UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Realisierung eines Schrittmotortreibers mit einem Atmel Mikrocontroller

Bachelorarbeit

zur Erlangung des Grades eines Bachelor of Science (B.Sc.)
im Studiengang Informatik

vorgelegt von
Marcel Ziganow

Erstgutachter:	Prof. Dr. Hannes Frey (Institut für Informatik, AG Rechnernetze)
Zweitgutachter:	Dr. Merten Joost (Institut für Integrierte Naturwissenschaften, Abteilung Physik)

Koblenz, im Januar 2016

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

.....

(Ort, Datum) (Unterschrift)

Zusammenfassung

Die vorliegende Arbeit befasst sich mit der Realisierung eines Schrittmotortreibers auf einem 8-Bit Mikrocontroller des Unternehmens Atmel. Der Schwerpunkt liegt hierbei in der Entwicklung einer Stromregelung, welche neben den grundlegenden Betriebsmodi wie Voll- und Halbschritt auch den Mikroschritt ermöglicht. Hierfür wird mithilfe physikalischer und regelungstechnischer Grundlagen ein PI-Regler hergeleitet, welcher auf dem Mikrocontroller implementiert wird. In diesem Zusammenhang wird auf die erforderlichen Kenntnisse für die praktische Umsetzung eingegangen. Zusätzlich wird die Entwicklung der Hardware dokumentiert, welche für die benötigte Strommessung von großer Bedeutung ist.

Abstract

The present thesis deals with the realization of a stepper motor driver on an 8-bit microcontroller by the company Atmel. The focus is on the development of a current control, which allows microstepping in addition to the basic modes of operation like full- and halfstep. For this purpose, a PI controller is derived using physical and control engineering principles, which is implemented on the microcontroller. In this context, essential knowledge for the practical implementation will be discussed. In addition, the development of the hardware is documented, which is of great significance for the current measurement.

Inhaltsverzeichnis

1 Einleitung

Schrittmotoren finden sich heut zu Tage in vielen Geräten und Maschinen wieder, welche sowohl im Alltag als auch in der Industrie verwendet werden. So sind zum Beispiel Uhren, Drucker und hoch genaue CNC-Maschinen mit Schrittmotoren ausgestattet. Da diese Motoren nicht einfach an eine Gleichspannung angeschlossen werden können, besteht Bedarf nach Lösungen zur Ansteuerung solcher. Bereits existierende Lösungen sind entweder teuer oder durch einfache, meist analoge, Bausteine realisiert, welche die Möglichkeiten und Leistung der Schrittmotoren limitieren. Denn mittels einer geeigneten Ansteuerung lässt sich ein solcher Motor in vielen Bereichen verbessern. So kann durch eine intelligente Ansteuerung unter anderem eine höhere Drehzahl, ein höheres Drehmoment und eine genauere Positionierung erfolgen.

Im Rahmen dieser Bachelorarbeit wird eine Möglichkeit zur Realisierung einer intelligenten Ansteuerung von Schrittmotoren entwickelt. Diese soll neben einer einfachen Logik auch die bereits erwähnten Vorteile erbringen. Als Besonderheit wird, im Gegensatz zu den meisten erhältlichen Treiberkarten, diese Ansteuerung größtenteils in Software erfolgen.

Hierfür wird zunächst auf die mechanischen und elektrotechnischen Grundlagen von Schrittmotoren und einer einfachen Ansteuerung eingegangen, da diese sowohl für die theoretische als auch praktische Realisierung eines Schrittmotortreibers vorausgesetzt werden.

Da bei dieser Realisierung einer Ansteuerung die Regelung des Stromes eine elementare Rolle spielt, folgen anschließend mathematische und regelungstechnische Herleitungen, welche die Grundlage der praktischen Umsetzung darstellen. Die Vorteile einer solchen Regelung werden im Zusammenhang hierzu ausführlich diskutiert.

Nach der genauen theoretischen Ausarbeitung folgt die softwaretechnische Umsetzung auf einem Mikrocontroller. Die Herausforderung besteht darin, einen kostengünstigen Mikrocontroller zu verwendet, welcher Limitierungen hinsichtlich der Rechengeschwindigkeit besitzt. Somit werden Optimierungen der Algorithmen durchgeführt, um diese in gewisser Weise zu kompensieren. Durch die Realisierung der Ansteuerung mittels Software besteht die Möglichkeit, diese auf viele verschiedene Schrittmotoren anzupassen und sie somit optimal zu betreiben.

Im Anschluss wird die Hardware des Schrittmotorreglers entwickelt, welche im engen Zusammenhang mit der Software steht. Hierbei wird sowohl die Hardware zur Messung von Strömen thematisiert als auch die Herstellung einer geeigneten Platine.

Abschließend folgt eine Auswertung anhand von Messwerten und ein Ausblick auf mögliche Verbesserungen und Erweiterungen.

2 Grundlagen Schrittmotor

2.1 Mechanischer Aufbau

Schrittmotoren sind sogenannte Synchronmotoren, welche durch ein rotierendes magnetisches Feld der Statorspulen[1] schrittweise gedreht werden können. Der Winkel pro Schritt ist hierbei konstant, weshalb sich Schrittmotoren besonders für Anwendungsbereiche eigenen, bei denen es auf eine hohe Genauigkeit ankommt. So findet man diese Art von Motoren in Druckern, Disketten und CD-Laufwerken, als auch in computergesteuerten Werkzeugmaschinen. Eine besondere Art der Schrittmotoren findet man in dem Antrieb jeder Quarzuhr (Lavet-Schrittmotor).

Neben der Möglichkeit den Rotor exakt zu positionieren, bieten Schrittmotoren nach [CJ04] weitere besondere Eigenschaften:

1. *Bürstenlos* - Kein Verschleiß wie bei konventionellen Motoren mit Schleifkontakten
2. *Unabhängig von Last* - Schrittmotoren drehen sich mit gegebener Geschwindigkeit unabhängig von der Last, solange das motorspezifische Drehmoment nicht überschritten wird
3. *Ohne Regelung positionierbar* - Werden die Drehmoment Angaben nicht überschritten, so braucht man keine Sensorik, um die aktuelle Position des Rotors zu bestimmen
4. *Haltemoment* - Schrittmotoren können im Stand eine bestimmte Last halten ohne ihre Position zu verändern

Grundsätzlich werden Schrittmotoren in Reluktanz-, Permanentmagnet- und Hybridschrittmotoren eingeteilt. Der Rotor des Reluktanzschrittmotors besteht aus einem gezahnten Weicheisenkern[2], wodurch das Magnetfeld nicht aufrecht erhalten werden kann wenn kein Strom durch die Statorspulen fließt. Bei dieser Bauweise entsteht die Drehbewegung dadurch, dass der nächstliegende Zahn des Rotors von dem des Stators angezogen wird (siehe Abbildung 1). Die Wirkungsweise ist namengebend für diese Art des Schrittmotors, denn das System ist bestrebt einen minimalen magnetischen Widerstand (Reluktanz) zu erreichen. Der Reluktanzschrittmotor hat im abgeschalteten Zustand auf Grund des fehlenden magnetischen Feldes kein Rastmoment [Rum07].

[1] Stator bezeichnet den stationären und unbeweglichen Teil eines Motors, wohingegen der Rotor den beweglichen, rotierenden Teil eines Motors darstellt.

[2] Weicheisen besteht z.B. aus unlegiertem Eisen und stellt ein weichmagnetisches Material dar, d.h. es lässt sich leicht magnetisieren [Fas05].

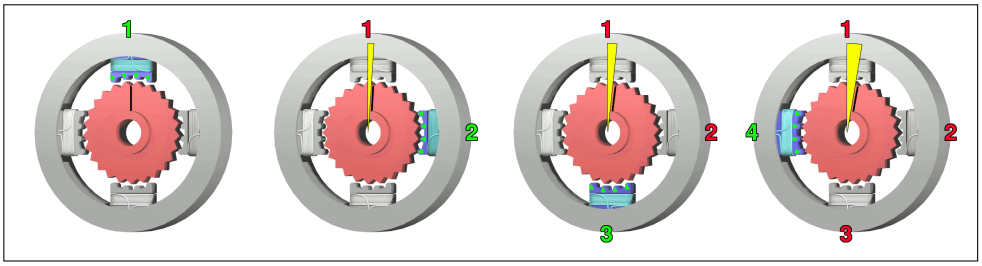


Abbildung 1: Rotationsdarstellung eines Reluktanzschrittmotors [Pie07]

Der Permenentmagnetschrittmotor hingegen besitzt einen Rotor mit Permanentmagneten, wodurch im Gegensatz zum Reluktanzschrittmotor ein Rastmoment existiert. So entsteht die Rotation durch das Ausrichten des Rotors mittels der Stator-Magnetfelder (siehe Abbildung 2). Ein Nachteil dieser Bauart ist die technisch begrenzte Anzahl der Pole und der damit einhergehenden niedrigen Auflösung.

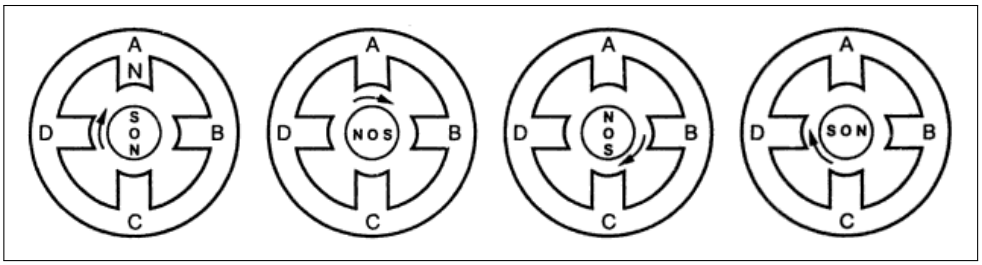


Abbildung 2: Rotation eines Permanentmagnetschrittmotors [BB09]

Die Kombination aus Reluktanz- und Permanentmagnetschrittmotor wird Hybridschrittmotor genannt. Der Rotor besteht hier aus einem Permanentmagnetkern und zwei gezahnten Weicheisenkränzen, welche um einen halben Schritt versetzt sind. Dabei stellt einer den magnetischen Nord- und der anderen den Südpol dar (siehe Abbildung 3). Der Stator ist genauso aufgebaut wie der des Reluktanzschrittmotors. So erreicht man eine hohe Auflösung, ein hohes Halte- und Drehmoment und ein Rastmoment, welches beim Reluktanzschrittmotor fehlt [BB09]. Die meisten heute erhältlichen Schrittmotoren sind von diesem Typ.

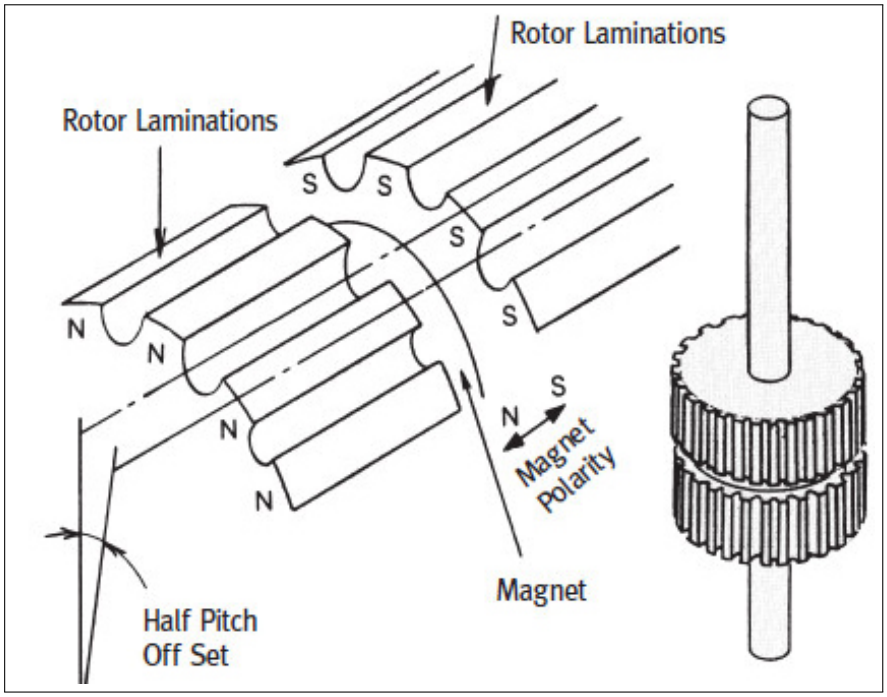


Abbildung 3: Rotor eines Hybridschrittmotors [BSD+]

2.2 Elektrischer Aufbau

Permanentmagnet- und Hybridschrittmotoren können in uni- oder bipolarer Ausführung aufgebaut sein und unterscheiden sich in der elektrischen Ansteuerung der Statorspulen.

Unipolar Ein Schrittmotor in unipolarer Ausführung besitzt an jeder Statorspule einen Mittelabgriff. Diese können entweder einzeln oder als zusammengefügte Kabel ausgeführt werden. Dementsprechend hat ein solcher Schrittmotor entweder sechs oder fünf Kabel. Angesteuert wird diese Art, unabhängig von der Anzahl der Kabel, durch ein Verbinden des Mittelabgriffs mit einer Spannungsquelle und abwechselndes Anlegen der Masse an eine der Spulenenden. Vorteilhaft hierbei ist der einfach zu realisierende Treiber mittels vier Transistoren (Low-Side Treiber). Nachteilig ist das niedrige Dreh- bzw. Haltemoment, welches durch die, jeweils nur zur Hälfte mit Strom durchflossenen, Spulen erzeugt wird. Der Vorteil des einfachen Treibers hat jedoch mittlerweile an Bedeutung verloren, weshalb der bipolare Aufbau weiter verbreitet ist [Ost11b].

Eine einfache Ansteuerung für unipolare Motoren sieht man in Abbildung 4. Der Strom durch die Spulen kann mithilfe der vier MOSFETs[3] geschaltet werden. Die Sperrdioden schützen die MOSFETs einerseits vor Spannungsspitzen als auch vor negativen Spannungen, die beim Abschalten einer Spule durch Selbstinduktion entstehen. Zu beachten ist, dass die Dioden eine möglichst kurze Sperrverzögerungszeit[4] besitzen.

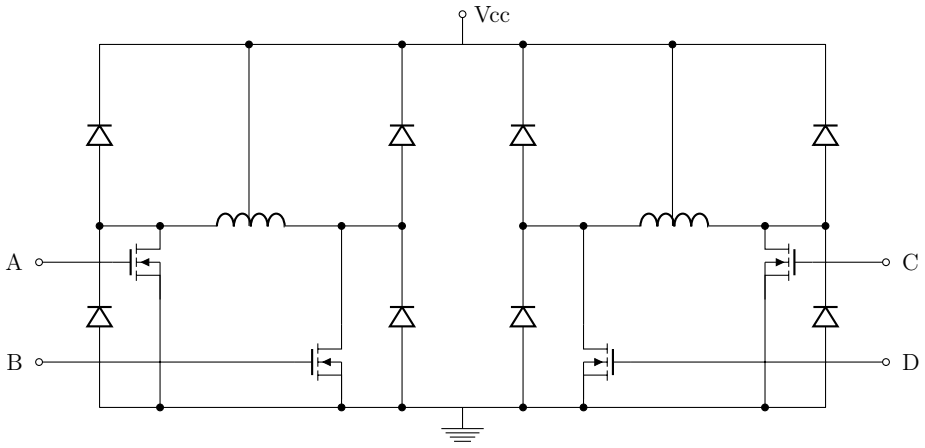


Abbildung 4: Schematischer Aufbau einer Ansteuerung für unipolare Motoren

Bipolar Im Gegensatz zum unipolaren Aufbau fließt der Strom von bipolaren Schrittmotoren durch die komplette Spule. Somit verringert sich auch die Anzahl der Anschlüsse auf vier Kabel. Da die Spulen nun umgepolt werden müssen, werden zwei Transistoren pro Spulenende benötigt. Dieser erhöhte Ansteuerungsaufwand stellt den Nachteil dar. Der Vorteil dieses Aufbaus ist das gesteigerte Dreh- und Haltemoment. So erreicht ein bipolarer Motor ungefähr 30% mehr Drehmoment als ein unipolarer [CJ04].

Die Ansteuerung (siehe Abbildung 5) unterscheidet sich von der des unipolaren Motors dahingehend, dass vier weitere MOSFETs benötigt werden um den Strom in den Spulen umpolen zu können. Es werden also 2 H-Brücken benötigt. Auch hier werden Sperrdioden antiparallel zur Spule eingesetzt.

[3]Metall-Oxid-Halbleiter-Feldeffekttransistoren sind wegen eines geringen Widerstands zwischen Drain und Source (R_{DSon} im mΩ-Bereich) geeignet hohe Ströme zu leiten [Sch07].

[4]Die Sperrverzögerungszeit t_{rr} (englisch *reverse recovery time*) ist die Zeit, die eine Diode braucht, um von Durchlass- in Sperrrichtung zu schalten [AS06].

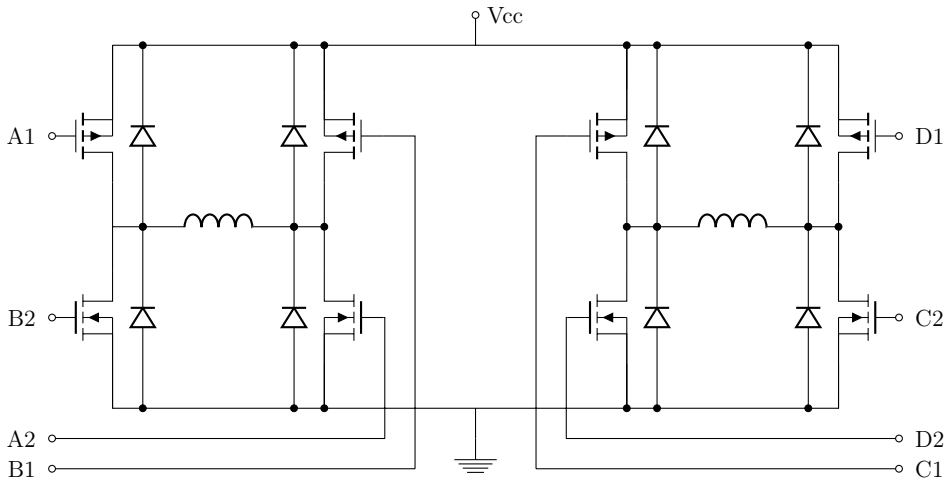


Abbildung 5: Ansteuerung eines Bipolaren Motors mittels H-Brücken

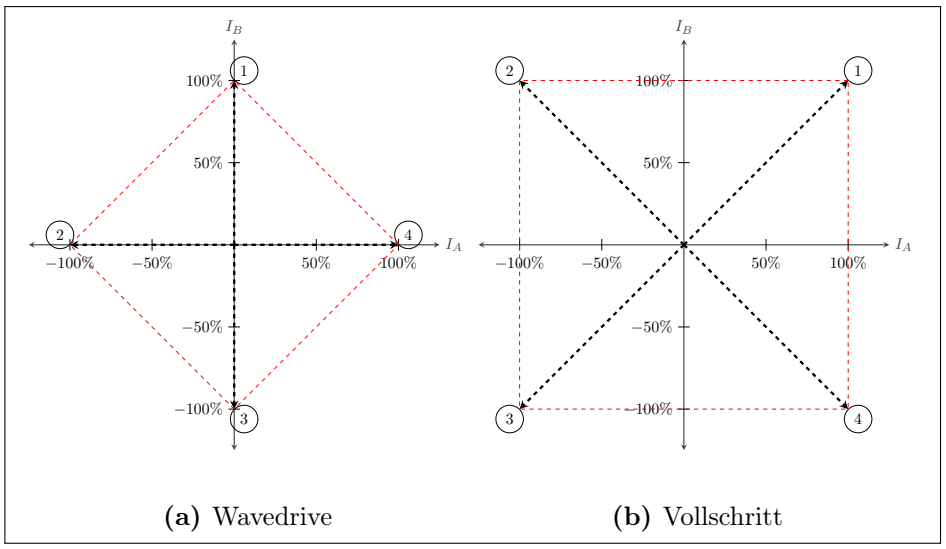
Da die Möglichkeit eines Kurzschlusses besteht, z.B. indem man A1 und B1 durchschaltet, wird bei einer MOSFET-H-Brücke häufig eine Ansteuerungslogik benötigt, welche sicherstellt, dass nicht beide Transistoren gleichzeitig eingeschaltet werden können. Zusätzlich muss verhindert werden, dass ein zeitlich zu nahe liegendes Umschalten der in Reihe liegenden Transistoren eintreten kann, denn MOSFETs besitzen eine Verzögerungszeit bis sie sperren.

2.3 Betriebsarten

Die im Datenblatt eines Schrittmotors zu findende Schrittanzahl bedeutet, wie viele Schritte für eine Umdrehung im Vollschrittbetrieb benötigt werden. Die meisten Schrittmotoren sind zweiphasig und besitzen daher eine Schrittanzahl, welche ein Vielfaches von 4 ist. Denn nach vier Schritten stehen sich die Zähne gleicher Polarität von Rotor und Stator wieder gegenüber. Durch elektronische Schrittteilung können aber auch feinere Auflösungen erreicht werden, wobei dann mehr als eine Phase gleichzeitig erregt wird [KEQ+94]. Durch gleichzeitige Regelung des Stromes, welcher durch die Spulen fließt, erreicht man den sogenannten Mikroschrittbetrieb. Nachfolgend werden die unterschiedlichen Betriebsarten kurz definiert.

Wavedrive Die einfachste Variante ist der Wavedrive. Hierbei wird immer nur eine Spule bestromt, weshalb das Drehmoment nicht den im Datenblatt angegebenen Wert erreicht [MW01]. Die Schrittanzahl ist dieselbe wie die des Vollschrittbetriebs. Aufgrund dessen wird diese Betriebsart eher selten genutzt.

Vollschritt Der Vollschrittbetrieb ist die standardmäßige Betriebsart. Es werden immer zwei Spulen gleichzeitig mit Strom durchflossen, weshalb das Drehmoment maximal ist. Des Weiteren wird die Genauigkeit im Vergleich zum Wavedrive-Betrieb erhöht, da die auf den Rotor wirkende Haltekraft, hier am höchsten ist. Die Unterschiede der Haltemomente zwischen Vollschritt und Wavedrive werden in Abbildung 6 verdeutlicht (die Länge der Zeiger ist proportional zum Haltemoment). Wegen der großen Änderung des magnetischen Flusses von 90° pro Schritt (Abbildung 6b) schwingt das System jedoch am Ende eines Vollschrittes stark aus. Die Frequenz des Ausschwingvorgangs ist die Eigenfrequenz des Schrittmotors. Sollten die Schritte pro Sekunde diese Frequenz erreichen, besteht die Gefahr des Auftretens von Resonanz und damit einhergehende Schrittverluste. Dieses Problem kann auch beim Vielfachen der Eigenfrequenz auftreten [Err95a].

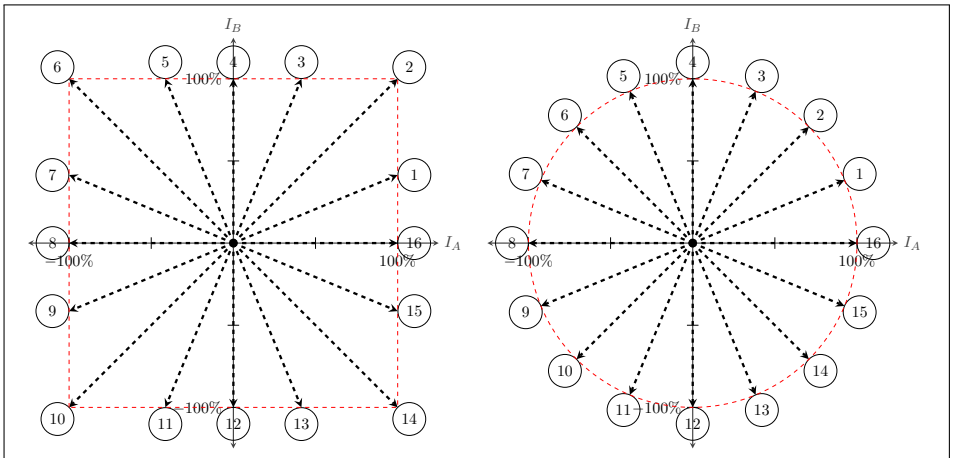


(a) Wavedrive **(b)** Vollschritt

Abbildung 6: Zeigermodell des Stromflusses durch Spule A und B

Halbschritt Der Halbschrittbetrieb ist eine Kombination aus Wavedrive- und Vollschrittbetrieb. Es werden abwechselnd jeweils eine oder zwei Spulen aktiviert. Durch die resultierende Halbierung des Schrittwinkel verdoppelt sich die Anzahl der Schritte pro Umdrehung (Überlagerung der Graphen in Abbildungen 6 a und b). Der Vorteil dieses Verfahrens besteht darin, dass die Ansteuerungselektronik die selbe sein kann wie die für den Vollschrittbetrieb. Aber auch die Gefahr der Resonanz wird verringert, da eine kleinere Stromänderung pro Schritt das schwingfähige System weniger angeregt. In den Phasen in denen nur eine Spule aktiv ist nimmt das Drehmoment jedoch ab ($\frac{1}{\sqrt{2}}$ des Maximalwertes [Hop12]).

Mikroschritt Durch eine Regelung des Spulenstromes ist es möglich noch weitere Schrittteilungen zu erzeugen. So können Positionen zwischen Voll- und Halbschritt genutzt werden. Hierfür gibt es zwei verschiedene Arten der Stromregelung. Wie in Abbildung 7a verdeutlicht, wird der Strom trapezförmig geregelt. Erreicht der Strom einer Spule den Maximalwert, so beginnt der Spulenstrom der anderen Phase entsprechend quantifiziert zu steigen bzw. zu sinken. Es wird somit immer das höchst mögliche Haltemoment im aktuellen Schrittwinkel erreicht. Abbildung 7b verdeutlicht hingegen eine Ansteuerung mit dem Bestreben möglichst immer das gleiche Haltemoment zu erzeugen. Dabei wird der Strom der einen Spule sinusförmig geregelt, der der anderen cosinusförmig. Das sich ändernde Drehmoment der trapezförmigen Ansteuerung ist insofern vom Nachteil, dass keine gleichmäßige, ruhige Bewegung entsteht. Daher wird ein sinusförmiger Stromverlauf, trotz des geringeren Drehmoments, meist bevorzugt. Denn der Mikroschritt soll neben der erhöhten Genauigkeit vor allem eine flüssigere Rotation ermöglichen. So löst es die Resonanz- und Vibrationsprobleme der anderen Betriebsarten durch Verringerung der Anregungsenergie und verhilft zu einem leiseren Betrieb. Es kann also auf teure und aufwendige mechanische Lösungen verzichtet werden. Ein Problem der digitalen Ansteuerung ist die Quantifizierung des Stromes und der damit einhergehenden Ungenauigkeiten. Denn die meisten Schrittmotoren haben eine Winkeltoleranz von bis zu $\pm 5\%$, was einem $\frac{1}{10}$-Mikroschritt entspricht [Err95a]. Ein Vollschritt wird also auf 10 Mikroschritte unterteilt. Eine Auflösung darüber hinaus dient einzig der Verminderung von Resonanz, Lautstärke und Vibrationen. Hierbei stellt ein $\frac{1}{32}$-Mikroschritt die sinnvolle Grenze dar, da nur noch 0.1% der Energie eines Vollschrittes an den Rotor übertragen wird. Diese wird durch Reibung innerhalb des Motors absorbiert.



