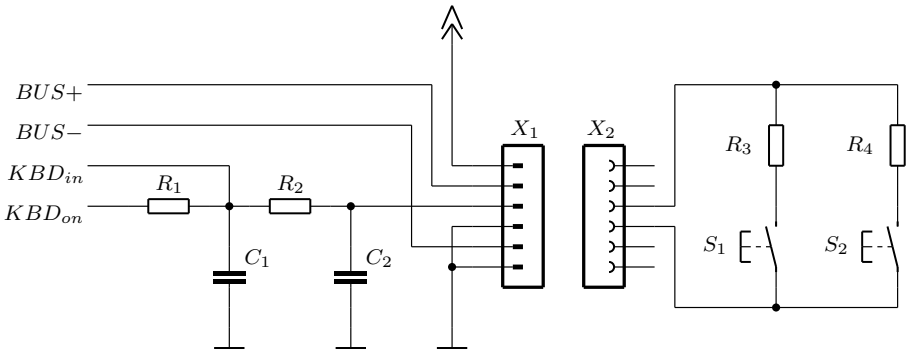

Abbildung 3.6.: Schaltung des 2-Tasten Bedienteils

Eine andere Möglichkeit mehrere Tasten über lediglich eine Signalleitung auszulesen besteht darin, dass jede gedrückte Taste eine andere Spannung auf der Signalleitung verursacht. Diese Spannung kann mit dem Analog-Digital-Wandler des Mikrocontrollers

gemessen und anschließend ausgewertet werden. Abbildung 3.6 zeigt die Schaltung des Bedienteils, welches nach diesem Prinzip funktioniert. Die Widerstände R_1 und R_2 bilden mit den Widerständen R_3 oder R_4 je nach gedrückter Taste einen Spannungsteiler, dessen Ausgangsspannung KBD_{in} vom Mikrocontroller erfasst wird. Die Widerstände werden so bemessen, dass die Differenz der Spannungen aus jeder Tastenkombination so groß wie möglich ist. Da eine analytische Dimensionierung der Widerstände in diesem Fall nicht einfach ist, wurden sie mit Hilfe einer Tabellenkalkulation iterativ mit $R_1 = 10k\Omega$, $R_2 = 1k\Omega$, $R_3 = 12k\Omega$ und $R_4 = 20k\Omega$ ermittelt.

$$\frac{U_{KBD_{in}\,S1}}{U_{KBD_{on}}} = \frac{R_3 + R_2}{R_3 + R_1 + R_2} = 0{,}68$$
$$\frac{U_{KBD_{in}\,S2}}{U_{KBD_{on}}} = \frac{R_4 + R_2}{R_4 + R_1 + R_2} = 0{,}57$$
$$\frac{U_{KBD_{in}\,S1,S2}}{U_{KBD_{on}}} = \frac{(R_3||R_4) + R_2}{(R_3||R_4) + R_1 + R_2} = 0{,}46$$

Für diese Bauteilwerte ergibt sich ein Spannungshub von jeweils 11% der am Teiler anliegenden Spannung $U_{KBD_{on}}$.

$$\left(\frac{U_{KBD_{in}\,S1}}{U_{KBD_{on}}} - \frac{U_{KBD_{in}\,S2}}{U_{KBD_{on}}}\right) \cdot 100\% = (0{,}68 - 0{,}57) \cdot 100\% = 11\%$$
$$\left(\frac{U_{KBD_{in}\,S2}}{U_{KBD_{on}}} - \frac{U_{KBD_{in}\,S1,S2}}{U_{KBD_{on}}}\right) \cdot 100\% = (0{,}57 - 0{,}46) \cdot 100\% = 11\%$$

Der Widerstand R_2 bildet zusammen mit den Kondensatoren C_1 und C_2 einen Filter gegen Störungen die über das Bedienteil eingefangen und auch abgestrahlt werden können. Die Kondensatoren sind jeweils mit $C_1 = C_2 = 10nF$ dimensioniert und bilden sowohl in die Schaltung als auch nach außen einen Tiefpass mit einer Grenzfrequenz bei $f_g \approx 16kHz$.

$$f_g = \frac{1}{2\pi \cdot R_2 \cdot C_1} = \frac{1}{2\pi \cdot 1k\Omega \cdot 10nF} = 15{,}92kHz$$

Die Variante mit dem Spannungsteiler hat einen Nachteil: Die Information über die gedrückte Taste steckt in den dazu gehörigen Widerständen. Der Strom durch den Spannungsteiler hängt mit dem Widerstand, der durch Drücken der Tasten gebildet wird zusammen. Ist dieser Widerstand groß, dann fließt ein kleinerer Strom und vice versa. Man sieht hier, dass der Spannungshub, bedingt durch die Änderung des Stroms reduziert wird. Abhilfe kann man schaffen, indem der Widerstand R_1 durch eine Konstantstromquelle ersetzt wird.

3.3.3. Datenschnittstelle

Die Datenschnittstelle dient dazu, mit anderen Motoransteuerungen und erweiterten Bedienteilen eine Verbindung aufzubauen. Zum Zeitpunkt der Entwicklung dieser Schaltung war noch nicht klar, wie die Kommunikation der einzelnen Komponenten realisiert

werden soll. Zur Auswahl stand einerseits ein bestehendes Protokoll, welches asynchron serielle Daten über RS485 transportiert, und andererseits die Möglichkeit, mit dem selben schaltungstechnischen Aufwand den CAN-Bus zu implementieren. Das bestehende Antriebskonzept ist so aufgebaut, dass eine Motorsteuerung gleichzeitig bis zu drei Motoren ansteuert und miteinander synchronisiert. Mehrere Motorsteuerungen können über die zuvor erwähnte serielle Schnittstelle kommunizieren, und ermöglichen somit den Synchronbetrieb von mehr als nur drei Motoren.

Der im Rahmen dieser Arbeit entwickelte Prototyp zur Ansteuerung bürstenloser Motoren sieht vom Konzept her anders aus: Jeder BLDC-Motor hat seine eigene Ansteuerung und kann somit als in sich abgeschlossene Komponente betrachtet werden. Im Gegensatz zum bestehenden Antriebskonzept, wo jeweils drei Motoren einen Busteilnehmer bilden, ist hier jeder BLDC-Motor ein eigenständiger Knoten am Bus. Somit teilen sich für dieselbe Anzahl an Aktuatoren zirka dreimal mehr Komponenten die selbe Schnittstelle.

Die Datenschnittstelle wurde letztendlich so aufgebaut, dass grundsätzlich sowohl das serielle Protokoll, als auch der CAN-Bus genutzt werden kann, jedoch nicht gleichzeitig. Gelöst wurde das, indem als Leitungstreiber ein CAN-Transceiver verwendet wird, und dieser auf der Seite des Mikrocontrollers mit dem CAN- sowie dem USART-Anschluss verbunden wird. Grundsätzlich handelt es sich bei diesem Aufbau um einen CAN-Bus, es ist jedoch auch möglich serielle Daten über dessen physikalische Schicht zu übertragen.

Für den CAN-Treiber gibt es entweder einen dominanten Zustand, bei dem die Anschlüsse CAN-high auf high-Pegel und CAN-low auf low-Pegel geschaltet werden oder einen rezessiven Zustand, bei dem die beiden Anschlüsse in der Luft hängen. Abschlusswiderstände an den Enden des Busses ziehen in diesem Fall die Spannungsdifferenz der CAN-Anschlüsse gegen Null. Grundsätzlich sollten am CAN-Bus ausschließlich CAN-Treiber angekoppelt sein, da diese den Standards des Busses genügen. Es gibt jedoch auch RS485-Tansceiver, welche die Pegel am CAN-Bus richtig erkennen und somit ein Empfangen des seriellen Signals ermöglichen. Man sollte diese Bauteile allerdings nicht zum Senden verwenden, da diese auf Grund eines Push-Pull-Betriebes den rezessiven Zustand zunichte machen.

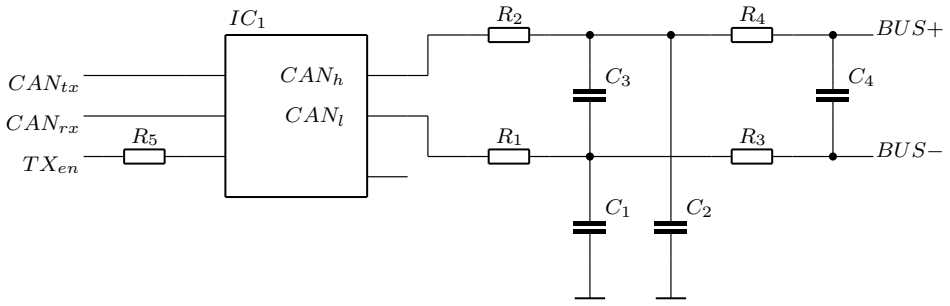

Abbildung 3.7.: Schaltung des CAN-Treibers

Als Treiberbaustein wurde der CAN-Transceiver *SN65HVD1050* verwendet. [28] Dieser CAN-Treiber überwacht die Dauer des dominanten Zustandes und trennt sich nach

Überschreiten einer Zeit von $280\mu s \ldots 700\mu s$ vom Bus. Dadurch wird sichergestellt, dass Fehler in der Hardware oder Software nicht dazu führen, dass der Bus blockiert wird. Für die gewöhnliche, asynchrone serielle Datenübertragung bedeutet das allerdings, dass diese für die Dauer der Übertragung eines Bytes nicht länger als $t_{max} = 280\mu s$ dauern darf. Daraus ergibt sich eine Symbolrate von mindestens $f_s \geq 30kBd$.

$$f_s \geq \frac{Bits\,per\,Byte}{t_{max}} = \frac{8}{280\mu s} = 28{,}57Bd \approx 30Bd$$

Abbildung 3.7 zeigt den CAN-Transceiver mit Filterschaltung, welche in erster Linie zur Reduzierung hochfrequenter Störungen dient, wobei Störungen ausgehend vom IC genauso gefiltert werden, wie jene, die über den Bus zum IC gelangen. Die Bauteile sind dabei so angeordnet, dass sie für jede Bus-Leitung einen Filter sowohl gegen Masse als auch gegeneinander bilden. Die Widerstände in dieser Filterschaltung haben den Nachteil, dass diese das Nutzsignal dämpfen. An der Signalleitung liegt somit eine kleinere Spannung an, als dies ohne Widerstände der Fall wäre. Wenn die Bus-Leitung nicht besonders lang ist, sodass selbst über die längste Distanz die Signalpegel innerhalb der Spezifikation liegen, dann spricht grundsätlich nichts gegen die Widerstände in der Filterschaltung. Für lange Leitungen empfiehlt es sich, jedoch Drosseln anstatt Widerstände zu verwenden. Noch besser wäre der Einsatz stromkompensierter Drosseln, da diese hauptsächlich die Gleichtaktstörung unterdrücken, welche durch nicht gleichzeitiges Schalten der beiden Ausgänge des Treibers entstehen. Hinsichtlich elektrostatischer Entladungen sind keine zusätzlichen Entstörbauteile vorgesehen, da der CAN-Transceiver *SN65HVD1050* über eine interne Schutzstruktur verfügt, sodass dieser Prüfimpulse nach EN61000-4-2 bei einer Spannung von $6kV$ standhält.

Die Filterschaltung wurde so dimensioniert, dass die Grenzfrequenz eines einfachen Tiefpasses mindestens eine Dekade über der Datenrate des Signals liegt. Angenommen wurde dabei die niedrigste für CAN übliche Symbolrate von $f_s = 125^{kbits}/_{s}$.

$$f_g = \frac{1}{2\pi \cdot R_1 \cdot C_1} = \frac{1}{2\pi \cdot 23{,}5\Omega \cdot 1nF} = 6{,}77MHz$$

Die Widerstände haben alle dieselben Bauteilwerte, während die Kondensatoren C_3 und C_4 jeweils mit der halben Kapazität von $C_1 = C_2$ gewählt werden. Damit wird erreicht, dass für ein differentielles Signal näherungsweise dieselbe Grenzfrequenz gilt wie für Gleichtaktsignale.

3.3.4. Elektrisch betätigte Bremse

Für Antriebe, bei denen das Getriebe in umgekehrter Richtung nicht blockiert, ist es notwendig, den Antrieb im Stillstand mechanisch zu verriegeln. Aus diesem Grund ist in der Schaltung die Ansteuerung einer elektrisch betätigten Bremse vorgesehen. Die Bauteile dafür wurden allerdings nicht dimensioniert, da zum Zeitpunkt der Entwicklung der Ansteuerelektronik für BLDC-Motoren kein Getriebe mit Bremse zur Verfügung stand, und somit auch keine konkreten Angaben dazu.

Der Vollständigkeit halber soll an dieser Stelle kurz die Idee hinter dieser Schaltung beschrieben werden: Wenn man annimmt, dass es sich bei der Bremse um einen Elektromagnet handelt, ähnlich wie bei einem Relais, dann liegt eine induktive Last vor. Dass sich in einer Induktivität der Strom nicht sprunghaft ändern kann, wurde bereits in Abschnitt 3.1.4 (Dynamische Verluste, Seite 21) erwähnt. Auch in diesem Fall ist es der Ausschaltvorgang, der Probleme bereiten kann. Als schaltendes Element wird ein N-Kanal MOSFET verwendet, der vom Mikrocontroller angesteuert wird. Der in der Wicklung des Elektromagneten eingeprägte Strom wird beim Abschalten noch so lange durch den MOSFET getrieben, bis dessen Kanal eine Impedanz aufweist, bei der der Strom einen Spannungsabfall an der Drain-Source-Strecke verursacht, der in der Größenordnung der Versorgungsspannung liegt. Ab diesem Zeitpunk übernimmt eine Freilaufdiode den Strom, und die restliche Energie wird in der Diode sowie dem resistiven Anteil der Spule abgebaut. Zusätzlich befindet sich parallel zur Diode eine Snubber-Schaltung zur Dämpfung hochfrequenter Störungen, bedingt durch die Schaltvorgänge.

3.4. Spannungsversorgung

Die zuvor beschriebenen Schaltungsteile benötigen, bis auf den Leistungsteil, alle eine geregelte Spannungsversorgung, wobei allerdings unterschiedliche Spannungen notwendig sind. Der Mikrocontroller wird beispielsweise mit $3{,}3V$ versorgt, hingegen benötigen andere Schaltungsteile wie der Bustreiber oder der Strommessverstärker eine Versorgungsspannung von $5V$. Weiters gibt es Schaltungsteile, die nur während des Betriebs des Motors von Relevanz sind und somit zur Reduzierung des Standby Energiebedarfs abgeschaltet werden können.

Aus diesen Anforderungen an die Spannungsversorgung ergibt sich ein „stufenförmiger“ Aufbau: Die Nennversorgungsspannung liegt bei $24V$ mit einer möglichen Toleranz von $\pm 20\%$. Hier tritt auch der größte Spannungsunterschied zur ersten Versorgungsspannung der Elektronik, welche bei $5V$ liegt, auf. Die Wahl zur Erzeugung der $5V$ fällt auf einen Abwärts-Schaltregler, vor allem weil es auf Grund der kleinen Baugröße durchaus Sinn macht, auf geringe Verlustleistung und somit auf geringe Erwärmung zu achten. Die zweite Versorgungsspannung der Elektronik von $3{,}3V$ dient ausschließlich zur Versorgung des Mikrocontrollers. Da dieser im Vergleich zu den anderen Schaltungsteilen einen geringeren Strom zieht, werden die $3{,}3V$ mit Hilfe eines Linearreglers aus der $5V$ Versorgung erzeugt.

Im Folgenden werden die einzelnen Teile der Spannungsversorgung näher beschrieben.

3.4.1. Schaltregler

An den Schaltregler sind unterschiedliche Anforderungen gestellt, denen dieser genügen muss:

- Der Eingangsspannungsbereich des Abwärtswandlers soll zumindest im Bereich der Nennspannung, unter Berücksichtigung der Toleranzen liegen. Wünschenswert

wäre allerdings vor allem nach oben hin eine höhere Spannungsfestigkeit, damit durch ein Ausreizen der Versorgungsspannung der Abwärtswandler nicht zu nahe seiner zulässigen Grenzen betrieben wird.

- Der Abwärtswandler muss entsprechend Gleichung (3.21) einer Strombelastbarkeit von $100mA$ standhalten. Zusätzlich muss auch gewährleistet sein, dass der Schaltregler bei sehr kleinen Strömen, wie diese im Standby Betrieb zu erwarten sind, zuverlässig funktioniert.

- Der Platzbedarf der gesamten Schaltung des Abwärtswandlers soll so gering wie möglich sein.

Der Strombedarf für die Elektronik setzt sich aus den Strömen des Winkelencoders I_{Enc}, des Strommessverstärkers I_{Amp} mit externer Beschaltung, des CAN-Bus-Treibers $ICAN$, des NOR-Gatters I_{drv} sowie des Mikrocontrollers $I_{\mu C}$ mit externer Beschaltung zusammen.

$$
\begin{aligned}
I_{Enc} &= 28mA \\
I_{Amp} &= I_{OV\,max} + I_{OV\,periph} = 1{,}2mA + 3 \cdot \left(\frac{5V}{470k\Omega} + \frac{5V}{100k\Omega} \right) = 1{,}38mA \\
I_{CAN} &= 10mA \ldots 70mA \\
I_{drv} &= 3 \cdot \Delta U_{GS} \cdot C_G \cdot f = 3 \cdot 5V \cdot 3{,}1nF \cdot 20kHz = 0{,}9mA \\
I_{\mu C} &= I_{\mu C} + I_{Brake} + I_{Key} + 3 \cdot I_B = I_{\mu C} + \frac{U}{R_{Brake}} + \frac{U}{R_{Key}} + 3 \cdot \frac{I_{C\,max}}{HFE} \\
&= 33mA \ldots 50mA + \frac{3{,}3V}{2{,}2k\Omega} + \frac{3{,}3V}{10k\Omega} + 3 \cdot \frac{11mA}{110} \\
&= 33mA \ldots 50mA + 1{,}5mA + 0{,}33mA + 0{,}3mA = 35{,}13mA \ldots 52{,}13mA
\end{aligned}
\tag{3.19}
$$

Die Ströme des Mikrocontrollers sowie des CAN-Transceivers zeigen eine große Varianz. Die beiden Ströme des Mikrocontrollers sind eine Abschätzung des ungünstigsten Falles und kommen jeweils bei einer maximalen Taktfrequenz von $72MHz$ zustande, wobei beim höheren Strom zusätzlich alle Peripherie-Komponenten aktiv sind. Tatsächlich liegt der Strom des Mikrocontrollers wesentlich niedriger. Zusammen mit den für die Schaltung relevanten Peripherie-Komponenten sowie externer Beschaltung liegt dieser bei $I_{\mu C} = 36mA$.

Der CAN-Transceiver zieht im Empfangsbetrieb den niedrigeren Strom, während der höhere Strom durch das Treiben einer Last von 60Ω bedingt ist. Die Filterschaltung der zuvor in Abschnitt 3.3.3 gezeigten Beschaltung des CAN-Transceivers (siehe Abbildung 3.7) verkleinert die Last des Treibers, sodass ein maximaler Strom von $I_{CAN\,max} = 33{,}5mA$ zu erwarten ist, wobei ein Tastverhältnis von $5:1$ berücksichtigt wurde, welches während der Übertragung ausschließlich dominanter Symbole auftritt.

$$\begin{aligned} I_{CAN\,max} &= I_{CAN\,Tx} \cdot \frac{U}{U + R \cdot I_{CAN\,Tx}} \cdot \frac{5}{6} + I_{CAN\,Rx} \\ &= 60mA \cdot \frac{5V}{5V + 94\Omega \cdot 60mA} \cdot \frac{5}{6} + 10mA = 33{,}5mA \end{aligned} \tag{3.20}$$

Entsprechend der hier gezeigten Ströme ergibt sich für den Abwärtswandler eine geforderte Belastbarkeit des Ausgangs von $I_{out} \approx 100mA$

$$\begin{aligned} I_{out} &= I_{Enc} + I_{Amp} + I_{CAN} + I_{drv} + I_{\mu C} \\ &= 28mA + 1{,}38mA + 33{,}5 + 0{,}9mA + 36mA = 99{,}78mA \end{aligned} \tag{3.21}$$

Die Integrierte Schaltung *LTC3631* [17] wird den gewünschten Anforderungen gerecht. Abbildung 3.8 zeigt die Schaltung dieses Abwärtswandlers, die im Anschluss etwas detaillierter beschrieben wird.

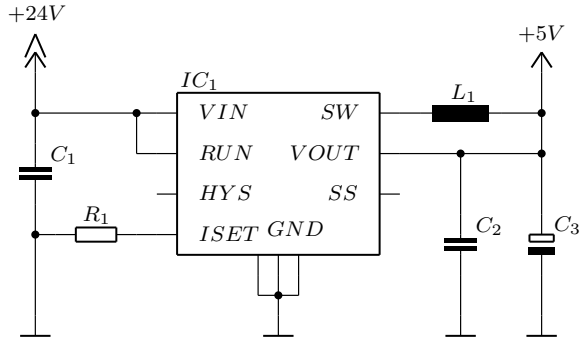

Abbildung 3.8.: Schaltung des Abwärtswandlers

Besonderheiten des Schaltreglers

variable / fixe Spannung Den Schaltregler *LTC3631* gibt es sowohl mit extern einstellbarer als auch mit fixer Ausgangsspannung, wobei bei der Variante mit fixer Spannung der externe Spannungsteiler, bestehend aus zwei Widerständen, wegfällt.

Strombegrenzung Sofern eine Begrenzung des Ausgangsstromes auf einen niedereren als den Nennbegrenzungsstrom gewünscht ist, kann dieser mit einem externen Widerstand eingestellt werden. Durch eine Verringerung des Nennbegrenzungsstroms ist es möglich, kleinere Speicherbauteile zu verwenden und Störungen durch den Schaltregler zu reduzieren.

Integrierte High- und Low-Side Schalter ermöglichen einen kontinuierlichen Stromfluss durch die Speicherdrossel. Eine externe Schottkydiode, um den Stromkreis während der Aus-Phase zu schließen, ist daher nicht notwendig.

Burst-Mode ist eine Betriebsart des Abwärtswandlers, die es ermöglicht, über einen weiten Lastbereich hohe Wirkungsgrade zu erzielen.

Hohe Schaltfrequenz Eine Schaltfrequenz im Bereich $f = 100kHz \ldots 600kHz$ erlaubt die Verwendung kleiner Speicherbauteile, weshalb der Platzbedarf, gegenüber Abwärtswandlern mit niederen Schaltfrequenzen, wesentlich geringer ist.

Dimensionierung des Schaltreglers

Neben den schaltenden Komponenten sind bei einem Abwärtswandler die wichtigsten Bauteile die Speicherdrossel sowie der Ladekondensator. Die zur Berechnung der Bauteilwerte notwendigen Formeln sind dem Datenblatt [17] des Abwärtswandlers *LTC3631* entnommen.

Im ersten Schritt wird mit dem Widerstand R_1 der Nennbegrenzungsstrom eingestellt. Der Wert für diesen Widerstand lässt sich anhand eines Diagramms des Datenblattes bestimmen. Wenn der maximal mögliche Strom von $I_{peak} = 225mA$ gewünscht ist, so wie im Falle dieser Schaltung, dann wird dieser Widerstand nicht bestückt.

Im zweiten Schritt wird die Speicherdrossel hinsichtlich der Differenz zwischen Ein- und Ausgangsspannung dimensioniert.

$$L_1 = \left(\frac{V_{out}}{f \cdot I_{peak}}\right) \cdot \left(1 - \frac{V_{out}}{V_{in}}\right) = \left(\frac{5V}{200kHz \cdot 225mA}\right) \cdot \left(1 - \frac{5V}{24V}\right) = 88{,}9\mu H$$

Im dritten Schritt wird nun überprüft, ob der für die Speicherdrossel berechnete Wert der minimalen Schaltzeit des Abwärtswandlers genügt:

$$L_{1\,min} = \frac{V_{in(max)} \cdot t_{on(min)}}{I_{peak(max)}} = \frac{24V \cdot 1{,}2 \cdot 100ns}{225mA} = 12{,}8\mu H$$

Auch diese Bedingung der minimal notwendigen Induktivität ist erfüllt. Dennoch wird die Drossel nicht mit dem nächst höheren Wert von $100\mu H$ gewählt, sondern mit $L_1 = 220\mu H$. Im Datenblatt gibt es dazu ein Diagramm, das für SMD-Drosseln einen Bereich zeigt, bei dem in Abhängigkeit des maximalen Stroms der Wirkungsgrad optimal ist. Hier liegt bei $L_1 = 220\mu H$ der günstigere Fall vor.

Im letzten Schritt wird der Ausgangskondensator dimensioniert. Von der Größe dieses Bauteils hängt die Restwelligkeit der Ausgangsspannung ab. Mit der nachstehenden Formel lässt sich die Kapazität des Kondensators berechnen, sodass die Restwelligkeit bei 1% liegt.

$$C_3 \geq 50 \cdot L \cdot \left(\frac{I_{peak}}{V_{out}}\right)^2 = 50 \cdot 220\mu H \cdot \left(\frac{225mA}{5V}\right)^2 = 22\mu F$$

3.4.2. Linearregler

Der Linearregler *MC78PC33* , dessen Schaltung in Abbildung 3.9 dargestellt ist, dient dazu, die Versorgungsspannung für den Mikrocontroller bereitzustellen. Es handelt sich hierbei um einen Regler mit geringem Spannungsverlust, auch bekannt unter der Abkürzung LDO-Spannungsregler (**L**ow **D**rop **O**ut).