(a) Trapezförmiger Stromverlauf **(b)** Sinus / Cosinus Stromverlauf

Abbildung 7: Zeigermodell des Stromflusses bei $\frac{1}{4}$-Mikroschrittbetrieb

2.4 Strombegrenzung

Durch das Anlegen einer höheren Spannung am Schrittmotor als die Nennspannung, steigt der Strom der Spule schneller an und ermöglicht sowohl höhere Schrittgeschwindigkeiten, als auch ein verbessertes Drehmoment. In diesem Kapitel wird auf die physikalischen Grundlagen und die Möglichkeiten der Strombegrenzung nach [CJ04] eingegangen.

2.4.1 Physikalische Grundlagen

Die Wicklungen eines Motors besitzen, wie eine *reale Spule*, neben einer Induktivität L auch einen Widerstand R_L. Dieser ist abhängig von Material, Länge und Durchmesser des Spulendrahtes. Dementsprechend gilt das ohmsche Gesetz (Formel 2.1).

$$U = R \cdot I \tag{2.1}$$

Induktivität, auch Selbstinduktivität genannt, beschreibt die Eigenschaft eines Leiters bzw. einer Spule eine Spannung durch ein sich änderndes Magnetfeld zu erzeugen. Fließt ein sich zeitlich ändernder Strom durch eine Spule, z.B. beim Einschaltvorgang, so erzeugt dieser ein sich änderndes Magnetfeld. Dadurch entsteht in der Spule wiederum eine Spannung, welche nach der *Lenz'schen Regel* der Ursache, also der Stromänderung, entgegenwirkt (Formel 2.2). Die sogenannte Selbstinduktion bremst den Anstieg des Stromes also ab. Entsprechend wirkt das sich ändernde Magnetfeld beim Abschalten des Stromes der Ursache, also der Abnahme des Magnetfeldes, entgegen und lässt den Strom langsam absinken. Die Induktivität ($[L] = 1$ Henry $= 1\,\mathrm{H}$), einer Spule ist unter anderem von der Anzahl der Windungen, dem Durchmesser und der Länge abhängig. [Dem13]

$$U_{ind}(t) = L \cdot \frac{dI}{dt} \tag{2.2}$$

Der Schaltplan in Abbildung 8 zeigt einen schematischen Aufbau einer realen Spule an einer Spannungsquelle. Die Induktivität und der Widerstand der Spule werden als eine Reihenschaltung dargestellt. Zum Zeitpunkt $t_0 = 0$ mit $I(t_0) = 0$ wird die Spannungsquelle eingeschaltet und bei $t = t_1$ mit $I(t_1) = I_{max}$ wieder abgeschaltet. Hierbei wird angenommen, dass der Stromkreislauf ohne zeitliche Unterbrechung kurzgeschlossen wird.

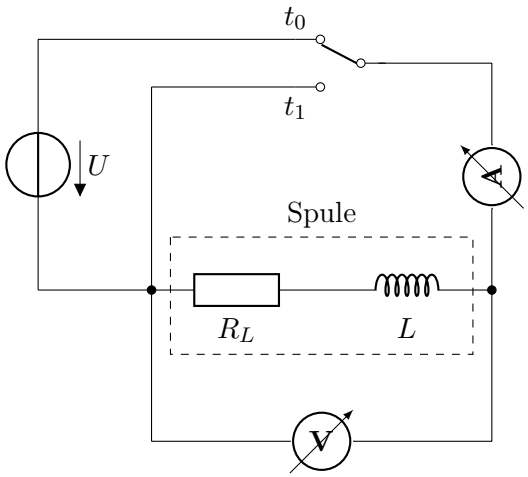


Abbildung 8: Schematischer Schaltplan einer realen Spule

Der Strom in der Wicklung steigt nach Formel 2.3 exponentiell an.

$$I(t) = I_{max} \cdot (1 - e^{-\frac{t}{\tau}}) = \frac{U}{R_L} \cdot (1 - e^{-t \cdot \frac{R_L}{L}}) \tag{2.3}$$

Hierbei ist $\tau = \frac{L}{R_L}$ die Zeitkonstante, welche die Dauer des exponentiell ansteigenden Systems angibt, bis dieses $1 - \frac{1}{e} \approx 63,2\%$ des Endwertes erreicht [Dem13]. Der maximal fließende Spulenstrom wird durch das ohmsche Gesetz in Formel 2.4 definiert.

$$I_{max} = \frac{U}{R_L} \tag{2.4}$$

Wie in Formel 2.4 zu erkennen ist, steigt der maximal zu erreichende Strom mit der Spannung und dem reziproken Widerstand. Aber auch die Steigung des Stromes zum Zeitpunkt $t = 0$ sollte beachtet werden. So sieht man in Formel 2.5 den Zusammenhang zur Spannung.

$$\begin{aligned} \frac{dI(t)}{dt} &= (\frac{U}{R_L} - \frac{U}{R_L} \cdot e^{-t \cdot \frac{R_L}{L}}) \frac{d}{dt} \\ &= \frac{U}{L} \cdot e^{-t \cdot \frac{R_L}{L}} \\ \frac{dI(t)}{dt}(t = 0) &= \frac{U}{L} \end{aligned} \tag{2.5}$$

Der Strom I_{max} fließt durch die Spule bis zum Zeitpunkt $t = t_1$, bei dem die Spannungsquelle abgeschaltet wird. Der Spulenstrom fällt ebenfalls exponentiell ab (Gleichung 2.6).

$$I(t \geq t_1) = I_{max} \cdot e^{-\frac{t}{\tau}} = \frac{U}{R_L} \cdot e^{-t \cdot \frac{R_L}{L}} \tag{2.6}$$

Es wird klar, dass die zeitliche Änderung des Spulenstromes in Abhängigkeit von der angelegten Spannung, dem Widerstand und der Induktivität der Spule steigt bzw. fällt. Da letztere jedoch durch den Motor vorgegebene Werte sind, lässt sich einzig die Spannung variieren. Eine höhere Spannung als die Nennspannung des Motors lässt den Strom zwar schneller steigen (Formel 2.5), jedoch wird dann der maximale Strom überschritten (Formel 2.4) und es droht eine Überhitzung und temporäre Entmagnetisierung des Rotors. Bei weiterer Steigerung der Temperatur kann der Motor endgültig zerstört werden. Man benötigt also eine Möglichkeit, die sowohl einen schnellen Anstieg als auch eine Limitierung des Stromes gewährleistet.

Abbildung 9 stellt die in diesem Kapitel erläuterten Formeln grafisch dar.

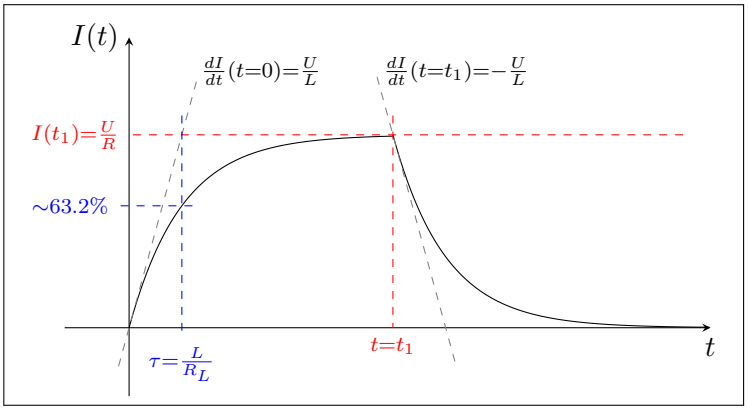


Abbildung 9: Stromverlauf einer realen Spule nach [Err95a]

2.4.2 Methoden der Strombegrenzung

Eine Möglichkeit die Steigung der Stromstärke zu erhöhen und I_{max} unverändert zu lassen, ist einen Widerstand in Reihe zu schalten. Zur Darstellung wird in Abbildung 10 vereinfacht der Spulenwiderstand durch einen Vorwiderstand verdoppelt. Deshalb kann die Spannung proportional erhöht werden: $I_{max} = \frac{2 \cdot U}{2 \cdot R_L} = \frac{U}{R}$ (dargestellt durch 2 Spannungsquellen mit gleicher Spannung).

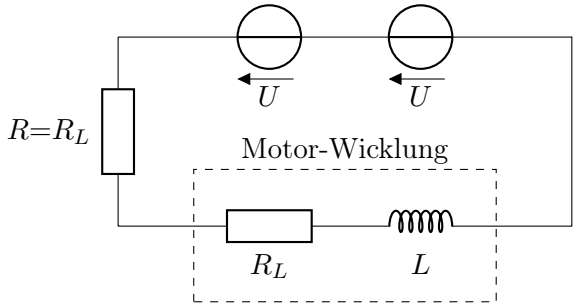


Abbildung 10: Strombegrenzung durch Vorwiderstand

Durch die in diesem Beispiel verdoppelte Spannung steigt nach Formel 2.5 der Spulenstrom wie folgt schneller: $\frac{dI}{dt}(t=0) = \frac{2U}{L}$. Abbildung 11 veranschaulicht den Stromverlauf und zeigt den Vorteil der verdoppelten Spannung. Es besteht jedoch der Nachteil der erhöhten Verlustleistung durch die Strombegrenzung mittels Vorwiderstand. So wird eine zusätzliche Leistung von bis zu $P = R \cdot I_{max}^2$ benötigt. Das System wird entsprechend ineffizienter und je nach Dimensionierung des Widerstands auch teurer.

Eine Möglichkeit der Effizienzsteigerung wäre das Verwenden von zwei unterschiedlichen Spannungsquellen. Eine mit höherer Spannung als die Nennspannung des Motors, die andere mit Nennspannung. So kann eine geeignete Steuerung die höhere Spannung bis zum Erreichen des Maximalstromes durchschalten, woraufhin die Spannungsquellen gewechselt werden. Diese Methode umgeht die Nachteile des Vorwiderstands, bietet aber weiterhin eine gesteigerte Schrittmotorleistung. Veranschaulicht wird diese Methode in Abbildung 11. Der Umschaltzeitpunkt t_1 wurde leicht verzögert, sodass eine kurze Überschreitung von I_{max} stattfindet. Im Vergleich zur Strombegrenzung mittels Vorwiderstand ist zu erkennen, dass die zuletzt beschriebene Methode den schnellsten Stromanstieg bietet und zugleich eine hohe Effizienz aufweist. Dennoch sind zwei Spannungsquellen unpraktisch und wiederum ein Kostenfaktor.

Um diese Nachteile zu umgehen und dennoch eine effektive Stromregelung zu erreichen, wird üblicherweise die sogenannte *Chopper-Steuerung* verwendet. Hierbei wird auf die zuvor erwähnte zweite Spannungsquelle verzichtet und dafür eine mit weit höherer Betriebsspannung als die Nennspannung verwendet. Dadurch steigt der Spulenstrom schnell auf maximalen Strangstrom I_{max}, woraufhin die Spannungsquelle abgeschaltet bzw. von der Spule getrennt wird. Sinkt der Strom unter einen bestimmten Wert, wird die Spannungsquelle wieder aktiviert.

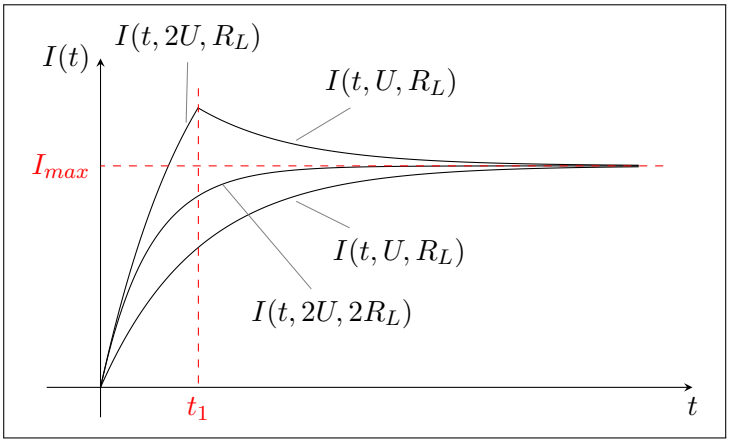


Abbildung 11: Stromverlauf im Vergleich

Eine dreieck- bzw. sägezahnartige Stromkurve entsteht, welche um die gewünschte Stromstärke pendelt. Wie in Abbildung 12 zu sehen ist, wird die Wicklung bis zum erreichen von I_{max} konstant mit Strom durchflossen, worauf bei ca. 2.5 A die Chopper-Steuerung einsetzt.

Es besteht die Möglichkeit eine solche Steuerung rein hardwarebasiert aufzubauen, ohne die Verwendung eines Mikrocontrollers. Hierbei wird die Stromstärke mithilfe eines Komparators mit einem Kontrollwert verglichen. Übersteigt die Stromstärke den gewünschten Wert, so unterbricht dieser die Verbindung zur Versorgungsspannung. Dieses Verfahren wird in Kapitel 3.1 genauer beschrieben.

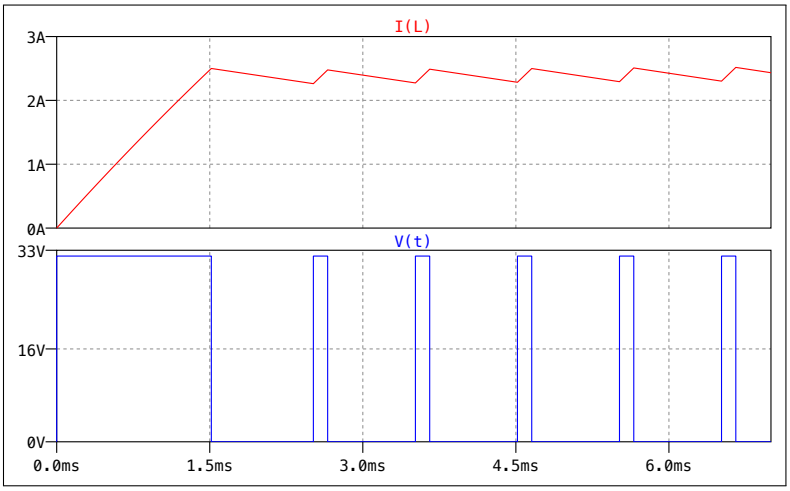


Abbildung 12: Simulation des Stromverlaufes bei Chopperbetrieb mit LTSpice

2.5 Decay Arten

Schaltet man eine Motorwicklung aus, so fällt der Strom nicht abrupt ab, sondern baut sich, wie in Kapitel 2.4.1 gezeigt, langsam ab. Bei der Verwendung einer H-Brücke ist es möglich, die Geschwindigkeit des Absinkens durch gezielte Ansteuerung der Transistoren zu beeinflussen. Im Falle eines Schrittmotors kann die gezielte Steuerung des Stromabbaus (englisch *decay*) sogar Vorteile erbringen. Unterschieden wird grundsätzlich zwischen *Fast Decay* und *Slow Decay*, welche jedoch auch in Kombination miteinander eingesetzt werden können.

Fast Decay Ein schnelles Absinken des Spulenstromes kann erzielt werden, wenn alle Transistoren der H-Brücke deaktiviert werden. Fließt der Strom wie in Abbildung 13a über die Transistoren *A1* und *A2* durch die Spule, bleibt die Stromrichtung in der Spule beim Abschalten erhalten (*Lenz'sche Regel*). Da die Transistoren nicht mehr leitend sind, bleibt nur der Weg durch Dioden parallel zu *B1* und *B2* (Abbildung 13b). Durch diese Freilaufdioden kann die in der Spule gespeicherte Energie wieder zurückgespeist werden. Zu beachten ist die Spannung U_R am Messwiderstand R_{Sense}, welche beim Decay-Vorgang negativ wird. Daher sollten Vorkehrungen getroffen werden um, die Hardware der Spannungsmessung zu schützen. Des Weiteren müssen möglichst schnelle Dioden eingesetzt werden (siehe Fußnote 4), andernfalls kann ein Transistor mit einer noch leitenden Diode einen Kurzschluss bilden. Beispielsweise könnte in Abbildung 13 der Transistor *A1* aktiv sein und wegen der kurzzeitig noch leitenden Diode parallel zum Transistor *B2* einen Kurzschluss erzeugen.

Vorteilhaft ist die Einfachheit der Strommessung. Denn sowohl beim Auf- als auch beim Abbau des Stromes kann dieser über einen Messwiderstand bestimmt werden. Das Zurückspeisen lässt jedoch einen verstärkten *Ripple*[5] in der Versorgungsspannung entstehen. Schrittmotoren können beim ausschließlichen Nutzen von *Fast Decay* unter Umständen nicht den benötigten Spulenstrom aufbauen, da dieser zu schnell absinken kann. [Mic01].

Slow Decay Im Gegensatz zum *Fast Decay* wird beim *Slow Decay* versucht, den Spulenstrom möglichst langsam abklingen zu lassen. Hierfür wird nach dem Stromanstieg nur ein bestimmter Transistor aktiviert, sodass der Strom durch die Spule, eine Diode und einem Transistor zirkulieren kann (Abbildung 14b). Er wird also kurzgeschlossen. Abhängig von der Stromrichtung wird entweder *A2* oder *B2* während des Decay-Vorgangs aktiviert. Alternativ kann der Decay-Vorgang auch im oberen Teil der H-Brücke stattfinden. Im Fall, dass leistungsfähige MOSFET-Treiber genutzt werden, besteht die Möglichkeit beide unteren MOSFETs zu aktivieren, um so ein noch

[5] Ripplestrom bezeichnet einen dem anliegenden Gleichstrom überlagerten Wechselstrom [Pry04]

langsameres Absinken zu ermöglichen. Durch den geringeren Ripple wird ein wesentlich ruhigerer Betrieb ermöglicht, aber auch die Hitzeentwicklung ist im Vergleich zum *Fast Decay* geringer.

Da der Strom während des Decay-Vorgangs nicht mehr durch den Messwiderstand fließt und so keine Spannung abfällt, kann kein Messvorgang durchgeführt werden. Bei hohen Drehzahlen wird die Spule des Motors häufig und schnell umgepolt. So kann bei diesem Vorgang der Strom unter Umständen nicht schnell genug abgebaut werden, was sich vor allem beim Mikroschritt durch eine verzerrte Sinuskurve bemerkbar macht [Mic01]. Dadurch werden nicht nur Ungenauigkeiten erzeugt, auch der Einfluss auf das Drehmoment ist negativ.

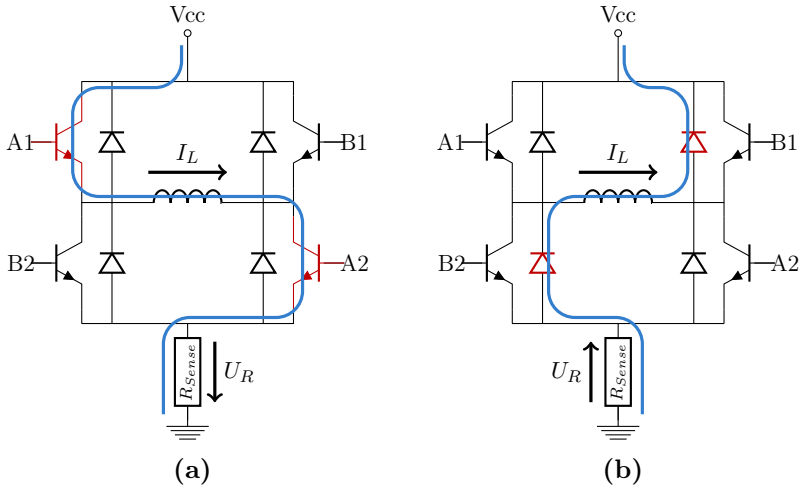


Abbildung 13: Stromlauf während des Chopperbetriebes mit Fast Decay

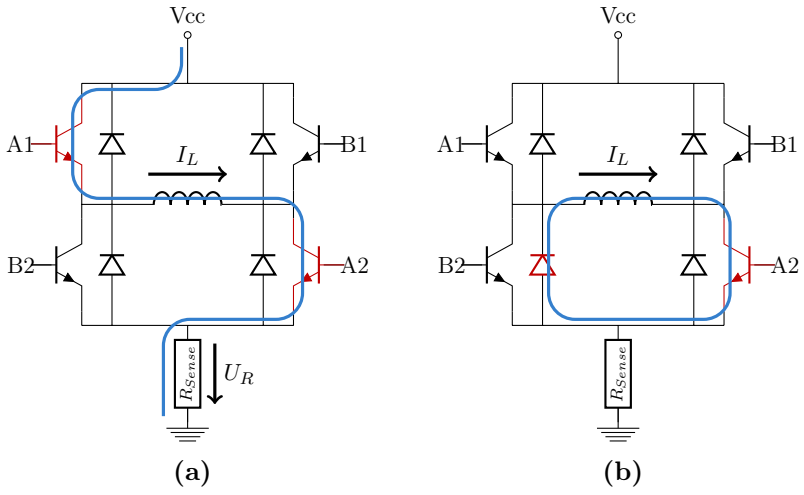


Abbildung 14: Stromlauf während des Chopperbetriebes mit Slow Decay

Um die Vorteile der beiden Decay-Arten zu vereinen, wurde der sogenannte *Mixed Decay* Vorgang entwickelt, welcher sowohl den *Fast Decay* als auch den *Slow Decay* verwendet. Hierbei wird beim Aufbau des Stromes der *Slow Decay* Vorgang verwendet, beim Sinus-Mikroschrittbetrieb also von 0 bis $\frac{1}{2}\pi$ und von π bis $\frac{3}{4}\pi$. Dies ermöglicht auch bei hohen Drehzahlen ein rasches Ansteigen des Spulenstromes und vermindert gleichzeitig den Ripple. In den übrigen Phasen setzt dann der *Mixed Decay* ein. Hierbei wird beim gewünschten Absinken des Stromes *Fast Decay* genutzt. Sobald der Sollwert erreicht wird, schaltet die Ansteuerung wieder auf den *Slow Decay* Vorgang um die Stromamplitude zu minimieren. Dieser Vorgang ermöglicht also auch bei hohen Drehzahlen stets den gewünschten Spulenstrom zu erzeugen.

3 Aufbau einer Schrittmotorsteuerung

Um eine optimale Schrittmotorsteuerung realisieren zu können, muss der Spulenstrom einstellbar sein. Das folgende Kapitel befasst sich mit dem theoretischen Aufbau eines Stromreglers. Hierbei wird die Problematik einer weit verbreiteten Schrittmotorsteuerung verdeutlicht, woraufhin ein optimiertes Regelsystem realisiert wird.

3.1 L297

Ein weit verbreitetes System zur Ansteuerung eines Schrittmotors ist der Baustein *L297* von STMicroelectronics in Verbindung mit dem Treiber *L298*. Der *L297* übernimmt dabei die Ansteuerung der Wicklungen eines Bi- oder Unipolaren Schrittmotors, je nach gewählter Konfiguration im Wavedrive, Vollschritt oder Halbschritt. So benötigt der Anwender nur ein *Clock-Signal*, welches dem *L297* bei jeder steigenden Flanke signalisiert, die entsprechenden Phasen, je nach Konfiguration, zu aktivieren. Auch die Richtung der Rotation kann durch eine einfache Pegeländerung an einem Eingang bestimmt werden. Darüber hinaus bietet dieser Baustein die Möglichkeit zur Stromlimitierung durch eine integrierte Chopper-Steuerung. Hierbei wird der Strom beider Spulen mittels eines Messwiderstands bestimmt, wobei eine hierzu proportionale Spannung abfällt. Diese wird mithilfe eines Komparators mit einer Referenzspannung verglichen. Übersteigt die gemessene Spannung den Referenzwert, so wird ein *RS-Flipflop* zurückgesetzt, dessen Ausgang der Ansteuerungslogik signalisiert, die aktivierten Phasen von der Stromversorgung zu unterbrechen. Sinkt der Spulenstrom unter den gewünschten Wert, so wird nach einem Taktzyklus der Flipflop wieder gesetzt und die Phasen wieder aktiviert [STM01].

Die Stromlimitierung wird ausschließlich dafür verwendet den Spulenstrom schnell ansteigen zu lassen ohne I_{max} zu übersteigen. Eine Verminderung des Stromes in Ruhephasen oder der Mikroschrittbetrieb ist hiermit nicht realisierbar. Außerdem muss die Referenzspannung bei einem Wechsel des Motors, welcher andere physikalische Eigenschaften besitzt, wieder angepasst werden. Dies führt zu einem, je nach Beschaffenheit der Platine, hohen Aufwand. Im weiteren Verlauf dieses Kapitels soll dieser Baustein durch einen Mikrocontroller mit digitaler Stromregelung ersetzt werden, welcher diese Nachteile kompensiert. Ein schematischer Aufbau des *L297* findet sich in Abbildung 15.

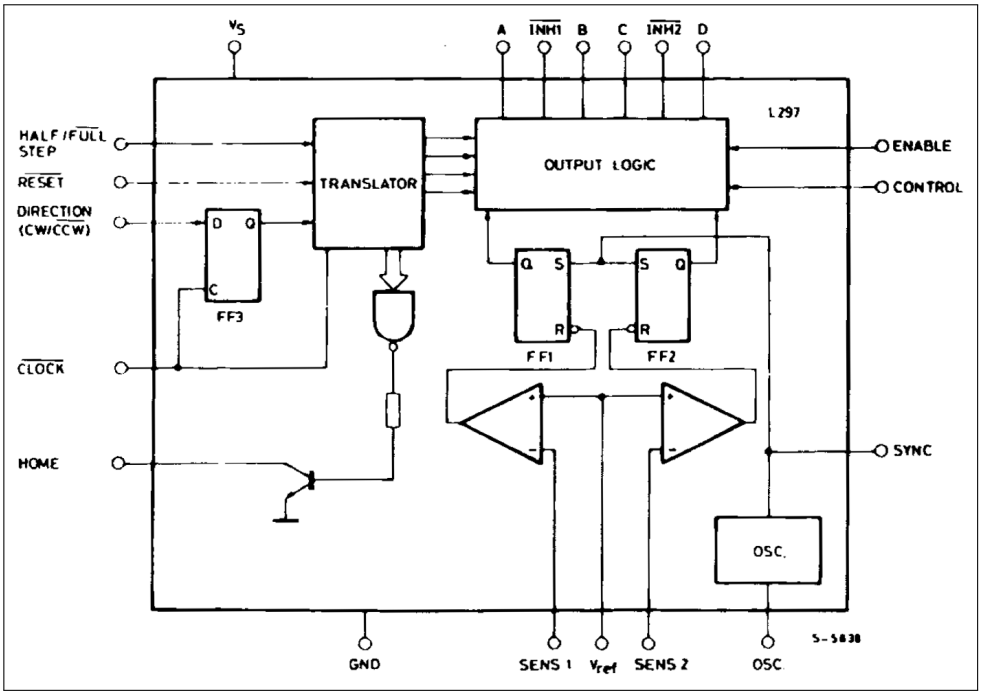


Abbildung 15: Blockschaltbild L297 [STM01]

3.2 L298

Zum Betreiben von Schrittmotoren benötigt der *L297* einen Treiberbaustein, welcher die benötigten Ströme schalten kann. Häufig wird hierzu der *L298* verwendet. Dieser besteht im Inneren aus zwei H-Brücken mit getrennten Masseanschlüssen. Dadurch besteht die Möglichkeit für jede Spule einen eigenen Messwiderstand zu verwenden, um so den Strom bestimmen und regulieren zu können. Neben der Fähigkeit einen Dauerstrom von bis zu 2 A zu führen, erlaubt der *L298* auch eine Versorgungsspannung von bis zu 50 V, was für die schnelle Steigung des Spulenstromes von Vorteil ist. Intern besitzt dieser Treiber eine Logik, welche einen Kurzschluss verhindert, sodass nie beide Transistoren auf einer Seite der H-Brücke leiten. Des Weiteren gibt es zwei *Enable*-Eingänge, welche die jeweilige H-Brücke komplett deaktivieren. Zu der Versorgungsspannung für die Ausgänge des Treibers wird eine weitere Spannung für die Logik benötigt, welche 7 V nicht überschreiten darf. Somit lässt sich der *L298* mit TTL-Spannung[6] betreiben.

Wichtig beim Betreiben einer induktiven Last ist die Verwendung von Sperrdioden. Schottky-Dioden werden auf Grund ihrer nicht vorhandenen Sperrverzögerungszeit empfohlen [STM00]. Die einfache Handhabung war ausschlaggebend den *L298* Treiberbaustein, trotz seines Alters, für die weitere Realisierung eines Schrittmotortreibers zu verwenden.

[6] Transistor-Transistor-Logik, Standard Spannung beträgt ca. 5 V.

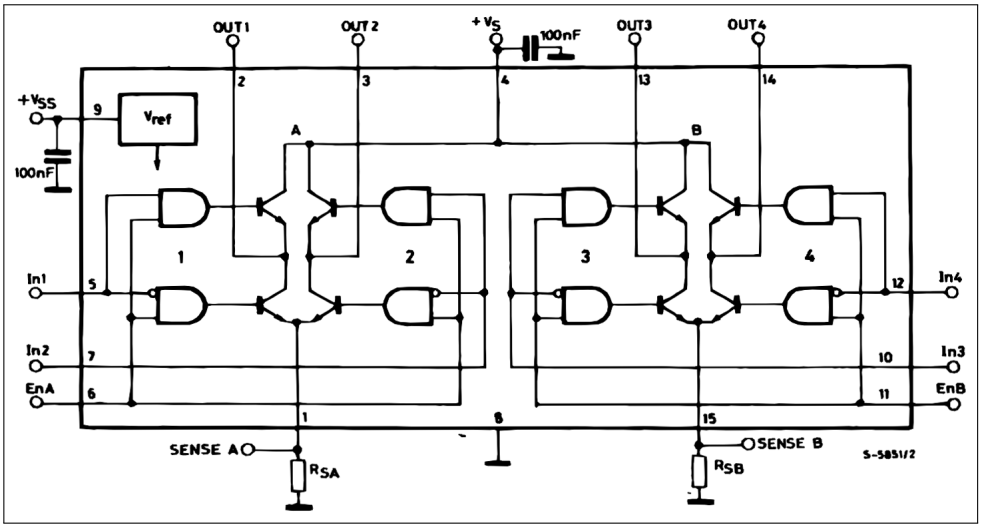


Abbildung 16: Blockschaltbild L298 [STM00]

3.3 Steuerung

Eine *Steuerung*, oder auch *Open Loop Controller*, kennzeichnet sich durch einen offenen Wirkungsweg. Hierbei wird die Ausgangsgröße durch eine Eingangsgröße gesteuert, wobei die Gesetzmäßigkeiten der Steuerstrecke (der Motor) diese Steuerung beeinflussen [Ste14]. Durch die fehlende Rückmeldung der Ausgangsgröße müssen Kenntnisse zur Steuerstrecke vorhanden sein, um sowohl den Sollwert zu erreichen als auch mögliche Störungen kompensieren zu können. Eine Möglichkeit zur Steuerung des Stromes eines Schrittmotors besteht darin, die benötigten Spannungen mithilfe von *Pulsweitenmodulation* (PWM) zu erzeugen. Die durch die PWM erzeugte mittlere Spannung lässt nach dem *ohmschen Gesetz* den gewünschten Strom fließen. Durch die Kenntnis des Stromaufbaus einer Spule (Kapitel 2.4.1) lassen sich die benötigten Spannungen für den Mikroschritt berechnen. Die Pulsweitenmodulation wird hierbei zur Realisierung einer Chopper-Steuerung (Kapitel 2.4.2) genutzt und ist eine Rechteckspannung mit konstanter Frequenz und veränderlicher Breite der Impulse. Beim Verwenden einer höheren Spannung muss der Tastgrad[7] entsprechend angepasst werden.

Durch die fehlende Rückkopplung kann die gewünschte Stromstärke jedoch nicht genau erreicht werden. So zeigen die Ergebnisse von [Man09] teilweise deutliche Abweichungen vom Referenzwert (Abbildung 17). Auch bei höheren Umdrehungszahlen entstehen Probleme, da durch die sich verkürzende Zeit zwischen den Schritten der Strom nicht ausreichend ansteigen kann. Je höher die Umdrehungszahl des Motors, desto kleiner wird die Stromamplitude (Abbildung 18). Um dem entgegenzuwirken, kann die Spannung

[7]Der Tastgrad ist das dimensionslose Verhältnis der Impulsdauer zur Periodendauer $Tastgrad = \frac{Impulsdauer}{Periodendauer}$ [Tas].

zwar erhöht werden, jedoch nicht unbegrenzt. So stellt dies einen wesentlichen Nachteil der Steuerung dar.

Diese Abweichungen rufen Ungenauigkeiten des Schrittwinkels hervor. Im Mikroschrittbetrieb ist es sogar möglich, dass deshalb Schritte übersprungen werden. Da die zu realisierende Schrittmotorsteuerung jedoch möglichst genau sein soll, wird eine Stromregelung realisiert, welche die Nachteile der Steuerung kompensieren kann.

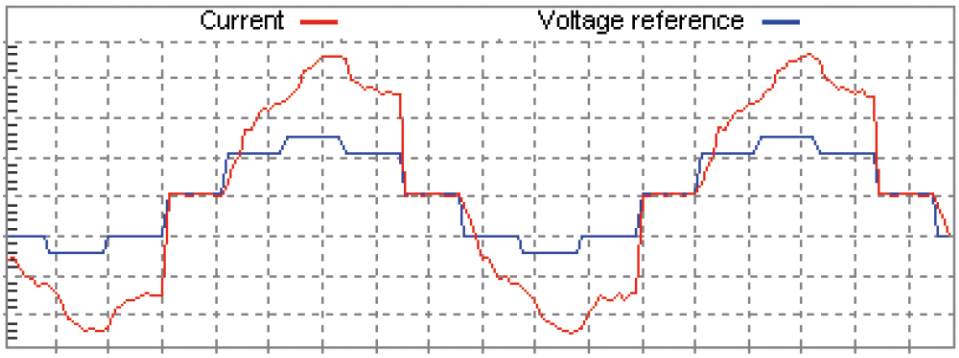


Abbildung 17: Spannungssteuerung im Halbschrittbetrieb [Man09]

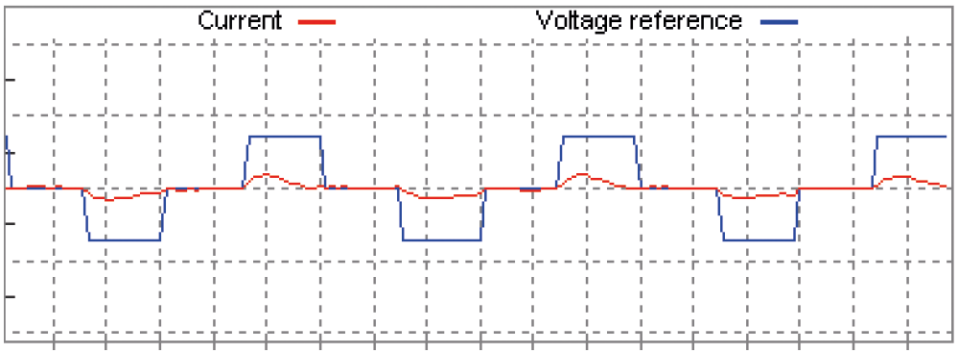


Abbildung 18: Spannungssteuerung im Vollschrittbetrieb bei $120\,\mathrm{min}^{-1}$ [Man09]

3.4 Stromgregelung

Eine Regelung unterscheidet sich von einer Steuerung durch die vorhandene Rückkopplung der Regelgröße, welche auf die Führungsgröße wirkt. Innerhalb eines Regelkreises vergleicht der Regler fortwährend die Führungsgröße $w(t)$ (Sollwert) mit der Regelgröße $y(t)$ (Istwert). Die Regelgröße wird hierbei mittels geeigneter Sensoren gemessen und zurückgeführt. Mit der Differenz dieser Werte, der Regelabweichung $e(t)$, wird die Stellgröße $u(t)$ berechnet, welche auf die Regelstrecke wirkt. So soll die Abweichung der physikalischen Größe der Regelstrecke minimiert werden. Im Gegensatz zur

Steuerung können durch die Rückführung auch eventuell vorhandene Störgrößen $d_s(t)$ kompensiert werden. Diese können beispielsweise temperaturabhängige Änderungen des Widerstands sein.

Bei einer Regelung des Spulenstromes eines Schrittmotors entspricht die Spule der Regelstrecke. Die Stellgröße $u(t)$ soll den Tastgrad der Pulsweitenmodulation angeben, um so die entsprechende Durchschnittsspannung an der Spule anlegen zu können. Der daraus resultierende Spulenstrom, also die Regelgröße $y(t)$, wird laufend gemessen und mit dem Sollwert $w(t)$ verglichen. Veranschaulicht wird dies in Abbildung 19. Der Regler wird in Software realisiert und auf einem *Atmel AVR* Mikrocontroller laufen. Da ein Schrittmotor jedoch zwei Phasen besitzt, werden dementsprechend zwei unabhängige Regler benötigt.

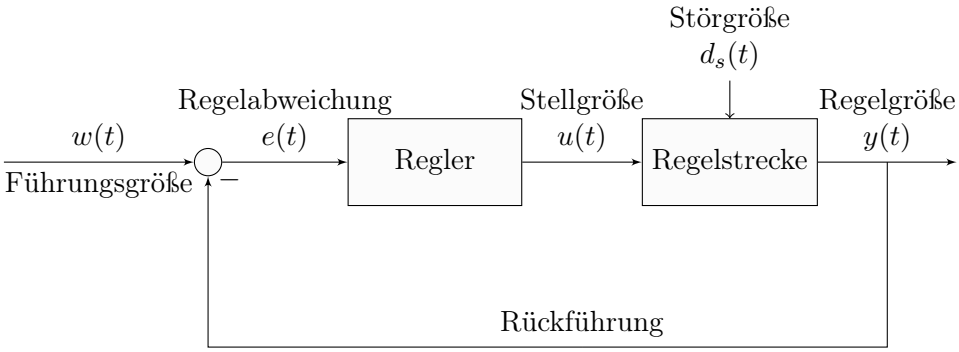


Abbildung 19: Blockschaltbild eines Regelkreises

Zunächst wird ein theoretisches Modell der Regelstrecke hergeleitet, woraufhin ein passender *analoger Regler* realisiert wird. Dieser Ansatz verspricht sowohl eine optimale Regelung als auch eine, im Gegensatz zur numerischen Lösung[8], einfache Anpassung an andere Motoren. Um den Regler auf einem Mikrocontroller laufen zu lassen, wird dieser daraufhin mittels geeigneter Verfahren diskretisiert, um so einen *digitalen Regler* zu erhalten.

3.4.1 Theoretisches Modell der Regelstrecke

Wie in Kapitel 2.4.1 bereits beschrieben, besitzt eine reale Spule neben ihrer Induktivität auch einen ohmschen Widerstand. Diese sind in Reihe geschaltet, weshalb die Summe ihrer Teilspannungen die Gesamtspannung ergibt. Da an die Spule jedoch keine zeitlich veränderbare Spannung angelegt werden kann, wird der Tastgrad $d(t)$ der Pulsweitenmodulation als dimensionsloser Faktor mit der Spannung multipliziert. Daraus ergibt sich eine Differentialgleichung.

[8]Beispielsweise durch die automatische PID-Optimierung der *Control System Toolbox* von MATLAB [TM].

$$U \cdot d(t) = R_L i(t) + L\frac{di(t)}{dt} \tag{3.1}$$

Der Tastgrad multipliziert mit der anliegenden Spannung ergibt die durch Pulsweitenmodulation erzeugte Durchschnittsspannung. Da bei Wechselspannungen und -strömen keine konstanten Werte vorliegen, spricht man bei $i(t)$ von einem Momentanwert. Dieser wird generell mit Kleinbuchstaben gekennzeichnet. Die entstandene Gleichung beschreibt das Übertragungsverhalten der Regelstrecke, also „den Zusammenhang zwischen den Ein- und Ausgangssignalen“ [Sch10]. Durch Überführung der Differentialgleichung vom Zeitbereich in den Frequenzbereich (Bildbereich) werden die weiteren Berechnungen ermöglicht. Dies geschieht mithilfe der *Laplace-Transformation*, welches ein mathematisches Verfahren zur Analyse von zeitinvarianten[9] dynamischen Systemen und kontinuierlichen Signalen ist. So ordnet die Laplace-Transformation einer „Funktion $f(t)$ mit $f(t) = 0$ für $t < 0$ wie folgt eine Funktion $F(s)$ der (komplexen) Variablen s“ zu [Pap09].

$$F(s) = \int_0^\infty f(t) \cdot e^{-st} dt \tag{3.2}$$

Hierbei ist $f(t)$ die zeitabhängige „*Original*- oder Oberfunktion“ und $F(s)$ die „*Bild*- oder Unterfunktion“ [Pap09]. Des Weiteren schreibt man $F(s) = \mathcal{L}\{f(t)\}$ als Laplace-Transformierte von $f(t)$. Durch Verwendung einer Tabelle der wichtigsten Laplace-Transformationen kann die Differentialgleichung der Regelstrecke nun überführt werden [Beu11].

$$\begin{aligned}
\mathcal{L}\{U \cdot d(t)\} = U \cdot \mathcal{L}\{d(t)\} &= \mathcal{L}\left\{R_L i(t) + L\frac{di(t)}{dt}\right\} \qquad (3.3)\\
U \cdot \mathcal{L}\{d(t)\} &= R_L \cdot \mathcal{L}\{i(t)\} + L \cdot \mathcal{L}\left\{\frac{di(t)}{dt}\right\}\\
U \cdot D(s) &= R_L I(s) + L \cdot \mathcal{L}\left\{\frac{di(t)}{dt}\right\}\\
U \cdot D(s) &= R_L I(s) + L \cdot (s \cdot I(s) - i(0))\\
U \cdot D(s) &= R_L I(s) + L \cdot s \cdot I(s)\\
U \cdot D(s) &= I(s) \cdot (R_L + L \cdot s) \qquad (3.4)
\end{aligned}$$

Die ersten beiden Schritt nutzen den *Linearitätssatz* der Laplace-Transformation aus [Pap09]. So wird $\mathcal{L}\{c_1 \cdot f_1(t) + c_2 \cdot f_2(t)\}$ zu $c_1 \cdot \mathcal{L}\{f_1(t)\} + c_2 \cdot \mathcal{L}\{f_2(t)\}$, wobei c_n Konstanten sind. Folgend werden $\mathcal{L}\{d(t)\}$ und $\mathcal{L}\{i(t)\}$ per Definition ersetzt mit $D(s)$ und $I(s)$. Der vierte Schritt überführt $\mathcal{L}\left\{\frac{di(t)}{d(t)}\right\}$ zu $sI(s) - i(0)$ mithilfe des Ableitungssatzes [Pap09]. Die Stromstärke $i(0)$

[9] Ein System ist zeitinvariant, wenn die Antwort auf eine Eingangsgröße zu jeder Zeit die selbe ist. Man kann also die Eingangsgröße um Δt verschieben und erhält die unveränderte Antwort ebenfalls um Δt verschoben. [GHS93]

ist jedoch 0 und fällt somit weg. Da eine Übertragungsfunktion definiert ist als „Quotient der Laplacetransformierten der Ausgangsgröße und der Eingangsgröße des Systems“ [Lun08], wird Gleichung 3.4 wie folgt umgestellt (Gleichung 3.5). Man erhält somit die Übertragungsfunktion der Regelstrecke.

$$\frac{I(s)}{D(s)} = \frac{U}{R_L + L \cdot s} \tag{3.5}$$

So hat die Regelstrecke den Tastgrad der Pulsweitenmodulation als Eingangsgröße und den resultierenden Strom als Ausgangsgröße. Zur Findung eines passenden Reglers wird die Regelstrecke als ein PT_1-Glied betrachtet, also „ein proportional wirkendes Verzögerungsglied erster Ordnung“ [Lun08]. Dieses hat eine Übertragungsfunktion, in der T als *Zeitkonstante* und K als *Verstärkungsfaktor* definiert sind (Gleichung 3.6).

$$G(s) = \frac{K}{T \cdot s + 1} \tag{3.6}$$

Beim Dividieren der Gleichung 3.5 durch R_L wird ersichtlich, weshalb die Regelstrecke ein PT_1-Glied ist. Die Zeitkonstante und der Verstärkungsfaktor betragen somit $T = \frac{L}{R_L}$ respektive $K = \frac{U}{R_L}$. Die Übertragungsfunktion der Regelstrecke $G_S(s)$ lässt sich somit durch Gleichung 3.7 darstellen.

$$G_S(s) = \frac{I(s)}{D(s)} = \frac{\frac{U}{R_L}}{\frac{L}{R_L} \cdot s + 1} \tag{3.7}$$

3.4.2 Herleitung des analogen Reglers

Da die Regelstrecke zeitkontinuierlich ist, wird der Regler vorerst als analoger Regler hergeleitet. Das heißt, dass dieser ebenfalls kontinuierlich arbeiten wird. Hierbei wird auf die Herleitung der Übertragungsfunktion verzichtet und direkt im Bildbereich mit der komplexen Variable s gearbeitet.

Um einen optimalen Regler zu realisieren, wird die *Pol-Nullstellen-Kompensation* angewandt, in der eine Polstelle der Regelstrecke gegen eine Nullstelle des Reglers gekürzt wird [Lun08]. Da die Regelstrecke ein PT_1-Glied ist, lässt sich deren Polstelle exakt mit der Nullstelle eines PI-Reglers kompensieren. Wie der Name schon verrät, besteht dieser aus einem proportionalwirkenden (*P-Glied*) und einem integralwirkenden (*I-Glied*) Regler. Die Übertragungsfunktion eines PI-Reglers findet sich in Gleichung 3.8 [Lun08].

$$G_{PI}(s) = K_P(1 + \frac{1}{T_I \cdot s}) = K_P \frac{T_I \cdot s + 1}{T_I \cdot s} \tag{3.8}$$

Hierbei steht K_P für den Verstärkungsfaktor des proportionalen Anteils. Er sagt aus, wie stark das Ausgangssignal im Verhältnis zum Eingangssignal sein soll. So ist der Ausgang eines P-Reglers definiert als $u(t) = K_P \cdot e(t)$,

mit $e(t)$ als Abweichung vom Eingangssignal. $u(t)$ folgt der Abweichung $e(t)$ somit ohne Verzögerung. T_I beschreibt die Nachstellzeit des I-Reglers, also nach welcher Zeit das Ausgangssignal den Sollwert erreicht. Im Gegensatz zum P-Regler besteht der Zusammenhang zwischen der Ausgangsgröße und der Regelabweichung darin, dass diese Abweichungen ständig kumulativ addiert werden. Wird der Sollwert erreicht, also wenn die Abweichung $e(t) = 0$ ist, behält der Regler seinen Wert so lange, bis die Regelabweichung ungleich 0 wird. Der I-Regler alleine ist ein langsamer Regler, hat aber keine bestehende Regelabweichung im eingeschwungenen Zustand. Im Gegensatz dazu ist der P-Regler schnell, hat jedoch eine bleibende Regelabweichung. Da der PI-Regler beide Regler vereint, ist dieser bei richtiger Auslegung sowohl reaktionsschnell (P-Glied) als auch genau (I-Glied).

Substituiert man in Gleichung 3.8 die Division $\frac{KP}{T_I}$ durch K_{PI} erhält man die Verstärkung des PI-Reglers.

$$G_{PI}(s) = K_{PI}\frac{T_I \cdot s + 1}{s} \tag{3.9}$$

Es wird klar, dass die Nullstelle des Reglers bei $s = -\frac{1}{T_I}$ liegt. Ebenso liegt die Polstelle des PT_1-Gliedes bei $s = -\frac{1}{T}$ (Gleichung 3.6), bzw. die der Regelstrecke bei $s = -\frac{R_L}{L}$ (Gleichung 3.7). Da die Nullstelle des Reglers an der selben Position liegen sollte, muss die Nachstellzeit T_I gleich der Zeitkonstante sein. Somit beträgt $T_I = T = \frac{L}{R_L}$. Durch Anwendung der Pol-Nullstellen-Kompensation bleibt der Regelkreis ein System erster Ordnung. Bei einem System höherer Ordnung träte ein ungewolltes Überschwingen auf und wäre somit eventuell instabil [Lun08].

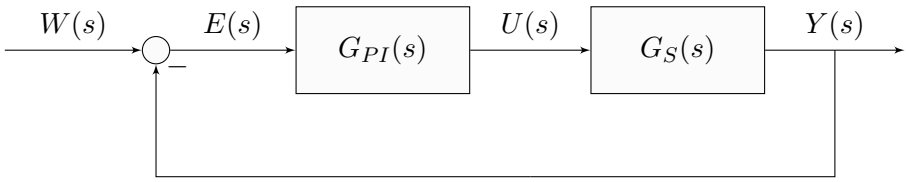


Abbildung 20: Blockschaltbild des Regelkreises

Dies wird deutlich, wenn der Verstärkungsfaktor des PI-Reglers berechnet wird. Hierfür wird der Regelkreis aus den Übertragungsfunktionen des Reglers und der Regelstrecke zusammengesetzt. Betrachtet man Abbildung 20, so kann man folgende Beziehungen definieren [Ott08].

$$Y(s) = U(s) \cdot G_S(s)$$
$$U(s) = E(s) \cdot G_{PI}(s)$$

Da die Übertragungssysteme als eine Reihenschaltung ausgeführt sind, werden diese multipliziert. Die Regelabweichung $E(s)$ lässt sich wegen der negative Rückkopplung beschreiben als $E(s) = W(s) - Y(s)$.