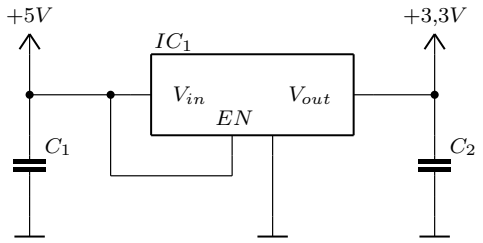

Abbildung 3.9.: Schaltung des Low-Drop-Out Linearreglers

[12] LDO Spannungsregler verwenden anstatt eines NPN-Transistors einen PNP-Transistor als Stellglied. Die niedrige Verlustspannung ergibt sich daraus, dass der PNP-Transistor bis in die Sättigung angesteuert werden kann. Diese Schaltung bringt aber auch gleichzeitig den Nachteil mit sich, dass der Basisstrom zur Ansteuerung nicht in den Ausgang fließt. Diesem Nachteil wirkt man entgegen, indem anstatt des PNP-Transistors ein P-Kanal MOSFET eingesetzt wird, so wie dies im integrierten Spannungsregler *MC78PC33* der Fall ist.

Hinsichtlich Stabilität sind LDO Regler im Gegensatz zu Spannungsreglern mit NPN-Transistor wesentlich empfindlicher. Es empfiehlt sich daher, am Ausgang einen Kondensator zu verwenden, dessen Kapazität und ESR den Angaben des Datenblattes entsprechen. [20] Im Falle des Spannungsreglers *MC78PC33* darf die Kapazität des Kondensators C_2 im Bereich $4{,}7\mu F \ldots 10\mu F$ liegen, wobei ein ESR von $0{,}3\Omega \ldots 10\Omega$ eingehalten werden soll. Um wirklich sicher zu gehen, ob die Schaltung tatsächlich stabil ist, wurde die Ausgangsspannung mit Oszilloskop auf Schwingungen untersucht. Dabei hat sich gezeigt, dass der Spannungsregler bei einer Kapazität von $2{,}2\mu F$ bereits stabil ist. Dies liegt vermutlich daran, dass durch die Last, welche der Mikrocontroller bildet, die Schaltung letztendlich einen stabilen Arbeitspunkt erreicht. Die Kapazität des Kondensators am Eingang ist dagegen unkritisch, sie wird im Datenblatt mit $C_1 = 1\mu F$ angegeben.

Die Verlustleistung des LDO-Spannungsreglers mit P-Kanal MOSFET ist direkt proportional zum Strom am Ausgang. Es handelt sich hier um den Strom des Mikrocontrollers und dessen externe Beschaltung, welcher entsprechend Gleichung (3.19) im ungünstigsten Fall $I_{\mu C} = 52{,}13mA$ beträgt. Unter der Annahme, dass die Verlustspannung innerhalb der Spezifikationen maximal ist, ergibt sich für diesen Strom eine Verlustleistung von $P_V = 104mW$.

$$P_V = (U_{in} - U_{out\,min}) \cdot I_{\mu C} = (5V - 3V) \cdot 52{,}13mA = 104mW$$

Die Verlustleistung des Spannungsreglers darf bis zu $P_{V\,max} = 250mW$ betragen, und demnach ist ein zuverlässiger Betrieb des Reglers gewährleistet.

3.4.3. Abschaltung bei Standby-Betrieb

Die Spannungsversorgung des Strommessverstärkers sowie des Winkelencoders ist nur während des Betriebs des Motors notwendig und ist daher so ausgeführt, dass sie über den Mikrocontroller ein- beziehungsweise ausgeschaltet werden kann. Als Schalter wird dafür ein P-Kanal MOSFET auf der $5V$ Seite verwendet, die Masse bleibt hingegen ständig verbunden.

An den Schalttransistor sind folgende Anforderungen gestellt: Dem Strom der zu schaltenden Komponenten muss der Transistor auf jeden Fall standhalten. Bedingt durch den Ein-Widerstand des Transistors fällt an diesem eine dem Strom proportionale Spannung ab. Dieser Spannungsabfall sollte nur so groß sein, dass am Ausgang immer noch eine Spannung anliegt, bei der sowohl der Winkelencoder als auch der Strommessverstärker zuverlässig funktionieren. Nimmt man an, dass die in Abschnitt 3.2.1 zur Einstellung der Ruhespannung des Strommessverstärkers dimensionierten Widerstände mit einer Toleranz von 5% gewählt werden, dann funktioniert der Messverstärker bis zu einer kleinsten Spannung von $4{,}13V$.

$$V_{min} = U_{bias\,min} \cdot \frac{R_{2\,min} + R_{3\,max}}{R_{2\,min}} = 0{,}044V \cdot \frac{5{,}6k\Omega \cdot 0{,}95 + 470k\Omega \cdot 1{,}05}{5{,}6k\Omega \cdot 0{,}95} = 4{,}13V$$

Dagegen ist für den Winkelencoder eine minimale Versorgungsspannung von $4{,}5V$ notwendig. Dementsprechend ergibt sich bei einer minimalen Ausgangsspannung des Schaltreglers von $U_{out\,min} = 4{,}91V$ ein zulässiger Spannungsabfall am Transistor von $U_{DS\,on} \leq 0{,}41V$.

$$U_{DS\,on} \leq 4{,}91V - 4{,}5V = 0{,}41V$$

Der Winkelencoder und der Strommessverstärker mit externer Beschaltung ziehen einen Strom von höchstens $30mA$. Die Ströme I_{ENC} und I_{Amp} wurden zuvor in Gleichung (3.19) berechnet.

$$I_{D\,max} = I_{ENC} + I_{Amp} = 28mA + 1{,}38mA = 29{,}38mA \approx 30mA$$

Der Transistor darf folglich einen maximalen Ein-Widerstand von $R_{DS\,on} \leq 13{,}7\Omega$ aufweisen.

$$R_{DS\,on} \leq \frac{U_{DS\,on}}{I_{D\,max}} = \frac{0{,}41V}{30mA} = 13{,}7\Omega$$

Diesen Anforderungen wird der Transistor *BSS84* gerecht. Aus dessen Datenblatt [23] geht dazu folgendes hervor: Der Drainstrom darf maximal $I_D \leq 130mA$ betragen. Die Sperrschichttemperatur darf höchstens auf $T_{j\,max} = 150°C$ ansteigen. Bei einer Sperrschichttemperatur von $T_j = 100°C$ beträgt der Ein-Widerstand $R_{DS\,on} \leq 13\Omega$. Ein Drainstrom von $I_D = 30mA$ führt dabei zu einem Temperaturanstieg der Sperrschicht gegenüber der Umgebung von $\Delta\vartheta = 5{,}85°C$.

$$\Delta\vartheta = P_V \cdot R_{th(j-a)} = I_D^2 \cdot R_{DS\,on} \cdot R_{th(j-a)} = (0{,}03A)^2 \cdot 13\Omega \cdot 500^K/_W = 5{,}85°C$$

Dieser Temperaturunterschied erlaubt demnach eine Umgebungstemperatur von zirka $T_a = 94°C$, bei der nach wie vor ein zuverlässiger Betrieb des Transistors gewährleistet ist, sodass auch die beiden Teile Winkelencoder und Strommessverstärker mit der zum Betrieb mindestens notwendigen Spannung versorgt werden.

3.5. Elektromagnetische Verträglichkeit

Das Thema „Elektromagnetische Verträglichkeit“ mag zwar auf den ersten Blick mit der eigentlichen Funktionalität einer Schaltung wenig zu tun haben, allerdings ist sie in der Geräteentwicklung nicht wegzudenken. [3] Abgesehen von den gesetzlichen Bestimmungen, welche Grenzwerte sowie Störfestigkeit für in Verkehr gebrachte Geräte regeln, bringt die Entwicklung EMV-konformer Schaltungen den Vorteil mit sich, dass diese einerseits benachbarte Geräte weniger beziehungsweise gar nicht stören, und andererseits wesentlich robuster gegenüber äußeren Störeinflüssen sind. Ob nun eine Schaltung EMV-konform sein wird, lässt sich zum Zeitpunkt der Entwicklung nur schwer vorhersagen. Um dennoch den Anforderungen elektromagnetischer Verträglichkeit gerecht zu werden, erweist es sich als sinnvoll entwicklungsbegleitend Messungen durchzuführen, damit entstörende Maßnahmen während des Entwicklungsprozesses so früh wie möglich getroffen werden können.

3.5.1. Schaltungstechnische Maßnahmen

Der entwickelte Prototyp wurde zwar nicht bis zur Serienreife perfektioniert, dennoch wurde darauf Wert gelegt, dass zumindest Grenzwerte bezüglich elektromagnetischer Emission eingehalten werden. Während der Entwicklung wurden dabei folgende Maßnahmen berücksichtigt, wobei das Hauptaugenmerk auf dem Leistungsteil liegt, da von diesem die größte Störemission ausgeht.

Schaltplan und Layout der Leiterplatte

Obwohl im Schaltplan nicht alle Koppeleffekte berücksichtigen werden können, sollte man bereits in dieser Phase des Schaltungsentwurfs das Thema EMV im Hinterkopf haben. Mit relativ einfachen Maßnahmen, wie Filterstrukturen bestehend aus Kondensatoren sowie Widerständen beziehungsweise Drosseln, lassen sich gegenseitige Beeinflussungen von einzelnen Schaltungsteilen genauso reduzieren, wie Störungen, die über Zuleitungen eingefangen, aber auch abgestrahlt werden können.

Genau so wichtig wie die Entstörbauteile selbst, ist deren sinnvolle Anordnung auf der Leiterplatte. Dabei wurde versucht, die Bauteile so zu platzieren, dass Leiterbahnen dazwischen kurz, und die aufspannenden Flächen klein sind. Bei der Leiterplatte selbst handelt es sich um einen vier-lagigen Print, bei dem eine der beiden inneren

Lagen durchgehend als Massefläche ausgeführt ist. Auf der zweiten inneren Lage sind Versorgungsleitungen großflächig ausgeführt.

Blockkondensator am MOSFET

Messungen an der Schaltung haben gezeigt, dass die MOSFETs der drei Halbbrücken während der Schaltvorgänge „klingeln“. Jeder Schaltvorgang bringt eine große Änderung des Stromes mit sich, weshalb an der parasitären Induktivität zum Source-Anschluss eine Spannung abfällt, die indirekt eine Änderung der Gate-Spannung zufolge hat. Diese wiederum steuert entsprechend der Steilheit des MOSFETs den Drain-Strom, womit sich der „Schwingkreis“ schließt.

Diesen Effekt kann man dämpfen, indem man den Blockkondensator C_2 (siehe Abbildung 3.1) zwischen die beiden Source-Anschlüsse jeder Halbbrücke schaltet. Der Kondensator soll dabei so nahe wie möglich am MOSFET platziert sein. Die Reduktion der Störung liegt nun darin begründet, dass die schnellen Stromänderungen vom Kondensator gepuffert werden, und somit die Spannung an den Source-Anschlüssen nicht so stark einbricht. Mit diesem Kondensator konnte bei einem Bauteilwerte im Bereich $C_2 = 1 \ldots 10nF$ eine optimale Dämpfung erreicht werden.

Snubber

Es gibt noch eine weitere Ursache für das „Klingeln“ der MOSFETs. [14] Die Transistoren jeder Halbbrücke werden komplementär geschaltet. Dabei gibt es eine kurze Zeit zwischen den Beiden Ein-Zuständen der MOSFETs, damit ein Querstrom durch die Brücke vermieden wird. In dieser Zeit fließt der Strom der Motorwicklung durch die Diode des einzuschaltenden Transistors, bis dessen Source-Drain-Strecke den Strom übernimmt. Damit die Diode wider in den Sperrbetrieb übergeht müssen die Ladungsträger ausgeräumt werden. Auf Grund parasitärer Kapazitäten und Induktivitäten verursacht diese Ladung ein Überschwingen an den Drain-Anschlüssen. Der Kondensator C_3 sowie der Widerstand R_4, aus Abbildung 3.1, bilden eine Snubber-Schaltung, welche diese Störung dämpft. Eine optimale Dämpfung konnte erzielt werden, wenn die Bauteile mit $C_3 = 470pF$ und $R_4 = 1\Omega$ dimensioniert werden.

Common-Mode Dämpfung mit Ferritkerne

Zu guter Letzt konnte gezeigt werden, dass mittels Ferritkernen, sowohl auf Seite der Versorgung, als auch zwischen Elektronik und Motor, hochfrequente Common-Mode Störungen gedämpft werden. Für diese Gleichtakt-Störungen wirken die Zuleitung und auch die Motorwicklung wie eine Antenne, die ein Abstrahlen der Störungen begünstigt. Auf Seite des Motors kann man, anstatt mit einem Ferritkern, genauso eine Dämpfung der Störaussendung erreichen, indem bei der Entwicklung eines Firmenspezifischen Motors darauf geachtet wird, das dessen Gehäuse, welches auch die Ansteuerelektronik beinhaltet, dicht gegen Hochfrequenz ist.

3.5.2. Messergebnisse

Während der Optimierung der Schaltung bezüglich elektromagnetischer Emission wurden mehrere Messungen durchgeführt. Für die Dokumentation erscheinen allerdings nur jene Messergebnisse von Relevanz, die hinsichtlich EMV positive Werte aufweisen. Im Anschluss wird auf die beiden Bereiche der EMV „Leitungsgeführte Hochfrequenz“ und „Funkstörstrahlung“ eingegangen. Weiters werden die Ergebnisse der Messungen entsprechend der Anwendung interpretiert.

Leitungsgeführte Hochfrequenz

Zur Bewertung leitungsgeführter Hochfrequenz wird grundsätzlich die Funkstörspannung gemessen, welche vom Prüfling über die Netzzuleitung auf eben diese zurück wirkt. Relevant ist hierbei ein Frequenzbereich von $150kHz$ bis $30MHz$. Mit einer Netznachbildung werden die kapazitiven und induktiven Eigenschaften eines Versorgungsnetzes bis hin zur Steckdose modelliert, sowie die vom Prüfling verursachte Störspannung innerhalb des zu erfassenden Frequenzbereichs an den Messempfänger ausgekoppelt.

Die entwickelte Schaltung zur Ansteuerung bürstenloser Motoren wird nicht direkt am Netz betrieben, sondern es werden mehrere Exemplare dieser Schaltung von einem gemeinsamen Netzteil versorgt. Dieser Aufbau bringt nun mit sich, dass die einzelnen Komponenten über Leitungen, ähnlich einer Bus-Topologie, miteinander verbunden sind. Für eine genaue Messung der Störemission müsste dieser Aufbau im Freifeld gemessen werden. Zwecks einer Abschätzung der Störemission im Frequenzbereich unter $30MHz$ wurde das Messverfahren für leitungsgeführte Hochfrequenz dahin gehend abgeändert, dass die Netznachbildung zwischen das Netzteil und dem Prüfling geschaltet wurde.

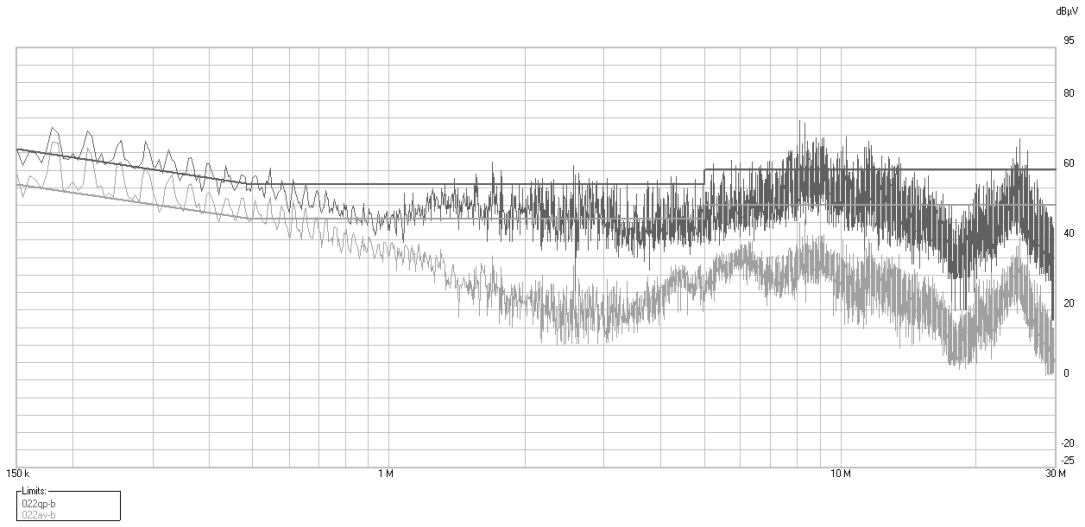

Abbildung 3.10.: Störspektrum leitungsgeführter Hochfrequenz bei halber Motorlast

Abbildung 3.10 zeigt das Ergebnis dieser Messung mit den Grenzwerten für *Quasi-Peak* (qp) und *Average* (av). Zwecks schnellerer Messung wurde anstatt des *Quasi-Peak-Detektors* der *Peak-Detektor* zur Bewertung des Störpegels verwendet. Das Ergebnis gilt

es nun entsprechend der Anwendung zu interpretieren: Wenn man davon ausgeht, dass eine Leitung mit einem Viertel der Wellenlänge als Antenne für eben diese Welle wirkt, so kann man folgendermaßen abschätzen, ab welcher Frequenz es mit Sicherheit zur Abstrahlung kommt. Als Beispiel wird eine Leitungslänge von $5m$ angenommen:

$$f = \frac{c}{\lambda} = \frac{c}{4 \cdot l_{Leitung}} = \frac{3 \cdot 10^{8 m/s}}{4 \cdot 5m} = 15MHz$$

Die hier gezeigte Frequenz stellt keine harte Grenze dar, dies wirft daher die Frage auf, welche Frequenz nicht mehr abgestrahlt wird.

Der Innenwiderstand R_q der Störquelle U_q bildet zusammen mit dem Strahlungswiderstand der Antenne R_s einen Spannungsteiler.

$$U_{R_s} = U_q \cdot \frac{R_s}{R_s + R_q} = \begin{cases} \approx U_q & R_s \gg R_q \\ \approx U_q \cdot \frac{R_s}{R_q} & R_s \ll R_q \end{cases}$$

[12] Der Strahlungswiderstand R_s einer Stabantenne hängt mit der relativen Länge l/λ zusammen. Dieser Widerstand wird für Antennenlängen $l < \lambda/8$ relativ schnell sehr klein. Wenn man nun annimmt, dass bei einer Wellenlänge von $\lambda/4$ Leistungsanpassung ($R_q = R_s$) vorliegt, dann hätte der abgestrahlte Störpegel Hochpasscharakter, mit der Knickfrequenz im Punkt der Leistungsanpassung. Wenn man weiters annimmt, dass bis zur Knickfrequenz kleinere Frequenzen pro Dekade bis zur Knickfrequenz um $20dB$ ansteigen, dann würde das für die Messung aus Abbildung 3.10 bedeuten, dass im Bereich $\approx 7 \ldots 30MHz$ der Grenzwert für *Quasi-Peak* überschritten ist. Um jedoch eine genaue Aussage bezüglich Störemission treffen zu können, empfiehlt es sich, im Freifeld eine Referenzmessung durchzuführen. Aus dieser Messung lässt sich dann schließen, wie die gewählte Messmethode mit der Netznachbildung zu korrigieren ist.

Funkstörstrahlung

Der zur Bewertung hochfrequenter Störstrahlung zu erfassende Frequenzbereich hängt entsprechend EN61000-6-3[2] von der höchsten in der Schaltung vorkommenden Frequenz ab. In der entwickelten Schaltung wird diese von der Taktfrequenz des Mikrocontrollers bestimmt, welche innerhalb des Chips bei höchstens $72MHz$ liegt. Für Systemfrequenzen bis $108MHz$ wird die Funkstörstrahlung normgerecht im Bereich $30MHz \ldots 1GHz$ erfasst.

Zur Bewertung der Störstrahlung wird für eine normgerechte Messung der *Quasi-Peak-Wert* (QPK) erfasst. Da dessen Messung sehr zeitintensiv ist, kann man alternativ dazu das Störspektrum anhand des *Peak-Wertes* (PK) beurteilen. Der Spitzenwert liegt im ungünstigsten Fall gleich mit dem Quasi-Spitzenwert, für gewöhnlich jedoch darüber. Abbildung 3.11 zeigt das Emissionsspektrum der Schaltung, wobei hier der Spitzenwert erfasst wurde. Kritisch sind nun jene Stellen, an denen der Grenzwert erreicht, beziehungsweise überschritten wird, wie beispielsweise bei eine Frequenz von $92MHz$. In

[2] EN61000-6-3: Fachgrundnorm Störaussendung für Wohnbereich, Geschäfts- und Gewerbebereich sowie Kleinbetriebe (Stand 2011)

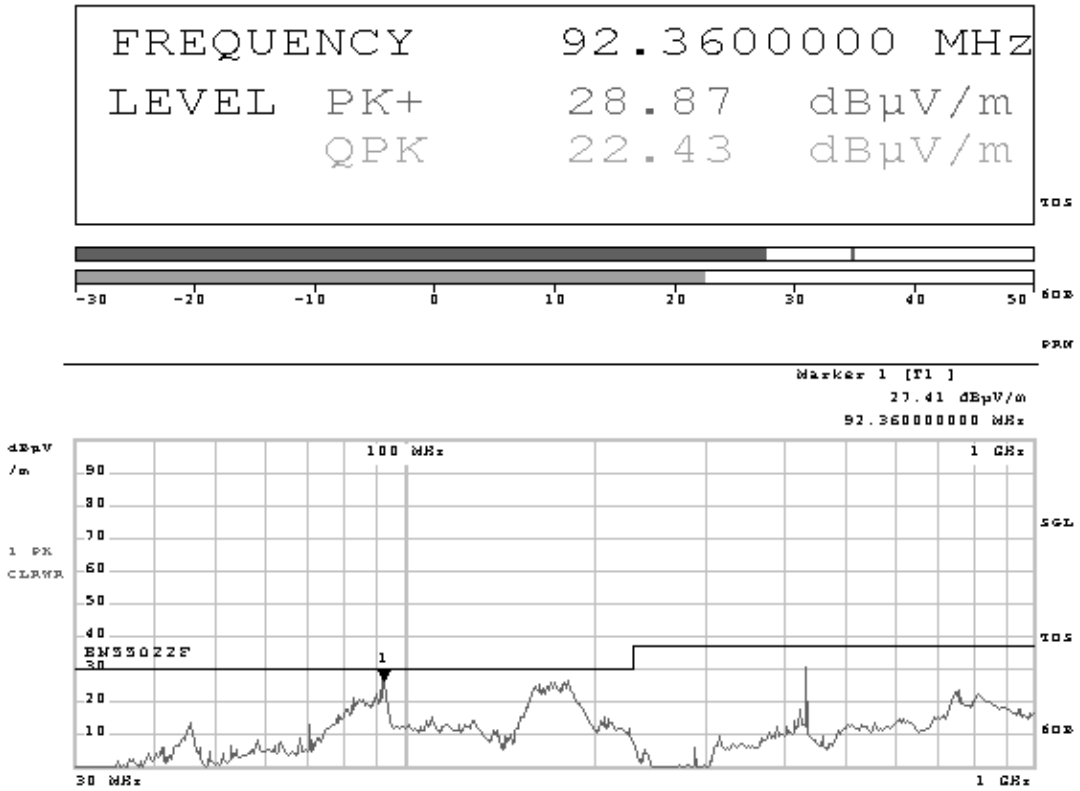

Abbildung 3.11.: Spektrum der Funkstörstrahlung bei halber Motorlast

diesem Fall kann man sich damit behelfen, indem man den Messempfänger auf diese Frequenz einstellt und eine Messung mit dem Quasi-Peak-Detektor durchführt, wie dies in der oberen Hälfte der Abbildung zu sehen ist. Man sieht hier, dass der Spitzenwert mit 28,87$dB_{\mu V}$ tatsächlich höher liegt als der Quasi-Spitzenwert mit 22,43$dB_{\mu V}$. Weiters zeigt der Schleppzeiger des Spitzenwertdetektors einen Pegel von zirka 35$dB_{\mu V}$ an. Diese kurzzeitigen Spitzen werden durch die Zeitkonstante des Quasi-Spitzenwertdetektors geringer bewertet als länger andauernde Pegel. Bei einer Spitzenwertmessung ist es daher durchaus möglich, dass an einer Stelle, an welcher der Grenzwert überschritten wird, der zulässige Pegel eingehalten wird.

Entsprechend des hier gezeigten Messergebnisses ist die Schaltung bezüglich Störstrahlung normgerecht. Die Messung wurde dabei mit den zuvor beschriebenen Entstörmaßnahmen „Blockkondensator am MOSFET“, „Snubber“ sowie „Common-Mode Dämpfung mit Ferritkerne“ durchgeführt.

3.6. Zuverlässigkeit

Die Zuverlässigkeit einer Schaltung hängt sehr stark damit zusammen, inwiefern einzelne Bauteile bezüglich Belastbarkeit an ihre Grenzen getrieben werden. Dabei stellt die Temperatur generell für fast jedes elektronische Bauteil eine Belastung dar. Vor allem

erhöhte Temperaturen führen zu beschleunigter Alterung. Dabei hängt es oft mit den verwendeten Materialien zusammen, wie die Lebensdauer eines Bauteils durch Temperatureinwirkung beeinflusst wird. Folgende Eigenschaften sind jedoch allen Bauteilen gemeinsam:

- Jedes Bauteil hat eine kritische Temperatur, deren Überschreitung dazu führt, dass die Funktion des Bauteils nicht mehr mit Sicherheit gegeben ist, oder dass das Bauteil sogar unwiderruflich zerstört wird.
- Das Gehäuse jeden Bauteils hat einen Temperaturübergangswiderstand. Demnach ist unter Belastung der Kern des Bauteils stets wärmer als die Umgebung.
- Der innere Temperaturanstieg eines Bauteils gegenüber der Umgebung ist immer eine Folge von Verlustleistung.

Viele Bauteile, der in den vorigen Abschnitten beschriebenen Komponenten der Schaltung, wurden hinsichtlich thermischer Belastung untersucht. Dabei konnte gezeigt werden, dass die Grenzwerte, entsprechend der jeweils relevanten Datenblätter unter Berücksichtigung einer erhöhten Umgebungstemperatur von zirka $85°C$, eingehalten werden.