Durch das Einsetzen der Gleichungen erhält man die Übertragungsfunktion des Regelkreises (Gleichung 3.10).

$$\begin{aligned} Y(s) &= E(s) \cdot G_{PI}(s) \cdot G_S(s) \\ Y(s) &= W(s) \cdot G_{PI}(s) \cdot G_S(s) - Y(s) \cdot G_{PI}(s) \cdot G_S(s) \\ (1 + G_{PI}(s) \cdot G_S(s)) \cdot Y(s) &= W(s) \cdot G_{PI}(s) \cdot G_S(s) \\ \frac{Y(s)}{W(s)} &= \frac{G_{PI}(s) \cdot G_S(s)}{1 + G_{PI}(s) \cdot G_S(s)} \end{aligned} \tag{3.10}$$

Die Multiplikation $G_{PI}(s) \cdot G_S(s)$ lässt klar werden, weshalb die Pol- und Nullstellen übereinstimmen sollten. Denn so kann die Polstelle der Regelstrecke mit der mit der Nullstelle des Reglers gekürzt werden, weshalb diese nicht mehr im Regelkreis vorkommen.

$$G_{PI}(s) \cdot G_S(s) = K_{PI} \frac{\frac{L}{R_L} \cdot s + 1}{s} \cdot \frac{\frac{U}{R_L}}{\frac{L}{R_L} \cdot s + 1} = K_{PI} \frac{\frac{U}{R_L}}{s} = K_{PI} \frac{U}{s \cdot R_L} \tag{3.11}$$

Eingesetzt in Gleichung 3.10 lässt sich der Regelkreis als ein PT_1-Glied mit der Verstärkung $K = 1$ darstellen.

$$\frac{Y(s)}{W(s)} = \frac{K_{PI} \frac{U}{s \cdot R_L}}{1 + K_{PI} \frac{U}{s \cdot R_L}} = \frac{1}{s \cdot \frac{R_L}{K_{PI} \cdot U} + 1} \tag{3.12}$$

Die Übergangsfunktion[10] eines solchen Übertragungsgliedes lautet nach [Lun08] $h(t) = K(1 - e^{-\frac{t}{T}})$ und zeigt, dass nach $t = 3 \cdot T$ (dem Dreifachen der Zeitkonstante) ungefähr 95% des Endwertes erreicht werden. Dieser Wert soll für die folgende Realisierung des Reglers eines Schrittmotors genügen. Die Zeitkonstante des Regelkreises beträgt $T = \frac{R_L}{K_{PI} \cdot U}$ (Gleichung 3.12), womit bei vorhandener gewünschter Anstiegszeit $t_{Anstieg}$ die Verstärkung des analogen PI-Reglers berechnet werden kann.

$$\begin{aligned} t_{Anstieg} &= 3 \cdot \frac{R_L}{K_{PI} \cdot U} \\ \Rightarrow K_{PI} &= 3 \cdot \frac{R_L}{t_{Anstieg} \cdot U} \end{aligned} \tag{3.13}$$

3.4.3 Digitaler Regler

Um diesen Regler nun auf einem Mikrocontroller zu verwenden, muss beachtet werden, dass dieser nicht zeitkontinuierlich arbeiten kann. So erfolgt zum Beispiel die Strommessung in bestimmten diskreten Zeitabständen, was in Kapitel 4.2 behandelt wird. Daher muss der zeitkontinuierliche (analoge) Regler durch geeignete Verfahren diskretisiert werden. Hierfür

[10] „Die Übergangsfunktion beschreibt, wie das System auf eine sprungförmige Eingangsgröße reagiert“ [Lun08].

bietet sich die sogenannte *bilineare Transformation* an, welche ohne komplexe Berechnungen direkt vom gegebenen Laplace-Bildbereich in den *z-Bildbereich* umwandeln kann. Die Analyse des *z-Bildbereichs* erfolgt mittels der *z-Transformation*, welche Ähnlichkeiten zur Laplace-Transformation aufweist, jedoch zur Behandlung von zeitinvarianten und -diskreten Systemen eingesetzt wird. Ohne auf eine genaue Herleitung[11] der bilinearen Transformation einzugehen, wird nach [GHS93] die komplexe Variable s wie in Gleichung 3.14 zu sehen substituiert.

$$s = \frac{2}{T_A} \cdot \frac{z-1}{z+1} \tag{3.14}$$

Der Parameter T_A steht hierbei für die Zeitdauer zwischen den diskreten Abtastungen (das Messen der Stromstärke). Nun kann man diese Substitution auf die Übertragungsfunktion des PI-Regler der Gleichung 3.9 anwenden und erhält eine *z-Übertragungsfunktion* (Gleichung 3.16). Hierbei wird die Zeitkonstante T_I bereits durch die Zeitkonstante der Spule $T = \frac{L}{R_L}$ ersetzt (siehe Absatz nach Gleichung 3.9).

$$\frac{U(z)}{E(z)} = K_{PI} \frac{\frac{L}{R_L} \cdot (\frac{2}{T_A} \cdot \frac{z-1}{z+1}) + 1}{\frac{2}{T_A} \cdot \frac{z-1}{z+1}} \tag{3.15}$$

$$\frac{U(z)}{E(z)} = K_{PI} \cdot \left(\frac{\frac{L}{R_L} \cdot \left(\frac{2}{\cancel{T_A}} \cdot \frac{\cancel{z-1}}{z+1} \right)}{\frac{2}{\cancel{T_A}} \cdot \frac{\cancel{z-1}}{z+1}} + \frac{1}{\frac{2}{T_A} \cdot \frac{z-1}{z+1}} \right)$$

$$\frac{U(z)}{E(z)} = K_{PI} \cdot \left(\frac{L}{R_L} + \frac{T_A}{2} \cdot \frac{z+1}{z-1} \right)$$

$$\frac{U(z)}{E(z)} = K_{PI} \cdot \frac{L}{R_L} + K_{PI} \cdot \frac{T_A}{2} \cdot \frac{z+1}{z-1} \tag{3.16}$$

Aus Gründen der besseren Lesbarkeit wurde der Verstärkungsfaktor K_{PI} nicht durch die Gleichung 3.13 ersetzt. Durch Aürzen und Anwenden des Distributivgesetzes erlangt man die vereinfachte Gleichung 3.16. Da die Ausgangsgröße der Übertragungsfunktion des PI-Reglers die Unbekannte ist, wird nach $U(z)$ aufgelöst (Gleichung 3.17).

$$U(z) = E(z) \cdot K_{PI} \cdot \frac{L}{R_L} + E(z) \cdot K_{PI} \cdot \frac{T_A}{2} \cdot \frac{z+1}{z-1}$$

$$U(z) = K_1 \cdot E(z) + K_2 \cdot E(z) \cdot \frac{z+1}{z-1} \tag{3.17}$$

$$K_1 = K_{PI} \cdot \frac{L}{R_L} \tag{3.18}$$

$$K_2 = K_{PI} \cdot \frac{T}{2} \tag{3.19}$$

[11] Eine Herleitung der sogenannten *Tustinschen Näherung* findet sich in [Ott08].

Durch das Zusammenfassen aller Konstanten zu K_1 respektive K_2 (Gleichungen 3.18 und 3.19) erreicht man sowohl bessere Lesbarkeit, als auch eine vereinfachte Berechnung in der späteren Programmierung. Um nun einen Regelalgorithmus zu erhalten, welcher einfach zu programmieren ist, wird eine sogenannte *Differenzengleichung* hergeleitet. Mit dieser lässt sich ein zeitdiskretes System rekursiv beschreiben. Hierfür wird wie in [Ott08] eine Rücktransformation definiert, welche aus dem *z-Bereich* in den diskreten Zeitbereich umwandelt.

$$\mathcal{Z}^{-1}\left\{F(z)\right\} = f_k \tag{3.20}$$

$$\mathcal{Z}^{-1}\left\{z^{-n} \cdot F(z)\right\} = f_{k-n} \tag{3.21}$$

Gleichung 3.21 definiert den *Verschiebungssatz nach rechts* und gilt nur für $n \geq 0$. Da in der vereinfachten Gleichung 3.17 jedoch kein z^{-n} vorhanden ist, wird der Bruch noch mit z^{-1} erweitert[12].

$$\begin{aligned} U(z) &= K_1 \cdot E(z) + K_2 \cdot E(z) \cdot \frac{1+z^{-1}}{1-z^{-1}} \\ U(z) \cdot \left(1+z^{-1}\right) &= K_1 \cdot E(z) \cdot \left(1-z^{-1}\right) + K_2 \cdot E(z) \cdot \left(1+z^{-1}\right) \\ U(z) - U(z) \cdot z^{-1} &= K_1 \cdot \left(E(z) - E(z) \cdot z^{-1}\right) + K_2 \cdot \left(E(z) + E(z) \cdot z^{-1}\right) \end{aligned} \tag{3.22}$$

Die dadurch resultierende Gleichung 3.22 lässt sich mittels Anwendung der Definitionen 3.20 und 3.21 schrittweise in den diskreten Zeitbereich umwandeln.

$$\begin{aligned} &\mathcal{Z}^{-1}\left\{U(z) - U(z) \cdot z^{-1}\right\} \\ =&\mathcal{Z}^{-1}\left\{U(z)\right\} - \mathcal{Z}^{-1}\left\{U(z) \cdot z^{-1}\right\} \\ =&u_k - u_{k-1} \end{aligned} \tag{3.23}$$

Analog zur Laplace-Transformation wird hier ebenfalls Gebrauch vom Linearitätssatz gemacht (vergleiche Kapitel 3.4.1).

$$\begin{aligned} &\mathcal{Z}^{-1}\left\{K_1 \cdot \left(E(z) - E(z) \cdot z^{-1}\right) + K_2 \cdot \left(E(z) + E(z) \cdot z^{-1}\right)\right\} \\ =&K_1 \cdot \left(\mathcal{Z}^{-1}\left\{E(z)\right\} - \mathcal{Z}^{-1}\left\{E(z) \cdot z^{-1}\right\}\right) + K_2 \cdot \left(\mathcal{Z}^{-1}\left\{E(z)\right\} + \mathcal{Z}^{-1}\left\{E(z) \cdot z^{-1}\right\}\right) \\ =&K_1 \cdot (e_k - e_{k-1}) + K_2 \cdot (e_k + e_{k-1}) \qquad (3.24) \\ =&e_k \cdot (K_1 + K_2) - e_{k-1} \cdot (K_1 - K_2) \qquad (3.25) \end{aligned}$$

Das Umformen der Gleichung 3.24 lässt erkennen, dass die Konstanten zusammengefasst werden können (Gleichung 3.25).

$$K_a = K_1 + K_2 = K_{PI} \cdot \frac{L}{R_L} + K_{PI} \cdot \frac{T_A}{2} = K_{PI} \cdot \left(\frac{L}{R_L} + \frac{T_A}{2}\right) \tag{3.26}$$

$$K_b = K_1 - K_2 = K_{PI} \cdot \frac{L}{R_L} - K_{PI} \cdot \frac{T_A}{2} = K_{PI} \cdot \left(\frac{L}{R_L} - \frac{T_A}{2}\right) \tag{3.27}$$

[12]Erweiterung wie folgt: $\frac{z^{-1} \cdot (z+1)}{z^{-1} \cdot (z-1)} = \frac{1+z^{-1}}{1-z^{-1}}$

Die finale Differenzengleichung erhält man durch gleichsetzen der Gleichungen 3.23 und 3.25. Gleichzeitig werden die Konstanten K_1 und K_2 aus genannten Gleichungen substituiert.

$$\begin{aligned} u_k - u_{k-1} &= e_k \cdot K_a - e_{k-1} \cdot K_b \\ \Rightarrow u_k &= e_k \cdot K_a - e_{k-1} \cdot K_b + u_{k-1} \end{aligned} \tag{3.28}$$

So kann nun mittels Gleichung 3.28 die abzugebende normierte Durchschnittsspannung des Reglers berechnet werden. Hierbei bezeichnet u_k die zu berechnende aktuelle Spannung, e_k die aktuelle Differenz der Stromstärke, e_{k-1} die Regelabweichung der vorherigen Iteration, ebenso wie u_{k-1} den vorherigen berechnete Wert darstellt. Die Gleichung ist also rekursiv und besitzt die Anfangswerte $e_0 = u_0 = 0$.

Der nun berechnete PI-Regler kann nun ohne großen Aufwand implementiert werden und benötigt nur eine Anpassung an den entsprechenden Motor und der angelegten Versorgungsspannung. Diese werden in den Konstanten K_{PI} (Gleichung 3.13), K_a (Gleichung 3.26) und K_b (Gleichung 3.27) eingesetzt. Die Anstiegszeit in K_{PI} ist ebenso abhängig vom verwendeten Motor und richtet sich nach dessen Zeitkonstante.

4 Implementierung auf einem Mikrocontroller

Im Folgenden werden zuerst die Grundlagen des verwendeten Mikrocontrollers bzw. der dabei verwendeten Funktionen erläutert. Daraufhin wird die Theorie der vorangegangen Kapitel auf diesem implementiert.

Ein passender Mikrocontroller für die Implementierung benötigt eine gewisse Rechengeschwindigkeit und sollte sowohl die Pulsweitenmodulation (PWM), als auch einen Analog-Digital-Wandler in der Hardware integriert haben. Des Weiteren werden Interrupts benötigt, um die Schritte des Motors in Echtzeit zu verarbeiten. Als nicht erforderlich, jedoch hilfreich, wäre eine DIP Gehäuseform, welche die Hardwarerealisierung vereinfacht.

Als geeignet haben sich die 8-Bit Mikrocontroller der Firma Atmel erwiesen. Diese sind kostengünstig und in vielen verschiedenen Formen und Kapazitäten vorhanden. Ebenso vorteilhaft ist deren einfache Programmierung und die kostenfreie Verfügbarkeit von Entwicklungsumgebungen. Außerdem sind die frei verfügbaren Dokumentationen jedes einzelnen Mikrocontrollertyps sehr ausführlich beschrieben. Aber auch schon vorhandene Kenntnisse im Umgang mit Mikrocontrollern der AVR-Produktfamilie[13] von Atmel waren ein ausschlaggebender Punkt.

Unter den verschiedenen Typen dieses Herstellers hat sich der *ATmega8* für am sinnvollsten erwiesen, da kein großer Flashspeicher (mehr als 8 Kbyte) benötigt wird. Die Rechenleistung ist bei gleicher Taktrate jedoch dieselbe wie die der anderen ATmega Modellen. Auch ist dieser Typ in einer einfach zu verlötenden Bauform vorhanden (DIP). Außerdem besitzt er neben einer Hardware-PWM, einen integrierten Analog-Digital-Wandler und Externe Interrupts. Programmiert wird dieser Controller in der Programmiersprache C, was durch Optimierungen des Compilers einen effizienten Maschinencode erzeugen lässt. Durch die RISC Architektur der ATmega-Modelle besteht der Vorteil, viele der Befehle in nur einem Takt ausführen zu können. So kann bei Taktraten mit maximal 16 MHz eine hohe Performanz von bis zu 16 MIPS erreicht werden [Cor13]. Der 1 Kbyte große *statische RAM* genügt für die Anwendung eines Schrittmotortreibers. Die benötigten Peripherien und Funktionen des Mikrocontrollers werden in den folgenden Unterkapiteln weiter erörtert.

4.1 Pulsweitenmodulation

Bei der Pulsweitenmodulation (PWM) wird ein Rechtecksignal mit konstanter Frequenz erzeugt. Variabel ist, wie in Kapitel 3.3 schon kurz erwähnt, die Impulsbreite. Ist die Frequenz hoch genug, so kann man je nach Trägheit des Systems eine Durchschnittsspannung erzeugen. Der verwendete Mikrocontroller hat die Möglichkeit bis zu sechs parallele PWM Signale zu erzeugen, je zwei mit gleicher Frequenz. Diese werden über sogenannte *Timer* erzeugt,

[13] `http://www.atmel.com/products/microcontrollers/avr/`

wovon zwei mit je 8 Bit und einer mit 16 Bit vorhanden sind. Im weiteren Verlauf wird grundsätzlich ein 8-Bit Timer verwendet.

Das PWM Signal wird erzeugt, indem ein Register (TCNT1) fortwährend inkrementiert wird, bis der maximale Wert (TOP = 0xFF = 255) erreicht wird, woraufhin das Register von vorne startet (BOTTOM = 0x00). Währenddessen wird dieser Wert mit variablen Registern verglichen (OCR1x, $x \in [0, 1]$) und schaltet den entsprechenden Ausgang (OC1x, $x \in [0, 1]$) bei Übereinstimmung (*Compare Match*) aus. Bei Erreichen des BOTTOM Wertes werden die Ausgänge wieder angeschaltet (sogenannter „non-inverting Compare Ouput mode“ [Cor13]). Dieser Vorgang wird auch *Fast-PWM* genannt, da das Zählregister nicht, wie bei andern PWM-Arten, wieder dekrementiert wird. So kann eine zweifache Frequenz erreicht werden, was bei Anwendungen wie Stromregulierungen von Vorteil ist.

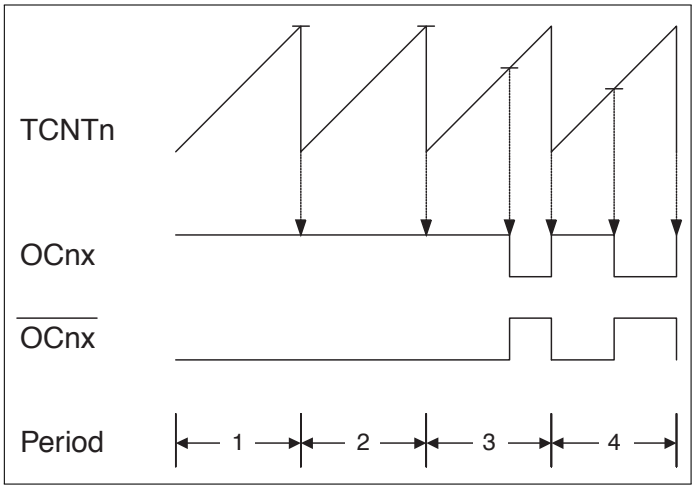


Abbildung 21: Fast PWM Zeitdiagramm, Ausschnitt aus [Cor13]

In Abbildung 21 sieht man ein Zeitdiagramm des Zählregisters und den Logikpegel des Ausgabepins. Die kleinen horizontalen Striche bei *TCNTn* zeigen ein *Compare Match* zwischen OCRnx und TCNTn an. So wurde in diesem Beispiel in der Periode 3 und 4 der Wert von OCRnx jeweils verringert.

Die PWM ist elementarer Bestandteil dieser Realisierung. Denn es wird, wie in Kapitel 2.4.2 erwähnt, eine effektive Stromregelung durch einen sogenannte *Chopper-Betrieb* realisiert. In Verbindung mit dem in Kapitel 3.4.3 hergeleiteten digitalen Regler wird die PWM benutzt, um die Stromstärke beliebig einstellen zu können. Grundlegend dafür wird ein Interrupt sein, welches bei jedem Erreichen von *TOP* des *TCNT1* Registers aufgerufen wird (also zu Beginn jedes Impulses der PWM). In diesem Interrupt werden die Tastgrade für die einzelnen Spulen berechnet, indem jeweils ein Messvorgang und ein Durchlauf des diskreten Reglers durchgeführt wird. Dadurch

kann, je nach PWM Frequenz, eine schnelle Reaktion gewährleistet werden. Hiermit wird aber auch der Nachteil der Verwendung der ATmega Baureihe ersichtlich. Denn der Messvorgang muss bei einer stromdurchflossenen Spule durchgeführt werden, was die Verwendung von *Slow Decay* verhindert. So werden hierfür vier synchrone PWM Signale benötigt, die der ATmega zwar besitzt, jedoch nur jeweils zwei synchron laufen lassen kann. Es kann somit nicht sichergestellt werden, dass bei zwei asynchronen laufenden Zählern die Messvorgänge bei eingeschaltetem Ausgang durchgeführt werden. Im Gegensatz dazu besteht beim *Fast Decay* die Möglichkeit nur zwei PWM Signale verwenden zu müssen. Diese liegen dann an den beiden Enable-Eingänge des *L298* an, welche alle Transistoren abschalten bzw. einschalten können. So wird nur noch ein Zähler benötigt und damit ein synchrones PWM Signal ermöglicht.

Der ATmega8 wird für den Schrittmotortreiber mit einer Frequenz von 16 MHz betrieben. So braucht der Zähler 256 Takte, um *TOP* (255) zu erreichen, wodurch die maximale PWM Frequenz bestimmt werden kann ($f_{OC1xPWM} = \frac{16\,\mathrm{MHz}}{256} = 62\,500\,\mathrm{Hz}$). Des Weiteren besteht die Möglichkeit einen Frequenzteiler zu aktivieren, welcher die Frequenz durch 1, 8, 64, 256 oder 1024 teilen kann. Somit lässt sich die PWM Frequenz wie in Gleichung 4.1 bestimmen, wobei N der Teiler ist.

$$f_{OC1xPWM} = \frac{16\,\mathrm{MHz}}{N \cdot 256} \tag{4.1}$$

Durch die PWM Frequenz kann T_A (die Zeit zwischen den diskreten Abtastungen) für die Konstanten K_a und K_b (Gleichung 3.26 und 3.27) berechnet werden. Da die Frequenz jedoch noch von der Dauer der Strommessung abhängt, wird diese erst in Kapitel 4.2 bestimmt.

Um mit dem Timer1 des ATmega8 ein PWM Signal zu erzeugen, müssen bestimmte Bits in Konfigurations-Registern gesetzt werden. In Register „*Timer/Counter 1 Control Register A - TCCR1A*“ und „*Timer/Counter 1 Control Register B - TCCR1B*“ wird zunächst durch Setzen der Bits „*WGM10*“ und „*WGM12*“ der Fast PWM Modus mit 8 Bit aktiviert. Darüber hinaus müssen im selben Register die Bits „*COM1A1/COM1B1*“ gesetzt werden, um den „non-inverting mode“ zu gewährleisten. Im „*Timer/Counter 1 Control Regiter B - TCCR1B*“ kann der Frequenzteiler eingestellt werden. Dies wird nach Evaluierung der benötigten Dauer der Analog-Ditigal-Wandlung getan. Um den Interrupt zu aktivieren, wird noch im Register „*Timer/Counter Interrupt Mask Register - TIMSK*“ das Bit „*TOIE1*“ gesetzt. Dieser aktiviert den Interrupt beim Überlauf des TCNT1 Registers. Die Konfiguration und der Interrupt Aufruf finden sich in Listing 1.

```
TCCR1A = (1 << WGM10) | (1 << COM1A1) | (1 << COM1B1);
TCCR1B = (1 << WGM12);

TIMSK = (1 << TOIE1);

DDRB |= (1 << PB1) | (1 << PB2); //Pin der PWM als Ausgang definieren

ISR(TIMER1_OVF_vect) //PWM Interrupt
{
    //Stub, Regler fuer Spule 1
    //OCR1A ist Tastverhaeltnis fuer Ausgang OC1A
    OCR1A = CalcPI1(...);

    //Stub, Regler fuer Spule 2
    //OCR1B ist Tastverhaeltnis fuer Ausgang OC1B
    OCR1B = CalcPI2(...);
}
```

Listing 1: PWM Konfiguration und Interruptroutine

4.2 Analog-Digital-Wandler

Ein Analog-Digital-Wandler (AD-Wandler) bietet, wie der Name schon sagt, die Möglichkeit analoge Signale in digitale Werte zu wandeln. Bei den ATmega Mikrocontrollern ist diese Möglichkeit bereits integriert, womit ein externer AD-Wandler wegfällt und so ein vereinfachter Aufbau ermöglicht wird. So kann eine Spannung zwischen 0 V und der Referenzspannung des AD-Wandlers gemessen werden. Diese beträgt bei der Schrittmotorsteuerung 5 V und gleicht damit der Versorgungsspannung des ATmega8. Die Messung erfolgt automatisch mittels einer sukzessiven Approximation. So wird eine erzeugte Vergleichsspannung mit der zur messenden Spannung verglichen. Falls diese größer als die angelegte Spannung ist, wird die Vergleichsspannung im nächsten Schritt um eine Bitstelle verringert und umgekehrt. So nähert sich in jedem Schritt die Referenzspannung an die zu messende Spannung an. Hierfür wird zwingend eine sogenannte *Sample & Hold Stufe* benötigt, welche die Spannung beim Messvorgang mittels eines Kondensators stabil hält. Dies ist bereits im Mikrocontroller integriert.

Die für das Projekt wichtigen Kenndaten des integrierten AD-Wandlers sind vor allem die Auflösung und die Abtastungen pro Sekunde (englisch *samples per second*, kurz SPS). So kann maximal eine 10-Bit Auflösung erreicht werden, was bei 5 V Referenzspannung eine Auflösung von ungefähr 0.004 88 V entspricht. Des Weiteren gibt es sechs verschiedene Hardwareeingänge, an welchen die Spannung gemessen werden kann. Diese sind jedoch mithilfe eines Multiplexers an den einzigen verfügbaren AD-Wandler durchgeschaltet, weshalb keine parallelen Messvorgänge durchgeführt werden können. Laut Datenblatt ist der AD-Wandler fähig, bis zu 15 kSPS durchzuführen, was für diesen Anwendungszweck jedoch zu wenig ist. Pro Abtastung

werden 13 Taktzyklen des AD-Wandler-Takts benötigt. Durch einen Frequenzteiler (englisch *prescaler*) mit sieben möglichen Teilungen wird dieser Takt aus dem CPU-Takt erzeugt. Im Normalfall soll dieser laut Datenblatt maximal 200 kHz betragen, was also ungefähr 15384 SPS entspricht. Daher wird die Möglichkeit genutzt den AD-Wandler zu *übertakten* und somit mehr SPS zu ermöglichen. Es muss jedoch in Kauf genommen werden eine niedrigere Auflösung zu erhalten [Cor13]. Die Übertaktung erfolgt durch geeignete Wahl des Frequenzteilers. Den Verlust der Genauigkeit wurde in [Lab15] untersucht und in einem Diagramm dargestellt (Abbildung 22).

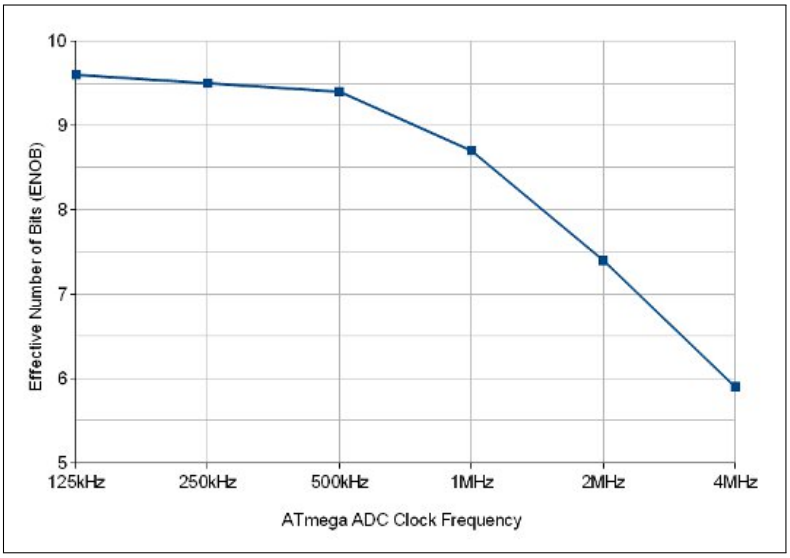


Abbildung 22: Vergleich von AD-Wandler Taktung und Genauigkeit [Lab15]

Die höchste Frequenz mit einer akzeptablen Genauigkeit ist somit 1 MHz, da eine Auflösung von 8 Bit noch garantiert werden kann. Somit dauert eine Wandlung 13 µs. Es wird also eine sechzehntel Teilung des CPU-Taktes benötigt. Die erste Wandlung dauert jedoch 25 Takte und muss daher vor dem Beginn der Regelung durchgeführt werden. Das Resultat wird, bedingt durch die Auflösung von 10 Bit, in zwei Registern gespeichert („*ADCH*“ und „*ADCL*“) und muss zusammengefügt werden.

Konfiguriert wird der AD-Wandler, wie auch die PWM, durch das Setzen bestimmter Bits in den Konfigurations-Registern. So wird als erstes die Spannungsreferenz auf die Versorgungsspannung des AD-Wandlers gesetzt indem das Bit „*REFS0*“ in dem Register „*ADC Multiplexer Selection Register ADMUX*“ gesetzt wird. Außerdem kann in diesem Register der Eingang des Multiplexers gewählt werden. Dies wird im Interrupt vor jeder Wandlung geschehen. Im Register „*ADC Control and Status Register A - ADCSRA*“ kann durch das Bit „*ADEN*“ der AD-Wandler aktiviert werden. Darüber hinaus muss hier der Frequenzteiler gewählt werden indem das Bit „*AD-*

PS2“ gesetzt wird. Dieser aktiviert einen Teilungsfaktor von 16. Die erste, länger andauernde Wandlung wird direkt nach der Konfiguration des AD-Wandlers durchgeführt. Dies geschieht durch das Setzen des Bits „*ADSC*“ im selben Register. Solange die Wandlung läuft bleibt das „*ADSC*“ Bit gesetzt. Daher wird eine Schleife benötigt, welche aktiv auf das Beenden der Wandlung wartet (englisch *busy waiting*). Listing 2 zeigt die Konfiguration des AD-Wandlers und die Messvorgänge innerhalb der PWM-Interruptroutine.

```
1  //globale Variablen
2  int16_t currentA = 0; //Stromlimit Spule 1
3  int16_t currentB = 0; //Stromlimit Spule 2
4
5  [...]
6
7  ADMUX = (1 << REFS0); //Referenzspannung
8  ADCSRA = (1 << ADPS2) | (1 << ADEN); //Frequenzteiler & ADC Aktivierung
9
10 ADCSRA |= (1 << ADSC); //Initialisierungswandlung starten
11 while (ADCSRA & (1 << ADSC)) //Aktives Warten
12     ;
13
14 [...]
15
16 ISR(TIMER1_OVF_vect) //PWM Interrupt
17 {
18     ADCSRA |= (1 << ADSC); //ADC-Messung starten Spule 1
19     while (ADCSRA & (1 << ADSC))
20         ; //Auf ADC warten ~13us
21     uint16_t cur1 = ADCW; //Messwert speichern
22
23     ADMUX &= ~(1 << MUX0);  //Setzen des Multiplexers auf Eingang 1
24
25     ADCSRA |= (1 << ADSC); //ADC-Messung starten Spule 2
26     while (ADCSRA & (1 << ADSC))
27         ;
28     uint16_t cur2 = ADCW;
29
30     ADMUX |= (1 << MUX0); //Setzen des Multiplexers auf Eingang 2
31
32     OCR1A = CalcPI1(cur1, currentA); //Berechnung des Reglers
33     OCR1B = CalcPI2(cur2, currentB); //Berechnung des Reglers
34 }
```

Listing 2: AD-Wandler Initialisierung und PWM Interrupt

„*ADCW*“ (Listing 2 Zeile 21 und 28) beinhaltet den gewandelten Wert und ist ein Makro, welches die Register „*ADCH*“ und „*ADCL*“ addiert und als ein 16-Bit Wert ausgibt. Durch das Setzen des Bit „*MUX0*“ in Zeile 23 schaltet der Multiplexer des AD-Wandlers auf den zweiten Eingang. Somit wird ab Zeile 25 die Stromstärke der zweiten Spule gemessen. Dies wird mit dem Löschen des Bits in Zeile 30 wieder rückgängig gemacht und schaltet den Multiplexer so wieder auf den ersten Eingang.

Durch die Kenntnis über die Dauer des Messvorgangs kann die Frequenz der PWM nun hergeleitet werden. So werden mindestens $2 \cdot 13\,\mu s$ benötigt um die Messvorgänge der einzelnen Spulen durchzuführen. Dies würde eine Frequenz von ungefähr $\frac{1}{2 \cdot 13\,\mu s} \approx 38\,461.54\,Hz$ ergeben. Da der Frequenzteiler des Timer1, welcher für die PWM verwendet wird, mit einem Divisionsfaktor von 1 eine Frequenz von $62\,500\,Hz$ (siehe Formel 4.1) ergeben würde, muss der nächst kleinere Faktor gewählt werden. Dieser wäre $N = 8$ und ergäbe eine Frequenz von $f_{OC1xPWM} = \frac{16\,MHz}{8 \cdot 256} = 7812.5\,Hz$. Die Konfiguration der PWM kann somit wie in Listing 3 zu sehen abgeschlossen werden. Durch das Setzen des Bits „*CS11*“ wird der Prescaler auf 8 gesetzt.

```
1 TCCR1B = (1 << WGM12) | (1 << CS11); //Prescaler auf 8 setzen
```

Listing 3: Setzen des Prescalers der PWM

Somit kann nun T_A der Konstanten K_a und K_b (Gleichungen 3.26 und 3.27) bestimmt werden ($T_A = \frac{1}{7812,5} = 128\,\mu s$). Die restliche Zeit bis zum nächsten PWM-Interrupt wird für die Berechnungen der Reglers verwendet, welche mit ungefähr $100\,\mu s$ ausreichend ist.

4.3 Software Regler

Die Funktionen *CalcPI1* und *CalcPI2* in Listing 2 werden nun näher erläutert. Wie schon in Kapitel 3.4 gezeigt, benötigt der Regler eine Regelabweichung als Eingang und gibt den Tastgrad als Stellgröße aus. Die hergeleitete Differenzengleichung 3.28 muss somit, möglichst effizient, implementiert werden. Dazu werden vorerst die Konstanten als Markos definiert, um so schon während der Kompilation vom Präprozessor berechnet zu werden. Im Folgenden werden die Induktivität und der Widerstand eines Motors verwendet, welcher während der Entwicklung als Versuchsmotor genutzt wurde.

```
1 #define MOTOR_L 0.205           //Motor Induktivitaet
2 #define MOTOR_R 82.5            //Motor Widerstand
3
4 #define DC 30                   //Versorgungsspannung
5
6 #define SAMPLE_TIME 0.000128    //Abtastzeit, Reziproke PWM Frequenz
7 #define RISE_TIME MOTOR_L/MOTOR_R   //Anstiegszeit
```

Listing 4: Konstanten des Schrittmotors

Die Spannung betrug während der Versuche stets $30\,V$. Die Anstiegszeit in der Gleichung 3.13 kann somit aus der Zeitkonstante der Motorspule berechnet werden ($t_{Anstieg} = \frac{L}{R_L} = \frac{0.205\,H}{82.5\,\Omega}$). Durch Kenntnis der Werte des Listings 4 kann als nächstes der Verstärkungsfaktor des PI-Reglers berechnet werden (Gleichung 4.2).

$$K_{PI} = 3 \cdot \frac{82.5\,\Omega}{\frac{0.205\,H}{82.5\,\Omega} \cdot 30\,V} \approx 3320 \tag{4.2}$$

Da der ATmega keine Gleitkommaeinheit (FPU) besitzt, welche die Berechnung von Gleitkommazahlen beschleunigt, wird der Performanz halber mit Festkommazahlen gerechnet. Dies wird bewerkstelligt, indem der Präprozessor die Konstanten mit einer Zweierpotenz multipliziert. Die Konstanten werden dann in der Berechnung des PI-Reglers zusammen mit den Variablen, wie zum Beispiel den Messwerten, verwendet. Anschließend wird das Ergebnis durch die gesamten Zweierpotenzen geteilt. Der Vorteil der Multiplikation und Division mit Zweierpotenzen ist, dass es sich bei diesem Vorgang um eine bitweise Verschiebung (englisch *bitwise shift*) handelt. Dieser Operator ist eine Grundkomponente von CPUs und auch bei ATmega8 als Assemblerbefehl integriert. So braucht eine Linksverschiebung einen Takt und ist, verglichen mit einer Multiplikation, welche zwei Takte benötigt, doppelt so schnell. Ein Befehl für Division existiert erst garnicht in den ATmega CPUs und müsste in Software realisiert werden. Es würde also, im Vergleich zu einer Rechtsverschiebung, wesentlich länger als einen Takt benötigen. Somit können nun die Konstanten K_a und K_b (Gleichungen 3.26 und 3.27) und der Verstärkungsfaktor implementiert werden.

```
1 #define PI_GAIN 3*MOTOR_R/(RISE_TIME * DC)          //Verstaerkungsfaktor
2 //Konstanten Ka und Kb
3 #define PI1 (int32_t)(PI_GAIN*(MOTOR_L/MOTOR_R + SAMPLE_TIME/2)*16)*255*5
4 #define PI2 (int32_t)(PI_GAIN*(MOTOR_L/MOTOR_R - SAMPLE_TIME/2)*16)*255*5
5
6 #define MIN_PWM 4   //entspricht ca. 2 Mikrosekunden
```

Listing 5: Konstanten des PI-Reglers

Auf die Definition der Zeile 6 wird nach der Erläuterung der Implementierung des PI-Reglers eingegangen. Der Faktor 16 in den Konstanten $PI1$ und $PI2$ entspricht einer Linksverschiebung um 4 Bits und dient der Genauigkeit der Konstanten. Der Faktor 255 dient zur Umrechnung der Stellgröße in einen vom Mikrocontroller verstanden Wert für den Tastgrad. Denn der PI-Regler gibt, wie schon erwähnt, eine Zahl zwischen 0 bis 1 aus, wohingegen der Mikrocontroller diesen Wert auf 0 bis 255 abbildet, da das Tastverhältnis der PWM dies verlangt. Der letzte Faktor in der Definition der Konstanten, also die 5, wird genutzt, um den gemessenen Wert des AD-Wandlers in einen vom PI-Regler verstandenen Wert (Ampere) zu wandeln. Im weiteren Verlauf dieses Kapitels wird davon ausgegangen, dass bei einer Steigerung der Stromstärke um 1 A die Spannung am AD-Wandler ebenfalls um 1 V steigt. Weitere Details bezüglich der Realisierung der Strommessung folgen in Kapitel 5.1. Da die gewandelten Werte des AD-Wandlers zwischen 0 V und der Referenzspannung (5 V) liegen und als 10-Bit Wert ausgegeben werden, können die Messwerte durch eine Multiplikation mit dem Bruch $\frac{Referenzspannung}{Bittiefe} = \frac{5\,\mathrm{V}}{1024}$ umgewandelt werden. Um auch hier eine Festkommazahl zu erhalten, wird eine Linksverschiebung um 10 Bits durchgeführt. Dies entspricht also einer Multiplikation mit $2^{10} = 1024$, womit sich der Bruch auflöst und der Fak-

tor 5 übrig bleibt (siehe Listing 5 Zeile 3f.). Die Bitbreite der Konstanten wurde auf 32 Bit und vorab als *signed* festgelegt, um so eine spätere Typumwandlung als auch ein arithmetischen Überlauf zu vermeiden. Denn bei der Berechnung der Stellgröße (Formel 3.28) treten sowohl größere Werte auf als ein 16-Bit Integer zulässt, als auch negative Zahlen.

Bei der Implementierung der Berechnung der Stellgrößen ist darauf zu achten, die Messwerte als Parameter der Funktion ebenfalls in vorzeichenbehaftete 32-Bit Integer umzuwandeln. Zuvor müssen noch die benötigten Variablen deklariert werden (Listing 6).

```
1 extern int16_t PI_sum1;     //Ergebnis der Differenzengl. Regler 1
2 extern int16_t PI_sum2;     //Ergebnis der Differenzengl. Regler 2
3 extern int16_t lastError1;  //Abweichung zurueckliegende Messung Regler 1
4 extern int16_t lastError2;  //Abweichung zurueckliegende Messung Regler 2
5 extern int16_t PI1_out;     //Ausgabe des PI-Reglers 1
6 extern int16_t PI2_out;     //Ausgabe des PI-Reglers 2
```

Listing 6: Globale Variablen

Diese Variablen wurden im Gegensatz zu den Konstanten als vorzeichenbehaftete 16-Bit Integer deklariert. Denn nach der Berechnung der Differenzengleichung wird die Festkommazahl wieder zurückgerechnet, um so ein verwendbares Ergebnis zu erhalten und somit den maximalen Wert eines 16-Bit Integers nicht zu überschreiten. Dies geschieht in Listing 7.

```
1  extern inline int16_t CalcPI1(uint16_t currADC, int16_t currRef)
2  {
3      //Berechnung der Differenz des Spulenstromes (Regelabweichung)
4      int16_t currentError = currRef - (int16_t) currADC;
5
6      //temporaere Variable
7      int32_t temp_PI_sum = (PI1 * (int32_t) currentError - PI2 * (int32_t)
       lastError1) >> 14;
8
9      //Differenzengleichung
10     PI_sum1 = PI_sum1 + (int16_t) temp_PI_sum - ((PI_sum1 - PI1_out) >> 4);
11
12     lastError1 = currentError;      //Speichern der zurueckliegenden Messung
13
14     //Limitierung der PWM
15     if (PI_sum1 > 255)
16         PI1_out = 255;
17     else if (PI_sum1 < MIN_PWM)
18         PI1_out = MIN_PWM;
19     else
20         PI1_out = PI_sum1;
21
22     return PI1_out;
23 }
24
25 extern inline int16_t CalcPI2(uint16_t currADC, int16_t currRef) { [...] }
```

Listing 7: Implementierung der PI-Regler

Die Funktion CalcPI2 (Listing 7 Zeile 25) berechnet wie CalcPI1 die Stellgröße für die entsprechende Motorspule, verwendet jedoch statt *PI_sum1* und *lastError1* entsprechend *PI_sum2* und *lastError2* als globale Variablen und wird hier deshalb nicht vollständig ausgeführt. Um möglichst schnell die Funktionen zur Berechnung der PI-Regler aufzurufen, wurden *inline*-Funktionen verwendet. So fügt der Compiler die inline-Funktion an der Stelle ein, an der diese aufgerufen wurde. Es muss also weder eine Rücksprungadresse gespeichert, noch ein Sprungbefehl ausgeführt werden. Des Weiteren wurde eine temporäre Variable eingeführt (Listing 7 Zeile 7). In der selben Zeile wird eine Rechtsverschiebung um 14 Bit vorgenommen (Division durch 16384), welche sich aus den Konstanten *PI1* respektive *PI2* ergibt und zur Umwandlung der Festkomma-Arithmetik dient.

In Zeile 10 wird die Berechnung der Differenzengleichung durchgeführt. Da die Stellgröße des Reglers durch die Spannung begrenz wird, kann es bei großen Sprüngen in der Führungsgröße passieren, dass die Summe zu große Werte annimmt. Diese Gefahr ist vor allem dann gegeben, wenn kein Mikroschritt verwendet wird. Um dem vorzubeugen wird von der Summe in Zeile 10 noch ein Bruchteil der Differenz zwischen der Stellgröße und der limitierten Stellgröße abgezogen. Die Division durch 8, bzw. Rechtsverschiebung um 4 Bit, vermindert den Effekt um keine negativen Auswirkungen zu erhalten.

Durch das Speichern der Differenz des Spulenstromes wird die nächste Iteration des PI-Reglers ermöglicht (Zeile 12). Der Rückgabewert der beiden Funktionen entspricht dem Tastgrad der Pulsweitenmodulation im Wertebereich von 0 bis 255 (8-Bit PWM). Da die PI-Regler jedoch auch Regelgrößen weit über 255 erzeugen, muss dieser für die PWM limitiert werden (Zeile 15). Da, wie schon erwähnt, ausschließlich *Fast Decay* verwendet wird, ist die minimale Regelgröße 0, also ein vollständiges Abschalten der Transistoren (Enable-Eingang des Treibers auf Low-Pegel). Dies würde jedoch verhindern, dass in der nächsten Iteration ein sauberer Wert gemessen werden kann. Um die Filterkondensatoren und Verstärkerschaltung (siehe Kapitel 5.1) mit ausreichend Spannung zu versorgen, wurde ein minimales PWM-Tastverhältnis verwendet (Listing 7 Zeile 18). Nach [Man09] wurde ein Wert von ca. 1.75 µs für ausreichend befunden. Dies entspricht ungefähr einem Tastverhältnis von 4 (siehe Listing 5 Zeile 6).

4.4 Externer Interrupt

Um die selbe Funktion wie der *L297* Baustein zu besitzen, wird bei der Implementierung ein externer Interrupt eingesetzt. Dieser wird dafür zuständig sein, um auf Schrittsignale zu reagieren. So wird ein Pin des Mikrocontrollers dafür verwendet, auf steigende oder sinkende Flanken zu reagieren, um daraufhin die Spulen entsprechend ansteuern zu können. Wird bei einem AVR Mikrocontroller ein Interrupt ausgelöst, so deaktiviert dieser das *Global Interrupt Enable I-bit*, womit alle weiteren Interrupts deaktiviert werden. Da

die Pulsweitenmodulation bei jedem Puls ein Interrupt ausführt, welche wegen der AD-Wandlung und PI-Regler Berechnungen einige Zeit in Anspruch nimmt, besteht die Möglichkeit, dass ein externes Signal während der Abarbeitung der Interruptroutine auftritt. In diesem Fall werden entsprechende *Flags* gesetzt und der externe Interrupt nach der Routine ausgeführt (englisch *pending interrupts*). Um zu vermeiden, dass solch ein Signal mehrfach innerhalb eines PWM-Interrupts auftritt und verloren geht, muss darauf geachtet werden, dass die Impulslänge des Schrittsignals eine minimale Länge aufweist. Dies kann in der Ansteuerungssoftware (zum Beispiel LinuxCNC[14]) eingestellt werden. Die minimale Impulslänge sollte der Pulsweite der PWM entsprechen, wodurch sichergestellt wird, dass nicht mehr als eine steigende Flanke pro PWM-Periode auftritt.

Die externen Interrupts können beim ATmega8 Mikrocontroller auf zwei Pins ausgelöst werden (*INT0* und *INT1*). Da ausschließlich das Schrittsignal ein Interrupt auslösen soll, wird nur INT0 verwendet. Dieses löst, je nach Konfiguration der entsprechenden Register, bei einer steigenden oder fallenden Flanke aus und kann somit an die Ansteuerungssoftware angepasst werden. Hierfür wird für fallende Flanken im Register „*MCUCR*“ das Bit „*ISC01*“ gesetzt (Listing 8 Zeile 9). Soll bei steigenden Flanken ein Interrupt ausgelöst werden, muss zusätzlich das „*ISC00*“ Bit gesetzt werden (Zeile 7). Zum Einschalten des externen Interrupts muss nun noch im „*GICR*“ Register das „*External Interrupt Request 0 Enable* “ Bit („*INT0* “) aktiviert werden (Zeile 12). Die Interruptroutine (Zeile 16) wird in Kapitel 4.5 implementiert.

```
1  //globale Konfigurationsvariablen
2  #define RISINGEDGE 1    //Schrittsignal bei steigender Flanke
3
4  [...]
5
6  #if RISINGEDGE == 1 //steigende Flanke
7      MCUCR |= (1 << ISC01) | (1 << ISC00);
8  #else   //fallende Flanke
9      MCUCR |= (1 << ISC01);
10 #endif
11
12 GICR |= (1 << INT0);    //Externen Interrupt aktivieren
13
14 [...]
15
16 ISR(INT0_vect)
17 {
18     //Interruptroutine fuer externen Interrupt
19 }
```

Listing 8: Externer Interrupt

[14] Ansteuerungssoftware für CNC Maschinen: `http://www.linuxcnc.org`

4.5 Mikroschritt

Die Schrittsteuerung soll alle in Kapitel 2.3 erwähnten Betriebsarten unterstützen. So wird zunächst der Mikroschritt implementiert, welcher beliebig eingestellt werden kann. Jedoch müssen die Grenzen der mechanischen Auflösung beachtet werden, weshalb die maximale Auflösung $\frac{1}{8}$-Mikroschritt beträgt. Um die Komplexität möglichst gering zu halten, wird die Berechnung der Spulenströme und Phasenpolarität optimiert. Als Grundlage dient eine Lookup-Tabelle (LUT) der diskreten Sinuswerte für einen $\frac{1}{8}$-Mikroschritt. Die Anzahl der Elemente in dem Array der LUT beträgt 16 Werte und entspricht somit 180° der Sinuskurve. Zuerst müssen die Werte der LUT an den Schrittmotor angepasst werden. Dies geschieht in der Initialisierungsphase des Programms (Listing 9).

```
1  #define MAXCURRENT 0.43 //max. Strom des Motors nach Datenblatt in Ampere
2
3  [...]
4
5  //Initialisierung
6  int16_t lutMotor[16];   //globale LUT, auf Motor angepasst
7  float lutSine[16] = {0, 0.2, 0.38, 0.56, 0.71, 0.83, 0.92, 0.98, 1, 0.98,
       0.92, 0.83, 0.71, 0.56, 0.38, 0.2};   //Diskrete Sinuswerte 0-180 Grad,
        1/8 Schritt
8
9  for (int i = 0; i < 16 ; i++)
10     lutMotor[i] = (int16_t)(lutSine[i]*(MAXCURRENT*1024/5)+512);
```

Listing 9: Anpassen der LUT an den Schrittmotor

Zunächst wird eine symbolische Konstante definiert, welche den maximalen Strom des Schrittmotors nach Spezifikation bzw. Anwendungsfall angibt (Listing 9 Zeile 1). Die LUT in Zeile 6 beinhaltet die diskreten Sinuswerte für $\frac{1}{8}$ Mikroschritte. Zur Berechnung der motorspezifischen Werte wurde über die Länge der LUT iteriert. Jeder Wert wurde mit $MAXCURRENT \cdot 1024/5$ multipliziert, womit die diskreten Sinuswerte in Werte umgewandelt werden, die auch vom AD-Wandler abgespeichert werden (also eine Abbildung des Stromes in Ampere in den Wertebereich von 0 bis 1024). Die Addition mit 512 hängt mit der Art und Weise zusammen, wie der Strom gemessen wird, da bei einer Stromstärke von 0 A eine Spannung von 2.5 V am AD-Wandler anliegt. Details diesbezüglich werden in Kapitel 5.1 weiter erläutert. Die resultierende Lookup-Tabelle beinhaltet somit die zu erreichenden Ströme der Spule, um bestimmte Schrittwinkel zu erhalten. Diese können, wegen der Konstanten des PI-Reglers (siehe Listing 5 und Erläuterung), direkt als Referenzwert genutzt werden und benötigen keine weiteren Umrechnungen. Somit kann also in jeder Ausführung des PI-Reglers Rechenzeit gespart werden.

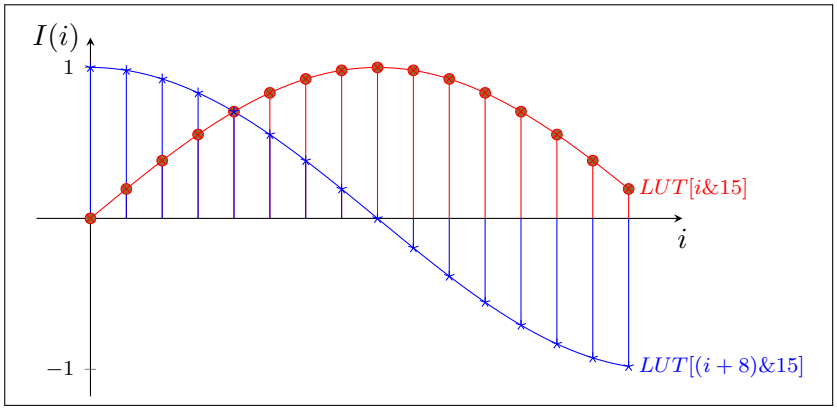


Abbildung 23: Diskreter Stromverlauf der Spulen

Im Folgenden wird nun eine Methode entwickelt, um den Spulen die passenden Ströme je nach Schrittindex zuzuteilen. Bei jedem externen Interrupt wird ein Schrittzähler (Listing 10 Zeile 16) inkrementiert, welcher als Index für die LUT dienen wird. Da jedoch auch die Richtung, in der sich der Motor drehen soll, bestimmt werden kann, lässt sich dieser Zähler ebenso dekrementieren. Für das Richtungssignal wird ein weiterer Pin des Mikrocontrollers verwendet, welcher bei jedem externen Interrupt ausgelesen wird und je nach logischem Pegel den Zähler entsprechend inkrementiert bzw. dekrementiert (Listing 10 Zeile 24 und 26). Um die Auflösung des Mikroschritts variieren zu können, wird eine symbolische Konstante definiert (Zeile 1), welche als Inkrement bzw. Dekrement dient. Die Spulen, im weiteren Verlauf *Spule A* und *Spule B* genannt, benötigen im Mikroschrittbetrieb einen um 90° verschobene Sinusfunktion (Abbildung 23). Somit entfällt der Bedarf einer weiteren LUT für *Spule B*. Stattdessen wird für *Spule B* der aktuelle Schrittzähler, addiert mit der halben Arraylänge der LUT (also 8), als Index verwendet (10 Zeile 30). Um nicht auf Speicherwerte außerhalb der LUT zuzugreifen, wird dieser Index noch mit der um 1 dekrementierten Länge der LUT bitweise maskiert (bitweises UND). Dies entspricht der modulo-Berechnung mit $2^4 = 16$. Somit werden immer alle restlichen Bits nach dem vierten Bit auf 0 gesetzt. Dies wird ebenfalls mit dem Index für *Spule A* getan (Zeile 29). Somit muss der vorzeichenlose 8-Bit Schrittzähler nicht bei jedem Erreichen von 16 (dem maximalen Wert) zurückgesetzt werden, sondern kann einen Über- bzw. Unterlauf erzeugen, ohne falsche Werte zu liefern. Die symbolische Konstante *STEPMODE* dient im späteren Verlauf dazu, die entsprechenden Betriebsarten festlegen zu können (Zeile 2). Diese Konstante wird mit der Präprozessordirektive *#if* in Zeile 22 ausgewertet.

```
#define STEPRESOLUTION  1   //Mikroschrittaufloesung: 1 -> 1/8 | 2 -> 1/4
#define STEPMODE        1   //Betriebsarten
                            //Mikroschritt: 1
                            //Halbschritt:  2
                            //Vollschritt:  3
                            //Wavedrive:    4

//globale Variablen
int16_t currentA = 0;   //Stromlimit Spule1
int16_t currentB = 0;   //Stromlimit Spule2

[...]

//Initialisierung
DDRD &= ~(1 << PD3);    //Pin D3 als Eingang definieren
uint8_t stepIndex = 0;  //Schrittindex

[...]

ISR(INT0_vect)  //externer Interrupt
{
#if (STEPMODE == 1)
    if(PIND & (1 << PD3))   //falls Pin D3 High Pegel: Rechts-Rotation
        stepIndex += STEPRESOLUTION;    //Schrittindex inkrementieren
    else    //sonst Links-Rotation
        stepIndex -= STEPRESOLUTION;    //Schrittindex dekrementieren
#endif

    currentA = lutMotor[stepIndex & 15];        //Strom Spule A (Sinus)
    currentB = lutMotor[(stepIndex + 8) & 15];  //Strom Spule B (Cosinus)
}
```