Ein weiterer kritischer Parameter ist die Spannungsfestigkeit. Auf Grund der kleinen Strukturen in Halbleitern und minimaler Abstände in Kondensatoren, treten hier bei relativ niedrigen Spannungen hohe Feldstärken auf, sodass wenig Spielraum nach oben bleibt, ehe es zu einem Durchschlag kommt, der das Bauteil schädigt. Dies betrifft vor allem jene Bauteile, die direkt an der Versorgung angeschlossen sind. In den vorigen Beschreibungen der einzelnen Schaltungsteile wurde das zwar nicht explizit erwähnt, dennoch wurde bei der Wahl der Bauteile darauf Wert gelegt, dass die maximal zulässige Spannung nicht überschritten wird, selbst wenn die Versorgungsspannung um zirka 60% erhöht ist.

Hinsichtlich thermischer und elektrischer Belastung ist die Schaltung insofern als Zuverlässig einzustufen, dass es zu keiner Zerstörung der Bauteile kommt. Es gibt jedoch Schwachstellen bezüglich einer zuverlässigen Funktion der Schaltung. Dazu zählt der in Abschnitt 3.4.1, ab Seite 37 beschriebene Schaltregler. Dieser ist zirka um 50% zu schwach, für den Fall dass alle Schaltungsteile in Betrieb sind, und dabei den ungünstigsten, größtmöglichen Strom ziehen. Der Schaltregler ist daher gegen eine stärkere Variante zu ersetzen, beziehungsweise könnte man alternativ dazu auch eine getrennte Spannungsversorgung von Leistungsteil und Steuerelektronik in Erwägung ziehen

4. Software

4.1. Feldorientierte Regelung – Implementation

Dieser Abschnitt widmet sich der Implementation der feldorientierten Regelung, deren Funktionsweise in Abschnitt 2.3 (Feldorientierte Regelung, Seite 6) bereits beschrieben wurde.

Eine detaillierte Beschreibung aller Einzelheiten zur Implementation der Firmware würde den Rahmen dieser Arbeit sprengen. In den nachfolgenden Abschnitten werden daher grundsätzliche Konzepte der Software beschrieben, und es wird auf Erweiterungen des Kerns der feldorientierten Regelung, um diesen an die entwickelte Hardware anzupassen, sowie auf ein paar interessante Teile der Firmware näher eingegangen. Zu nennen sind hier Algorithmen, die auf den in Abbildung 2.2 auf Seite 6 gezeigten Kern der feldorientierten Regelung aufbauen, und eine Optimierung des Betriebsverhaltens des Motors ermöglichen. Zu nennen sind hier *Flux weakening Control* und *Maximum Torque per Ampere*, welche Teil der Motor-Control-Library [10] sind, sowie *Position Conditioning*. Abbildung 4.1 zeigt, wie diese Funktionsblöcke um die feldorientierte Regelung angeordnet sind. Im konkreten Fall handelt es sich hier um einen Positionsregler, der die Basis für eine konkrete Anwendung als Aktuator beispielsweise für höhenverstellbare Tische darstellt. Etwas näher wird im Folgenden auch auf die Struktur der Software eingegangen, welche dem Paradigma der objektorientierten Programmierung genügt.

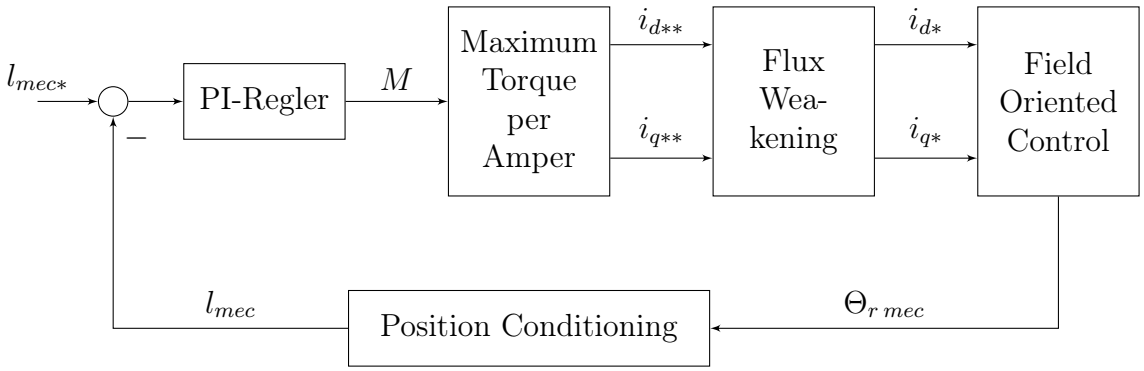

Abbildung 4.1.: Blockschaltbild Positionsregler

Zu erwähnen sei hier noch, dass die Firmware auf dem Software Development Kit *STM32F103xx/STM32F100xx PMSM single/dual FOC SDK v3.0* der Firma STMicroelectronics aufbaut. Verwendet wird die Programmiersprache *C*, und als Entwicklungs-

werkzeug kommt die auf ARM portierte freie Version des GNU C-Compilers *arm-none-eabi*, welche von Mentor Graphics entwickelt und gewartet wird, zum Einsatz.

4.1.1. Objektorientierte Programmierung

Der vorige Abschnitt hat bereits anklingen lassen, dass die Firmware in der Programmiersprache C implementiert wurde. Da C den Paradigmen imperativ und strukturiert genügt, sieht hier einerseits die Überschrift dieses Abschnitts auf den ersten Blick eher widersprüchlich aus, andererseits vertreten professionelle Softwareentwickler die Meinung, dass es möglich sei, in jeder Programmiersprache objektorientiert zu programmieren, auch in Assembler. Die soeben erwähnte Sprache ist ein für Menschen lesbares Abbild dessen, was ein Prozessor verarbeitet und ist somit dem Maschinencode ebenbürtig. Ungeachtet der verwendeten Programmiersprache muss Software, sofern sie mit dem Ziel entwickelt wurde auf einem Prozessor lauffähig zu sein, im letzten Schritt in eine für die Hardware verständliche Sprache übersetzt werden. Auch objektorientierte Software liegt somit in letzter Konsequenz als Assemblerprogramm vor, und somit hat der Umkehrschluss durchaus seine Richtigkeit.

Eigenschaften objektorientierter Programmierung

[13] Das Paradigma objektorientierte Programmierung baut darauf auf, Software entsprechend eines relitätsnahen Musters zu entwickeln. Das Objekt ist dabei elementarer Bestandteil des Paradigmas. Es besitzt *Attribute*, welche seine Eigenschaft bestimmen, sowie *Methoden*, die sein Verhalten beschreiben. Anhand des Beispiels eines Reglers werden die Begriffe der objektorientierten Programmierung näher beschrieben:

Abstraktion Da es umständlich wäre jedes Objekt extra zu implementieren, werden ähnliche in einer *Klasse* zusammengefasst. Die Klasse versteht sich somit als Bauplan eines Objektes. Ein Regler besitzt interne Zustandsvariablen, sowie Methoden zur Berechnung des Ausgangs, zum Schreiben der Eingangsgröße und zum Lesen der Ausgangsgröße,

Kapselung Die Attribute und Methoden der Klasse sind so zusammengefasst, dass nur auf jene zugegriffen werden kann, die auch tatsächlich nach außen hin notwendig sind. Die Eingangs- und Ausgangsgröße des Reglers sind für die Anwendung sichtbar, während die Berechnung des Ausgangs sowie die dafür notwendigen Zustandsvariablen verborgen bleiben.

Vererbbarkeit Eine Spezielle Klasse kann die Eigenschaften der Basisklasse erben. Ein PI-Regler erbt demnach die Schnittstellen der generellen Regler-Klasse, haltet jedoch seine spezielle Methode zur Berechnung des Ausgangs, sowie gegebenenfalls weitere Zustandsvariablen.

Polymorphie Die Fähigkeit, dass ein Bezeichner zur Laufzeit unterschiedliche Datentypen annehmen kann, nennt man Polymorphie. Der PI-Regler besitzt die geerbte Bezeichnung zum Aufruf der Methode "Berechnung des Ausgangs„, es handelt sich dabei aber nicht um die Methode der allgemeinen Klasse, sonder stattdessen wird beim Aufruf die spezielle Methode des PI-Reglers ausgeführt.

Objektorientierte Programmierung bietet gegenüber strukturierter Programmierung mehr Modularität, wodurch das Austauschen beziehungsweise Erweitern bestehender Klassen vereinfacht wird. Durch die Kapselung ist sichergestellt, dass nicht aus Versehen eigene Daten des Objekts manipuliert werden

Objektorientierter Programmierung in C

Man kann sich nun an dieser Stelle berechtigterweise fragen, warum man nicht von vornherein auf eine Programmiersprache zurückgreift, welche für eine objektorientierte Softwareentwicklung geschaffen wurde. Begründet liegt die Wahl der Programmiersprache C darin, dass diese gerade im Embedded Bereich anderen Sprachen überlegen ist. Höhere Programmiersprachen sind oft auf der Abstraktionsschicht eines Betriebssystems aufgebaut. Jeder Zugriff auf Hardware muss in diesem Fall durch das Betriebssystem geleitet werden, was besonders bei minimalistisch gehaltenen Architekturen, wie sie bei Mikrocontrollern anzutreffen sind, zu unerwünschten Einbusen der Performance führt.

Mit C ist es möglich, hardwarenahe zu entwickeln, und sofern es sich um eine sogenannte freistehende Implementation von C handelt, ist auch kein Betriebssystem notwendig, um das Programm auf der Zielhardware laufen zu lassen. Die freistehende Implementation hat allerdings den Nachteil, dass es keine Standardbibliothek und somit auch keine für die objektorientierte Programmierung notwendige dynamische Speicherverwaltung gibt. Damit es nun dennoch möglich ist, in C objektorientiert zu entwickeln, bedarf es einiger Tricks, um einerseits Klassen mit öffentlichen als auch privaten Daten und Methoden zu realisieren, und andererseits den Speicher für die einzelnen Instanzen der Klasse bereitzustellen. [11] In der Firmware zur Regelung bürstenloser Motoren wird dies folgendermaßen bewerkstelligt: Zu jeder Klasse gibt es zwei Headerdateien, wobei eine die Strukturen der privaten Daten beschreibt, während die andere sowohl Prototypen öffentlicher Methoden und Daten enthält, als auch gleichzeitig die Schnittstelle der Klasse nach außen hin repräsentiert. Des Weiteren gibt es noch eine Sourcedatei, die den Quellcode aller Methoden enthält. Zum Compilieren der Klasse sind alle drei Dateien notwendig, während zur Einbindung der Klasse in andere Teile der Software lediglich die öffentliche Header-Datei verwendet wird, und somit alle privaten Daten und Methoden verborgen bleiben. Dem Umstand, dass es keine dynamische Speicherverwaltung gibt wirkt man entgegen, indem jede Klasse für eine begrenzte Anzahl ihrer Instanzen den dafür notwendigen Speicher bereits zur Compilezeit reserviert. Eine beispielhafte Implementation dazu zeigen die Auflistungen A.2, A.3 und A.4 im Anhang A.1 (Erweiterung der Motor-Control-Library, Seite 69).

4.1.2. Firmware

[10] Die Firmware zur Ansteuerung von BLDC-Motoren setzt sich im Wesentlichen aus der Motor-Control-Library (MCLibrary) und der Motor-Control-Application (MCApplication) zusammen. Diese Teile wiederum sind auf der Standardbibliothek von ST (STM32F10x StdLib) sowie dem Cortex Mikrocontroller Software Interface Standard (CMSIS) aufgebaut. Abbildung 4.2 zeigt die Struktur der Firmware mit dem dazugehörigen Application Programming Interface (API), welches die Verbindung zum Anwendungsprogramm herstellt.

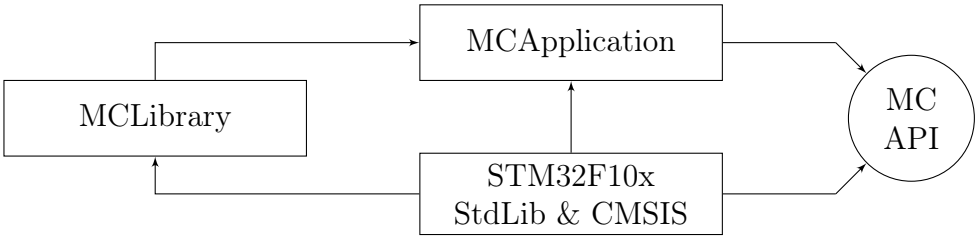

Abbildung 4.2.: Struktur der Firmware zur Regelung bürstenloser Motoren

In der frei zugänglichen Version der Firmware ist der Teil Motor-Control-Library nur als kompilierte Bibliothek enthalten. Der proprietäre Quellcode wurde seitens STMicroelectronics zur Verfügung gestellt, da einerseits die kompilierte Bibliothek zum verwendeten Compiler nicht kompatibel ist, und andererseits die Bibliothek erst um den verwendeten Winkelencoder sowie einige Schnittstellen erweitert werden musste.

Zeitkritische Tasks

In Abschnitt 3.3.1 (PWM-Modul zur Motoransteuerung, 32) wurde bereits erwähnt, dass die Messung des Motorstroms synchron zur PWM läuft. Nachdem die Digital-Analog-Umsetzung mit hoher Priorität abgeschlossen ist, wird ein Interrupt ausgelöst, in dessen Routine die zeitkritischen Tasks der feldorientierten Regelung ausgeführt werden. [11] Dazu zählen die Methoden *SPD_ GetElAngle()*, welche den elektrischen Winkel aus der Rotorposition berechnet, sowie *FOC_ CurrController()*. Letztere berechnet den auf den Rotor bezogenen Stromvektor I_{dq}, und wendet darauf die PI-Regler an, welche die Spannung U_{dq} zurück liefern. Weiters ermittelt die Methode aus dem Spannungsvektor die auf den Stator bezogene Spannung, und schließt somit den Kreis der Vektorregelung.

Die zeitkritischen Tasks werden synchron zur Pulsweitenmodulation ausgeführt, welche mit einer Frequenz von $18KHz$ arbeitet. Es bleibt somit recht wenig Zeit für die Ausführung der Tasks. Aus diesem Grund ist es wichtig, dass die Algorithmen effizient implementiert sind, und sich daraus eine kurze Laufzeit ergibt.

QUAD_SpeednPosFdbkClass

Bei der Einbindung des magnetischen Winkelencoders hat sich das Konzept objektorientierter Programmierung bewährt. So konnten die Schnittstellen der Basisklasse *Speed-*

nPosFdbkClass übernommen werden, und die Implementation der für den Encoder notwendigen Algorithmen war ohne größere Änderungen der Bibliothek möglich. Die für die feldorientierte Regelung wichtige Methode *SPD_GetElAngle()* wird wie zuvor erwähnt als erste in der Interrupt Routine der zeitkritischen Tasks ausgeführt. Zum Zeitpunkt des Interrupts sind die gemessenen Werte des Winkelencoders bereits im Register des Analog-Digital-Wandlers gespeichert. Aus diesen Werten wird der mechanische Rotorwinkel berechnet. Eine detaillierte Beschreibung dazu folgt in Abschnitt 4.3 (Erfassung des Rotorwinkels, Seite 59). Weiters wird aus dem Rotorwinkel, durch Multiplikation mit der Polpaarzahl des Motors, der elektrische Winkel ermittelt.

Die Basisklasse *SpeednPosFdbkClass* wurde um die Methoden *SPD_GetRotations()* und *SPD_ClearRotations()* erweitert. Die erste Methode retourniert die Anzahl der zurückgelegten Umdrehungen des Motors, während mit der zweiten der dahinter liegende Rundenzähler auf Null gesetzt wird. Der Zähler ist so implementiert, dass je nach Drehrichtung nach oben beziehungsweise nach unten gezählt wird. In weiterer Folge lässt sich daraus beispielsweise die Position eines dem Motor nachgeschalteten Antriebs ermitteln, worauf im Abschnitt 4.1.5 näher eingegangen wird.

Der Quellcode der Klasse *QUAD_SpeednPosFdbkClass*, bestehend aus den Dateien in den Auflistungen A.2, A.3 und A.4, befindet sich im Anhang A.1 (Erweiterung der Motor-Control-Library, Seite 69) Zusätzlich zur Implementation der Klasse *QUAD_SpeednPosFdbkClass* musste die bestehende Klasse *PWMnCurrFdbkClass* angepasst werden. Dabei wurden die Methoden zur Konfiguration des Analog-Digital-Wandlers für die jeweils nächste Strommessung so geändert, dass zusätzlich die Rohdaten des Winkelencoders mit hoher Priorität gemessen werden.

Zustandsmaschine

Die Motor-Control-Application ist als Zustandsmaschine implementiert, deren struktureller Aufbau Abbildung 4.3 entnommen werden kann. Die Zustände unterscheiden sich grundsätzlich in ihrer „Verweilzeit“. Dauerhafte Zustände sind quasi eingeschwungene Zustände, aus denen die Zustandsmaschine erst nach einer externen Eingabe wechselt. Bei den flüchtigen Zuständen handelt es sich um „Durchläufer“, welche beim Übergang von einem dauerhaften Zustand zum nächsten einmal abgearbeitet werden und ohne externe Beeinflussung des Benutzers wechseln. Im Folgenden werden die in Abbildung 4.3 gezeigten Zustände genauer beschrieben:

IDLE dauerhafter Zustand

Der Motor ist ausgeschaltet. Nachfolgende Zustände können sein: IDLE_START oder IDLE_ALIGNMENT, sofern ein Motor-Start-Befehl beziehungsweise ein Motor-Alignment-Befehl gegeben wurde.

IDLE_ALIGNMENT flüchtiger Zustand

In diesem Zustand wird der Code nur einmal ausgeführt, nämlich beim Übergang von IDLE nach ALIGNMENT. Der gewöhnliche Folgezustand ist ALIGNMENT, wenn jedoch ein Motor-Stop-Befehl ansteht, wird dieser ausgeführt und es folgt der Zustand ANY_STOP.

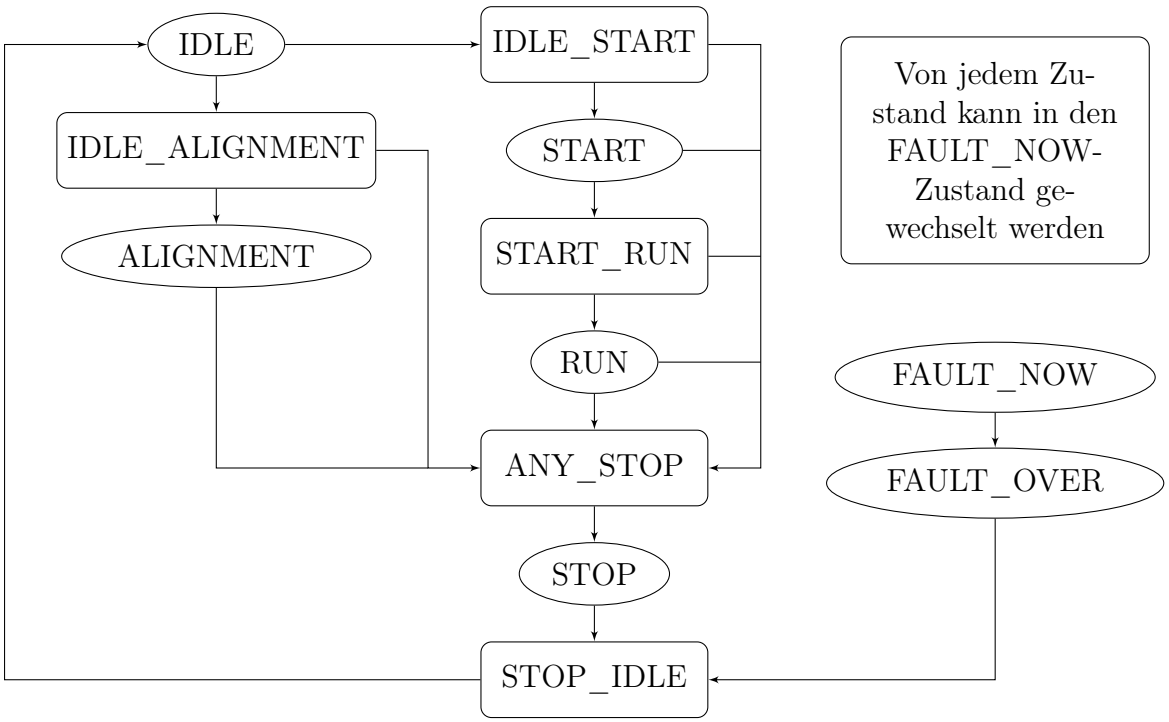

Abbildung 4.3.: Zustandsdiagramm der Motor-Control-Application

ALIGNMENT dauerhafter Zustand
Ausrichten der Motorwelle an den Winkelsensor. Der folgende Zustand kann nur ANY_STOP sein.

IDLE_START flüchtiger Zustand
Code, der beim Übergang von IDLE nach START nur einmal ausgeführt werden soll, wird in diesem Zustand abgearbeitet. Der gewöhnliche Folgezustand ist START, wenn jedoch ein Motor-Stop-Befehl ansteht wird dieser ausgeführt und es folgt der Zustand ANY_STOP.

START dauerhafter Zustand
In diesem Zustand wird der Motor hochgefahren. Wenn die Software eine gültige Drehzahl erfasst hat, schaltet die Zustandsmaschine weiter nach START_RUN. Ein Motor-Stop-Befehl bringt einen Wechsel nach ANY_STOP mit sich.

START_RUN flüchtiger Zustand
Code, der beim Übergang von START nach RUN nur einmal ausgeführt werden soll, wird in diesem Zustand abgearbeitet. Der gewöhnliche Folgezustand ist RUN, wenn jedoch ein Motor-Stop-Befehl ansteht, wird dieser ausgeführt und es folgt der Zustand ANY_STOP.

RUN dauerhafter Zustand
In diesem Zustand läuft der Motor. Sobald ein Motor-Stop-Befehl ausgeführt wird,

wechselt der Zustand auf ANY_STOP.

ANY_STOP flüchtiger Zustand
Der Code in diesem Zustand wird vom Übergang eines beliebigen Zustandes nach STOP genau einmal ausgeführt.

STOP dauerhafter Zustand
In diesem Zustand wird der Motor entschleunigt. Der darauf folgende Zustand ist für gewöhnlich STOP_IDLE, sobald eine Bedingung zum Wechseln des Zustandes ansteht.

STOP_IDLE flüchtiger Zustand
Der Code in diesem Zustand wird beim Übergang von STOP nach IDLE genau einmal ausgeführt.

FAULT_NOW dauerhafter Zustand
Im Falle eines Fehlers kann die Zustandsmaschine von jedem beliebigen Zustand hierher wechseln. Ausgelöst wird dieser Sprung durch die Methode STM_FaultProcessing, welche auch den einzig möglichen Wechsel nach FAULT_OVER behandelt.

FAULT_OVER dauerhafter Zustand
Sobald der eigentliche Fehler nicht mehr ansteht, schaltet die Zustandsmaschine hierher. Wenn nun der Fehler quittiert wird, wechselt die Zustandsmaschine nach STOP_IDLE.

Fehlermeldungen

Während des Betriebs des Motors kann es zu Fehlern kommen, welche in den beiden zuvor beschrieben Zuständen *FAULT_NOW* und *FAULT_OVER* behandelt werden. [11] Über die Methode *STM_GetFaultState()* des Objektes *oSTM* kann die Ursache des Fehlers abgefragt werden. Im Rückgabewert sind an den Positionen der Bits die Fehler entsprechend Auflistung 4.1 codiert. Mit der Methode *MCI_FaultAcknowledged()* des Motor-Control-Interfaces wird die Fehlermeldung quittiert, und sofern keine weiteren Fehler anstehen, wechselt die Zustandsmaschine wie zuvor beschrieben in den Zustand STOP_IDLE. Von diesem Zustand aus ist wieder ein normaler Betrieb der Firmware möglich.

Auflistung 4.1: Definition der Fault States

```
1 #define MC_NO_FAULTS      (uint16_t)(0x0000u)
2 #define MC_FOC_DURATION   (uint16_t)(0x0001u)
3 #define MC_OVER_VOLT      (uint16_t)(0x0002u)
4 #define MC_UNDER_VOLT     (uint16_t)(0x0004u)
5 #define MC_OVER_TEMP      (uint16_t)(0x0008u)
6 #define MC_START_UP       (uint16_t)(0x0010u)
7 #define MC_SPEED_FDBK     (uint16_t)(0x0020u)
8 #define MC_BREAK_IN       (uint16_t)(0x0040u)
```

```
9 #define MC_SW_ERROR      (uint16_t)(0x0080u)
```

4.1.3. Flux weakening Control

Die geschlossene Schleife der feldorientierten Regelung berechnet als Stellgröße für den Motor dessen transformierte Rotorspannung u_{dq}, deren Betrag unter anderem von der Last am Motor abhängt. Ist bei geringer Last der gemessene auf den Rotor bezogene Strom i_{dq} geringer als der Sollstrom i_{dq*}, dann gehen die Ausgänge der PI-Regler der feldorientierten Regelung, welche die auf den Rotor bezogene Statorspannung u_{dq} einstellen, in Sättigung. In diesem Fall besteht nun grundsätzlich die Möglichkeit, die direkte Stromkomponente i_{d*} betragsmäßig zu erhöhen, sodass in weiterer Folge der tatsächliche Strom i_d ansteigt, bis der Betrag der Spannung u_{dq} unter die Sättigungsspannung absinkt. Ein negativer direkter Strom i_{d*} beschleunigt den Motor über die Nenndrehzahl hinaus, während ein positiver Strom den Motor unter Anstieg der Verluste abbremst. Abbildung 4.4 zeigt die Regelung der Flussabschwächung, welche diese Eigenschaft von BLDC-Motoren ausnutzt und dadurch den Betriebsbereich dieser Motoren erweitert.