Listing 10: Stromberechnung der Spulen

Zur Herleitung der Polarität der Phasen kann der Schrittzähler verwendet werden. Am Beispiel der *Spule A* erkennt man in Abbildung 23, dass die negativen Werte der Sinusfunktion nach Erreichen von $i = 16$ vorkommen. Durch Maskieren mit der Zahl 2^4 bzw. mit $0x10 = 0b1000$ wird somit die Information über die Polarität erlangt. Denn sobald das Resultat ungleich 0 ist, ist die Polarität umgekehrt. Damit lässt sich die Ansteuerung des Treiberbausteins *L298* realisieren, da die Pins des Mikrocontrollers entsprechend der Polarität geschaltet werden müssen. In Listing 11 werden zunächst die Pins *PB4*, *PB5*, *PD6* und *PD7* als Ausgänge definiert (Zeile 3f.). Diese dienen der Ansteuerung der Transistoren der zwei H-Brücken des *L298* Treibers (siehe 3.2). In den Zeilen 12 bis 15 für *Spule A* und den Zeilen 17 bis 20 für *Spule B* werden, nach der Information bezüglich der Polarität, entsprechende Pins geschaltet, um den Stromfluss der Spule zu steuern. Hierbei wird als Phasensequenz die Betriebsart *Vollschritt* genutzt (zwei Phasen sind dauerhaft aktiv). Der Mikroschritt wird dann, wie schon erwähnt, erreicht, indem der Spulenstrom individuell geregelt wird.

```
//Initialisierung
//Ausgaenge definieren fuer L298 Input
DDRB |= (1 << PB4) | (1 << PB5);    //Spule A
DDRD |= (1 << PD6) | (1 << PD7);    //Spule B

[...]

ISR(INT0_vect)  //externer Interrupt
{
    [...]

    if ((stepIndex & 0x10))
        PORTB = (PORTB | (1 << PB4)) & ~(1 << PB5);
    else
        PORTB = (PORTB | (1 << PB5)) & ~(1 << PB4);

    if (((stepIndex + 8) & 0x10))
        PORTD = (PORTD | (1 << PD7)) & ~(1 << PD6);
    else
        PORTD = (PORTD | (1 << PD6)) & ~(1 << PD7);
}
```

Listing 11: Bestimmung der Phasenpolarität

4.6 Voll- und Halbschritt

Des Weiteren sollen die Schrittmodi Vollschritt und Halbschritt implementiert werden. Um maximale Drehmomente zur Verfügung zu stellen, werden beide Spulen mit der maximalen Stromstärke angesteuert. Würde man die selbe LUT verwenden, welche die Sinuswerte für den Mikroschritt beinhaltet, so wäre das Drehmoment vermindert (das 0.71-fache des maximalen Drehmomentes). Daher muss Listing 9 geändert werden, sodass diese Tabellen entsprechend definiert werden (Listing 12).

```
int16_t lutMotor[16];   //globale LUT, an Motor angepasst

//Initialisierung
int16_t curr = (MAXCURRENT*1024/5)+512; //Max. Strom von 512 bis 1024
#if (STEPMODE == 1) //Mikroschritt
float lutSine[16] = {0, 0.2, 0.38, 0.56, 0.71, 0.83, 0.92, 0.98, 1, 0.98,
    0.92, 0.83, 0.71, 0.56, 0.38, 0.2};   //Diskrete Sinuswerte 0-180 Grad

for (int uint8_t = 0; i < 16 ; i++)
    lutMotor[i] = (int16_t)(lutSine[i]*MAXCURRENT*1024/5)+512;
#elif (STEPMODE == 2 || STEPMODE == 4)  //Halbschritt und Wavedrive
    lutMotor[4] = lutMotor[8] = lutMotor[12] = curr;
#elif (STEPMODE == 3)   //Vollschritt
    lutMotor[0] = lutMotor[4] = lutMotor[8] = lutMotor[12] = curr;
#endif
```

Listing 12: LUT für restliche Schrittmodi

Darüber hinaus muss die Routine des externen Interrupts (Listing 10) entsprechend angepasst werden. Denn im Falle des Vollschritt- und Wavedrivebetriebes, muss jeweils jeder achte Werte der LUT genutzt werden (Listing 13 Zeile 7 und 9). Im Falle des Halbschrittbetriebes muss der Inkrement bzw. Dekrement 4 betragen (Zeile 12 und 14).

```
1  ISR(INT0_vect)  //externer Interrupt
2  {
3  #if (STEPMODE == 1)
4      [...]
5  #elif (STEPMODE == 3 || STEPMODE == 4)  //Vollschritt und Wavedrive
6      if(PIND & (1 << PD3))   //falls Pin D3 High Pegel: Rechts-Rotation
7          stepIndex += 8;    //Schrittindex inkrementieren
8      else    //sonst Links-Rotation
9          stepIndex -= 8;    //Schrittindex dekrementieren
10 #elif (STEPMODE == 2)   //Halbschritt
11     if(PIND & (1 << PD3))   //falls Pin D3 High Pegel: Rechts-Rotation
12         stepIndex += 4;    //Schrittindex inkrementieren
13     else    //sonst Links-Rotation
14         stepIndex -= 4;    //Schrittindex dekrementieren
15 #endif
16
17     [...]
18 }
```

Listing 13: Schrittweite für restliche Schrittmodi

Durch die in Listing 12 angepassten Lookup-Tabellen und der Verwendung von Präprozessordirektiven wird die Verwendung unterschiedlicher Betriebsmodi ermöglicht und die Konfiguration vereinfacht. Zudem werden so in der Routine des externen Interrupts keine unnötigen Sprungbefehle ausgeführt, da nur der entsprechende Code kompiliert wird.

4.7 Stromabsenkung

Um ein Überhitzen des Motors bei Stillstand zu vermeiden, wird eine Funktionalität hinzugefügt, welche den Strom automatisch absenkt. Dies geschieht, wenn innerhalb einer zuvor definierten Zeit kein Schrittsignal, also keine steigende bzw. sinkende Signalflanke am Pin *INT0*, auftritt.

Hierfür wird eine globale Variable *reduceCounter* eingefügt, welche bei jedem PWM Interrupt inkrementiert wird (Listing 14 Zeile 20). Wird ein bestimmter Wert erreicht (*REDUCE_TIME*), so halbiert sich die Stromstärke der beiden Spulen. Dabei muss vor dem Halbieren der Verstärkeroffset (2.5 V) abgezogen und nach dem halbieren wieder hinzu addiert werden (Zeile 24 und 25). Wird ein externer Interrupt registriert, so wird der *reduceCounter* zurückgesetzt (Zeile 10). Durch das Setzten der Variable auf den Wert 0 wird erreicht, dass der Zähler nach bereits abgesenktem Strom nicht wieder anfängt von vorne zu Zählen (Zeile 23). Die Stromstärke kann also nur nach einem erneuten Schrittsignal wieder abgesenkt werden.

Außerdem wurde eine symbolische Konstante (*REDUCE_CURRENT*) eingeführt mit welcher sich diese Funktionalität aktivieren oder deaktivieren lässt (Listing 14 Zeile 3). Die Anzahl der Interrupts und somit die benötigte Zeit, bis ein Absenken des Stromes eintritt, lässt sich über die symbolische Konstante *REDUCE_TIME* einstellen (Zeile 4).

```
1  uint16_t reduceCounter = 0;
2
3  #define REDUCE_CURRENT 1    //bei Stillstand Stromreduzierung, 0 = disable
4  #define REDUCE_TIME 7813    //Interrupts bis Stromreduzierung, 1 = 128us
5
6  ISR(INT0_vect)  //externer Interrupt
7  {
8      [...]
9
10     reduceCounter = 1;  //Aktivieren/Zuruecksetzen des Zaehlers
11 }
12
13 ISR(TIMER1_OVF_vect)
14 {
15     [...]
16
17 #if REDUCE_CURRENT == 1
18     if (reduceCounter != 0)
19     {
20         reduceCounter++;
21         if (reduceCounter > REDUCE_TIME)
22         {
23             reduceCounter = 0;
24             currentA = ((currentA - 512)>>1)+512;   //Spule 1 auf 50%
25             currentB = ((currentB - 512)>>1)+512;   //Spule 2 auf 50%
26
27         }
28     }
29 #endif
30 }
```

Listing 14: Absenken des Stromes bei Stillstand

4.8 Programmablaufpläne

Im Folgenden wird der Programmablauf in Diagrammen aufgezeigt. In Abbildung 24 links ist das Flussdiagramm des Hauptprogramms mit dessen Unterprogrammen zu sehen. Es werden zunächst globale Variablen definiert, wie zum Beispiel die globalen Variablen des Stromlimits der beiden Spulen. Darauf folgt die Konfiguration der Hardware des Mikrocontrollers (Abbildung 24 mitte). Hier werden die entsprechenden Register der Ports, PWM, des AD-Wandler und externen Interrupts passend gesetzt. In der Initialisierung der Software (Abbildung 24 rechts) wird je nach definiertem Betriebsmodus die passende Lookup-Tabelle für den Motor bestimmt bzw. berechnet. Da

Auswahlvorgang der Modi während der Kompilierung passiert, entspricht dies nicht dem eigentlichen Programmcode, sondern soll nur verdeutlichen, dass die LUT unterschiedliche Werte annehmen kann.

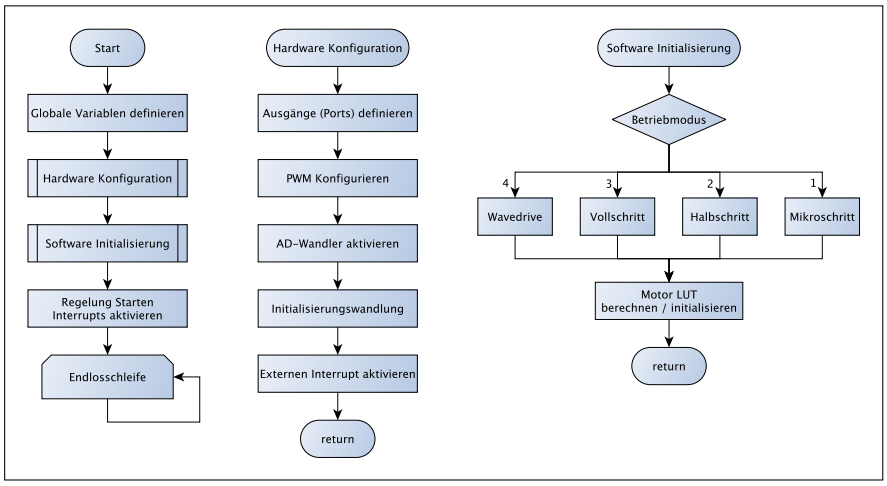


Abbildung 24: Flussdiagram des Hauptprogramms und der Unterprogramme

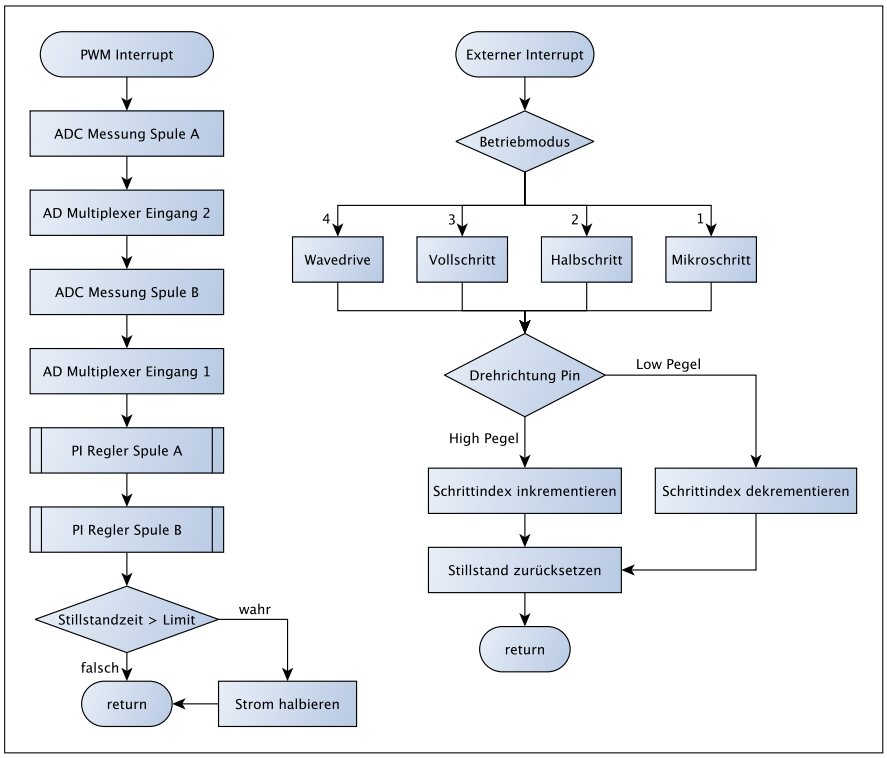


Abbildung 25: Flussdiagram der Interruptroutinen

Das linke Diagramm in Abbildung 25 zeigt den Ablauf der PWM Interruptroutine. Die Subroutinen der beiden PI Regler wurden dabei nicht in einem eigenen Ablaufplan dargestellt, da diese ausschließlich Berechnungen durchführen. Ebenso wie im vorherigen Ablaufplan wurde im rechten Diagramm der Abbildung 25 eine Unterscheidung der Betriebsmodi dargestellt. Diese wird jedoch auch vom Präprozessor durchgeführt und ist im eigentlichen Programmcode nicht vorhanden.

5 Aufbau der Hardware und Auswertung

In diesem Kapitel geht es um den Aufbau der benötigten Hardware. Dies beinhaltet sowohl die Messung des Stromes als auch das Entwerfen der Platine. Im Anschluss werden Messungen ausgewertet, um den Vorteil der verwendeten Methoden darzustellen.

5.1 Strommessung

Zur Messung des Stromes existieren mehrere Möglichkeiten. Herauskristallisiert haben sich die Methoden zur Strommessung über einen Messwiderstand und die Messung durch Nutzung des Halleffekts.

Geeignet hat sich für dieses Projekt die Verwendung eines Messwiderstands, da dieser im Aufbau kostengünstiger zu realisieren ist, aber auch eine hinreichende Genauigkeit bietet. Um den Halleffekt messen zu können werden spezielle Sensoren benötigt, was den Aufbau des Schrittmotortreibers verkompliziert. Als Vorteil gegenüber eines Messwiderstands sei zu erwähnen, dass die Nutzung eines Hallsensors berührungsfrei stattfindet.

Wie in Abbildung 26 gezeigt, fließt bei eingeschalteten Transistoren ein Strom sowohl durch die Spule als auch über einen Messwiderstand (R_{Sense}). Da die Stromstärke in einer Reihenschaltung an allen Verbrauchern identisch ist, fällt am Messwiderstand nach dem ohmschen Gesetz eine Spannung ab, die proportional zur Stromstärke ist.

Ein Messwiderstand, auch *Shunt* genannt, zeichnet sich durch einen geringen ohmschen Widerstand aus. Die Wahl des Widerstands hängt nach [Lep03] von mehreren Faktoren ab. So ist die Messgenauigkeit bei Verwendung eines größeren Widerstands erhöht, da die Verstärkung der Spannung geringer sein kann. Allerdings führt dieser auch zu einem höheren Spannungsverlust. Durch geeignetes Einsetzen der Formel des ohmschen Widestands ($U = R \cdot I$) in die Formel für die elektrische Leistung ($P = U \cdot I$), wird der Zusammenhang zur Verlustleistung deutlich ($P = R \cdot I^2$). Je kleiner der Widerstand, desto geringer ist die in Wärme umgewandelte elektrische Energie. Als Besonderheit sollte der Widerstand zudem eine geringe Induktivität besitzen, da dies bei hohen Schaltfrequenzen negative Auswirkungen haben kann. Bei Verwendung von normalen Widerständen statt speziellen Shunts wird nach [Lep03] empfohlen, Metallschichtwiderstände zu verwenden, da diese eine geringere Induktivität im Vergleich zu Drahtwiderständen aufweisen. Als Alternative besteht die Möglichkeit der Herstellung eines Widerstands durch passende Leiterbahnen auf einer Platine. Diese weisen jedoch eine erhöhte Ungenauigkeit bei sich ändernder Temperatur auf. So liegt der Temperaturkoeffizient bei ungefähr $0.39\frac{\%}{°C}$ [Lep03].

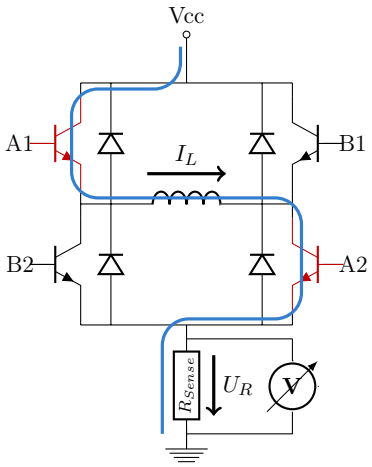


Abbildung 26: Strommessung durch Messwiderstand

In der Regel liegt der Wert des Widerstands unter 1 Ω, sodass hier nach dem ohmschen Gesetz nur eine geringe Spannung abfällt und die umgesetzte thermische Energie gering gehalten wird. Nach testweiser Verwendung eines 0.1 Ω Widerstands mit einer Genauigkeit von 5%, hat sich herausgestellt, dass die Messergebnisse zu stark vom Sollwert abweichen. Besonders deutlich wird dies beim Betrachten der diskreten Sinuswerte (z.B. in Listing 12). So muss eine 2-prozentige Abweichung der Stromstärke messbar sein, was bei einer Genauigkeit von 5% nicht möglich ist. Daher wurde als Messwiderstand ein 0.1 Ω Metallschichtwiderstand der Firma *Vishay Intertechnology* verwendet. Dieser besitzt eine Toleranz von ±0.1% und eignet sich für hochfrequente Anwendungen [Dal13]. Zudem verträgt der Widerstand eine Leistung von 3 W und eignet sich somit für Ströme bis zu 5.47 A. Ein Messwiderstand mit 0.1 Ω hat zudem keinen großen Einfluss auf die Ansteuerung des Motors, da sich der Widerstand von Schrittmotoren im Bereich von ungefähr 2 Ω bis 30 Ω oder mehr bewegt.

5.1.1 Verstärker

Der Nachteil eines solch geringen Widerstands besteht darin, dass die Messung am Mikrocontroller ungenauer wird. So liegt bei einer maximalen Stromstärke von 2.5 A eine Spannung von 0.25 V am Eingang des AD-Wandlers an. Bei entsprechend geringerer Stromstärke fällt die Spannung proportional am Shunt ab und kann unter Umständen nicht mehr messbar sein.

Da die Auflösung des AD-Wandlers 10 Bit beträgt, kann mit einer Referenzspannung von 5 V eine Genauigkeit von maximal 0.0488 A erreicht werden. Benötigt ein Schrittmotor beispielsweise 2 A, so ist bei einem $\frac{1}{8}$-Mikroschritt mindestens eine Genauigkeit von 0.06 A erforderlich. Um dies zu Kompensieren wird in dieser Realisierung eine Verstärkerschaltung nach

[Mic] aufgebaut, welche die zu messende Spannung um den Faktor 10 verstärkt.

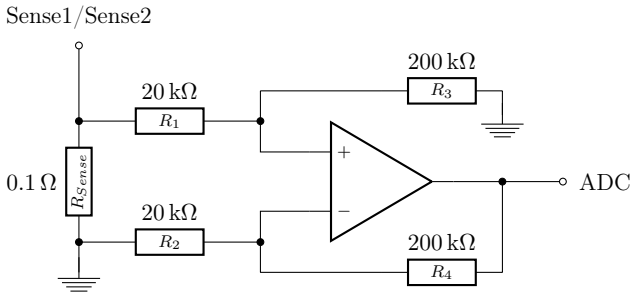


Abbildung 27: Schaltplan des Differenzverstärkers

Mittels eines Operationsverstärkers wird ein Differenzverstärker aufgebaut (Abbildung 27). Dieser gibt am Ausgang die Differenz der Eingangsspannungen aus, für den Fall, dass $R_1 = R_2$ und $R_3 = R_4$ sind [Fed12]. Da hier die Masse an den negativen Eingang des Operationsverstärkers angeschlossen wird, kann die Schaltung auch durch einen nichtinvertierenden Verstärker ersetzt werden. Jedoch besteht so die Möglichkeit, eine Vierleiter Messanordnung[15] anzuschließen, um so die Genauigkeit weiter zu erhöhen. Die Verstärkung des, in Abbildung 27 dargestellten Verstärkers, lässt sich wie in Gleichung 5.1 gezeigt beschreiben [Fed12].

$$\begin{aligned} V &= \frac{R_1 + R_3}{R_1} \cdot \frac{R_4}{R_2 + R_3} \\ &= \frac{R_1 + R_3}{R_1} \cdot \frac{R_3}{R_1 + R_3} \\ &= \frac{R_3}{R_1} = \frac{200\,\text{k}\Omega}{20\,\text{k}\Omega} = 10 \end{aligned} \tag{5.1}$$

Somit wird der geringe Widerstand des Shunts (0.1 Ω) durch die Verstärkung kompensiert. Eine Stromstärke von 1 A ergibt also eine messbare Spannung von 1 V ($1\,\frac{\text{V}}{\text{A}}$). Der Anschluss *Sense1/Sense2* wird an die Masseanschlüsse für die Spulen des *L298* Treiberbausteins verbunden. Pro Spule wird also eine Verstärkerschaltung benötigt.

[15] Bei einer Vierleiter-Messanordnung wird das Messgerät über zwei eigene Leitungen vor und nach dem zu messenden Widerstand angeschlossen. Leitungs- und Kontaktwiderstände „haben somit einen geringeren Einfluss auf die Messung“[GF14].

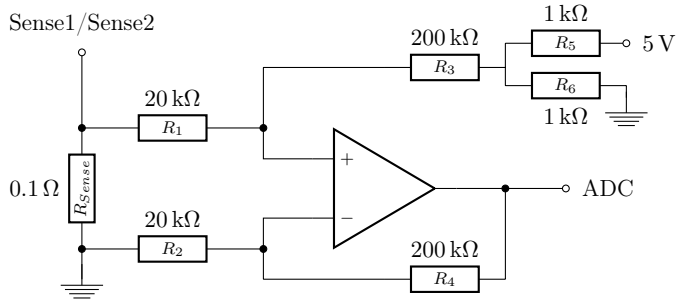


Abbildung 28: Differenzverstärker mit Offsetspannung

Da bei der Verwendung von *Fast Decay* auch negative Ströme auftreten können wird eine Offset-Spannung hinzugefügt. Diese hebt die Spannung am Ausgang des Operationsverstärkers auf 2.5 V, wenn kein Strom durch den Messwiderstand fließt, also die Differenz der Eingangsspannungen 0 V beträgt. Hierfür wird ein Spannungsteiler eingefügt, welcher die Referenzspannung des AD-Wandlers (5 V) halbiert (Abbildung 28). Dies geschieht durch die Widerstände R_5 und R_6, welche nach Formel 5.2 die benötigte Spannung erzeugen [SP10].

$$U_{Teil} = U_{ADC_{Ref}} \cdot \frac{R_6}{R_5 + R_6} = 5\,\mathrm{V} \cdot \frac{1\,\mathrm{k\Omega}}{1\,\mathrm{k\Omega} + 1\,\mathrm{k\Omega}} = 2.5\,\mathrm{V} \tag{5.2}$$

Da in dieser Realisierung der negative Eingang auf Masse liegt, vereinfacht sich die Berechnung der Ausgangsspannung U_a (Gleichung 5.3) unter der Bedingung, dass die Widerstände R_3,R_5 und R_6 nicht beachtet werden (dies entspricht der Ausgangsspannung eines nichtinvertierenden Verstärkers) [Fed12]. Die Spannung U_+ soll für die Spannung stehen, die am positiven Eingang anliegt.

$$U_a = U_+ \cdot \left(1 + \frac{R_4}{R_2}\right) \tag{5.3}$$

Um die Offset-Spannung zu berechnen, wird angenommen, dass kein Strom durch den Shunt fließt und somit keine Spannung abfällt. So liegt wegen der Widerstände R_3 und R_1, welche einen Spannungsteiler darstellen, und der erzeugten Spannung von 2.5 V (Gleichung 5.2) dennoch eine Spannung am positiven Eingang an. Im Folgenden wird die Spannung U_+ in Gleichung 5.3 durch diese Spannung ersetzt (Gleichung 5.4).

$$U_a = U_{Offset} = U_{Teil} \cdot \frac{R_1}{R_1 + R_3} \cdot \left(1 + \frac{R_4}{R_2}\right) \tag{5.4}$$

$$U_{Offset} = U_{Teil} \cdot \frac{R_1}{R_1 + R_3} \cdot \left(1 + \frac{R_3}{R_1}\right)$$

$$U_{Offset} = U_{Teil} \cdot \frac{R_1}{R_1 + R_3} \cdot \frac{R_1 + R_3}{R_1}$$

$$U_{Offset} = U_{Teil} \tag{5.5}$$

Hierbei stellt $U_{Teil} \cdot \frac{R_1}{R_1+R_3}$ die resultierende Spannung des Spannungsteilers dar. Da $R_4 = R_3$ und $R_2 = R_1$ lässt sich Gleichung 5.4 vereinfachen. Es folgt, dass die Offset-Spannung gleich der halben Referenzspannung des AD-Wandlers ist (Gleichung 5.5). Somit lässt sich nach [Fed12] die Spannung am Ausgang der Differenzverstärkerschaltung (Abbildung 28) herleiten. Hierfür werden die zuvor berechneten Spannungen in die Gleichung 5.3 eingesetzt. Da hier jedoch der Widerstand R_3 nicht beachtet wurde, muss die Spannung, welche durch den Spannungsteiler R_1/R_3 erzeugt wird, noch hinzugefügt werden (Gleichung 5.6).

$$U_a = U_+ \cdot \frac{R_3}{R_1 + R_3} \cdot (1 + \frac{R4}{R_2}) = U_+ \cdot \frac{R_3}{R_1} \tag{5.6}$$

Wiederum lässt sich durch Gleichsetzen der Widerstände und geeignetes Kürzen eine verkürzte Formel herleiten. Folgend wird die Offset-Spannung U_{Offset} addiert und die Spannung am positiven Eingang U_+ ersetzt durch die Spannung U_{Sense}, welche am Shunt abfällt (Gleichung 5.7).

$$U_a = U_{Offset} + U_{Sense} \cdot \frac{R_3}{R_1} = 2.5\,\mathrm{V} + (R_{Sense} \cdot I_{Spule1/2}) \cdot 10 \tag{5.7}$$

Die Spannung U_{Sense} lässt sich durch das ohmsche Gesetz ersetzen mit $R_{Sense} \cdot I_{Spule1/2}$, wobei $I_{Spule1/2}$ für den Strom der jeweiligen Motoren steht.