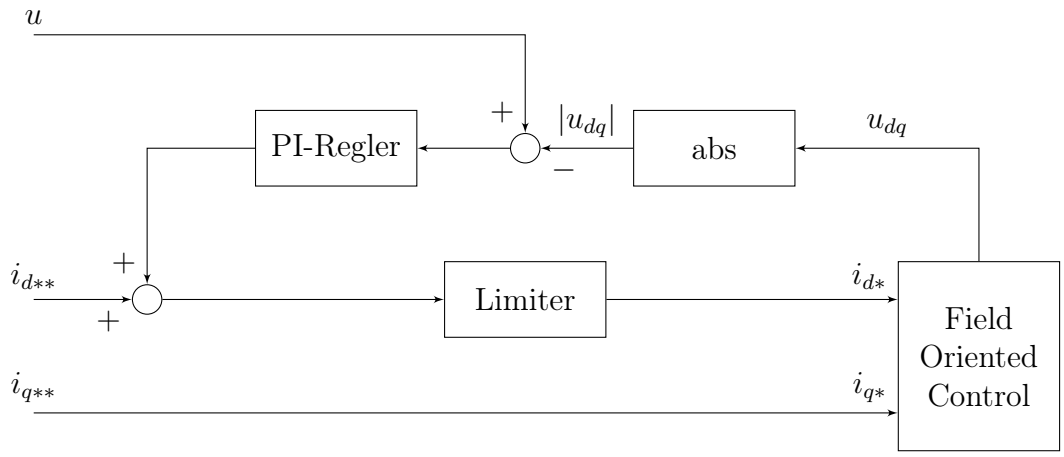

Abbildung 4.4.: Regelung der Flussabschwächung

Die Regelung des direkten Stromes i_{d*} passiert folgendermaßen: Der Betrag, der auf den Rotor bezogenen Spannung $|u_{dq}|$ wird mit der Sollspannung u verglichen, welche etwas geringer als die Sättigungsspannung gewählt wird. Wenn die Spannung $|u_{dq}|$ über der Sollspannung liegt, dann stellt sich am Ausgang des PI-Reglers eine negative Größe ein, welche zur Stromkomponente i_{d**} addiert wird. Sofern die Summe beider Werte negativ ist, kann diese den Limiter passieren, und stellt somit die neue Sollgröße i_{d*} für die feldorientierte Regelung dar. Liegt die Spannung $|u_{dq}|$ hingegen unter der Sollspannung, dann wird der Ausgang des PI-Reglers positiv. Für den Fall dass am Limiter ein positiver Wert anliegt, begrenzt dieser die Sollgröße i_{d*} auf 0, damit ein Abbremsen des Motors vermieden wird.

4.1.4. Maximum Torque per Ampere

In Abschnitt 2.1.2 (Eingebettete Rotormagnete, Seite 4) wurde bereits erwähnt, dass auf Grund der Reluktanz auch die direkte Stromkomponente i_d zur Drehmomentbildung beiträgt. Derartige Motoren weisen ein Drehmoment-Kennlinienfeld auf, ähnlich wie in Abbildung 4.5 zu sehen ist. Im Vergleich dazu würden bei einem Motor mit oberflächig montierten Magneten die Drehmomentkennlinien näherungsweise parallel zu I_d verlaufen.

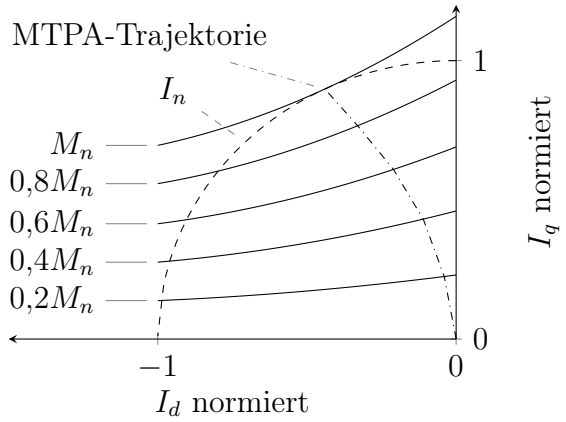

Abbildung 4.5.: Kennlinienfeld eines BLDC-Motors mit Reluktanzmoment

Aus diesem Diagramm lassen sich für jedes Drehmoment die dafür notwendigen Ströme i_{d**} und i_{q**} ablesen. Der minimale Strom, beispielsweise für das Nenndrehmoment M_n, lässt sich ermitteln, indem der Betrag der Stromkomponenten so gewählt wird, dass dieser die Drehmomentkennlinie gerade berührt, wobei dies bei Nennstrom $|i_{dq**}| = I_n$ der Fall ist. Führt man diese Prozedur für jede Drehmomentkennlinie durch, dann erhaltet man die **M**aximum-**T**orque-**p**er-**A**mpere-Trajektorie, aus welcher sich wiederum die Ströme i_{d**} und i_{q**} für maximales Drehmoment ablesen lassen. Die MTPA-Trajektorie lässt sich zwar graphisch recht einfach ermitteln, allerdings gibt es keinen effizienten Algorithmus, der die Berechnung in Echtzeit mit den beschränkten Ressourcen eines Mikrocontrollers ermöglicht. Aus diesem Grund ist der MTPA-Block so implementiert, dass mittels linearer Interpolation die Stromkomponenten ermittelt werden. Die dafür notwendigen Stützstellen können bereits im Vorfeld, außerhalb des Mikrocontrollers berechnet werden.

4.1.5. Position Conditioning

Der Block *Position Conditioning* dient dazu, die Anzahl der Umdrehungen des Motors $\Theta_{r\,mec}$ auf einen real zurückgelegten Weg l_{mec} am Ausgang des Getriebes abzubilden. Im einfachsten Fall, beispielsweise bei einem Lineargetriebe, handelt es sich bei der Funktion des Blocks um einen konstanten Faktor. Es reicht hier die Anzahl der Umdrehungen mit dem Übersetzungsverhältnis des Getriebes zu multiplizieren. Die Implementierung

eines eigenen Blocks erscheint möglicherweise etwas übertrieben, sie bringt jedoch den Vorteil der Modularität mit sich. Je nach Art des Getriebes können so auch komplexere nichtlineare Funktionen realisiert werden. Dadurch ist es möglich stets eine „echte“ Position anzufahren, ohne die Struktur des Positionsreglers für jeden Antrieb ändern zu müssen.

4.2. Kompatibilität vs. Performance

Die Motorcontrol Library ist so implementiert, dass diese dem MISRA-C[1] Standard genügt. Daraus ergeben sich Konstrukte, die zwar die Kompatibilität unterschiedlicher Compiler gewährt, jedoch geht das teilweise zu Lasten der Performance. Beispielsweise werden zwecks Genauigkeit in der Firmware Berechnungen mit 32-Bit durchgeführt, allerdings liegen die Eingangsvariablen in einer anderen Auflösung wie etwa 16-Bit oder 12-Bit vor. Meist wird anschließend das Ergebnis wieder auf eine geringere Genauigkeit reduziert. Die Shift-Operation mag zwar in diesem Fall als mögliche Lösung praktikabel erscheinen, jedoch kann dies auch durchaus Probleme bereiten:

[7] Für eine vorzeichenlose Variable ist das Verschieben trivial: Die Variable wird je nach vorgegebener Richtung und zusätzlich mit übergebenen Operanden um genau diese Anzahl an Stellen verschoben, wobei „rausgeschobene Stellen“ gelöscht, und die neu eingeschobenen Stellen mit Null befüllt werden. Die Shift-Operation entspricht im wesentlichen einer Multiplikation beziehungsweise einer Division mit einer Zahl 2^n, wobei n die Anzahl der zu schiebenden Stellen ist. Im Fachjargon wird diese Art des Verschiebens als *Logic-Shift* bezeichnet.

Bei einer vorzeichenbehafteten Variable ist das anders: In die Variable wird abhängig davon, ob das Vorzeichen-Bit gesetzt ist, entweder 0 oder 1 geschoben, wobei das Vorzeichen-Bit selbst unangetastet bleibt. Diese Art des Verschiebens ist auch unter der Bezeichnung *Arithmetic-Shift* bekannt.

Man kann hier erkennen, dass ein Verschieben einer vorzeichenbehafteten Variable, zwecks Darstellung in einer anderen Bit-Breite, nicht unbedingt das gewünschte Ergebnis bringt.

Abhilfe kann man dadurch schaffen, indem anstatt des Shift-Befehls multipliziert bzw. dividiert wird, oder man greift auf den passenden Shift-Operator des Maschinenbefehlssatzes zurück, welcher beispielsweise über Inline-Assembler aufgerufen wird. Die erste Variante hat den Nachteil, dass der Compiler eventuell schlecht optimiert, und somit Maschinencode mit längerer Laufzeit entsteht. Letztere hingegen hat den Nachteil, dass es im C-Standard keinerlei Richtlinien zur Syntax des integrierten Assemblers gibt. Auflistung 4.2 zeigt, wie für den *GNU-C Compiler* Assembler-Code in den C-Code integriert werden kann. In diesem Beispiel werden auf verschiedene Variablen die auf der ARM-Architektur verfügbaren Schiebefunktionen ASR (**A**rithmetic **S**hift **R**ight), LSR (**L**ogic **S**hift **R**ight) sowie LSL (**L**ogic **S**hift **L**eft) ausgeführt. Es handelt sich hier um

[1] MISRA (**M**otor **I**ndustry **S**oftware **R**eliability **A**ssociation) ist ein Programmierstandard aus der Automabilindustrie

ein Compiler-spezifisches Konstrukt, wodurch die Kompatibilität zu anderen Compilern nicht mehr gegeben ist.

Auflistung 4.2: *GNU-C*-Syntax für Inline Assembler

```
1  void inlineAssembler(int b, unsigned int d, unsigned int f)
2  {
3    int a;
4    unsigned int c, e;
5    asm(
6      "ASR %[asr_output], %[asr_input],#16\n"
7      "LSR %[lsr_output], %[lsr_input],#16\n"
8      "LSL %[lsl_output], %[lsl_input],#16\n"
9
10     // output registers
11     : [asr_output]"=r" (a),
12       [lsr_output]"=r" (c),
13       [lsl_output]"=r" (e)
14     // input registers
15     : [asr_input]"r" (b),
16       [lsr_input]"r" (d),
17       [lsl_input]"r" (f)
18   );
19   // do something with the shifted variables 'a', 'c' and 'e'
20 }
```

4.3. Erfassung des Rotorwinkels

Im Abschnitt 2.3.1 (Rotorwinkel) auf Seite 6 wurde bereits darauf eingegangen, welche Bedeutung der Rotorwinkel in der feldorientierten Regelung eines synchronen Drehstrommotors hat. Dieser Abschnitt widmet sich nun der Erfassung des Rotorwinkels, wobei unterschiedliche Techniken sowie deren Vor- und Nachteile näher beschrieben werden. Zu guter Letzt wird auf die tatsächlich implementierte Variante im Detail eingegangen.

4.3.1. Drehencoder

Drehencoder sind Bauteile, welche eine Drehbewegung in Impulse umformen. Meist werden die Impulse über zwei Signalleitungen ausgegeben, die hinsichtlich ihrer Phasenlage je nach Drehrichtung um $\pm 90°$ versetzt sind. Für Drehencoder gibt es unterschiedliche Anwendungsgebiete, zum Beispiel als Bedienelement für elektronische Geräte. Diese lösen je nach Type eine Umdrehung mit $15 \ldots 40$ Impulsen auf, welche mit mechanischen Kontakten erzeugt werden.

Für Positionierungsanwendungen gibt es Drehencoder, welche eine Umdrehung mit mehreren Hundert Impulsen auflösen. Die Impulse werden im Gegensatz zu den vorhin erwähnten Drehencodern über eine Infrarot Lichtschranke geformt. Bedingt durch diesen

Aufbau gibt es keine mechanische Reibung, und somit sind diese Bauteile verschleißfrei und langlebig. Verglichen mit anderen Methoden zur Erfassung des Rotorwinkels sind Drehencoder wesentlich größer, sie scheiden daher für die Anwendung aus.

4.3.2. Hallsensoren

Bei einem Großteil der am Markt erhältlichen Standard-BLDC-Motoren sind drei Hallsensoren integriert. Üblicherweise sind die Sensoren so angeordnet, dass diese jeweils um 120° elektrisch versetzt sind. Diese Konfiguration erlaubt quasi direkt eine Blockkommutierung des Motors, indem über den jeweils angeregten Hallsensor die räumlich darauf folgende Wicklung angesteuert wird. Mittels Interpolation kann der Rotorwinkel, beim drehenden Rotor über den zeitlichen Verlauf zwischen den drei Hallsensoren, wesentlich genauer bestimmt werden als auf die geometrisch bedingten 120°. Somit ist diese Methode zur Erfassung des Rotorwinkels für die feldorientierte Regelung geeignet.

Die Hallsensoren reagieren auf das magnetische Feld des Rotors, die Positionserfassung erfolgt somit berührungslos, wodurch ein langlebiger robuster Aufbau möglich ist. Hinsichtlich einer Serienfertigung, bei der auch ein kundenspezifischer Motor zum Einsatz kommen soll, bringen die drei Hallsensoren den Nachteil einer aufwändigen Verdrahtung und Platzierung im Gehäuse des Motors mit sich. Wünschenswert wäre daher ein Möglichkeit zur Erfassung des Rotorwinkels, welche die Vorteile der Hallsensoren mit einer einfachen Bestückung am Print ohne zusätzliche Verdrahtung vereint.

4.3.3. Magnetischer Winkelencoder

Magnetische Winkelencoder sind eine Sonderform der zuvor beschriebenen Hallsensoren. Es handelt hierbei um Integrierte Schaltungen, in denen ein 4-Quadranten Hallsensor-Array mit einer entsprechenden Signalverarbeitung gekoppelt ist. Die Erfassung des Winkels erfolgt indem am Ende der Rotorwelle ein radial ausgerichteter Magnet montiert ist, und der Sensor sich anschließen in Verlängerung der Rotorachse mit einem Abstand von zirka 1,5mm befindet. Das Ausgangssignal des Sensors entspricht bei dieser Anordnung für jede Umdrehung einer Periode eines Sinus- und Kosinussignals. Aus diesem Quadratursignal kann mittels Trigonometrischer Funktion Arkustangens der Winkel berechnet werden. Ein Vertreter aus der Palette der magnetischen Winkelencoder ist der IC *AS5115* . [15] Die Signalaufbereitung dieser Integrierten Schaltung kann von außen über eine serielle Schnittstelle parametriert werden. Hinsichtlich der am Chip vorliegenden magnetischen Feldstärke ist somit eine optimale Anpassung an den Messaufbau möglich. Eine detaillierte Beschreibung zur Parametrierung des *AS5115* folgt in Abschnitt 4.4.

Der Vollständigkeit halber sei noch erwähnt, dass es Magnetische Winkelencoder gibt, bei denen bereits die Signalverarbeitung zur Berechnung des Winkels enthalten ist. Auf diese Funktionalität kann allerdings verzichtet werden, da der verwendete Mikrocontroller über genügend Rechenleistung verfügt.

Arkustangens

Für die Berechnung des Rotorwinkels aus den beiden Komponenten $\cos(\Theta)$ und $\sin(\Theta)$ würde man nach der klassischen Methode zuerst den Quotienten bilden und daraus mittels Arkustangens den Winkel bestimmen. Dieser Ansatz hat mehrere Nachteile:

- Divisionen sind in der Digitaltechnik nur mit großem schaltungstechnischen Aufwand möglich, beziehungsweise benötigen viele Taktzyklen.
- Es kann zu Divisionen durch Null kommen.
- Der Definitionsbereich des Arkustangens liegt im ersten und vierten Quadranten. Wünschenswert wäre die Berechnung des Winkel über alle 4 Quadranten.

Es ist daher nicht besonders zielführend mit den auf einem Mikrocontroller verfügbaren Ressourcen den Winkel entsprechend der Schulmethode zu ermitteln. Eine Alternative dazu stellt der Arkustangens mit zwei Argumenten dar, welcher in der Computerarithmetik auch als *atan2(x,y)* bekannt ist. Dahinter verbirgt sich meist eine Implementation des CORDIC-Algorithmus, auf den im Folgenden näher eingegangen wird.

CORDIC-Algorithmus

CORDIC ist ein effizienter iterativer Algorithmus mit dem sich unterschiedliche Funktionen implementieren lassen, wie beispielsweise trigonometrische, logarithmische sowie Exponentialfunktionen. Die Abkürzung CORDIC steht für **CO**ordinate **R**otation **DI**gital **C**omputer.

[5] Der Algorithmus wurde entwickelt, um mit möglichst wenig Aufwand Koordinatenrotationen, sowie die Umwandlung von kartesischen in Polarkoordinaten bewerkstelligen zu können. Das Design des Algorithmus ist dabei so ausgelegt, dass die dafür notwendigen trigonometrischen Funktionen ausschließlich mit Additionen und Schiebeoperationen gelöst werden können. Diese Operatoren sind fixer Bestandteil der Rechenwerke von Mikroprozessoren und Mikrocontrollern.

Bei 16-Bit zeigen Benchmarktests [2] real eine Auflösung von 12 Bit. Angesichts der Tatsache, dass der Analog-Digital-Umsetzer ebenfalls mit 12 Bit auflöst, liegt bedingt durch den CORDIC-Algorithmus keine nennenswerte Beeinflussung des Ergebnisses vor. Viel mehr hängt die Exaktheit des berechneten Winkels davon ab, wie genau die Analogwerte der beiden Komponenten $\cos(\Theta)$ und $\sin(\Theta)$ sind.

Im Anhang A.1 (Erweiterung der Motor-Control-Library) zeigt die Auflistung A.1, beginnend ab Seite 69, die Implementation des 4-Quadranten Arcustangens.

[2] `http://www.mikrocontroller.net/articles/AVR_Arithmetik/Sinus_und_Cosinus_%28CORDIC%29#Genauigkeit`, (Abgerufen am 26.06.2012)

Offsetkorrektur

Nach der Analog-Digital-Umsetzung liegen die Cosinus- und Sinuskomponenten des Winkelencoders mit einem überlagerten Gleichanteil vor. Dieser entspricht der Offsetspannung des Winkelencoders. Laut Datenblatt [15] kann die Offsetspannung des ICs *AS5115* um $\pm 2\%$ vom nominellen Wert abweichen. Zusätzlich kann der Offset in Abhängigkeit von der Umgebungstemperatur im Bereich $\pm 50^{\mu V}/_{\circ C}$ driften.

Winkelfunktionen, wie auch immer diese realisiert sind, können nur so genau sein, wie die Eingangswerte auf die sie ausgeführt werden. Im Fall der Bestimmung des Winkels aus den Cosinus- und Sinuskomponenten ist es notwendig, dass die beiden Signale keinen Gleichanteil aufweisen. Wie bereits beschrieben wurde, handelt es sich beim Offset viel mehr um einen „Quasigleichanteil“, der im Vergleich zum Nutzsignal über lange Zeit schwankt. Um diesem variablen Gleichanteil entgegenzuwirken, wurde ein Algorithmus zur dynamischen Offsetkorrektur nach folgenden Überlegungen implementiert: Das Nutzsignal ist sinusförmig und hat daher die Eigenschaft, dass der arithmetische Mittelwert Null sein muss. Es ist in diesem Fall ausreichend, die Betrachtung auf die Scheitel zu reduzieren, da die Summe aus Maximum und Minimum ebenfalls Null sein muss. Extremwerte können in Software mit wenig Rechen- und Speicheraufwand erfasst werden. Auflistung 4.3 zeigt anhand von Pseudocode, wie die Offsetkorrektur funktioniert.

Auflistung 4.3: Pseudoalgorithmus zur Offsetkorrektur

```
1  procedure OffsetCorrection
2  set offset_x, max_x, min_x, offset_y, max_y, min_y to NOMINAL_OFFSET;
3  set i to NUMBER_OF_SAMPLES;
4  set last_x to x;
5  set last_y to y;
6
7  {Finde Extrema}
8  if x < min_x then min_x := x;
9  if x > max_x then max_x := x;
10 if y < min_y then min_y := y;
11 if y > max_y then max_y := y;
12
13 {Berechne Offset sobald genügend Samples erfasst wurden}
14 if i <= 0 then
15 begin
16   offset_x := (max_x + min_x) / 2;
17   offset_y := (max_y + min_y) / 2;
18   set max_x, min_x, max_y, min_y to NOMINAL_OFFSET;
19   i := NUMBER_OF_SAMPLES;
20 end
21
22 {Dekrementiere Samplezähler (i) nur wenn sich das Signal hinreichend ändert}
23 if abs(last_x − x) >= DELTA_MIN or abs(last_y − y) >= DELTA_MIN then
24   i := i − 1;
25 last_x := x;
```

```
26 last_y := y;
```

4.4. Parametrierung des Winkelencoders

Im Abschnitt 4.3.3 (Magnetischer Winkelencoder, Seite 60) wurde bereits erwähnt, dass der Winkelencoder *AS5115* zur Anpassung an die Feldstärke parametriert werden kann. [15] Weiters lassen sich am Signalausgang zwei verschiedene Offsetspannungen, sowie ein invertiertes Signal einstellen. Die Daten zur Parametrierung des Winkelencoders können sowohl temporär in einen flüchtigen Speicher, als auch dauerhaft in einen einmal programmierbaren Speicher geschrieben werden. Zur Programmierung dieses so genannten **O**ne-**T**ime **P**rogrammable (OTP) Speichers ist eine Spannung von $8V \ldots 8{,}5V$ am Programmiereingang des Winkelencoders notwendig. Zusätzlich sind zum Stützen dieser Spannung Blockkondensatoren notwendig, da während der Programmierung des OTP-Speichers hohe Ströme am Programmiereingang auftreten. Angesichts der Tatsache, dass die Parametrierung des Winkelencoders genau einmal erfolgt, erscheint es nicht besonders sinnvoll die Spannungsversorgung zur Programmierung des OTP-Speichers in die Schaltung mit einzubinden. Stattdessen wurde ein Programmieradapter aufgebaut, der alle zur einmaligen Parametrierung des Winkelencoders notwendigen Signale und Spannungen zu Verfügung stellt. Den Kern des Programmieradapters bildet der Mikrocontroller *ATMEGA128*. Es gibt mehrere Gründe, die für die Wahl dieses Controllers sprechen: Dieser Mikrocontroller, sowie sämtliche Entwicklungswerkzeuge wie Programmiergerät und Compiler (*avr-gcc*) und eine gewisse Routine im Umgang mit diesen Tools waren bereits vorhanden. Der *ATMEGA128* kann mit $5V$ versorgt werden, und stellt somit gleichzeitig an den Pins die selben Spannungspegel zur Verfügung, die auch der Winkelencoder *AS5115* benötigt. Der Mikrocontroller ist in einem DIL-Gehäuse untergebracht, wodurch der Aufbau des Programmieradapters auf einem Bread-Board möglich war. Im Anhang C.1 zeigt die Abbildung C.1, auf Seite 100, den Schaltplan des Programmieradapters.

Der Winkelencoder *AS5115* verfügt über eine synchrone serielle Schnittstelle, über die eine Parametrierung möglich ist. Diese Schnittstelle ist an keinen speziellen Standard wie SPI, I^2C oder ähnliche angelehnt, weshalb keine der für gewöhnlich in Mikrocontrollern enthaltenen seriellen Schnittstellen geeignet ist, um mit dem Winkelencoder eine Verbindung aufzubauen. Aus diesem Grund wurde die Verbindung zwischen Mikrocontroller und Winkelencoder über gewöhnliche Port-Pins hergestellt, und die Schnittstelle ausschließlich in Software implementiert. Im Anhang A.2 zeigen die Auflistungen A.5 und A.6 ab Seite 81 den Quell-Code mit allen notwendigen Funktionen zur Kommunikation und Parametrierung des Winkelencoders *AS5115* .

5. Ausblick

5.1. Änderungen in der Schaltung

Für eine serienreife Schaltung sind ein paar Änderungen des Prototyps notwendig. Dies liegt zum Einen begründet in den Erkenntnissen, die aus Versuchen mit dem Prototyp hervorgingen, und zum Anderen darin, dass hinsichtlich einer angedachten Serienfertigung sämtliche Optionen bereits in der Schaltung und somit auch im Layout mitberücksichtigt sein sollten. Hinsichtlich der Firmware ist es wünschenswert, dass für mögliche Erweiterungen die Ressourcen des Mikrocontrollers ausreichend sind, während insbesondere der Leistungsteil der Schaltung eventuellen Änderungen der Bauteile standhalten sollte, ohne dass ein neues Layout entworfen werden muss. Man denke nur daran, was passiert, wenn die Leistungstransistoren, aus welchen Gründen auch immer, geändert werden müssen: Für zukünftig hergestellte MOSFETs ist damit zu rechnen, dass bei selber Stromtragfähigkeit die Gatekapazität eher kleiner sein wird, als bei derzeitigen Transistoren. Die daraus resultierenden kürzeren Schaltzeiten können durchaus Probleme hinsichtlich EMV bereiten.

Im Folgenden werden nun Änderungen der Schaltung beschrieben, die den Anforderungen eines serienreifen Produktes Rechnung tragen. Im Anhang C.3 zeigen die Abbildungen C.7, C.8, C.9 und C.10 ab Seite 106 alle Teile des gesamten Schaltplans, wobei hier bereits alle Änderungen berücksichtigt sind.

5.1.1. Mikrocontroller

Der für den Prototyp verwendete Mikrocontroller ist in seiner Gehäusevariante jener mit dem größten Programmspeicher. Für den Fall, dass künftige Ideen in Software umgesetzt werden, und dabei das Platzangebot des Programmspeichers überschritten wird, müsste die Leiterplatte neu entworfen werden, weil vergleichbare Mikrocontroller mit mehr Speicher in einem größeren Gehäuse untergebracht sind. Die Bezeichnung „größer“ bezieht sich genau genommen auf die Anzahl der Pins. Ein Chipgehäuse, bei dem die Pins an der Unterseite angeordnet sind, ein so genanntes **B**all **G**rid **A**rray (kurz BGA), ist von den geometrischen Abmessungen her sogar kleiner als der beim Prototypen eingesetzte Mikrocontroller.

Die Verwendung eines Chips im BGA-Gehäuse erfordert am Print jedoch feinere Strukturen und andere Durchkontaktierungen, als dies beim Prototyp-Print der Fall ist.

5.1.2. Motortreiber

In der Treiberstufe (Abbildung 3.1) werden die Transistoren T_2 und T_3 gegen den komplementären Transistor *BC847BPN* getauscht, wo sowohl NPN als auch PNP in einem Gehäuse integriert sind. Auf die restliche Schaltung hat das keine nennenswerte Auswirkungen, da die Angaben in den Datenblättern [22] (*BC847*), [19] (*BC857*) und [25] (*BC847BPN*) sehr ähnlich sind. Lediglich die thermische Belastung des komplementären Transistors darf nicht so groß sein wie bei den einzelnen Bauteilen, allerdings verursacht die in Abschnitt 3.1.2 (Verlustleistung des Gatetreibers) auf Seite 17 für T_2 und T_3 gezeigte Verlustleistung beim *BC847BPN* nur eine geringe Erwärmung, sodass auch mit diesem Transistor ein sicherer Betrieb gewährleistet ist:

$$\Delta\vartheta = 2 \cdot P_V \cdot R_{th(j-a)} = 2 \cdot 5{,}04mW \cdot 568\frac{K}{W} = 5{,}7°C$$

Eine weitere Änderung betrifft die Ansteuerung des N-Kanal MOSFETs: Wie in Abschnitt 3.1.3 (Gatetreiber) auf Seite 19 bereits erwähnt wurde, sind die tatsächlichen Schaltzeiten des N-Kanal MOSFETs wesentlich kürzer als erwartet, sprich mit $t_{rise} = t_{fall} \approx 80ns$ liegen die Zeiten in einem Bereich, der hinsichtlich EMV noch vertretbar ist. Wenn nun andere Leistungstransistoren zum Einsatz kommen, deren Gateladung geringer ist als jene des Transistors *si4564dy*, dann werden diese in noch kürzerer Zeit schalten, und somit auch die mögliche Störemission vergrößern. Die Schaltzeiten lassen sich jedoch recht einfach vergrößern, indem ein Widerstand dem Gate vorgeschaltet wird. Wenn nun dieser Widerstand im Schaltplan, und somit auch im Layout, berücksichtigt wird, lässt sich die Schaltung leichter an andere Leistungstransistoren anpassen, ohne dass deshalb das Layout geändert werden muss.

5.1.3. Winkelencoder

Im Abschitt 4.4 (Parametrierung des Winkelencoders) auf Seite 63 wurde bereits darauf eingegangen, dass der Winkelencoder *AS5115* zwecks Anpassung des Ausgangssignals an die nachgeschaltete Signalverarbeitung, sowie an die Feldstärke des rotierenden Magneten eine einmalige Parametrierung benötigt. Neben einer dauerhaften Konfiguration des ICs, indem das OTP programmiert wird, besteht auch die Möglichkeit einer temporären Parametrierung. In diesem Fall geht die Information nach Abschalten der Versorgung des Chips verloren.

Die temporäre erfordert im Gegensatz zur dauerhaften Parametrierung keine hohe Programmierspannung, weshalb es möglich ist, die Konfiguration des Winkelencoders in der Schaltung mit dem ohnehin vorhandenen Mikrocontroller durchzuführen. Da der Mikrocontroller mit $3{,}3V$ betrieben wird, jedoch der Logikpegel des *AS5115* sich auf $5V$ bezieht, ist allerdings eine Pegelanpassung notwendig. Diese Anpassung lässt sich mit einem Trick sehr leicht bewerkstelligen, sodass nur für jede der drei Datenleitungen ein Pullup-Widerstand gegen $5V$ notwendig ist: Für die Kommunikation werden $5V$-tolerante Port-Pins verwendet. Diese werden in der Software als Open-Drain-Ausgang konfiguriert. Wenn die Software ein *low*-Signal am Pin ausgibt, dann zieht der Ausgang des Mikro-

controllers die Datenleitung gegen $0V$. Wenn hingegen ein *high*-Signal ausgegeben wird, dann zieht der Pullup-Widerstand die Datenleitung auf $5V$.

Letztendlich braucht nur noch der Pullup-Widerstand richtig dimensioniert werden. Von diesem und dem kapazitiven Belag der Datenleitung hängt die Anstiegszeit des Signals ab:

$$\tau = R_{Pullup} \cdot C_{Dataline}$$

Unter der Annahme, das die Datenleitung nach einer Zeit von 5τ sicher den *high*-Pegel erreicht hat, und noch einmal die selbe Zeit lang auf dem *high*-Pegel verweilen soll, darf eine Zeit von $t = 10\tau$ für die halbe Periodendauer nicht überschritten werden. Weiters wird angenommen, dass der gesamte kapazitive Belag der Datenleitung $100pF$ ausmacht. Somit lässt sich der Pullup-Widerstand bei gegebener Taktfrequenz des Datensignals folgendermaßen berechnen:

$$R_{Pullup} = \frac{1}{20 \cdot f \cdot C_{Dataline}} = \frac{1}{20 \cdot 200kHz \cdot 100pF} = 2{,}5k\Omega$$

Es ist jedoch darauf zu achten, dass der Widerstand keinesfalls zu klein gewählt wird, da sonst der für die Port-Pins maximal zulässige Strom überschritten wird. Entsprechend des Datenblattes des Mikrocontrollers [29] ergibt sich folgender minimal zulässiger Wert:

$$R_{Pullup} \geq \frac{U}{I_{max}} = \frac{5V}{25mA} = 200\Omega$$

Zu guter Letzt muss man noch berücksichtigen, dass durch die Pullup-Widerstände im ungünstigsten Fall der Strombedarf steigt. Dieser Strom muss zusätzlich vom Abwärtsregler aufgebracht werden, und beträgt bei Verwendung von Widerständen mit $R_{Pullup} = 2{,}2k\Omega$:

$$I_{Pullup} = 3 \cdot \frac{U}{R_{Pullup}} = 3 \cdot \frac{5V}{2{,}2k\Omega} = 6{,}8mA$$

5.1.4. Schaltbare Spannungsversorgung

Die zuvor gezeigte Möglichkeit einen „quasi-Pegelwandler“ mit einem $5V$-toleranten Pin des Mikrocontrollers und einem externen Pullup aufzubauen, lässt sich auch dazu verwenden, die im Abschnitt 3.4.3 (Abschaltung bei Standby-Betrieb) auf Seite 42 beschriebene Schaltung zu vereinfachen. Der Transistor zur Ansteuerung des MOSFETs kann dabei entfallen, sodass lediglich der zum Schalten der $5V$-Versorgung notwendige P-Kanal MOSFET und der Pullup-Widerstand zwischen den Anschlüssen Gate und Source übrig bleiben. In der Software ist dann noch zu berücksichtigen, dass der Pin des Mikrocontrollers als Open-Drain Ausgang konfiguriert wird, und die Schaltrichtung invertiert werden muss, das heißt, dass zum Einschalten der $5V$-Versorgung ein *Low*-Signal seitens des Mikrocontrollers notwendig ist, und vice versa.

5.1.5. EMV

In Abschnitt 3.5.1 (Schaltungstechnische Maßnahmen, Seite 43) wurde stillschweigend angenommen, dass die beschriebene Snubber-Schaltung sowie der Blockkondensator zur Dämpfung des Klingelns der MOSFETs ein Teil der Schaltung jeder Halbbrücke sei. Tatsächlich wurde die Snubber-Schaltung erst im Zuge von Maßnahmen zur Störungsunterdrückung entwickelt. Tatsächlich wurden der Leistungsteil erst im Zuge von Maßnahmen zur Störungsunterdrückung um diese Bauteile erweitert. Aus diesem Grund sind sie im Layout des Prototypen Prints nicht enthalten, und müssen daher für die nächste Revision berücksichtigt werden.

5.2. Änderungen der Software

In der Firmware steckt noch etwas Verbesserungspotential bezüglich Performance und Speicherbedarf, wie bereits in Abschnitt 4.2 (Kompatibilität vs. Performance, Seite 58) erwähnt wurde. Weiters steht auch eine komplexere Änderung betreffend des Kerns der Vektorregelung an: Beginnend ab dem Block *Reverse Park & circle limitation* erscheint es sinnvoller in Polarkoordinaten anstatt in kartesischen Koordinaten zu rechnen. Der Spannungsvektor U_{dq} kann, beispielsweise mittels Cordic-Algorithmus in Amplitude und Phase umgerechnet werden. Die Funktion *Reverse Park* ist dann trivial, da hier lediglich der Rotorwinkel zur Phase addiert wird. Für die Funktion *circle limitation* ist es dann nur noch notwendig die Amplitude zu beschränken.

In weiterer Folge vereinfacht sich die Raumvektor-Pulsweitenmodulation ($SVPWM$): Die Feldverteilung des Motors entspricht einer Funktion des Winkels, und ist für alle drei Wicklungen, nur mit unterschiedlicher Phasenlage, die selbe. Der Funktionswert der Feldverteilung lässt sich für jeden Strang aus der Summe der Phase, $\arg(U_{\alpha\beta})$ sowie dem räumlichen Versatz der Wicklungen ermitteln. Die Funktionswerte brauchen anschließend nur noch mit der Amplitude der Spannung $U_{\alpha\beta}$ multipliziert und in die Register des PWM-Moduls geladen werden. Dieser Ansatz zur Implementation des Blocks $SVPWM$ bietet eine hohe Flexibilität und Performance, wenn beispielsweise die Feldverteilung in einer Tabelle abgelegt ist.

A. Source Code

In diesem Abschnitt befindet sich der Quell-Code einiger ausgewählter Teile der Firmware zur feldorientierten Regelung und zur Parametrierung des Winkelencoders *AS5115* . Unter anderem soll dieser Code einen Einblick geben, wie das Softwareparadigma öbjektorientierte Programmierung in C"bewerkstelligt wurde.

Die Auflistung des gesamten Quell-Codes würde mehr als ein ganzes Buch füllen und ist deshalb an dieser Stelle nicht angebracht. Zusätzlich dazu wäre gedruckter Code im Gegensatz zu seinem Original in digitaler Form wohl kaum von Nutzen.

A.1. Erweiterung der Motor-Control-Library

In diesem Abschnitt des Anhangs befinden sich Teile der Firmware, die zur Gänze neu implementiert wurden, um die Motor-Control-Library an die entwickelte Hardware anzupassen. Es handelt sich dabei um die Quell-Codes des CORDIC-Algorithmus (Auflistung A.1), sowie des Winkelencoders (Auflistungen A.2, A.3, A.4).

Auflistung A.1: Implementation des CORDIC-Algorithmus zur Winkelberechnung

```
1  /**
2   *  @file     Cordic.c
3   *  @author   Clemens Treichler
4   *  @version  0.1
5   *  @date     May 2012
6   *  @brief    This file contains a 16-Bit implementation of the
7   *            CORDIC-Algorithm to compute the arcus-tangent.
8   */
9
10 #include "MC_type.h"
11 #include "Cordic.h"
12
13
14 #ifndef CORDIC_ITERATIONS
15  #error "The number of CORDIC_ITERATIONS has to be defined."
16 #endif
17
18 #if CORDIC_ITERATIONS > 14
19  #error "16-Bit CORDIC: Not more than 14 iterations allowed."
20 #endif
21
22
```

```
23 /**
24  *  @brief  This function computes the angle of the given X- and
25  *          Y-components of a cartesian coordinate system.
26  *          The angle is calculated over all 4 quadrants.
27  *  @param  X-Component of the cartesian coordinate system.
28  *  @param  Y-Component of the cartesian coordinate system.
29  *  @retval The calculated angle.
30  */
31 int16_t cordicAtan2(int16_t x , int16_t y)
32 {
33   unsigned char i;
34   int16_t temp = x;
35   const static const int16_t atan_tab[] = {8192, 4836, 2555, 1297, 651, 326, 163, 81, 41, 20,
       10, 5, 3, 1};
36   int16_t angle = 0x8000;
37
38   if (y < 0)
39   {
40     x = -y;
41     y = temp;
42     angle += 0x4000;
43   }
44   else
45   {
46     x = y;
47     y = -temp;
48     angle -= 0x4000;
49   }
50
51   for (i = 0; i < CORDIC_ITERATIONS; i++)
52   {
53     temp = x;
54     if (y < 0)
55     {
56       x -= y >> i;
57       y += temp >> i;
58       angle -= atan_tab[i];
59     }
60     else
61     {
62       x += y >> i;
63       y -= temp >> i;
64       angle += atan_tab[i];
65     }
66   }
67 //  cordic_abs = x; //length of the vector
```

```
68   return angle;
69 }
```

Auflistung A.2: Winkelencoder, Interface-Datenstrukturen

```
1  /**
2   *  @file     QUAD_SpeednPosFdbkClass.h
3   *  @author   Clemens Treichler
4   *  @version  0.1
5   *  @date     Feb 2012
6   *  @brief    This file contains puplic implementation of QUAD class
7   *
8   *  QUAD is an quadrature encoder wich measures the angle of a magnet
9   *  placed on the rotor.
10  *  Its output is sine and cosine of the rotor position.
11  *  Rotor angle calculation is done by arctan2(cosine, sine).
12  */
13
14 #ifndef __QUAD_SPEEDNPOSFDBKCLASS_H__
15 #define __QUAD_SPEEDNPOSFDBKCLASS_H__
16
17 #include "PWMnCurrFdbkClass.h"
18
19
20 /**
21  *  @brief  Public QUAD class definition
22  */
23 typedef struct CQUAD_SPD_t *CQUAD_SPD;
24
25
26 /**
27  *  @brief  QUAD class parameters definition
28  */
29 typedef const struct
30 {
31   GPIO_TypeDef*  quadXPort;         /*!< Port where the cosine component of the
       rotary encoder is connected. */
32   uint16_t     quadXPin;          /*!< Pin where the cosine component of the
       rotary encoder is connected. */
33
34   GPIO_TypeDef*  quadYPort;         /*!< Port where the sine component of the
       rotary encoder is connected. */
35   uint16_t     quadYPin;          /*!< Pin where the sine component of the rotary
        encoder is connected. */
36
37   uint8_t      quadSamplingTime;     /*!< Sampling time for the Encoder. Must be
       equal to ADC_SampleTime_xCycles5 x=1,7,... */
```

```
38  int16_t      quadOffset;         /*!< Nominal offset of the sine and cosine
      component of rotary Encoder. */
39  int16_t      quadOffsetLimit;     /*!< Maximal deviation offset is allowed to be
      within */
40  int16_t      quadSamples;        /*!< Number of samples reqired for each offset
      calculation */
41  int16_t      quadDeltaLimit;      /*!< Minimal difference of sine and cosine to
      previouse sampled components to count as sample */
42  int16_t      quadDeltaError;      /*!< Difference to previous sample that will be
       handled as error */
43  uint8_t      bSpeedBufferSize;     /*!< Size of the buffer used to calculate the
      average speed. It must be <= 16.*/
44  uint16_t     hSpeedSamplingFreq01Hz; /*!< Frequency (01Hz) at which motor speed
      is to be computed. It must be equal to the frequency at which function
      SPD_CalcAvrgMecSpeed01Hz is called.*/
45  FunctionalState RevertSignal;        /*!< To be enabled if measured speed is
      opposite to real one (ENABLE/DISABLE)*/
46 }QUADParams_t, *pQUADParams_t;
47
48
49 /**
50  *  @brief  Creates an object of the class QUAD
51  *  @param  pSpeednPosFdbkParams pointer to an SpeednPosFdbk parameters
      structure
52  *  @param  pQUADarams pointer to an QUAD parameters structure
53  *  @retval CQUAD_SPD new instance of QUAD object
54  */
55 CQUAD_SPD QUAD_NewObject(pSpeednPosFdbkParams_t pSpeednPosFdbkParams,
      pQUADParams_t pQUADParams);
56
57
58 #endif // __QUAD_SPEEDNPOSFDBKCLASS_H__
```

Auflistung A.3: Winkelencoder, private Datenstrukturen

```
1  /**
2   *  @file     QUAD_SpeednPosFdbkPrivate.h
3   *  @author   Clemens Treichler
4   *  @version  0.1
5   *  @date     Feb 2012
6   *  @brief    This file contains private implementation of QUAD class
7   *
8   *  QUAD is an quadrature encoder wich measures the angle of a magnet
9   *  placed on the rotor.
10  *  Its output is sine and cosine of the rotor position.
11  *  Rotor angle calculation is done by arctan2(cosine, sine).
12  */
```

```
13
14 #ifndef __QUAD_SPEEDNPOSFDBKPRIVATE_H__
15 #define __QUAD_SPEEDNPOSFDBKPRIVATE_H__
16
17 #define QUAD_SPEED_ARRAY_SIZE        ((uint8_t)16)
18
19 /**
20  *  @brief  QUAD class members definition
21  */
22 typedef struct
23 {
24   bool          SensorIsReliable;       /*!< Flag to indicate sensor/decoding is not
       properly working. */
25   int16_t wDeltaCapturesBuffer[QUAD_SPEED_ARRAY_SIZE]; /*!< Buffer used to
       store captured variations of timer counter */
26   volatile uint8_t  bDeltaCapturesIndex;      /*!< Buffer index */
27   uint16_t       hSpeedSamplingFreqHz;     /*!< Frequency (Hz) at which motor speed
        is to be computed. */
28
29   int16_t  mecAngle;       /*!< Real measured mechanical angle */
30   int16_t  lastMecAngle;    /*!< Last measured mechanical angle */
31   int16_t  angleOffset;     /*!< Offset to stator angle */
32
33   int32_t  angleDelta;      /*!< Difference to the last measured angle */
34   int64_t  angleAccumulated; /*!< Holds the over all accumulated angle, it is used
       to calculate number of rotor rotations */
35
36   int16_t  offset_x;       /*!< Actual offset of x component */
37   int16_t  offset_y;       /*!< Actual offset of y component */
38   int16_t  min_x;          /*!< Minimum of x */
39   int16_t  min_y;          /*!< Minimum of y */
40   int16_t  max_x;          /*!< Maximum of x */
41   int16_t  max_y;          /*!< Maximum of y */
42   int16_t  last_x;         /*!< Previouse sampled x */
43   int16_t  last_y;         /*!< Previouse sampled y */
44   int16_t  samples;        /*!< Samples counter */
45 }DVars_t,*pDVars_t;
46
47 /**
48  *  @brief  Redefinition of parameter structure
49  */
50 typedef QUADParams_t DParams_t, *pDParams_t;
51
52 /**
53  *  @brief Private QUAD class definition
54  */
```

```
55 typedef struct
56 {
57  DVars_t    DVars_str;       /*!< Derived class members container */
58  pDParams_t  pDParams_str;     /*!< Derived class parameters container */
59 }_DCQUAD_SPD_t, *_DCQUAD_SPD;
60
61
62 #endif // __QUAD_SPEEDNPOSFDBKPRIVATE_H__
```

Auflistung A.4: Winkelencoder, Quell-Code

```
1  /**
2   *  @file     QUAD_SpeednPosFdbkClass.c
3   *  @author   Clemens Treichler
4   *  @version  0.1
5   *  @date     Feb 2012
6   *  @brief    This file contains private implementation of QUAD class
7   */
8
9  /*  QUAD is a quadrature encoder wich measures the angle of a magnet
10  *  placed on the rotor.
11  *  Its output is sine and cosine of the rotor position.
12  *  Rotor angle calculation is done by arctan2(cosine, sine).
13  */
14
15
16 // Includes
17
18 #include "SpeednPosFdbkClass.h"
19 #include "SpeednPosFdbkPrivate.h"
20 #include "QUAD_SpeednPosFdbkClass.h"
21 #include "QUAD_SpeednPosFdbkPrivate.h"
22 #include "MC_type.h"
23 #include "MCLibraryConf.h"
24 #include "Cordic.h"
25 //#include "MCIRQHandlerPrivate.h"
26
27
28 // Private definitions
29
30 #define  DCLASS_PARAM  ((_DCQUAD_SPD)(((_CSPD) this)->DerivedClass))->
       pDParams_str
31 #define  DCLASS_VARS   &(((_DCQUAD_SPD)(((_CSPD) this)->DerivedClass))->
       DVars_str)
32 #define  CLASS_VARS    &(((_CSPD)this)->Vars_str)
33 #define  CLASS_PARAM   (((_CSPD)this)->pParams_str)
34
```

```
35
36 #ifdef MC_CLASS_DYNAMIC
37  #include "stdlib.h"  // Used for dynamic allocation
38 #else
39  _DCQUAD_SPD_t QUAD_SPDpool[MAX_QUAD_SPD_NUM];
40  unsigned char QUAD_SPD_Allocated = 0u;
41 #endif
42
43
44 // Prototypes
45
46 static void     QUAD_Init(CSPD this);
47 static void     QUAD_Clear(CSPD this);
48 static int16_t  QUAD_CalcAngle(CSPD this, void *pInputVars_str);
49 static bool     QUAD_CalcAvrgMecSpeed01Hz(CSPD this, int16_t *pMecSpeed01Hz);
50 static void     QUAD_SetMecAngle(CSPD this, int16_t hMecAngle);
51 static int32_t  QUAD_GetRotations(CSPD this);
52 static void     QUAD_ClearRotations(CSPD this);
53
54 /**
55  *  @brief  Creates an object of the class QUAD
56  *  @param  pSpeednPosFdbkParams pointer to an SpeednPosFdbk parameters
      structure
57  *  @param  pQUADarams pointer to an QUAD parameters structure
58  *  @retval CQUAD_SPD new instance of QUAD object
59  */
60 CQUAD_SPD QUAD_NewObject(pSpeednPosFdbkParams_t pSpeednPosFdbkParams,
      pQUADParams_t pQUADParams)
61 {
62   _CSPD _oSpeednPosFdbk;
63   _DCQUAD_SPD _oQUAD;
64
65   _oSpeednPosFdbk = (_CSPD)SPD_NewObject(pSpeednPosFdbkParams);
66
67 #ifdef MC_CLASS_DYNAMIC
68   _oQUAD = (_DCQUAD_SPD)calloc(1u,sizeof(_DCQUAD_SPD_t));
69 #else
70   if (QUAD_SPD_Allocated < MAX_QUAD_SPD_NUM)
71   {
72     _oQUAD = &QUAD_SPDpool[QUAD_SPD_Allocated++];
73   }
74   else
75   {
76     _oQUAD = MC_NULL;
77   }
78 #endif
```

```
79
80    _oQUAD->pDParams_str = pQUADParams;
81    _oSpeednPosFdbk->DerivedClass = (void*)_oQUAD;
82
83    _oSpeednPosFdbk->Methods_str.pSPD_Init = &QUAD_Init;
84    _oSpeednPosFdbk->Methods_str.pSPD_Clear = &QUAD_Clear;
85    _oSpeednPosFdbk->Methods_str.pSPD_CalcAngle = &QUAD_CalcAngle;
86    _oSpeednPosFdbk->Methods_str.pSPD_CalcAvrgMecSpeed01Hz = &
        QUAD_CalcAvrgMecSpeed01Hz;
87    _oSpeednPosFdbk->Methods_str.pSPD_SetMecAngle = &QUAD_SetMecAngle;
88
89    _oSpeednPosFdbk->Methods_str.pSPD_GetRotations = &QUAD_GetRotations;
90    _oSpeednPosFdbk->Methods_str.pSPD_ClearRotations = &QUAD_ClearRotations;
91
92    return ((CQUAD_SPD)_oSpeednPosFdbk);
93  }
94
95
96  /**
97   *  @brief  It initializes the analog Quadrature rotation encoder.
98   *          It must be called only after current sensor initialization (
        PWMC_Init)
99   *  @param  this related object of class CSPD
100  *  @retval none
101  */
102 static void QUAD_Init(CSPD this)
103 {
104   GPIO_InitTypeDef GPIO_InitStructure;
105   ADConv_t ADConv_struct;
106
107   pDParams_t pDParams = DCLASS_PARAM;
108   pDVars_t   pDVars   = DCLASS_VARS;
109
110   uint8_t bBufferIndex, bBufferSize = pDParams->bSpeedBufferSize;
111
112   // GPIOs configs
113   GPIO_StructInit(&GPIO_InitStructure);
114
115   // Configure Encoder input X
116   GPIO_InitStructure.GPIO_Pin = pDParams->quadXPin;
117   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
118   GPIO_Init(pDParams->quadXPort, &GPIO_InitStructure);
119
120   // Configure Encoder input Y
121   GPIO_InitStructure.GPIO_Pin = pDParams->quadYPin;
122   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
```

```
123   GPIO_Init(pDParams->quadYPort, &GPIO_InitStructure);
124
125   // initialize private variables
126   pDVars->max_x = pDVars->min_x = pDVars->max_y = pDVars->min_y = pDParams
        ->quadOffset;
127   pDVars->offset_x = pDVars->offset_y = pDParams->quadOffset;
128   pDVars->last_x = pDVars->last_y = pDParams->quadOffset;
129   pDVars->bDeltaCapturesIndex = 0u;
130   pDVars->hSpeedSamplingFreqHz = pDParams->hSpeedSamplingFreq01Hz / 10u;
131   pDVars->angleDelta = 0;
132   pDVars->angleAccumulated = 0;
133   pDVars->angleOffset = 820; // 8680; // 300; // -290; // -24293;
134
135   // Erase speed averaging buffer
136   for (bBufferIndex = 0u; bBufferIndex < bBufferSize; bBufferIndex++)
137     pDVars->wDeltaCapturesBuffer[bBufferIndex] = 0;
138 }
139
140
141 static void QUAD_Clear(CSPD this)
142 {
143   pDVars_t pDVars = DCLASS_VARS;
144   pDVars->SensorIsReliable = TRUE;
145 }
146
147
148 int16_t diff(int16_t a, int16_t b)
149 {
150   return (a >= b) ? (a - b) : (b - a);
151 }
152
153
154 static int16_t QUAD_CalcAngle(CSPD this, void *pInputVars_str)
155 {
156   pDParams_t pDParams = DCLASS_PARAM;
157   pDVars_t   pDVars   = DCLASS_VARS;
158   pParams_t  pParams  = CLASS_PARAM;
159
160   int32_t ElTemp;
161   int16_t delta, hElAngle, hMecAngle, lastMecAngle = pDVars->lastMecAngle;
162   int16_t adc_x, adc_y, x, y, offset_x, offset_y;
163   uint8_t i;
164   uint32_t temp_x, temp_y;
165
166   // read the components cosine and sine, stored in the injected ADC registers
167   x = (int16_t)(ADC1->JDR2);
```

```
168   y = (int16_t)(ADC2->JDR2);
169
170   // Dynamic offset correction:
171   // decrement sample counter only if the difference to the last sample is at
        least quadDeltaLimit.
172   if((diff(x, pDVars->last_x) >= pDParams->quadDeltaLimit) || \
173     (diff(y, pDVars->last_y) >= pDParams->quadDeltaLimit))
174   {
175     pDVars->samples--;
176   }
177   pDVars->last_x = x;
178   pDVars->last_y = y;
179
180   if(pDVars->samples <= 0)
181   {
182     // Start calculation of offset if there are enought samples
183     offset_x = (pDVars->max_x + pDVars->min_x) >> 1;
184     offset_y = (pDVars->max_y + pDVars->min_y) >> 1;
185     if((diff(offset_x, pDParams->quadOffset) <= pDParams->quadOffsetLimit) && \
186       (diff(offset_y, pDParams->quadOffset) <= pDParams->quadOffsetLimit))
187     {
188       pDVars->offset_x = offset_x;
189       pDVars->offset_y = offset_y;
190     }
191     pDVars->max_x = pDVars->min_x = pDVars->max_y = pDVars->min_y =
        pDParams->quadOffset;
192     pDVars->samples = pDParams->quadSamples;
193   }
194   else
195   {
196     // find minima and maxima
197     if(x > pDVars->max_x) pDVars->max_x = x;
198     if(x < pDVars->min_x) pDVars->min_x = x;
199     if(y > pDVars->max_y) pDVars->max_y = y;
200     if(y < pDVars->min_y) pDVars->min_y = y;
201   }
202
203   // Compute and store the mechanical angle
204   hMecAngle = cordicAtan2(x - pDVars->offset_x, y - pDVars->offset_y);
205   if (pDParams->RevertSignal)
206     hMecAngle = -hMecAngle;
207   pDVars->mecAngle = hMecAngle;
208   hMecAngle += pDVars->angleOffset;
209   ((_CSPD)this)->Vars_str.hMecAngle = hMecAngle;
210
211   // Calculate and store the electrical angle
```

```
212   hElAngle = hMecAngle * pParams->bElToMecRatio;
213   ((_CSPD)this)->Vars_str.hElAngle = hElAngle;
214
215   // Capture delta angle
216   delta = hMecAngle - lastMecAngle;
217   pDVars->angleDelta += delta;
218   pDVars->angleAccumulated += delta;
219
220   // Store actual angle (to calculate delta angle in next round)
221   pDVars->lastMecAngle = hMecAngle;
222
223   return hElAngle;
224 }
225
226 static bool QUAD_CalcAvrgMecSpeed01Hz(CSPD this, int16_t *pMecSpeed01Hz)
227 {
228   pDParams_t pDParams = DCLASS_PARAM;
229   pDVars_t   pDVars   = DCLASS_VARS;
230   pParams_t  pParams  = CLASS_PARAM;
231   pVars_t    pVars    = CLASS_VARS;
232
233   int32_t temp, angleDelta, wOverallAngleVariation = 0;
234   uint8_t bBufferIndex, bBufferSize = pDParams->bSpeedBufferSize;
235
236   angleDelta = pDVars->angleDelta;
237   pDVars->angleDelta = 0;
238   pDVars->wDeltaCapturesBuffer[pDVars->bDeltaCapturesIndex] = angleDelta;
239
240   // Compute & return average mechanical speed [01Hz]
241   for (bBufferIndex = 0u; bBufferIndex < bBufferSize; bBufferIndex++)
242     wOverallAngleVariation += (int32_t)pDVars->wDeltaCapturesBuffer[bBufferIndex];
243
244   temp = ((wOverallAngleVariation * (int32_t)(pDParams->hSpeedSamplingFreq01Hz)) / (
        int32_t)(bBufferSize)) >> 16;
245   *pMecSpeed01Hz = (int16_t)(temp);
246
247   // Compute & store average mechanical acceleration [01Hz/SpeedSamplingFreq]
248   pVars->hMecAccel01HzP = (int16_t)(temp) - pVars->hAvrMecSpeed01Hz;
249
250   // Store average mechanical speed [01Hz]
251   pVars->hAvrMecSpeed01Hz = (int16_t)(temp);
252
253   // Compute & store instantaneous electrical speed [dpp], var wtemp1
254   temp = angleDelta * (int32_t)(pDVars->hSpeedSamplingFreqHz) * \
255         (int32_t)(pParams->bElToMecRatio) / (int32_t)(pParams->hMeasurementFrequency);
256   pVars->hElSpeedDpp = (int16_t)(temp);
```

```
257
258   // Update buffer index
259   if((pDVars->bDeltaCapturesIndex + 1) < bBufferSize)
260     pDVars->bDeltaCapturesIndex++;
261   else
262     pDVars->bDeltaCapturesIndex = 0u;
263
264   return TRUE;
265 }
266
267
268 /**
269  *  @brief  This method sets the QUAD-Encoder to a given angle
270  *  @param  this related object of class CSPD
271  *  @param  angle the Encoder has to be set
272  */
273 static void QUAD_SetMecAngle(CSPD this, int16_t hMecAngle)
274 {
275   pDVars_t pDVars = DCLASS_VARS;
276   pDVars->angleOffset = hMecAngle - pDVars->mecAngle;
277 }
278
279
280 /**
281  *  @brief  This method returns absolute number of rotor rotations.
282  *          each positive full rotation increments and each negative
283  *          full rotation decrements the returned value.
284  *  @param  this related object of class CSPD.
285  *  @retval int32_t The number of rotor rotations.
286  */
287 static int32_t QUAD_GetRotations(CSPD this)
288 {
289   pDVars_t pDVars = DCLASS_VARS;
290   return (int32_t)(pDVars->angleAccumulated >> 16);
291 }
292
293
294 /**
295  *  @brief  This method clears the rotations counter.
296  *  @param  this related object of class CSPD.
297  */
298 static void QUAD_ClearRotations(CSPD this)
299 {
300   pDVars_t pDVars = DCLASS_VARS;
301   pDVars->angleAccumulated = 0;
302 }
```