Die resultierende Spannung am Ausgang des Operationsverstärkers gibt somit eine Spannung zwischen 0 V und 5 V aus, wobei 2.5 V einer Stromstärke von 0 A entsprechen. Es können also sowohl negative als auch positive Ströme gemessen werden. Ein weitere Vorteil liegt darin, dass bei negativen Strömen (bzw. den daraus resultierenden negativen Spannungen) der AD-Wandler nicht beschädigt wird. Weiterhin wird durch die Verstärkung eine Spannung von $1\,\frac{\mathrm{V}}{\mathrm{A}}$ an den AD-Wandler angelegt. Die maximale Stromstärke beträgt somit 2.5 A, da die Versorgungsspannung des Operationsverstärkers das Limit der Ausgangsspannung vorgibt. Um dies zu gewährleisten, wird der *OPA2340* der Firma *Texas Instruments* verwendet, welcher ein sogenannter *Rail-to-Rail* Operationsverstärker ist. Das bedeutet, dass, im Gegensatz zu konventionellen Operationsverstärkern, die Ausgangsspannung hier „bis an die Versorgungsspannung heranreichen“ kann [Fed12]. Des Weiteren besitzt dieser Verstärker eine *Anstiegsrate* (englisch *slew rate*) von $6\,\frac{\mathrm{V}}{\mu\mathrm{s}}$, was die „maximal mögliche Anstiegsgeschwindigkeit der Ausgangsspannung“ darstellt [Fed12]. Da in dieser Realisierung eines Schrittmotortreibers eine PWM-Frequenz von 7812.5 Hz verwendet wird, ist diese Anstiegsrate mehr als ausreichend.

5.1.2 Tiefpassfilter

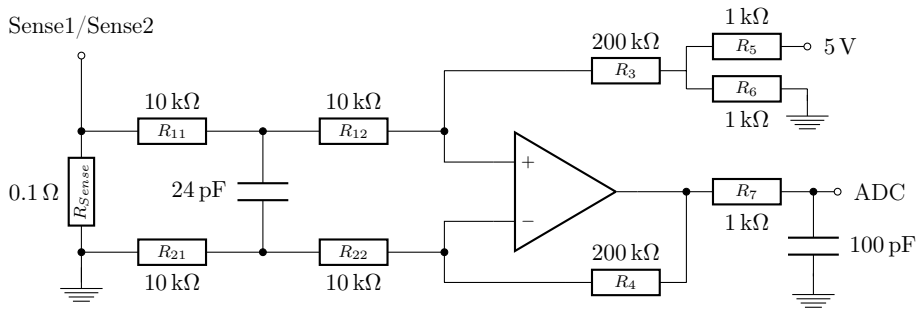


Abbildung 29: Differenzverstärker mit Tiefpassfilter

Durch die Verstärkung und die hohe Schaltfrequenz ist das dem Verstärker zugeführte und ausgegebene Signal verrauscht. Dies kann die Messung mit dem AD-Wandler beeinflussen, was wiederum die Genauigkeit des Mikroschritts beeinträchtigen kann. Daher werden wie in [Mic] sogenannte *Tiefpassfilter* jeweils vor und nach dem Operationsverstärker aufgebaut. Dieser Filter hat „bei tiefen Frequenzen seinen Durchlass- und bei hohen seinen Sperrbereich“ [Büt09]. Die Grenzfrequenz f_c beschreibt die Frequenz, ab welcher die Spannung eines Signals auf das $\frac{1}{\sqrt{2}}$-fache abgeschwächt wird[16]. Diese lässt sich bei einem *RC-Filter*, also einem Tiefpassfilters mit einem Widerstand und einem Kondensator, wie in Formel 5.8 berechnen. Hierbei steht R für die Größe des Widerstands und C für die Kapazität des Kondensators.

$$f_c = \frac{1}{2 \cdot \pi \cdot R \cdot C} \tag{5.8}$$

Der schematische Schaltplan in Abbildung 29 zeigt die beiden Tiefpassfilter. Hierfür wurden die Widerstände R_1 und R_2 mit je 20 kΩ wie in [Mic] geteilt, sodass jeweils 10 kΩ in Reihe liegen. Somit lässt sich bei einem Kondensator mit einer Kapazität von 24 pF die Grenzfrequenz des Tiefpassfilters vor dem Operationsverstärker berechnen (Gleichung 5.9).

$$f_c = \frac{1}{2 \cdot \pi \cdot R \cdot C} = \frac{1}{2 \cdot \pi \cdot 10\,\text{k}\Omega \cdot 24\,\text{pF}} \simeq 663\,145.6\,\text{Hz} \tag{5.9}$$

Dieser Tiefpass soll zunächst kurze Spannungsspitzen abschwächen, welche ansonsten um den Faktor 10 verstärkt würden. Der verwendete Operationsverstärker *OPA2340* kann durch die hohe Anstiegsrate bereits bei einer solchen Frequenz den Ausgang bis zur Versorgungsspannung von 5 V ansteigen lassen. Um diesen Vorgang sichtbar zu machen, wurde das Programm

[16] Dies entspricht einer Abschwächung um 3 dB.

LTSpice[17] verwendet, welches die Schaltung des Verstärkers mit den Filtern simuliert. So wurde eine Wechselstrom-Analyse durchgeführt, um den Frequenzgang des ersten Tiefpassfilters darzustellen (Abbildung 30). Hierfür floss ein Wechselstrom mit einer Amplitude von 2.5 A durch den Messwiderstand ($R5$ in Abbildung 32).

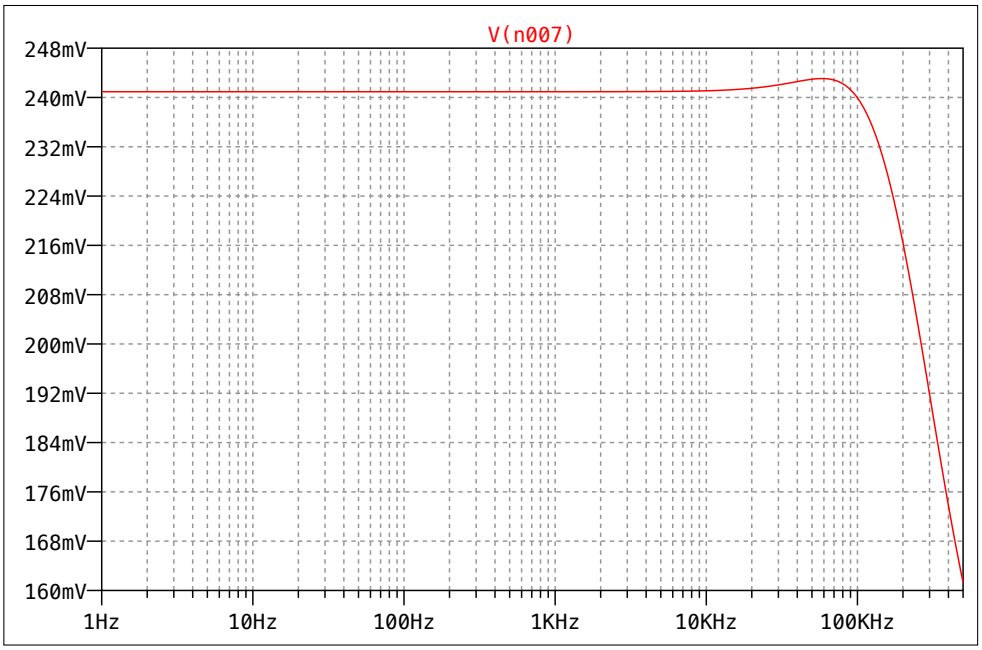


Abbildung 30: Frequenzgang des ersten Tiefpassfilters

Man erkennt die Abschwächung oberhalb einer Frequenz von ungefähr 100 kHz. Jedoch werden erst deutlich über 500 kHz Spannungen durchgelassen, welche nur geringe Auswirkungen haben.

Darüber hinaus erzeugen reale Operationsverstärker, im Gegensatz zum idealen Operationsverstärker, am Ausgang ein Rauschen. Dieses wird durch den zweiten Tiefpassfilter unterdrückt. Durch das Einsetzen der Werte des Widerstands und des Kondensators aus Abbildung 29 in Formel 5.8 lässt sich auch diese Grenzfrequenz berechnen (Gleichung 5.10).

$$f_c = \frac{1}{2 \cdot \pi \cdot R \cdot C} = \frac{1}{2 \cdot \pi \cdot 1\,\mathrm{k\Omega} \cdot 100\,\mathrm{pF}} \simeq 1\,591\,549.4\,\mathrm{Hz} \tag{5.10}$$

Ebenso wurde der Frequenzgang nach dem zweiten Tiefpassfilter erzeugt (Abbildung 31). Hier wird deutlich, dass ab einer Frequenz von ungefähr 400 kHz Rauschsignale mit einer Amplitude von 250 mV, welche vor der Verstärkerschaltung auftreten, ausreichend abgeschwächt wurden (auf ungefähr 0.2 V).

[17] Software zur Simulation von Schaltungen (`http://www.linear.com/designtools/software/`).

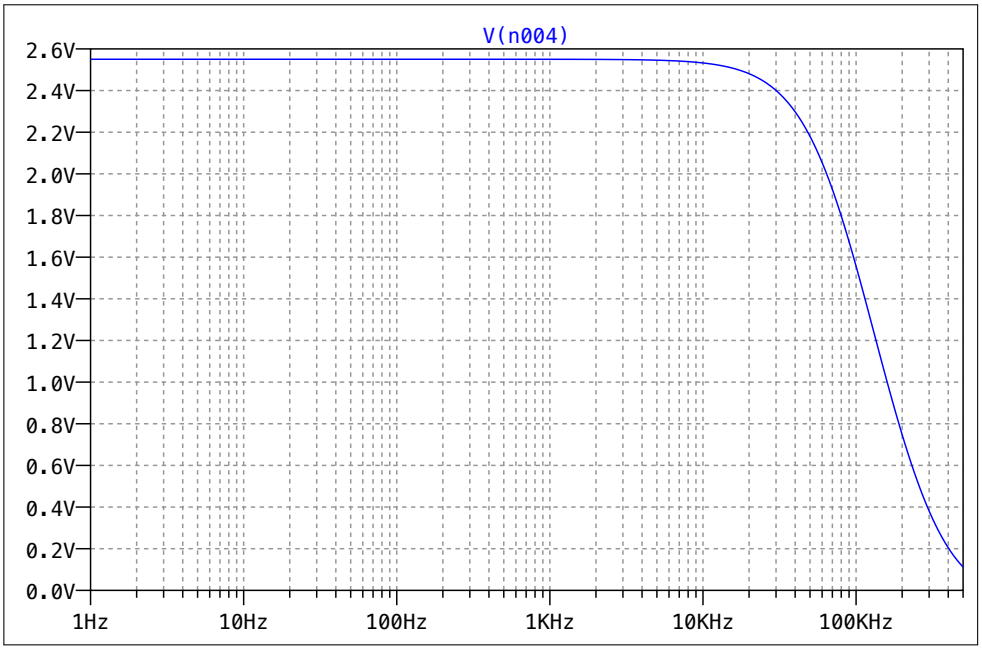


Abbildung 31: Frequenzgang nach zweitem Tiefpassfilter

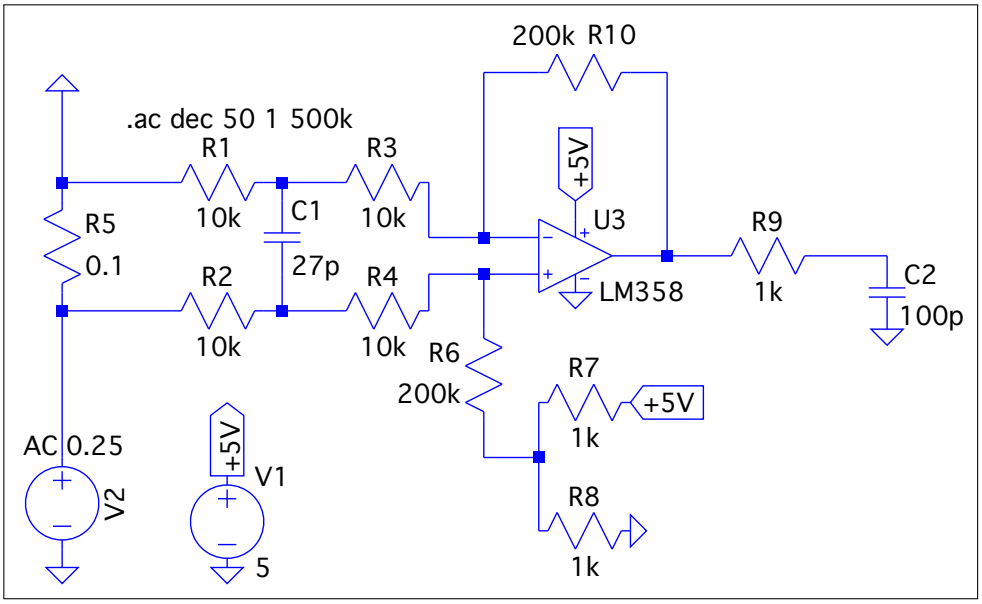


Abbildung 32: Schaltungsaufbau zur Simulation in LTSpice

5.1.3 Messwerte

Im Anschluss an die Simulationen wurde die Platine aufgebaut und mit einem Oszilloskop Messungen durchgeführt. Diese zeigen die Vorteile der Verwendung der Tiefpassfilter. Hierfür wurde nur eine Spule des Motors aktiviert und auf eine Stromstärke von 110 mA geregelt (erkennbar ist die Pulsweitenmodulation des Reglers). So wird in Abbildung 33 deutlich, dass der verwendete Schrittmotor einen Einschaltstrom besitzt, welcher eine deutliche Spannungsspitze am zweiten Kanal (blaues Signal) des Oszilloskops erzeugt. Hier wird die Spannung, welche am Messwiderstand abfällt, angezeigt. Das Signal des ersten Kanals (gelbes Signal) wurde nach dem ersten Tiefpassfilter abgegriffen und zeigt, dass der Tiefpass diese Spannungsspitze herausfiltert. Im Allgemeinen wirkt dieser Spannungsverlauf sauberer.

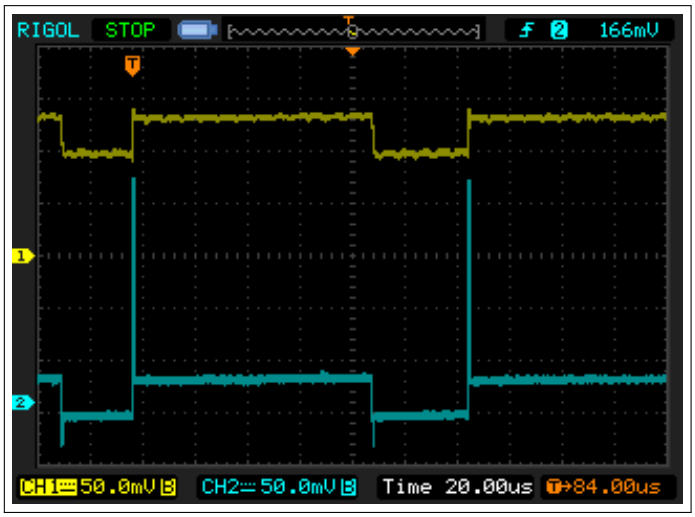


Abbildung 33: Frequenzgang nach zweitem Tiefpassfilter

Die zweite Messung (Abbildung 34) zeigt den Vergleich der Spannung direkt am Shunt (blaues Signal) und nach dem Differenzverstärker, jedoch vor dem zweiten Tiefpass (gelbes Signal). Zu beachten ist, dass sich die Skalierung der Y-Achse bei den beiden Signalen unterscheidet, was damit zusammenhängt, dass die Spannung an Kanal 1 bereits verstärkt und mit einem Offset versehen wurde. Hier ist eine Ähnlichkeit zu einer Oberschwingung zu erkennen und resultiert wahrscheinlich aus den nichtlinearen Eigenschaften der Bauteile.

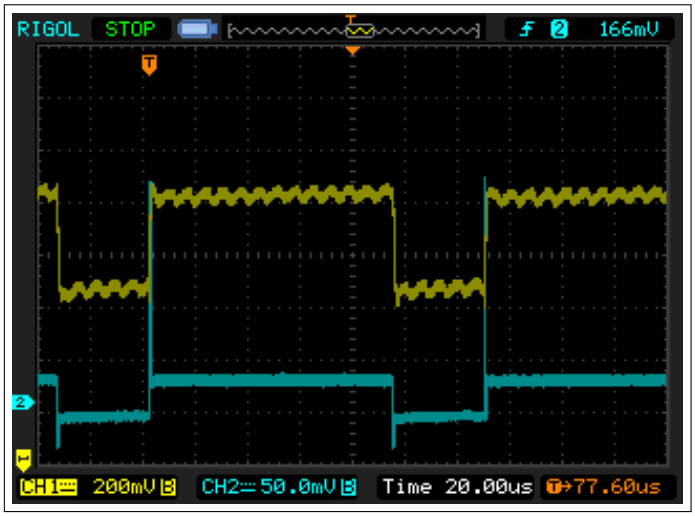


Abbildung 34: Frequenzgang nach Verstärkung, vor zweitem Tiefpassfilter

Der zweite Tiefpass hat, wie in der Gleichung 5.10 gezeigt, eine hohe Grenzfrequenz, weshalb für diese Messung der Ausschnitt vergrößert werden musste. So erkennt man bei dieser Messung wiederum einen saubereren Spannungsverlauf (gelbes Signal), nach dem das hochfrequente Rauschen rausgefiltert wurde (Abbildung 35).

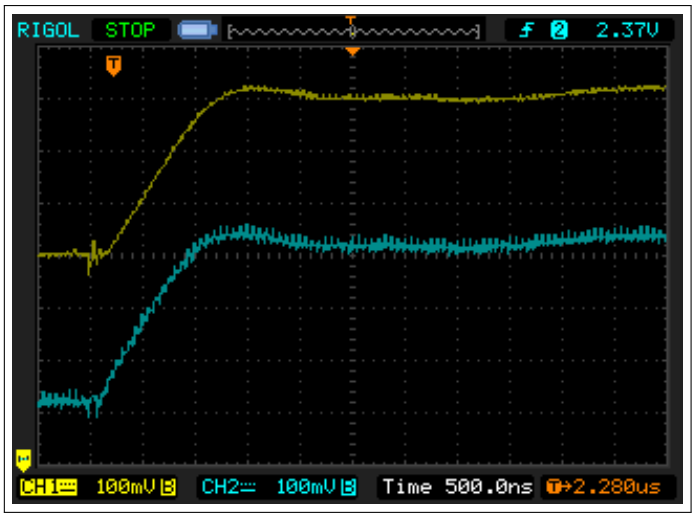


Abbildung 35: Frequenzgang nach zweitem Tiefpassfilter

Die Messungen zeigen, dass sowohl die Verstärkerschaltungen als auch Tiefpassfilter wie benötigt arbeiten und so die Messung der Stromstärken von Schrittmotoren ermöglichen.

5.2 Entwurf der Platine

Zuletzt wurde die Platine mit einer *Electronic Design Automation* (EDA) Software entworfen. Hierfür wurde das Programm *Eagle*[18] verwendet, welches die Herstellung einer solchen Leiterplatine vereinfacht. Zunächst wird dafür ein Schaltplan erstellt (Abbildung 36), indem alle Bauteile entsprechend elektrisch verbunden werden. So erkennt man die Erzeugung der Versorgungsspannung für den Mikrocontroller und den *L298* Baustein in dem oberen linken Bereich. Es wurde ein *7805* Spannungsregler verwendet, welcher aus einer Versorgungsspannungen von bis zu 35 V eine positive Spannung von 5 V erzeugt. Der Elektrolytkondensator bzw. die Kondensatoren dienen zur Unterdrückung von Schwingungen und zur Glättung der Ausgangsspannung. Die Darstellung von 12 V am Eingang dieses Spannungsreglers dient nur zur Orientierung, es sind auch höhere Spannungen möglich.

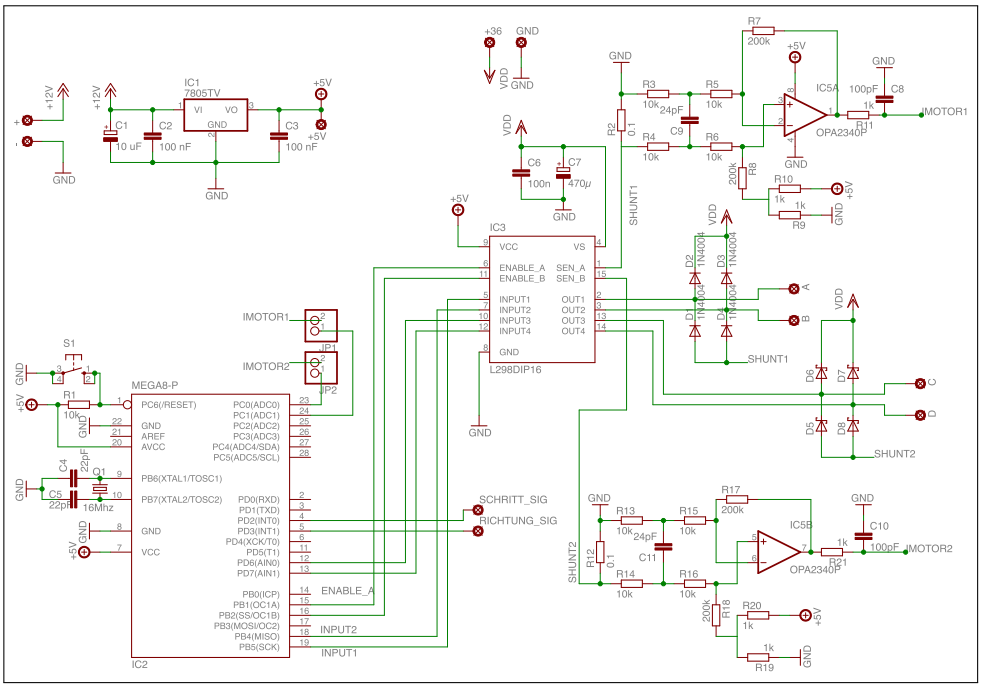


Abbildung 36: Schaltplan in Eagle

Der Mikrocontroller *ATMega8* ist unten links zu finden. Um eine Taktfrequenz von 16 MHz zu erhalten, wird ein sogenannter *Schwingquarz* verwendet, da der interne Oszillator zu langsam ist. Auch wurde ein Taster zum Neustarten (*Reset*-Funktion) eingebaut. Die Verbindungen zwischen den verstärkten Messspannungen, welche die Stromstärke der Spulen angibt, und den AD-Wandler Eingängen wurde mit Stiftleisten getrennt. Somit können

[18] EDA-Programm der Firma CadSoft (`http://www.cadsoft.de`).

Kurzschlussbrücken (englisch *Jumper*) verwendet werden, womit ein Abgreifen der Spannung mit einem Oszilloskop vereinfacht wird (*JP1* und *JP2*).

Ebenfalls wurden die Anschlüsse für das Schritt- und Richtungssignal durch Lötanschlüsse ausgeführt. Diese können somit direkt auf der Platine verlötet oder durch Anschlüsse im 5 mm Rastermaß verbunden werden.

Die Verstärker- und Filterschaltungen der jeweiligen Spulen sind rechts oben bzw. unten zu finden (*SHUNT1* und *SHUNT2*). Die Ausgänge dieser werden durch definierte Leitungsnamen mit den AD-Wandler-Eingängen verbunden (*IMOTOR1* und *IMOTOR2*). Zu beachten ist, dass die Anode der Freilaufdioden (rechts mittig) nicht an die Masse der Spannungsversorgung, sondern vor den Shunts angeschlossen wird. Somit können die negativen Ströme auch Spannungen am Shunt erzeugen, welche dann auch messbar sind. Auch die Ausgänge für die Anschlüsse des Schrittmotors sind einfache Lötanschlüsse im 5 mm Rastermaß.

Im Anschluß an die Herstellung des Schaltplans wurde die Platine geplant. Hierbei wurden die Bauteile passend platziert und mit Leiterbahnen verbunden. Die Schwierigkeit hierin besteht in dem sogenannten *Routing*, also der *Leiterplattenentflechtung*. So sollten die Leiterbahnen möglichst effizient und ohne viele Lötbrücken[19] verlegt werden. Da die Platine mit einfachen Mitteln herzustellen sein soll, wurde eine einseitige Platine verwendet. Dies hat zur Folge, dass drei Lötbrücken nötig waren, welche mittels Lötpunkten und Drähten verbunden wurden. Die Abstände der Widerstände und Kondensatoren um den Operationsverstärker sind möglichst kurz gehalten, um mögliche negative Auswirkungen langer Leiterbahnen zu vermeiden. Der *L298* wurde über eine Adapterplatine angeschlossen. Dies erleichterte den Austausch bei Defekt, vereinfachte aber auch das Routing, da mehr Platz zwischen den Anschlüssen vorhanden war. Die Breite der Leiterbahnen, durch welche hohe Ströme fließen, also zum Beispiel, jene die vom Treiberbaustein zu den Spulen führen, wurden mit einer Breite von ungefähr 1.68 mm versehen. Dies hat eine Verringerung des Leiterbahnwiderstands zur Folge und mindert damit die Erwärmung bei hohen Strömen.

Wegen der erhöhten Schwierigkeit des Routings einseitiger Platinen wurde eine Massefläche angelegt, statt jeweils Leiterbahnen zu den Masseanschlüssen zu legen. Dabei muss darauf geachtet werden, dass keine abgeschnittenen Flächen entstehen und somit die Masse bei Bauteilen fehlt. In Abbildung 37 sind die Anordnungen der Bauteile und die Leiterbahnen der Platinenunterseite zu sehen. Die Lötanschlüsse für die Lötbrücken sind an den großen grünen Flächen zu erkennen, welche das Anlöten geeigneter Drähte vereinfacht (jeweils rechts und unten mittig). Zusätzlich wurde ein Lötanschluss für die 5 V Spannungsversorgung eingefügt. Somit kann, falls bereits eine 5 V Spannungsquelle vorhanden ist, die Verwendung eines Spannungsreglers umgangen werden.

[19] Lötbrücken verbinden unterbrochene Leiterbahnen miteinander.