A.2. Programmieradapter für Winkelencoder

Auflistungen A.5 und A.6 zeigen den Header sowie den Quellcode zur Kommunikation mit dem Winkelencoder. Die hier implementierten Funktionen ermöglichen das Auslesen des OTP-Speichers auf zwei unterschiedliche Varianten. Einerseits lässt sich der Speicher über die digitale Schnittstelle auslesen, und andererseits kann er auch über die Spannungspegel der Speicherzellen ausgelesen werden. Letztere dient dazu den Inhalt des OTP-Speichers auf korrekte Programmierung zu verifizieren. Die Funktionen zum Beschreiben des Winkelencoders erlauben sowohl eine temporäre als auch eine dauerhafte Parametrierung. Das Hauptprogramm (Auflistung A.7) zeigt beispielhaft, wie die Funktionen zur Parametrierung des Winkelencoders angewendet werden.

Auflistung A.5: Funktionen zur Parametrierung des Winkelencoders, Header

```
1  /**
2   *  @file     as5115.h
3   *  @author   Clemens Treichler
4   *  @version  0.1
5   *  @date     Feb 2012
6   *  @brief    This file contains definitions, function prototypes and the
7   *            data-structure of the AS5115 magnetic angle encoder.
8   */
9
10 #ifndef __AS5115_H__
11 #define __AS5115_H__
12
13 /**
14  *  Pin definitions for the serial data bus
15  */
16 #define AS_PORT         PORTA
17 #define AS_PIN          PINA
18 #define AS_DDR          DDRA
19 #define AS_CS           0
20 #define AS_DIO          1
21 #define AS_DCLK         2
22 #define AS_VPROG        3
23
24 /**
25  *  Define the analog channel to be used for EasyZap readback
26  */
27 #define AS_ADC_CHANNEL    4
28
29
30 /**
31  *  Definitions for AS5115's commands
32  */
33 #define AS_EN_PROG_COMMAND   0x10
```

```
34 #define AS_EN_PROG_DATA     0x8CAE
35 #define AS_WRITE_OTP        0x1F
36 #define AS_PROG_OTP         0x19
37 #define AS_RD_OTP           0x0F
38 #define AS_RD_OTP_ANALOG    0x09
39
40
41 /**
42  *  Extended readout OK return value
43  */
44 #define AS_RD_EXT_OK        0xF0
45
46
47 //-------------------------------------------------------------------------
48 //                          Some useful macros
49 //-------------------------------------------------------------------------
50 #define AS_VPROG_ON    AS_PORT |= (1 << AS_VPROG)
51 #define AS_VPROG_OFF   AS_PORT &= ~(1 << AS_VPROG)
52
53
54 /**
55  *  The AS5115 data structure
56  */
57 typedef struct _AS5115_OTP_t
58 {
59   uint32_t as;
60   uint16_t user;
61 } AS5115_OTP_t;
62
63
64 //  user data bits
65 //  +----------------+--------+--------+-------+-----------+-----------+
66 //  |      <11>      |  <10>  |  <9>   | <8:7> |    <6>    |   <5:0>   |
67 //  +----------------+--------+--------+-------+-----------+-----------+
68 //  | invert_channel | cm_sin | cm_cos | gain  | dc_offset | hall_bias |
69 //  +----------------+--------+--------+-------+-----------+-----------+
70 //  | 0: inverted    | 0: dif | 0: dif |       | 0: 1,5V   |           |
71 //  | 1: normal      | 1: com | 1: com |       | 1: 2,5V   |           |
72 //  +----------------+--------+--------+-------+-----------+-----------+
73
74
75 /**
76  *  @brief  Initializes the port pins used to communicate with the
77  *          angle encoder
78  */
79 void as5115Init(void);
```

```
80
81
82 /**
83  *  @brief  This function is the standard write function, where the special
84  *          command asWriteNormal(AS_EN_PROG_COMMAND, AS_EN_PROG_DATA)
85  *          unlocks the extended write and read funcions.
86  *  @param  command to be executed on the angle encoder.
87  *  @param  date to be sent to the angle encoder.
88  */
89 void asWriteNormal(uint8_t command, uint16_t data);
90
91
92 /**
93  *  @brief  This function writes the data from the AS5115_OTP_t structure to
94  *          the angle encoder.
95  *  @param  command to be executed on the angle encoder.
96  *  @param  data is the AS5115_OTP_t structure containing data to be written
97  *          to the angle encoder.
98  */
99 void asWriteExtended(uint8_t command, AS5115_OTP_t * data);
100
101
102 /**
103  *  @brief  This function reads the data into the AS5115_OTP_t structure.
104  *          This funcion is also used to check the voltage levels of the
105  *          programmed EasyZap OTP.
106  *  @param  data is the AS5115_OTP_t structure where data is read to.
107  *  @retval Returns AS_RD_EXT_OK. In case that there are errors during analog
108  *          readout it returns the address where the last error occured.
109  */
110 uint8_t asReadExtended(uint8_t command, AS5115_OTP_t * data);
111
112 #endif // __AS5115_H__
```

Auflistung A.6: Funktionen zur Parametrierung des Winkelencoders

```
1  /**
2   *  @file     as5115.c
3   *  @author   Clemens Treichler
4   *  @version  0.1
5   *  @date     Feb 2012
6   *  @brief    This file contains functions to communicate with the AS5115
7   *            magnetic angle encoder.
8   */
9
10 #include <avr/io.h>
11 #include <avr/interrupt.h>
```

```
12 #include <util/delay.h>
13 #include "adc.h"
14 #include "as5115.h"
15
16 #include <stdio.h>
17
18
19 #define AS_STARTUP_DELAY    5 // 5us
20 #define AS_NORMAL_DELAY     1 // 1us -> 500 kHz Periode
21 #define AS_OTP_DELAY       1 // 1us -> 500 kHz Periode
22 #define AS_OTP_ANALOG_DELAY  4 // 4us -> 125 kHz Periode
23
24 #define AS_ANALOG_LOW_LIMIT         (uint8_t)(0.5/5.0*256.0)
25 #define AS_ANALOG_HIGH_UPPER_LIMIT   (uint8_t)(3.5/5.0*256.0)
26 #define AS_ANALOG_HIGH_LOWER_LIMIT   (uint8_t)(2.0/5.0*256.0)
27
28 //-------------------------------------------------------------------------
29 //                          Some useful macros
30 //-------------------------------------------------------------------------
31 #define AS_CS_LO       AS_PORT &= ~(1 << AS_CS)
32 #define AS_CS_HI       AS_PORT |= (1 << AS_CS)
33 #define AS_DCLK_LO     AS_PORT &= ~(1 << AS_DCLK)
34 #define AS_DCLK_HI     AS_PORT |= (1 << AS_DCLK)
35 #define AS_DIO_SET_IN   AS_DDR  &= ~(1 << AS_DIO)
36 #define AS_DIO_SET_OUT  AS_DDR  |= (1 << AS_DIO)
37
38 #define AS_PIN_MASK   ((1<<AS_CS) | (1<<AS_DIO) | (1<<AS_DCLK) | (1<<
       AS_VPROG))
39
40
41 //-------------------------------------------------------------------------
42 //                          Low level functions
43 //-------------------------------------------------------------------------
44
45 void as5115Init(void)
46 {
47   AS_DDR  |= AS_PIN_MASK;
48   AS_PORT &= (~AS_PIN_MASK); // set AS-Chip data lines to low
49 }
50
51 void asWriteDIO(uint8_t val)
52 {
53   AS_PORT = val ? AS_PORT | (1 << AS_DIO) : AS_PORT & ~(1 << AS_DIO);
54 }
55
56 uint8_t asReadDIO(void)
```

```
57 {
58   return (AS_PIN & (1 << AS_DIO)) ? 0x01 : 0x00;
59 }
60
61 #define ANALOG_ERROR  0xFE
62
63 uint8_t asReadAnalog(uint8_t i)
64 {
65   uint8_t adc, ret = 0;
66
67   adc = (uint8_t)(adcGet(AS_ADC_CHANNEL) >> 8);
68   if(adc <= AS_ANALOG_LOW_LIMIT)
69     ret = 0x1;
70   else
71   {
72     // Return error if analog is out of range
73     if((adc < AS_ANALOG_HIGH_LOWER_LIMIT) || (adc >
        AS_ANALOG_HIGH_UPPER_LIMIT))
74       ret = ANALOG_ERROR;
75   }
76   if((i & 0xF) == 0)
77     printf("\r\nanalog %2d:", i);
78   printf(" %X,", adc);
79   return ret;
80 }
81
82
83 //---------------------------------------------------------------------------
84 //                           High level functions
85 //---------------------------------------------------------------------------
86
87 // Normal Protocol: 5 command bit + 16 data input output
88 // Command: 5-bit command: cmd<4:0> <- bit<20:16>
89 // Data:    16-bit data: data<15:0> <- bit<15:0>
90 void asWriteNormal(uint8_t command, uint16_t data)
91 {
92   uint8_t i;
93   _delay_us(AS_STARTUP_DELAY);
94   AS_DIO_SET_OUT;
95   _delay_us(AS_NORMAL_DELAY);
96   AS_DCLK_LO;
97   _delay_us(AS_NORMAL_DELAY);
98   AS_CS_HI;
99   _delay_us(AS_NORMAL_DELAY);
100  for(i = 0; i < 5; i++)
101  {
```

```
102     asWriteDIO((command << i) & 0x10);
103     _delay_us(AS_NORMAL_DELAY);
104     AS_DCLK_HI;
105     _delay_us(AS_NORMAL_DELAY);
106     AS_DCLK_LO;
107   }
108   for(i = 0; i < 16; i++)
109   {
110     asWriteDIO( ((data << i) & 0x8000) >> 15 );
111     _delay_us(AS_NORMAL_DELAY);
112     AS_DCLK_HI;
113     _delay_us(AS_NORMAL_DELAY);
114     AS_DCLK_LO;
115   }
116   _delay_us(AS_NORMAL_DELAY);
117   AS_CS_LO;
118 }
119
120
121 // Extended Protocol: 5 command bit + 46 data input output
122 // Command:   5-bit command: cmd<4:0> <- bit<50:46>
123 // Data:      46-bit data: data<45:0> <- bit<45:0>
124 // AS-Data:   34-bit <45:12>
125 // UserData:  12-bit <11:0>
126 // Each data bit needs 4 clock cycles
127 void asWriteExtended(uint8_t command, AS5115_OTP_t * data)
128 {
129   uint8_t i, j;
130   uint32_t temp;
131   _delay_us(AS_STARTUP_DELAY);
132   AS_DIO_SET_OUT;
133   _delay_us(AS_OTP_DELAY);
134   AS_DCLK_LO;
135   _delay_us(AS_OTP_DELAY);
136   AS_CS_HI;
137   _delay_us(AS_OTP_DELAY);
138   for(i = 0; i < 5; i++)
139   {
140     asWriteDIO((command << i) & 0x10);
141     _delay_us(AS_OTP_DELAY);
142     AS_DCLK_HI;
143     _delay_us(AS_OTP_DELAY);
144     AS_DCLK_LO;
145   }
146   temp = data->as;
147   for(i = 0; i < 34; i++)
```

```
148   {
149     if(i < 2)
150       asWriteDIO(1);
151     else
152     {
153       asWriteDIO( (temp & 0x80000000) >> 31 );
154       temp <<= 1;
155     }
156     for(j = 0; j < 4; j++)
157     {
158       _delay_us(AS_OTP_DELAY);
159       AS_DCLK_HI;
160       _delay_us(AS_OTP_DELAY);
161       AS_DCLK_LO;
162     }
163   }
164   temp = data->user;
165   for(i = 0; i < 12; i++)
166   {
167     asWriteDIO( ((temp << i) & 0x0800) >> 11 );
168     for(j = 0; j < 4; j++)
169     {
170       _delay_us(AS_OTP_DELAY);
171       AS_DCLK_HI;
172       _delay_us(AS_OTP_DELAY);
173       AS_DCLK_LO;
174     }
175   }
176   _delay_us(AS_OTP_DELAY);
177   AS_CS_LO;
178 }
179
180
181 uint8_t asReadExtended(uint8_t command, AS5115_OTP_t * data)
182 {
183   uint8_t i, j, t, analog, err_pos = 45, ret = AS_RD_EXT_OK;
184   uint8_t rd_as, rd_usr;
185   uint32_t temp = 0;
186
187   t = (command == AS_RD_OTP_ANALOG) ? AS_OTP_ANALOG_DELAY :
        AS_OTP_DELAY;
188
189   _delay_us(AS_STARTUP_DELAY);
190   AS_DIO_SET_OUT;
191   _delay_us(t);
192   AS_DCLK_LO;
```

```
193   _delay_us(t);
194   AS_CS_HI;
195   _delay_us(t);
196   for(i = 0; i < 5; i++)
197   {
198     asWriteDIO((command << i) & 0x10);
199     _delay_us(t);
200     AS_DCLK_HI;
201     _delay_us(t);
202     AS_DCLK_LO;
203   }
204   AS_DIO_SET_IN;
205   _delay_us(t);
206   for(i = 0; i < 46; i++)
207   {
208     err_pos--;
209     temp <<= 1;
210     if(command != AS_RD_OTP_ANALOG)
211       temp |= (uint32_t) asReadDIO();
212
213     for(j = 0; j < 4; j++)
214     {
215       AS_DCLK_HI;
216       if((command == AS_RD_OTP_ANALOG) && (j == 3))
217       {
218         analog = asReadAnalog(i);
219         if(analog == ANALOG_ERROR)
220           ret = err_pos;
221         temp |= (uint32_t)(analog & 0x1);
222       }
223       _delay_us(t);
224       AS_DCLK_LO;
225       _delay_us(t);
226     }
227
228     if(i == 33)
229     {
230       data->as = temp;
231       temp = 0;
232     }
233     if(i == 45)
234       data->user = (uint16_t) temp;
235   }
236   AS_CS_LO;
237   return ret;
238 }
```

Auflistung A.7: Programmieradapter Hauptprogramm

```
/**
 * @file main.c
 */

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/wdt.h>

#include <stdio.h>

#include "as5115.h"
#include "usart.h"
#include "adc.h"


//---------------------------------------------------------------------------
//                      program specific definitions
//---------------------------------------------------------------------------
#define LEDS_PORT       PORTA
#define LEDS_DDR        DDRA
#define BUSY_LED        7
#define GOOD_LED        6
#define ERROR_LED       5

#define KEYS_PORT       PORTB
#define KEYS_PIN        PINB
#define KEYS_DDR        DDRB
#define PROG_TEMP       0
#define PROG_OTP        1


//---------------------------------------------------------------------------
//                          Some useful macros
//---------------------------------------------------------------------------
#define BUSY_LED_ON     LEDS_PORT |= (1<<BUSY_LED)
#define BUSY_LED_OFF    LEDS_PORT &= ~(1<<BUSY_LED)
#define GOOD_LED_ON     LEDS_PORT |= (1<<GOOD_LED)
#define GOOD_LED_OFF    LEDS_PORT &= ~(1<<GOOD_LED)
#define ERROR_LED_ON    LEDS_PORT |= (1<<ERROR_LED)
#define ERROR_LED_OFF   LEDS_PORT &= ~(1<<ERROR_LED)

#define LEDS_MASK       ((1<<BUSY_LED) | (1<<GOOD_LED) | (1<<ERROR_LED))

#define KEYS_MASK       ((1<<PROG_TEMP) | (1<<PROG_OTP))

```

```
46
47 #define AS_CONFIG_VALUE  0x1B0 //  0x1BF  0x0640
48
49 int main(void)
50 {
51   uint8_t last_key = 0, key, test, read_ret;
52   AS5115_OTP_t as5115_data, as5115_verify;
53
54   KEYS_DDR  &= (~KEYS_MASK);
55   KEYS_PORT &= (~KEYS_MASK);
56   key = KEYS_PIN & KEYS_MASK;
57   last_key = key;
58
59   LEDS_DDR  |= LEDS_MASK;
60   LEDS_PORT &= (~LEDS_MASK);
61
62   adcInit(1 << AS_ADC_CHANNEL);
63   as5115Init();
64   usartInit();
65
66 // init watchdog
67   wdt_enable(WDTO_2S);
68
69 // enable interrupts
70   sei();
71
72   printf("\r\n\r\nProgrammer for AS5115 magnetic angle encoder\r\n");
73   while(1)
74   {
75     wdt_reset();
76     key = KEYS_PIN & KEYS_MASK;
77     if(key != last_key)
78     {
79       last_key = key;
80       if(key == (1<<PROG_TEMP))
81       {
82         // temporary overwriting of OTP content
83         ERROR_LED_OFF;
84         GOOD_LED_OFF;
85         BUSY_LED_ON;
86         asWriteNormal(AS_EN_PROG_COMMAND, AS_EN_PROG_DATA);
87         asReadExtended(AS_RD_OTP, & as5115_data);
88         as5115_data.user = AS_CONFIG_VALUE;
89 //          as5115_data.as = 0x12345678;
90         asWriteExtended(AS_WRITE_OTP, & as5115_data);
```

```
91        printf("\r\nTemporary overwritten - AS-Data: %08lX, User-Data: %03X",
      as5115_data.as, as5115_data.user);
92        BUSY_LED_OFF;
93       }
94       if(key == (1<<PROG_OTP))
95       {
96        // permanent programming of OTP
97        ERROR_LED_OFF;
98        GOOD_LED_OFF;
99        BUSY_LED_ON;
100       AS_VPROG_ON;
101       asWriteNormal(AS_EN_PROG_COMMAND, AS_EN_PROG_DATA);
102       asReadExtended(AS_RD_OTP, & as5115_data);
103       printf("\r\nBefore Prog ## AS-Data: %08lX, User-Data: %03X", as5115_data.as,
      as5115_data.user);
104       as5115_data.user = AS_CONFIG_VALUE;
105       asWriteExtended(AS_PROG_OTP, & as5115_data);
106       AS_VPROG_OFF;
107       // verify
108
109       asWriteNormal(AS_EN_PROG_COMMAND, AS_EN_PROG_DATA);
110       read_ret = asReadExtended(AS_RD_OTP, & as5115_verify);
111       printf("\r\nDigital      ## AS-Data: %08lX, User-Data: %03X", as5115_verify.as,
      as5115_verify.user);
112
113       asWriteNormal(AS_EN_PROG_COMMAND, AS_EN_PROG_DATA);
114       read_ret = asReadExtended(AS_RD_OTP_ANALOG, & as5115_verify);
115       if(read_ret != AS_RD_EXT_OK)
116       {
117        printf("\r\nerror @ position: %d", read_ret);
118        ERROR_LED_ON;
119       }
120       else
121       {
122        if(as5115_verify.user == as5115_data.user)
123         GOOD_LED_ON;
124        else
125         ERROR_LED_ON;
126       }
127       printf("\r\nAnalog       ## AS-Data: %08lX, User-Data: %03X", as5115_data.as,
      as5115_data.user);
128
129       BUSY_LED_OFF;
130      }
131     }
132    }
```

```
133 }
```

B. Skripte

Im Rahmen dieser Arbeit sind einige durchaus nützliche Skripte entstanden, mit deren Hilfe es möglich ist, Daten aus dem Programm *Eagle* so zu exportieren, dass diese sich mittels LaTeX formschön und vor allem konsistent in diese Arbeit einbinden haben lassen.

B.1. Schaltpläne nach TikZ exportieren

Auflistung B.1 zeigt ein sogenanntes *User Language Program*. Dieses wird innerhalb des Schaltplaneditors aufgerufen, und generiert eine Datei, die der TikZ-Syntax genügt. Diese Datei wiederum lässt sich mit dem Paket *tikz* in das LaTeX Dokument einbinden. Mit dem Makro `\du` lässt sich der Schaltplan beliebig skalieren, während über das Makro `\textSize` die Größe des Textes innerhalb des Schaltplanes eingestellt werden kann. Anwenden lassen sich die beiden Makros indem diese zu Beginn des Dokumentes folgendermaßen definiert werden:

```
\newlength{\du}
\setlength{\du}{3\unitlength}
\newcommand{\textSize}{\scriptsize}
```

In der Auflistung B.1 ist nicht der gesamte Quell-Code des User Language Programs enthalten, sondern lediglich die für den Export in das *tikz*- Format relevanten high-Level Funktionen. Die low-Level Funktionen wurden direkt vom User Language Program *eaglelatex*[1] übernommen.

Auflistung B.1: ULP zum Exportieren von Schaltplänen

```
1  #usage "en: <b>TikZ Converter for Latex</b>"
2  "<p>Converts a schematic into a file.tex.<br>"
3  "The resulting file is a TikZ drawing which can be used with <b>Latex</b>."
4  "<p>LateX-usage: requires packages \\usepackage{tikz}.<br>"
5  "Drawing is implemented into a <i>figure</i>environment then as: "
6  "\\input{file.tex}.<br>"
7  "Best output performance with PDFLatex (fills circle and rectangle).<br>"
8  "Design tips: SMASH Names to appropriate position; "
9  "suppress names by using numbers w/o literals; plain text allowed in layer 91."
10 "<p>Tested with Eagle V5.6."
11 "<p><author>Author: clemens.treichler@aon.at</author>"
```

[1] Die Datei *eaglelatex.ulp* ist frei verfügbar und kann von der Seite `http://www.cadsoft.de/downloads/ulps` (Abgerufen am 02.08.2012) herunter geladen werden.

```
12 " adapted from eaglelatex.ulp by m.haselberger@aon.at",
13 "de: <b>TikZ Konverter für Latex, V1.0</b>"
14 "<p>Konvertiert einen Schaltplan nach file.tex<br>"
15 "Die Datei ist eine TikZ-Zeichnung, welche in <b>Latex</b> inkludiert werden "
16 "kann."
17 "<p>LateX-Einsatz: benötigt folgende Files: \\usepackage{tikz}<br>"
18 "Zeichnung wird dann in eine <i>figure</i>-Umgebung eingefügt mit: "
19 "\\input{file.tex}.<br>"
20 "Das beste Ergebnis wird mit PDFLatex erreicht (circle und rectangle gefüllt)."
21 "<br>"
22 "Design Tipps: Funktion SMASH verwenden, um Name am Bauteil passend zu "
23 "platzieren; um Name zu unterdrücken, nur Zahlen verwenden; normaler Text im "
24 "Layer 91 möglich."
25 "<p>Getestet mit Eagle V5.6"
26 "<p><author>Autor: clemens.treichler@aon.at</author> "
27 "adaptiert von eaglelatex.ulp - m.haselberger@aon.at"
28
29 // Thanks to jeff@uClinux.org (pic.ulp) and m.haselberger@aon.at (eaglelatex.
       ulp)
30
31 // Feel free to improve this converter
32
33 enum { NO, YES };
34
35 int PadColor = 0, ViaColor = 0;
36
37 // Some tools we need later:
38
39 real PicUnit(int n)
40 {
41 //  return u2inch(n);
42   return u2mm(n);
43 //  return u2mil(n);
44 }
45
46 int LayerActive[] = {1};
47
48 real max_x1 = 0.0;
49 real max_y1 = 0.0;
50 real max_x2 = 0.0;
51 real max_y2 = 0.0;
52
53 /**
54  *  @brief  Generates the header of the output-file.
55  *          This header contains some abstracted TikZ commands.
56  */
```

```
57 void PicStart()
58 {
59   printf("\
60 %% Some definitions and functions\n\n\
61 \\ifx\\du\\undefined\n\
62   \\newlength{\\du}\n\
63 \\fi\n\
64 \\ifdim\\du<\\unitlength\n\
65   \\ifdim\\du<10\\unitlength\n\
66     \\setlength{\\du}{4\\unitlength}\n\
67   \\fi\n\
68 \\fi\n\
69 \\ifx\\textSize\\undefined\n\
70   \\newcommand{\\textSize}{\\small}\n\
71 \\fi\n\
72 \n\
73 \\begin{tikzpicture}[font=\\textSize]\n\
74 \\pgftransformxscale{1.0}\n\
75 \\pgftransformyscale{1.0}\n\
76 \n\
77 \\newcommand{\\DrawLine}[5]{%%\n\
78   \\pgfsetlinewidth{#1\\du}{%%\n\
79     \\definecolor{linecolor}{rgb}{0.00, 0.00, 0.00}\n\
80     \\pgfsetfillcolor{linecolor}\n\
81     \\definecolor{linecolor}{rgb}{0.00, 0.00, 0.00}\n\
82     \\pgfsetstrokecolor{linecolor}\n\
83     \\draw (#2\\du,#3\\du)--(#4\\du,#5\\du);\n\
84   }\n\
85 }\n\
86 \n\
87 \\newcommand{\\DrawBox}[5]{%%\n\
88   \\pgfsetlinewidth{0.0\\du}{%%\n\
89     \\fill[black,rotate around={#5:(#3\\du/2+#1\\du/2,#4\\du/2+#2\\du/2)}]
      (#1\\du,#2\\du) rectangle (#3\\du,#4\\du);\n\
90   }\n\
91 }\n\
92 \n\
93 \\newcommand{\\DrawCircle}[4]{%%\n\
94   \\pgfsetlinewidth{#1\\du}{%%\n\
95     \\draw (#2\\du,#3\\du) circle (#4\\du);\n\
96   }\n\
97 }\n\
98 \n\
99 \\newcommand{\\DrawFilledCircle}[3]{%%\n\
100   \\pgfsetlinewidth{0.0\\du}{%%\n\
101     \\fill[black] (#1\\du,#2\\du) circle (#3\\du);\n\
```

```
102   }\n\
103 }\n\
104 \n\
105 %% This is a workaround to draw arcs with an angle bigger than 360 degrees.\n\
106 %% Newer versions of TikZ schould be able to do this directly.\n\
107 \\newcommand{\\DrawArc}[6]{%%\n\
108   \\pgfsetlinewidth{#1\\du}{%%\n\
109     \\definecolor{linecolor}{rgb}{0.00, 0.00, 0.00}\n\
110     \\pgfsetfillcolor{linecolor}\n\
111     \\definecolor{linecolor}{rgb}{0.00, 0.00, 0.00}\n\
112     \\pgfsetstrokecolor{linecolor}\n\
113     \\ifthenelse{#5 > 360}%%\n\
114     { \\draw (#2\\du,#3\\du) arc(#4:360:#6\\du) arc(0:(#5-360):#6\\du); }%%\n\
115     { \\draw (#2\\du,#3\\du) arc(#4:#5:#6\\du); }%%\n\
116   }\n\
117 }\n\
118 \n\
119 \\newcommand{\\DrawText}[6]{%%\n\
120   \\definecolor{linecolor}{rgb}{0.00, 0.00, 0.00}\n\
121   \\pgfsetstrokecolor{linecolor}\n\
122   \\ifthenelse{#1 = 180}\n\
123   { \\node[anchor=base east] at (#3\\du+1\\du,#4\\du-#6\\du){#5}; }\n\
124   { \\ifthenelse{#2 = 1}\n\
125     { \\node[anchor=base east] at (#3\\du+1\\du,#4\\du){#5}; }\n\
126     { \\node[anchor=base west] at (#3\\du-1\\du,#4\\du){#5}; }\n\
127   }\n\
128 }\n\
129 \n\n\
130 %% Draw Picture\n\n\
131 ");
132 }
133
134 /**
135  *  @brief  This is the ending of the output-file
136  */
137 void PicEnd()
138 {
139   printf("\n\\end{tikzpicture}\n");
140 }
141
142 void PicLine(int layer, int width, int x1, int y1, int x2, int y2)
143 {
144   if (width <= 0) width = -1;
145     printf("\\DrawLine{%2.2f}{%f}{%f}{%f}{%f};\n",
146     PicUnit(width), PicUnit(x1), PicUnit(y1), PicUnit(x2), PicUnit(y2));
147 }
```

```
148
149 void PicBox(real angle, int layer, int x1, int y1, int x2, int y2)
150 {
151   real centerX,centerY, widthX, heightY, angleC;
152
153   centerX=(PicUnit(x1)+ PicUnit(x2))/2.0; //center of box
154   centerY=(PicUnit(y1)+ PicUnit(y2))/2.0; //center of box
155   widthX=(PicUnit(x2) - PicUnit(x1));
156   heightY=(PicUnit(y2) - PicUnit(y1));
157   angleC= PicUnit(angle);
158
159   printf("\\DrawBox{%f}{%f}{%f}{%f}{%f};\n", PicUnit(x1), PicUnit(y1), PicUnit(x2), PicUnit
        (y2), angleC*10000);
160 }
161
162 void PicCircle(int layer, int width, int x, int y, int r)
163 {
164   if (width <= 0) width = -1;
165   if (layer == 91)  //only net-circles are filled
166   {
167     printf("\\DrawFilledCircle{%f}{%f}{%f};\n",
168       PicUnit(x), PicUnit(y), PicUnit(r)*2);
169   }
170   else
171   {
172     printf("\\DrawCircle{%2.2f}{%f}{%f}{%f};\n",
173       PicUnit(width), PicUnit(x), PicUnit(y), PicUnit(r) );
174   }
175 }
176
177 void DrawArc(UL_ARC A)
178 {
179   printf("\\DrawArc{%2.2f}{%f}{%f}{%1.0f}{%1.0f}{%f};\n",
180     PicUnit(A.width), /* * 72.0, */ PicUnit(A.x1), PicUnit(A.y1),
181     PicUnit(A.angle1)*10000, PicUnit(A.angle2)*10000, PicUnit(A.radius) );
182 }
183
184 // TODO adapt to TikZ
185 void PicArc(int layer, int width, int xc, int yc, int x1, int y1, int x2, int y2)
186 {
187 //  printf("linethick = %f; arc at (%f, %f) from (%f, %f) to (%f, %f)\n",
188 //           PicUnit(width) * 72.0, PicUnit(xc), PicUnit(yc),
189 //           PicUnit(x1), PicUnit(y1), PicUnit(x2), PicUnit(y2));
190 }
```

B.2. Farbiger PostScript Drucker

Die Datei *$EAGLEDIR/bin/eagle.def* enthält Definitionen für Ausgabegeräte wie Drucker, Plotter, Bohrer, Fräser sowie generische Geräte. Die Ausgabe in eine Datei in einem Vektorformat kann mit den vorgegebenen Treibern lediglich auf einen Farbkanal durchgeführt werden. Zwecks besserer Lesbarkeit des Bestückungsplans sollte die Darstellung zumindest mit zwei Farben erfolgen. Aus diesem Grund wurde der vorhandene Treiber für *Encapsulated PostScrip* erweitert, damit die Ausgabe in mehreren Farbkanälen erfolgen kann. Angesteuert werden die Farben auf ähnliche Weise, wie bei einem Plotter das Wechseln des Stiftes erfolgt.

Im Layout-Programm kann mit Hilfe des sogenannten CAM-Prozessors eine automatisierte und individuell angepasste Ausgabe der einzelnen Lagen in allen möglichen Kombinationen erfolgen. Die in dieser Dokumentation verwendeten Abbildungen zum Layout wurden auf diese Weise generiert.

Auflistung B.2: PostScript Treiber mit mehreren Farbkanälen

```
1  [EPSCOLOR]
2
3  @EPS
4  Long    = "Encapsulated PostScript with color-map"
5  HwColor = "%u color\n" ; (Pen-Number)
6
7  Header2 = "%% color-map:\n"\
8          "\n"\
9          "/color {  %% set color depending on map number\n"\
10         "   /i exch def\n"\
11         "   i  0 eq {0 0 0 setrgbcolor} if\n"\
12         "   i  1 eq {1 0 0 setrgbcolor} if\n"\
13         "   i  2 eq {0 1 0 setrgbcolor} if\n"\
14         "   i  3 eq {0 0 1 setrgbcolor} if\n"\
15         "   i  4 eq {1 1 0 setrgbcolor} if\n"\
16         "   i  5 eq {0 1 1 setrgbcolor} if\n"\
17         "   i  6 eq {1 0 1 setrgbcolor} if\n"\
18         "   i  7 eq {0.25 0.25 0.25 setrgbcolor} if\n"\
19         "   i  8 eq {0.5 0.5 0.5 setrgbcolor} if\n"\
20         "   i  9 eq {0.75 0.75 0.75 setrgbcolor} if\n"\
21         "   i 10 ge {1 1 1 setrgbcolor} if\n"\
22         "   } def\n"\
23         "\n"
```

C. Schaltpläne und Layout-Lagen

C.1. Schaltung des Programmieradapters für Winkelencoder

Abbildung C.1, auf Seite 100, zeigt die Schaltung des Programmieradapters zur Parametrierung des Winkelencoders, der in Abschnitt 4.4 (Parametrierung des Winkelencoders, Seite 63) beschrieben wurde. Für diese Schaltung wurde keine Printplatte entwickelt, sondern sie wurde lediglich auf einem Breadboard aufgebaut.

C.2. Schaltung und Layout des Prototypen

Die Abbildungen C.2, C.3, C.4 und C.5 ab Seite 101 zeigen die Teile des Schaltplans des realisierten Prototyps. Abbildung C.6 zeigt die wichtigsten Lagen des Layouts.

C.3. Schaltung mit Änderungen

Die Abbildungen C.7, C.8, C.9 und C.10 ab Seite 106 zeigen alle Teile des Schaltplans, wobei hier die unter Abschnitt 5.1 (Änderungen in der Schaltung) beschriebenen Erweiterungen und Verbesserungen berücksichtigt sind.

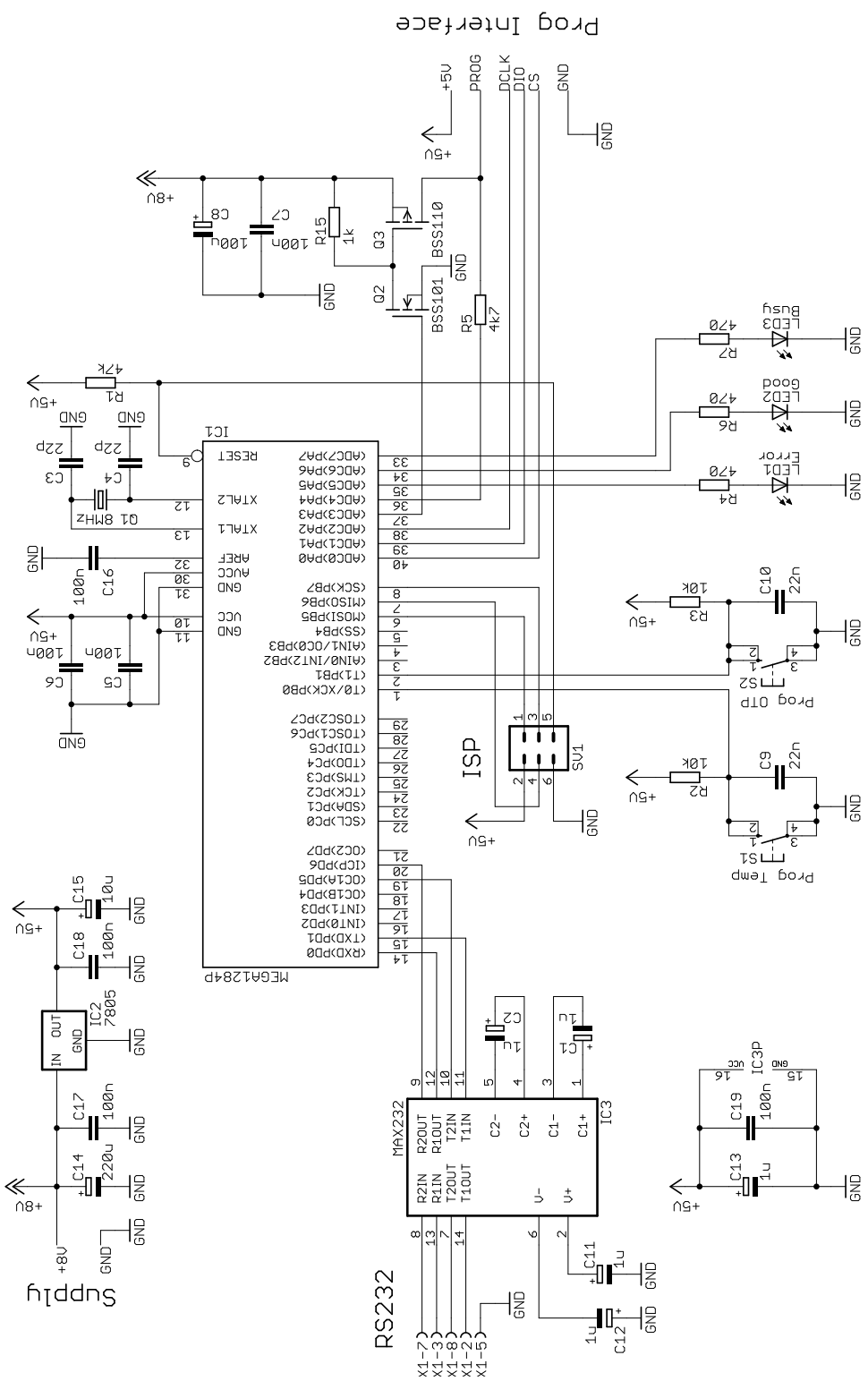

Abbildung C.1.: Schaltplan Programmieradapter für Winkelencoder

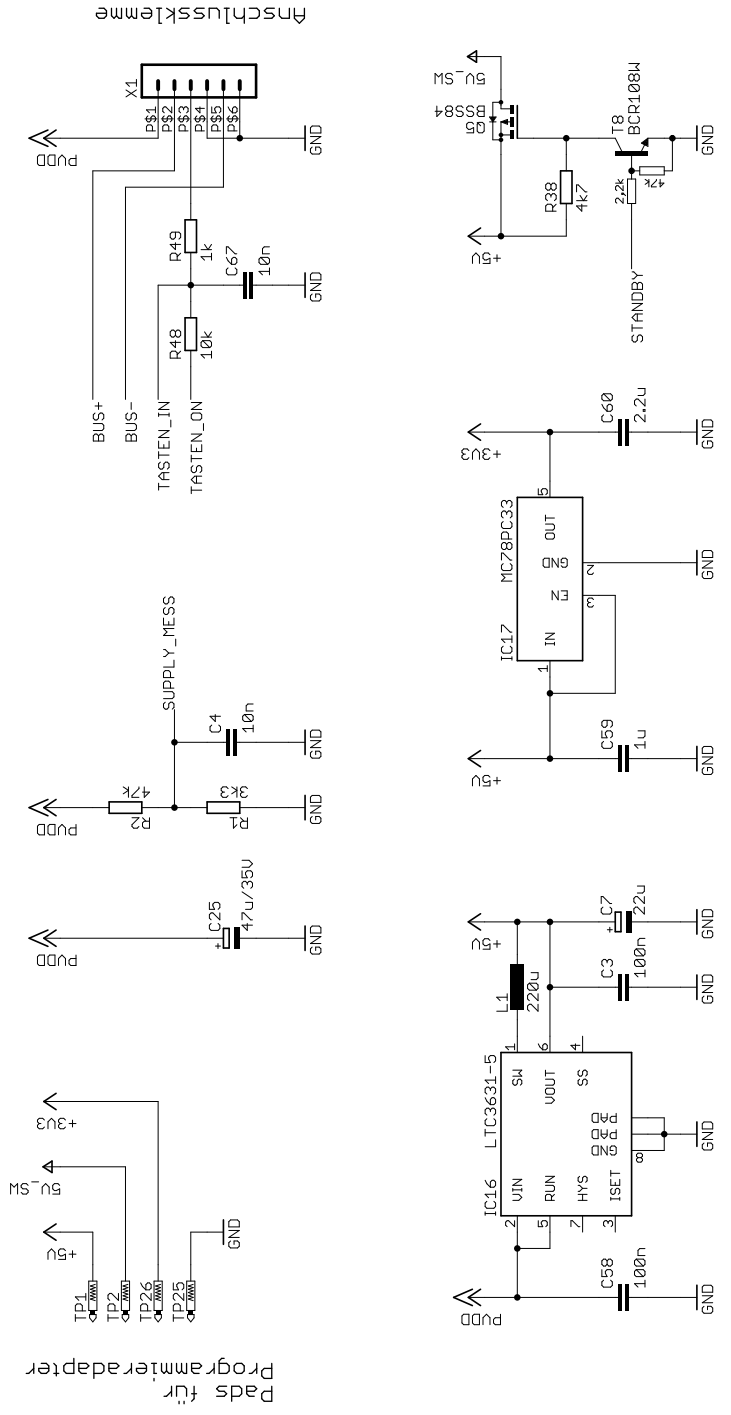

Abbildung C.2.: Schaltplan Stromversorgung (Prototyp)

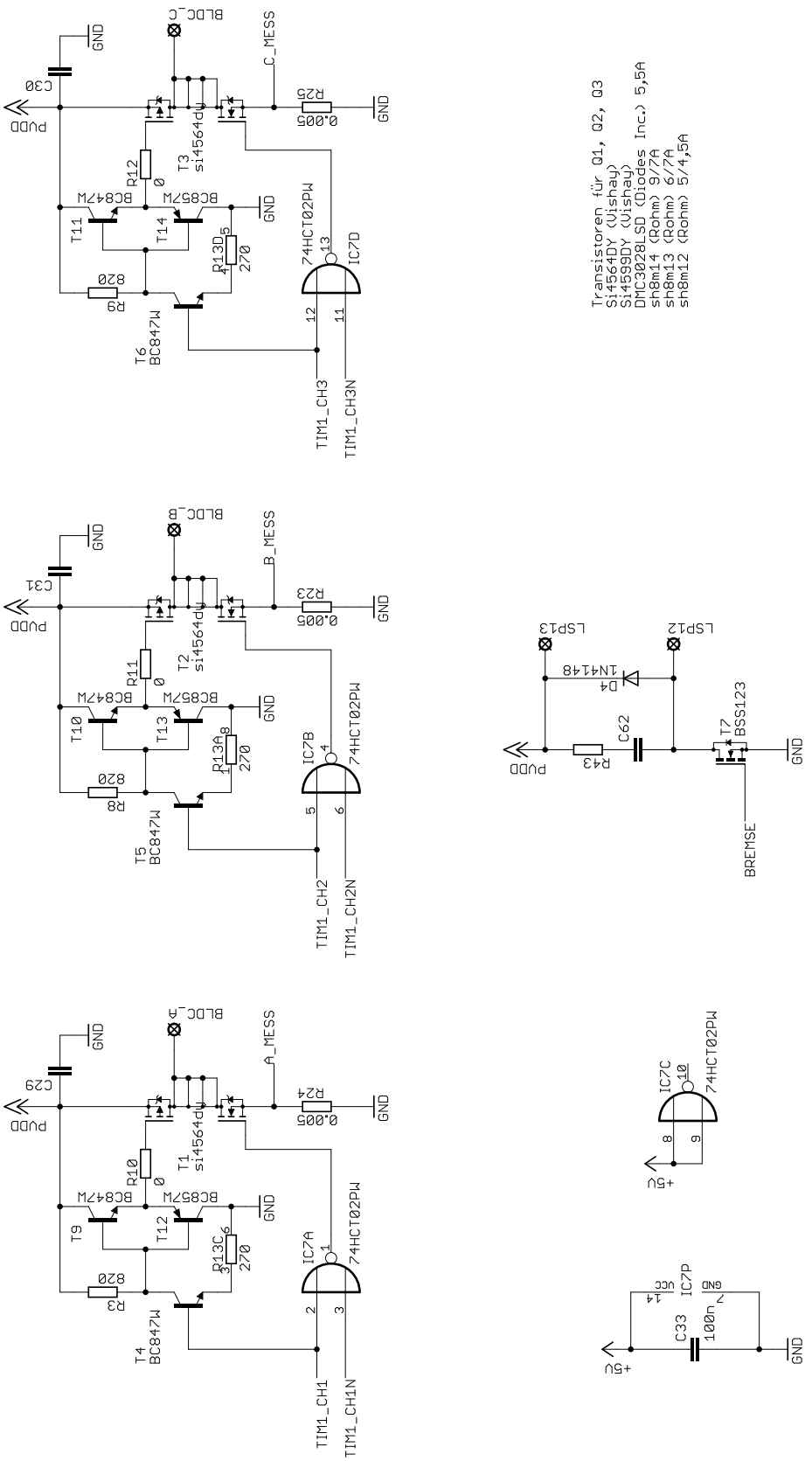

Abbildung C.3.: Schaltplan Treiberstufe für BLDC-Motor (Prototyp)

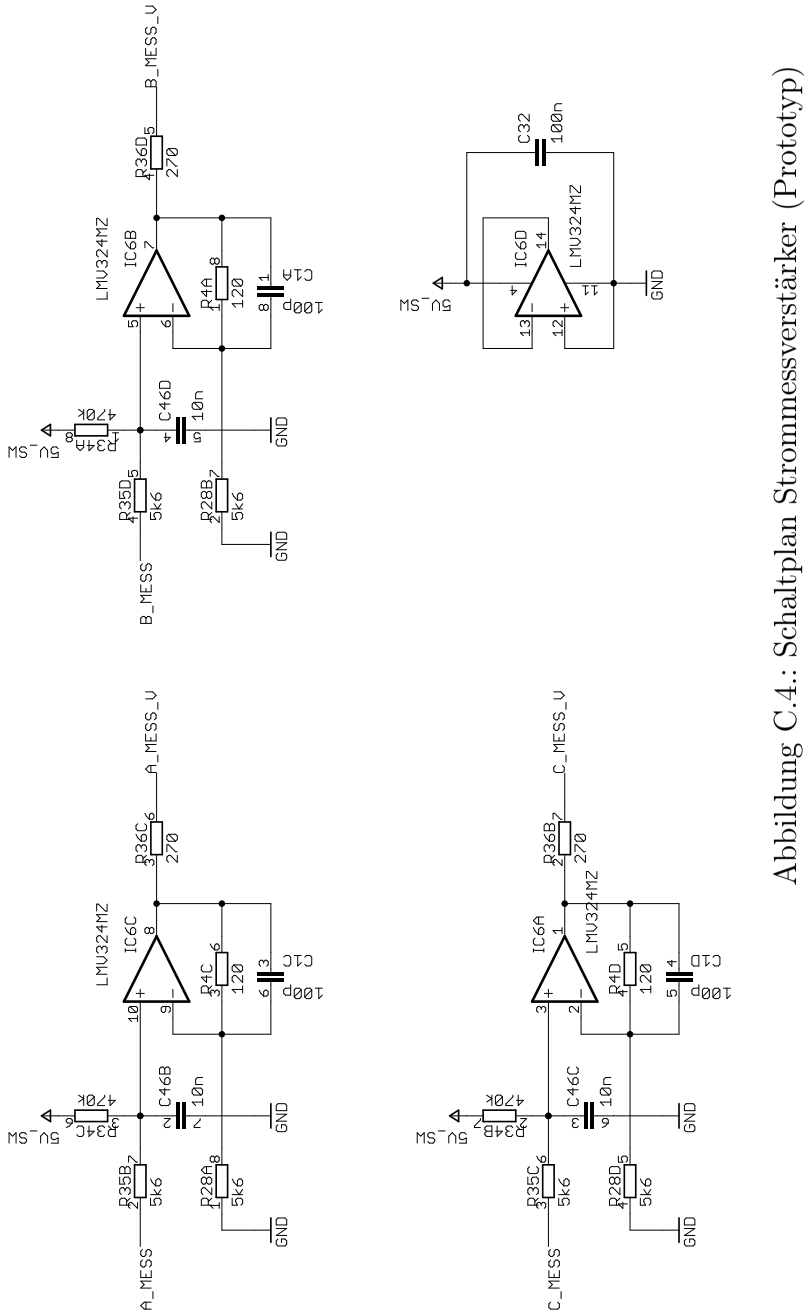

Abbildung C.4.: Schaltplan Strommessverstärker (Prototyp)

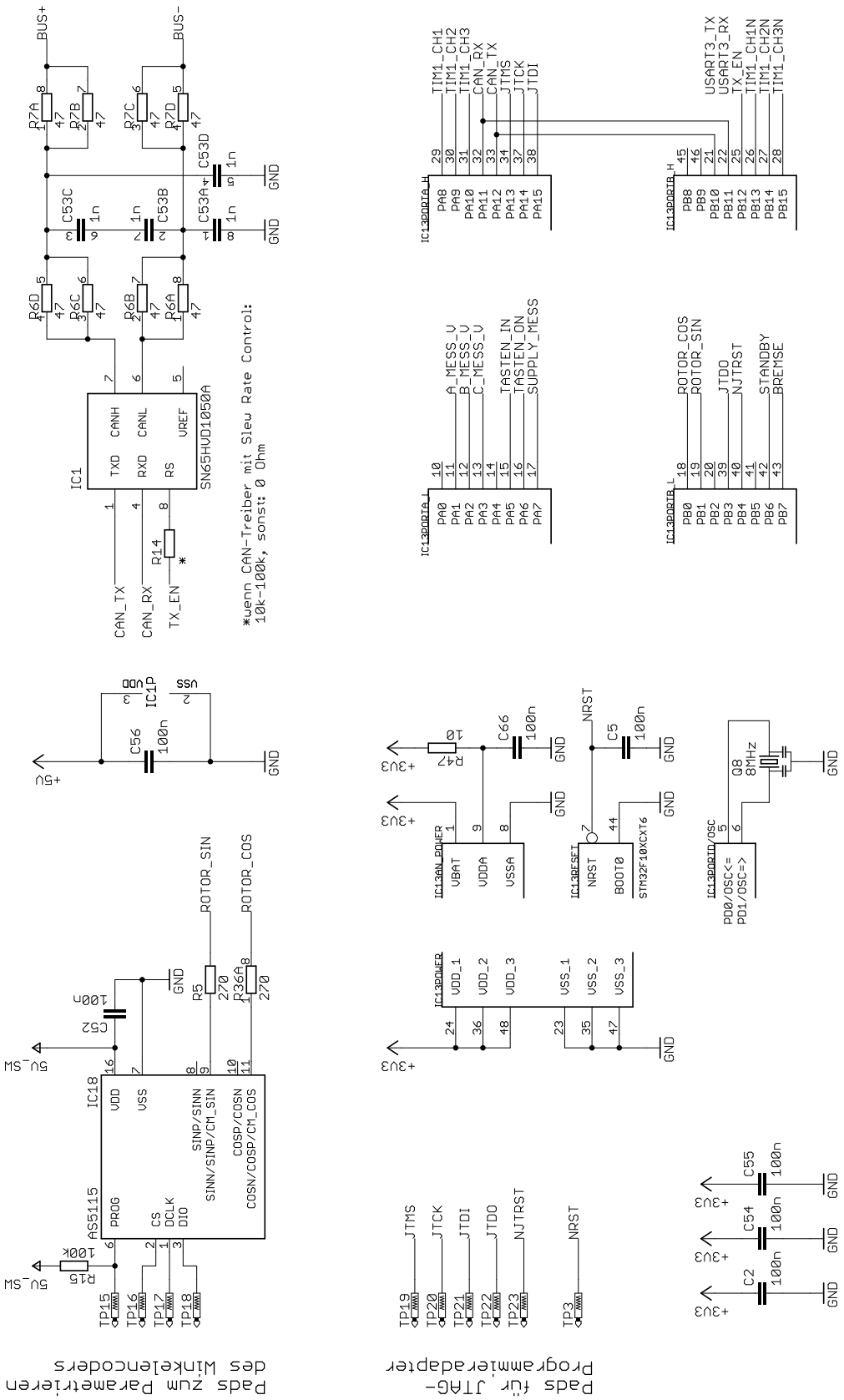

Abbildung C.5.: Schaltplan Mikrocontroller, Drehwinkelgeber, CAN-Treiber (Prototyp)

(a) Oben

(b) Unten

(c) Layer 2

(d) Layer 3

(e) Bestückung oben

(f) Bestückung unten

(g) Bauteilnamen oben

(h) Bauteilnamen unten

Abbildung C.6.: Lagen des Layouts (Prototyp)

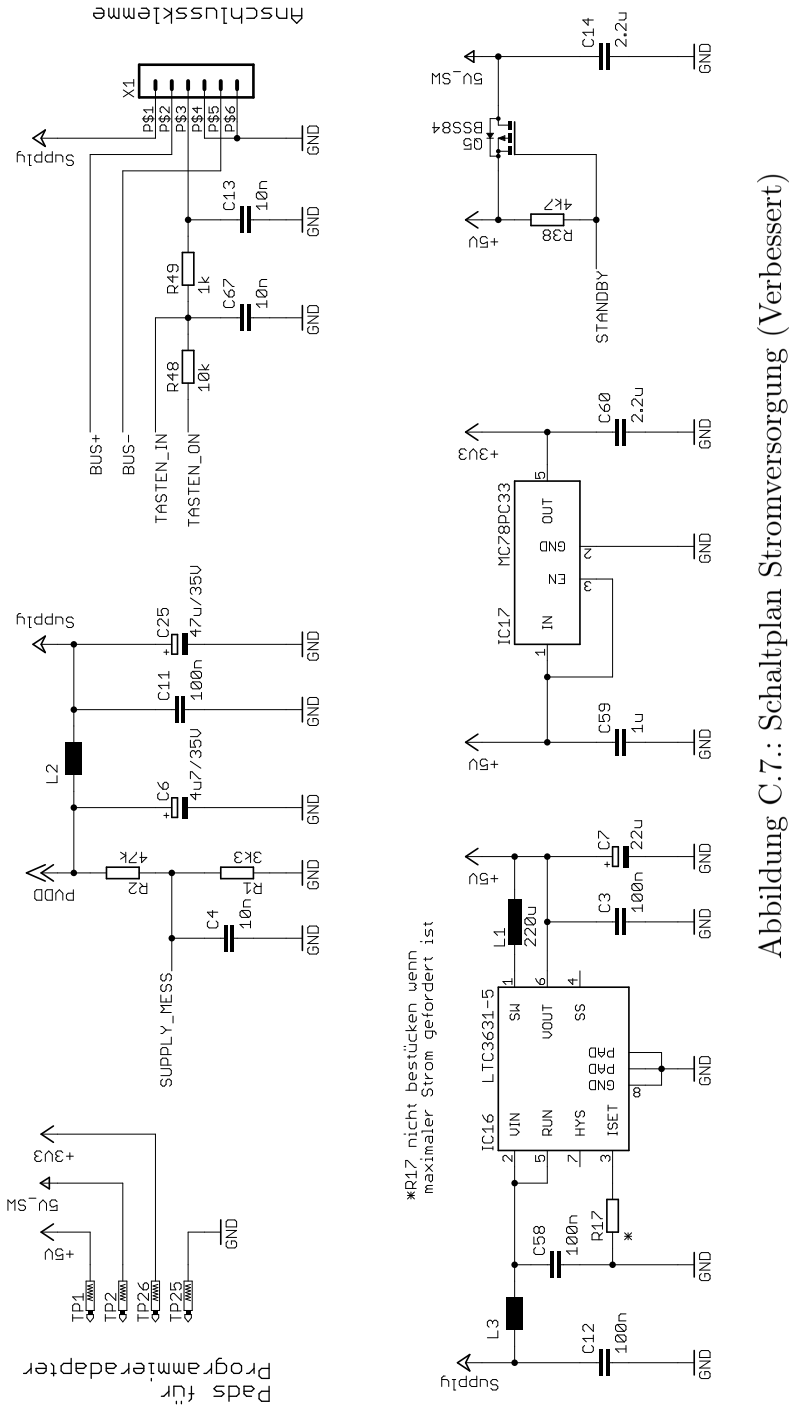

Abbildung C.7.: Schaltplan Stromversorgung (Verbessert)

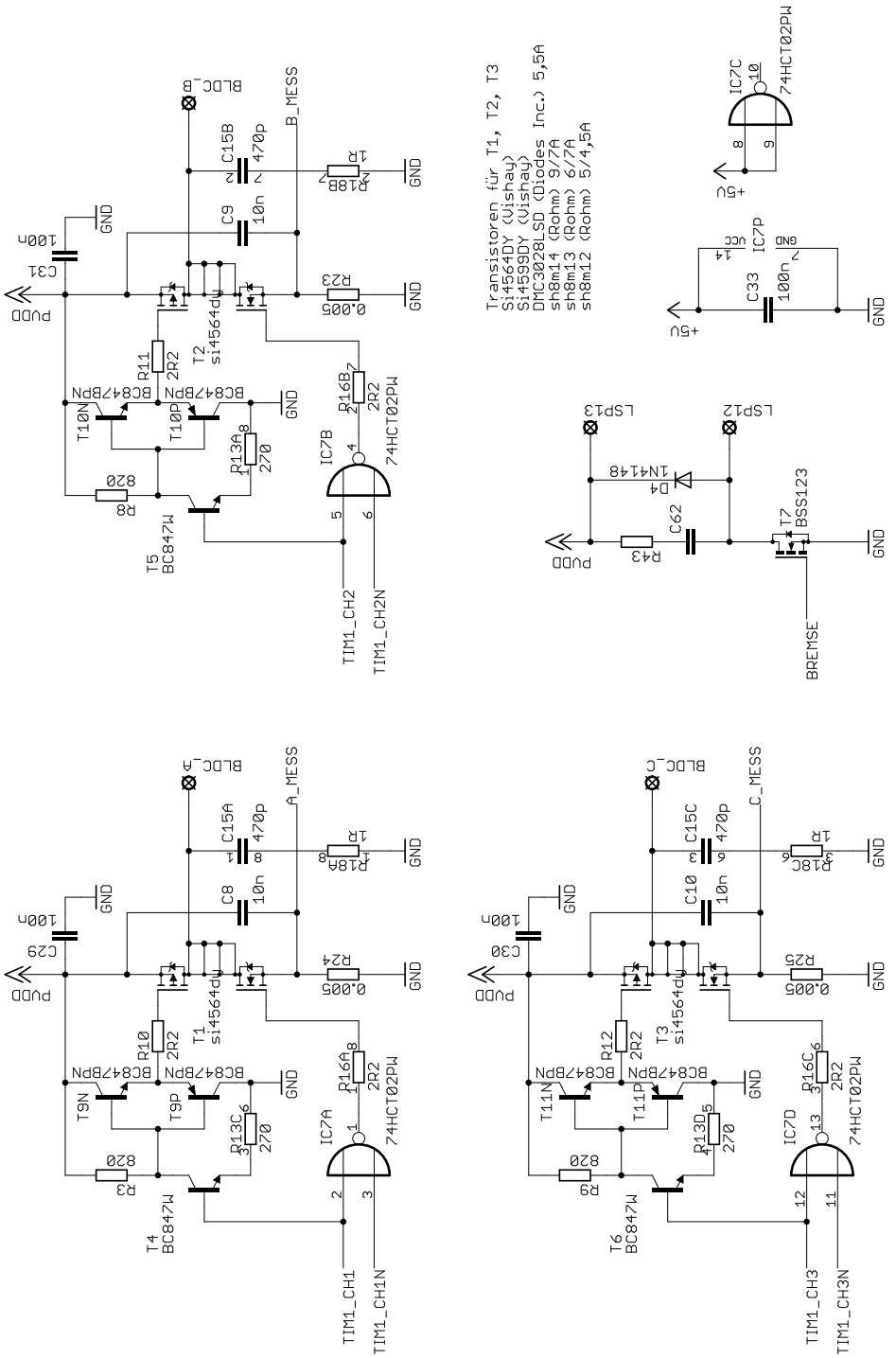

Abbildung C.8.: Schaltplan Treiberstufe für BLDC-Motor (Verbessert)

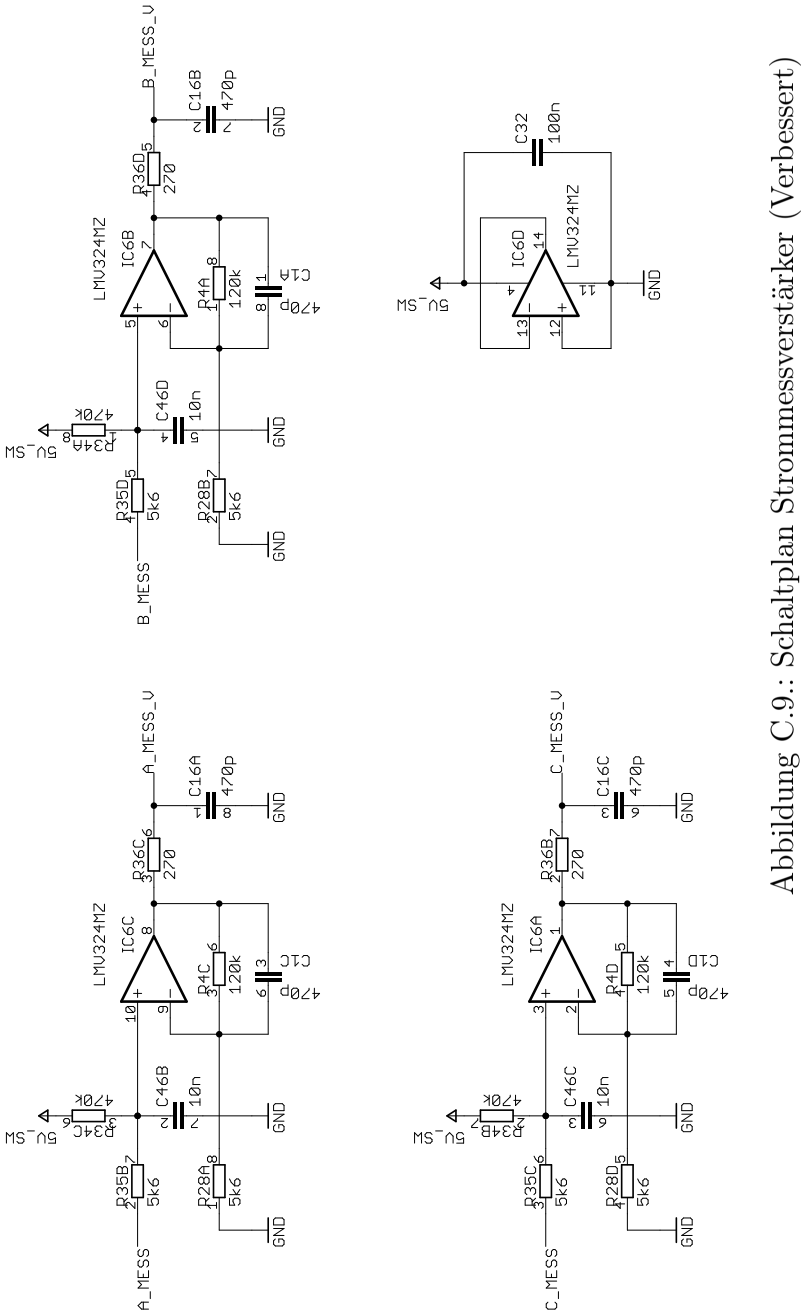

Abbildung C.9.: Schaltplan Strommessverstärker (Verbessert)

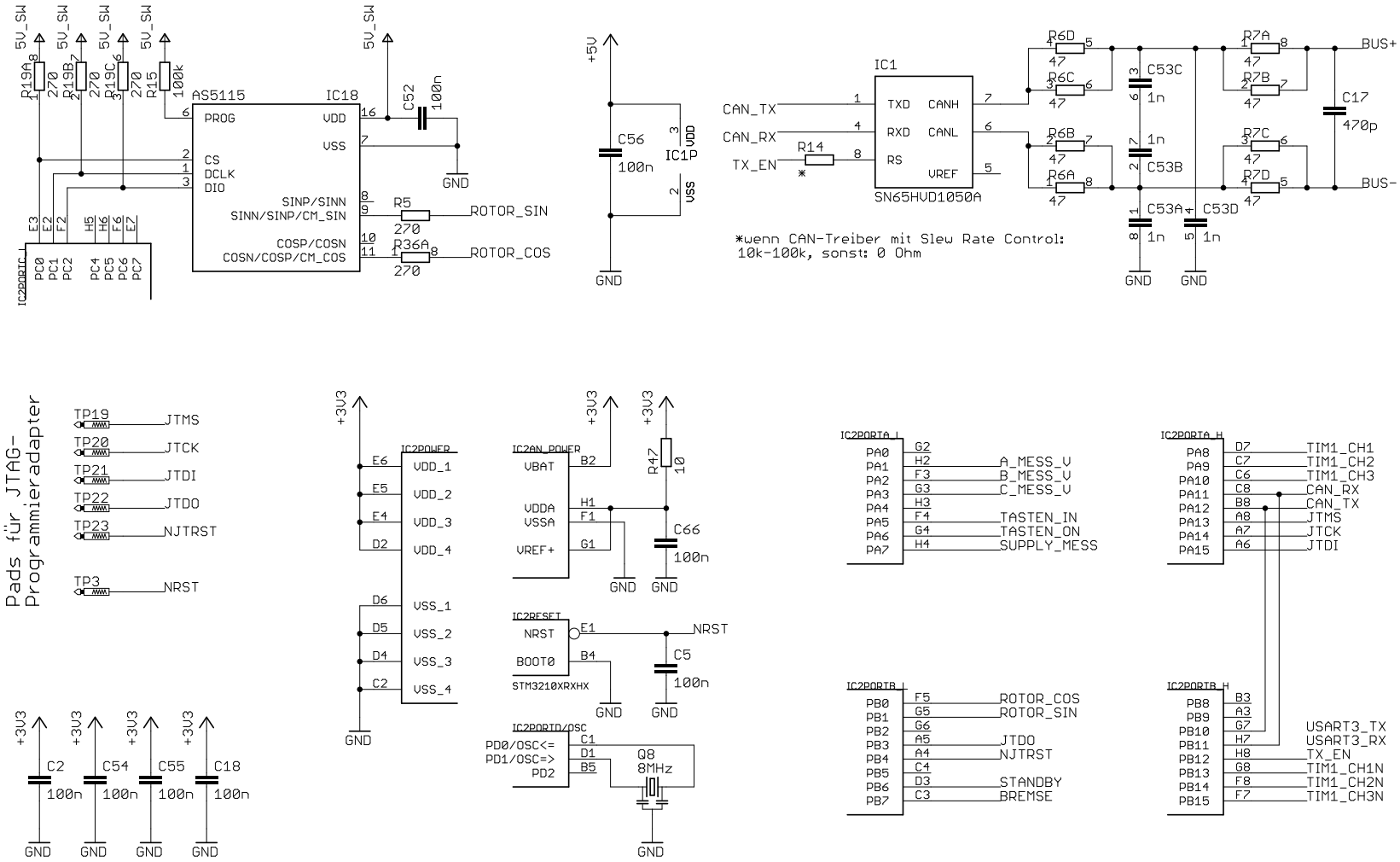

Abbildung C.10.: Schaltplan Mikrocontroller, Drehwinkelgeber, CAN-Treiber (Verbessert)

Abbildungsverzeichnis

Literaturverzeichnis

[1] ARM Limited, Cambridge, United Kingdom: *Cortex^{TM}-M3 Technical Reference Manual*, 2006. `http://infocenter.arm.com`, Doc ID: ARM DDI 0337E, Rev: r1p1.

[2] Balogh, L.: *Design And Application Guide For High Speed MOSFET Gate Drive Circuits*. `http://www.ti.com`, Doc ID: SLUP169.

[3] Hartl, H., E. Krasser, G. Winkler, W. Pribyl und P. Söser: *Elektronische Schaltungstechnik: mit Beispielen in PSpice*. Pearson Studium, 2008, ISBN 9783827373212.

[4] Horn, M. und N. Dourdoumas: *Regelungstechnik: Rechnergestützter Entwurf zeitkontinuierlicher und zeitdiskreter Regelkreise*. Pearson Studium, 2006, ISBN 9783827372604.

[5] Jack E. Volder: *The CORDIC Trigonometric Computing Technique*. Electronic Computers, IRE Transactions on, EC-8(3):330 –334, sept. 1959, ISSN 0367-9950.

[6] NXP, Eindhoven, Netherlands: *User Guide for the 74HC/HCT/HCU Family*, Nov 1997. `http://www.nxp.com`, File under Integrated Circuits, IC06.

[7] Schmaranz, K.: *Softwareentwicklung in C*. Xpert. press Series. Springer, 2001, ISBN 9783540419587.

[8] Söser, P.: *Integrierte Schaltungen*. Vorlesungsskript, 2009. Version: 1.1.4.

[9] STMicroelectronics, Geneva, Switzerland: *RM0008, STM32F1 Reference Manual advanced ARM-based 32-bit MCUs*, Oct. 2011. `http://www.st.com`, Doc ID: 13902, Rev: 14.

[10] STMicroelectronics, Geneva, Switzerland: *UM1052, User Manual PMSM single/-dual FOC SDK v3.0*, May 2011. `http://www.st.com`, Doc ID: 18458, Rev: 2.

[11] STMicroelectronics, Geneva, Switzerland: *UM1053, Advanced developers guide for STM32F103xx/STM32F100xx PMSM single/dual FOC library*, May 2011. `http://www.st.com`, Doc ID: 18459, Rev: 2.

[12] Tietze, U. und C. Schenk: *Halbleiter-Schaltungstechnik*. Springer, 2002, ISBN 9783540428497.

[13] Willemer, A.: *Einstieg in C++*. Galileo computing. Galileo Press, 2005, ISBN 9783898426497.

[14] *Ringing Reduction Techniques for NexFET^{TM} High Performance MOSFETs*. http://www.ti.com, Doc ID: SLPA010.

[15] *AS5115 Magnetic Angle Encoder*. Datasheet. http://www.austriamicrosystems.com.

[16] *LM124/LM224/LM324/LM2902 Low Power Quad Operational Amplifiers*. Datasheet. http://www.national.com.

[17] *LTC3631 High Efficiency, High Voltage 100mA Synchronous Step-Down Converter*. Datasheet. http://www.linear.com.

[18] *si4564dy Complementary Power MOSFET*. Datasheet. http://www.vishay.com.

[19] *BC857 PNP general purpose transistors*. Datasheet, Feb 2002. http://www.nxp.com.

[20] *MC78PC00-Series Low Noise 150 mA Low Drop Out Linear Voltage Regulator*. Datasheet, Jun 2005. http://www.onsemi.com.

[21] *74HC02; 74HCT02 Quad 2-input NOR gate*. Datasheet, Sep 2008. http://www.nxp.com.

[22] *BC847 NPN general-purpose transistors*. Datasheet, Dec 2008. http://www.nxp.com.

[23] *BSS84 P-channel enhancement mode vertical DMOS transistor*. Datasheet, Dec 2008. http://www.nxp.com.

[24] *CR0603/CR0805/CR1206 - Chip Resistors*. Datasheet, Feb 2008. http://www.bourns.com.

[25] *BC847BPN NPN/PNP general-purpose transistors*. Datasheet, Feb 2009. http://www.nxp.com.

[26] *Brushless DC Motor DB42L01*. Datasheet, Oct 2009. http://de.nanotec.com.

[27] *CAT/CAY 16 Series - Chip Resistor Arrays*. Datasheet, Aug 2010. http://www.bourns.com.

[28] *SN65HVD1050 EMC Optimized CAN Transceiver*. Datasheet, Mar 2010. http://ti.com.

[29] *STM32F103x8/STM32F103xB - Medium-density performance line ARM-based 32-bit MCU*. Datasheet, Apr. 2011. http://www.st.com, Doc ID 13587 Rev 13.