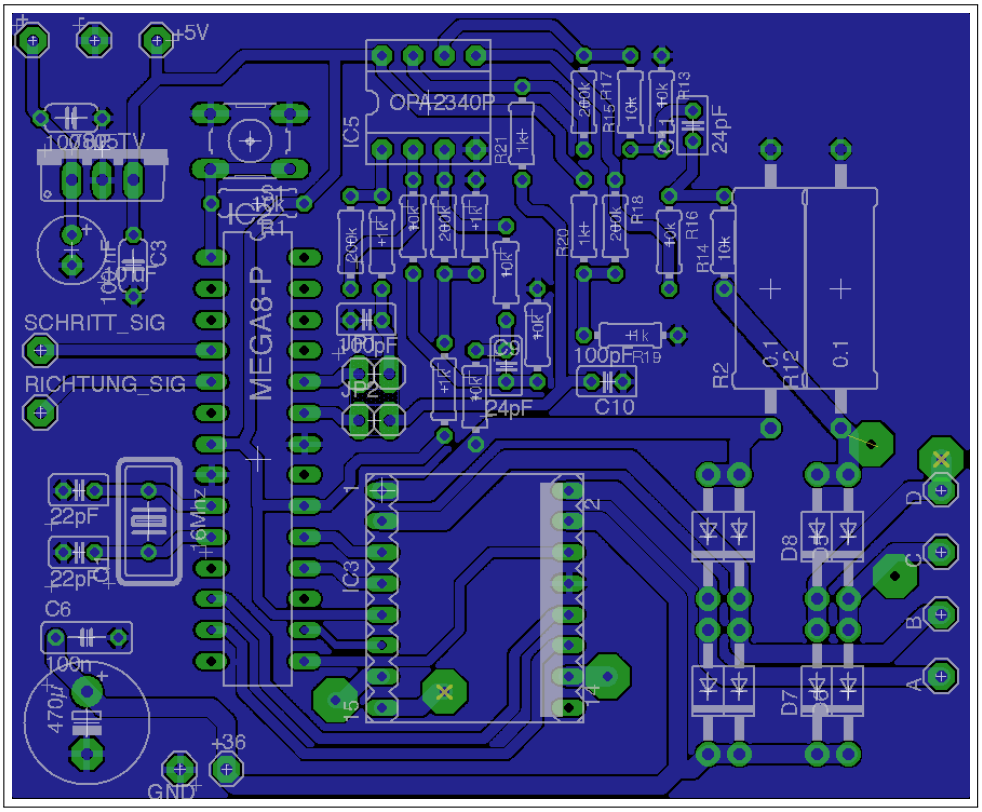


Abbildung 37: Platinenentwurf des Schrittmotortreibers

5.3 Auswertung

Bei der Realisierung wurde, wie schon erwähnt, ein Schrittmotor mit einem Widerstand von 82.5 Ω und einer Induktivität von 0.205 H verwendet. Die folgenden Messungen und Auswertungen basieren alle auf diesem Motor.

Da ein Mikrocontroller der AVR Serie keine Möglichkeit des Debuggings bietet, wurde zur Darstellung des Regelvorgangs ein vereinfachter Aufbau realisiert. Hierzu wurde nur eine Spule des Motors angeschlossen und dementsprechend auch nur ein PI-Regler aktiviert. Die angestrebte Stromstärke betrug ungefähr 0.23 A, was für den AD-Wandler einen Wert von 559 darstellt[20]. Darüber hinaus wurde die Funktion *CalcPI1* (siehe Listing 7) modifiziert, sodass diese sowohl die gemessene Spannung des AD-Wandlers als auch die berechnete PI-Regler Summe (*PI_sum1*) über die serielle Schnittstelle *UART*[21] ausgibt (siehe Listing 15).

```
#define DEBUG 1          //Senden der Werte aktivieren
char buffer[10];         //Buffer zum Senden von Strings

extern inline int16_t CalcPI1(uint16_t currADC, int16_t currRef)
{
    [...]
#if DEBUG == 1                          //Sende Werte, falls Debug aktiviert
    //ADC Messwerte ausgeben
    while (!(UCSR0A & (1 << UDRE0)))    //auf UART warten
        ;
    UDR0 = currADC >> 8;         //obere 8 Bit senden
    while (!(UCSR0A & (1 << UDRE0)))    //auf UART warten
        ;
    UDR0 = currADC & (0xFF);     //untere 8 Bit senden
    while (!(UCSR0A & (1 << UDRE0)))    //auf UART warten
        ;
    UDR0 = ',';                         //Komma senden
    itoa(PI_sum1, buffer, 10);          //PI Summe zu String wandeln
    for (uint8_t i = 0; buffer[i] != 0; i++)    //durch Buffer iterieren
    {
        while (!(UCSR0A & (1 << UDRE0)))//auf UART warten
            ;
        UDR0 = buffer[i];               //Zeichen aus String senden
    }
    while (!(UCSR0A & (1 << UDRE0)))    //auf UART warten
        ;
    UDR0 = '\n';                        //Zeilenumbruch senden
#endif
    [...]
}
```

Listing 15: Ausgabe des Messwertes und der Regler Summe

[20] $559 \cdot \frac{1024}{5\,\mathrm{V}} = 2.729\,\mathrm{V}$ - Dies entspricht, abzüglich des Verstärkeroffsets von 2.5 V einer Stromstärke von 0.229 A

[21] Die Konfiguration des UART findet sich im Anhang.

Da der verwendete Mikrocontroller ein 8 Bit breites Senderegister besitzt, muss die 16-Bit Variable *currADC* (Listing 15 Zeile 11 und 14) in zwei Paketen gesendet werden. Darüber hinaus wird mit der Funktion *itoa()* (Zeile 18) die vorzeichenbehaftete 16 Bit breite Variable *PI_sum1* zu einem String gewandelt (Zeile 18). Somit wird beim Empfang direkt ersichtlich, ob diese Zahl negativ oder positiv ist. Das Senden eines Kommas (Zeile 17) dient zur Trennung der Zahlen, ebenso wie der Zeilenumbruch (Zeile 27) zur Trennung der Abtastungen.

Die Messwerte werden am PC empfangen und mithilfe eines Scripts in das CSV Dateiformat gespeichert. Diese können nun graphisch dargestellt werden. In Abbildung 38 wurden die beiden Messwerte als Graphen gegenüber gestellt (oben: AD-Wandler Werte, unten: PI-Regler Summe). Man erkennt, dass der Regler zu Beginn die höchste Stellgröße berechnet, welche jedoch auf einen Wert von 255 begrenzt wird. Bevor der Sollwert erreicht ist, versucht der Regler sich zu stabilisieren und mindert die Stellgröße. Da bei dem Versuch ein Motor mit hoher Induktivität gewählt wurde, steigt der Strom entpsrechend langsam an. Trotzdem erreicht der Regler die für die gewählte Stromstärke passende Stellgröße recht schnell und schwankt nur leicht zu Beginn um diesen Wert. Dieser liegt laut Abbildung 38 knapp über 200. Im weiteren Verlauf stabilisiert sich der Regler.

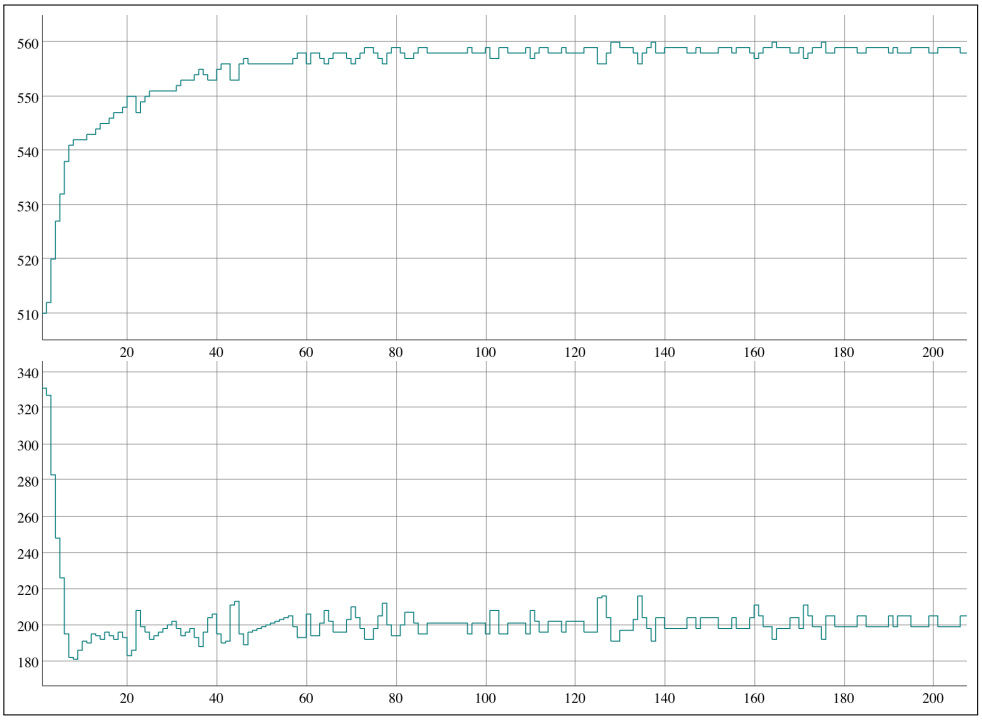


Abbildung 38: AD-Wandler Werte (oben) und Summe des PI-Reglers (unten)

Des Weiteren wird kein Überschwingen der Stromstärke erzeugt. Dies und der doch langsame Anstieg der Stromstärke zeigen, dass der verwendete Verstärkungsfaktor des PI-Reglers noch höher sein darf. Dies würde ein aggressiveres Regelverhalten bewirken, kann jedoch sowohl zum Überschwingen führen als auch ein Schwingen des Systems bewirken.

Im Folgenden wurde die zweite Spule des Schrittmotors angeschlossen und der $\frac{1}{8}$-Mikroschritt aktiviert. Das angelegte Schrittsignal hat eine Frequenz von 1000 Hz. Die zur Stromstärke proportionalen Spannungen an beiden AD-Wandler-Eingängen wurde mit einem Oszilloskop über einen Zeitraum von 60 ms festgehalten (Abbildung 39). So erkennt man die sinusförmigen Stromverläufe beider Spulen, welche eine Phasendifferenz von 90° aufweisen. Hiermit wird gleichzeitig das funktionierende Erzeugen des Mikroschritts und die erfolgreiche digitale Stromregelung belegt, was die Herausforderung dieser Bachelorarbeit darstellte.

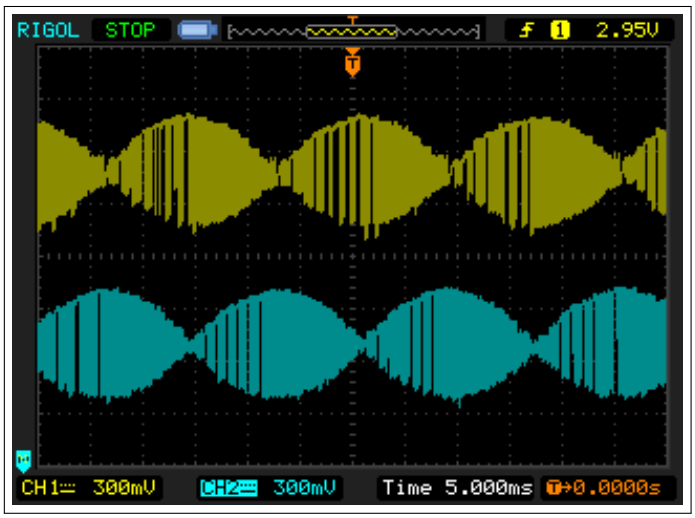


Abbildung 39: Stromverlauf beider Spulen bei $\frac{1}{8}$-Mikroschritt

Zur weiteren Auswertung wurde das Oszilloskop auf die *X-Y-Darstellung* eingestellt. Hierbei wird das Signal auf dem ersten Kanal als X-Wert verwendet, dass des zweiten Kanals als Y-Wert. Anhand der dabei entstandenen *Lissajous-Figur* erhält man Informationen über die Phasendifferenz der beiden Signale [AKA12]. So zeigt Abbildung 40 einen Kreis, was bedeutet, dass zwei Sinussignale mit einer Phasendifferenz von 90° anliegen. Die nahezu kreisförmige Figur gibt außerdem Auskunft darüber, dass die beiden Signale sich in Amplitude und Frequenz gleichen. Das *Rauschen* des Signals entsteht durch die Verwendung der Pulsweitenmodulation. Dennoch wird hiermit deutlich, wie genau die PI-Regler den Spulenstrom sinusförmig regeln können. Bei genauer Betrachtung lässt sich im oberen rechten Quadranten der $\frac{1}{8}$-Mikroschritt erkennen (acht Anhäufungen von Punkten).

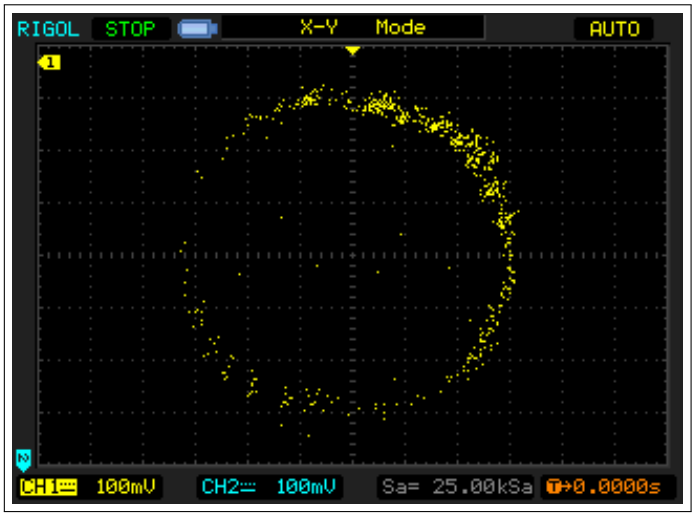


Abbildung 40: Lissajous-Figur bei $\frac{1}{8}$-Mikroschritt

Dennoch besitzt der hergestellte Schrittmotortreiber, abhängig vom Motor, Grenzen. So wurde die Schrittfrequenz im Folgenden auf 3333.33 Hz erhöht. Es lässt sich in Abbildung 41 nach wie vor ein sinusförmiger Stromverlauf erkennen. Jedoch schafft der PI-Regler es nicht, bedingt durch die hohe Induktivität, den Strom schnell genug zu erhöhen. Dies erkennt man am Ausbleiben der Pulsweitenmodulation beim Ansteigen der Sinuskurven. Auch die Amplitude der Stromstärke ist verringert, weshalb nicht das maximale Drehmoment garantiert werden kann. Man könnte dieses Problem jedoch mit einer höheren Betriebsspannung vermindern. Erhöht man die Schrittfrequenz weiter, so kann es passieren, dass der Spulenstrom selbst beim *Fast Decay* nicht schnell genug absinken kann. In diesem Fall bremst sich der Motor selbst aus, da der noch vorhandene Strom beim Wechsel der Polarität in die falsche Richtung fließt.

Abschließend wurde die automatische Absenkung des Stromes betrachtet. Das Oszilloskop befand sich im *Roll Modus*, in dem die gemessenen Spannungen konstant nach links geschoben und neue Messwerte von rechts hinzugefügt werden. Dieser Modus wurde deshalb genutzt, da das Signal nicht anders getriggert werden konnte. Zu sehen ist in Abbildung 42, dass die Stromstärke in beiden Spulen wie gewünscht auf ungefähr die Hälfte absinkt.

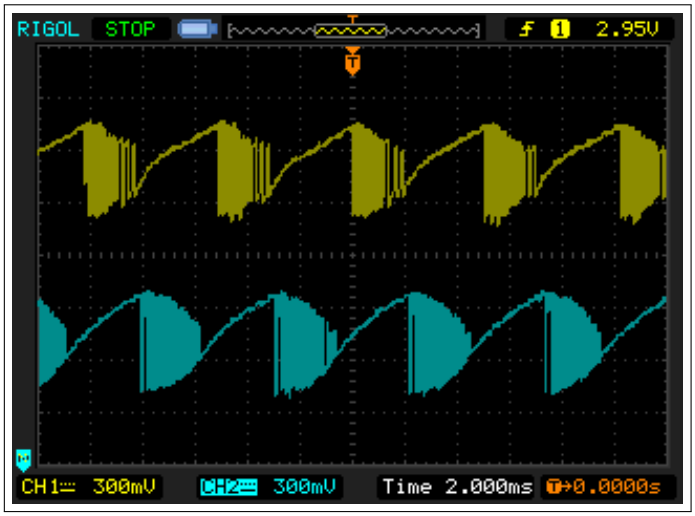


Abbildung 41: Stromverlauf bei $\frac{1}{8}$-Mikroschritt mit $\sim 3333.3\,\mathrm{Hz}$

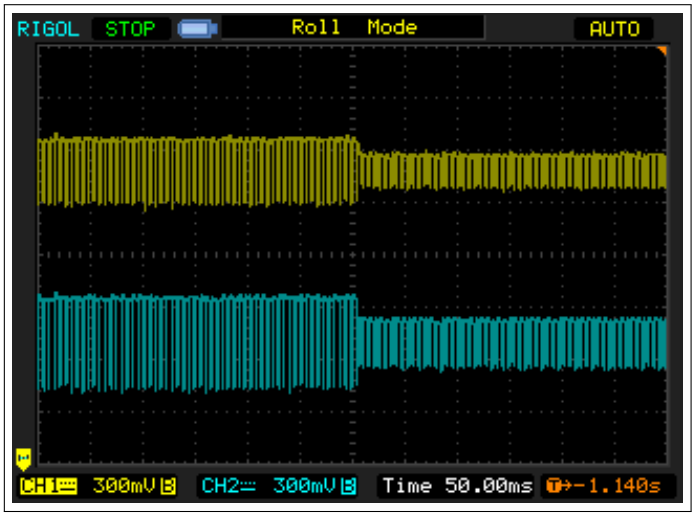


Abbildung 42: Stromverlauf mit aktiver automatischer Absenkung

6 Fazit und Ausblick

Im Fokus dieser Bachelorarbeit stand die Entwicklung einer Ansteuerung von Schrittmotoren, welche die grundlegenden Betriebsmodi wie Voll- und Halbschritt beherrschen, aber auch die Möglichkeit zur Verwendung von Mikroschritten bieten sollte. Letzteres erfordert eine einstellbare Limitierung des Stromes, weshalb hierzu ein in Software implementier PI-Regler entstand, welcher den Kern dieser Schrittmotorsteuerung darstellt. So kann deshalb auf einen höheren Anlaufstrom verzichtet werden, da durch die Regelung des Stromes und Verwendung eines vielfachen der Nennspannung des Schrittmotors, das maximale Drehmoment schnellst möglich erreicht wird.

Die Besonderheit bei der Verwendung eines digitalen Reglers besteht in der einfachen Anpassbarkeit an diverse Schrittmotoren. So können aktuelle Treiber wie der *TB6560AHQ* von Toshiba zwar auf verschiedene Schrittmotoren angepasst werden, jedoch muss hierfür eine Änderung an der Hardware vorgenommen werden. Experimentelle oder an Gegebenheiten angepasste Änderungen der maximalen Stromstärke sind somit mit erhöhten Aufwand verbunden. Im Gegensatz dazu kann bei dieser Realisierung die Stromstärke *on-the-fly*, also während des laufenden Programms, geändert werden. Darüber hinaus wird durch den theoretischen Ansatz eine optimale Regelung erreicht, da diese sich auf die physikalischen Konstanten des Motors anpasst.

So ist es mithilfe von theoretischen Grundlagen im Bereich Physik und Regelungstechnik im Rahmen dieser Bachelorarbeit gelungen, eine intelligente Ansteuerung mit den geforderten Funktionen zu realisieren.

Obwohl der gewählte Mikrocontroller einige Nachteile, wie zum Beispiel einen langsamen AD-Wandler oder die Limitierung von passenden Interrupts, aufweist, konnte eine genaue Stromregelung erreicht werden. Der damit angesteuerte Schrittmotor lässt sich somit in diversen Bereichen, wie zum Beispiel in einer CNC-Maschine, verwenden.

Jedoch werden durch die Verwendung des Mikrocontrollers und Treiberbausteins Limitierungen gesetzt. So kann durch Verwendung einer alternativen Architektur des Mikrocontrollers Verbesserungen erreicht werden, welche durch den langsamen AD-Wandler und die fehlenden Synchronisation der Pulsweitenmodulationen nicht möglich waren. Eine höhere Schrittfrequenz kann erst dann erreicht werden, wenn auch die PWM-Frequenz steigt, welche jedoch durch die Geschwindigkeit des Analog-Digital-Wandlers begrenz ist. Dieser Nachteil macht sich, neben der verminderten maximalen Schrittfrequenz, vor allem im Geräuschpegel bemerkbar, denn die verwendete Frequenz von 7812.5 Hz liegt im hörbaren Bereich und erzeugt ein deutlich hörbares Pfeifen. Auch die maximal zulässige Stromstärke des verwendeten Treiberbausteins setzt Grenzen in den Anwendungsbereichen. Hierzu könnte ein anderer Treiber verwendet werden, welcher eine höhere maximale Stromstärke zulässt. So gibt es bereits Treiber, welche Sperrdioden integriert haben und den *Slow Decay* durch MOSFETs unterstützen.

Durch Betrachtung der Nachteile wird die Flexibilität dieser Realisierung deutlich, da zum Beispiel ein Austausch des Treiberbausteins kaum Veränderungen der Software oder Hardware nötig macht.

Aber auch eine Umsetzung auf einem anderen Mikrocontroller, selbst mit anderer Architektur, benötigt nur geringe Änderungen der Software, welche wegen einer anderen Ansteuerung der Peripherie erforderlich sind. Denn der Kern dieses Schrittmotortreibers besteht im grundlegenden Verfahren der Regelung und dem digitalen Regler.

Literatur

[AKA12] Al-Khazali, H ; Askari, M: Geometrical and graphical representations analysis of lissajous figures in rotor dynamic system. In: *IOSR J. of Engineering* 2 (2012), Nr. 5, S. 971–978

[AS06] APD Semiconductor, Applications D.: *AN-1012: Reverse Recovery Time (T_{rr}) of the Super Barrier Rectifier™*. http://www.diodes.com/pdfs/apd/AN-1012_TRRComparison_v0.2.pdf, 2006

[BB09] Bakshi, Uday A. ; Bakshi, Varsha U.: *Basic Electrical Engineering Seiten 8-24*. Technical Publications, 2009

[Beu11] Beucher, Ottmar: *Übungsbuch Signale und Systeme*. Springer Berlin Heidelberg, 2011

[BSD+] Basnet, Nesh ; Shrestha, Subash ; Dhondup, Lhakpa ; Lekshey, Lobsang ; Macuff, Shiraz: *Introduction to Stepper Motors*. http://www.iaasr.com/introduction-to-stepper-motors/, . – [Online; Stand 23. November 2014]

[Büt09] Büttner, Wolf-Ewald: *Grundlagen der Elektrotechnik 2*. Oldenbourg Verlag München, 2009

[CJ04] Condit, Reston ; Jones, Douglas W.: *Stepping Motors Fundamentals*. http://homepage.cs.uiowa.edu/~jones/step/an907a.pdf, 2004

[Cor13] Corporation, Atmel: *ATmega8*. http://www.atmel.com/Images/Atmel-2486-8-bit-AVR-microcontroller-ATmega8_L_datasheet.pdf, 2013

[Dal13] Dale, Vishay: *Metal Film Resistors, Industrial Power, Precision, Flameproof*. http://www.vishay.com/docs/31021/cpf.pdf, 9 2013

[Dem13] Demtröder, Wolfgang: *Experimentalphysik 2 Elektrizität und Optik*. Springer Spektrum, 2013

[Err95a] Erricsson Components AB: *Industrial Circuits Data Book and Stepper Motor Control Handbook*. http://users.ece.utexas.edu/~valvano/Datasheets/StepperMicrostep.pdf http://library.solarbotics.net/pdflib/pdf/drive.pdf http://library.solarbotics.net/pdflib/pdf/motorbas.pdf, 1995

[Err95b] Erricsson Components AB: *Industrial Circuits Data Book and Stepper Motor Control Handbook*. http://library.solarbotics.net/pdflib/pdf/motorbas.pdf, 1995

[Err95c] Erricsson Components AB: *Industrial Circuits Data Book and Stepper Motor Control Handbook.* http://library.solarbotics.net/pdflib/pdf/drive.pdf, 1995

[Fas05] Fasching, Gerhard M.: *Werkstoffe für die Elektrotechnik: Mikrophysik, Struktur, Eigenschaften.* Springer-Verlag, 2005

[Fed12] Federau, Joachim: *Operationsverstärker.* Springer Vieweg, 2012

[GF14] Gühmann, Clemens ; Funck, Jürgen: *Grundlagen der elektronischen Messtechnik - Praktikum.* https://www.mdt.tu-berlin.de/fileadmin/fg184/Lehre/Messtechnik/Praktikum/LaborskriptMDT.pdf, 2014

[GHS93] Gausch, Felix ; Hofer, Anton ; Schlacher, Kurt: *Digitale Regelkreise: Ein einfacher Einstieg mit dem Programm uLINSY.* Oldenbourg Wissenschaftsverlag, 1993

[Hop12] Hopkins, Thomas: *Reducing torque variation in half step drive.* http://www.st.com/st-web-ui/static/active/en/resource/technical/document/design_tip/DM00053453.pdf, 2012

[Ins07] Instruments, Texas: *Datenblatt OPA2340.* http://www.ti.com.cn/cn/lit/ds/symlink/opa340.pdf, 2007

[KEQ+94] Kallenbach, Eberhard ; Eick, Rüdiger ; Quendt, Peer ; Ströhla, Tom ; Feindt, Karsten ; Kallenbach, Matthias: *Elektromagnete.* Teubner Stuttgart, 1994

[Lab15] Labs, Open M.: *ATmega ADC.* http://www.openmusiclabs.com/learning/digital/atmega-adc/, 2015

[Lep03] Lepkowski, Jim: *Motor Control Sensor Feedback Circuits.* http://ww1.microchip.com/downloads/en/AppNotes/00894a.pdf, 2003

[Lun08] Lunze, Jan: *Regelungstechnik 1: Systemtheoretische Grundlagen, Analyse und Entwurf einschleifiger Regelungen.* Springer Berlin Heidelberg, 2008

[Man09] Manea, Sorin: *Stepper Motor Control with dsPIC® DSCs.* http://ww1.microchip.com/downloads/en/AppNotes/1307A.pdf, 2009

[Mic] Microchip Technology Inc.: *AN1307 Tuning Procedure ReadMe.* http://ww1.microchip.com/downloads/en/AppNotes/AN1307%20Tuning%20Procedure%20ReadMe.pdf, 2009

[Mic01] MicroSystems, Allegro: *A New Microstepping Motor-Driver IC with Integrated Step and Direction Translator Interface*. http://www.allegromicro.com/~/media/Files/Technical-Documents/stp01-2-Microstepping-Motor-Driver-IC.ashx?la=en, 2001

[MW01] Maske, Rudolph J. ; Woods, David M.: *Use of digital current ramping to reduce audible noise in stepper motor*. März 27 2001. – US Patent 6,208,107

[Ost11a] Ostermann, Thorsten: *Ansteuertechniken*. http://www.ostermann-net.de/electronic/schritt/sm_ansteuer.htm, 2011. – [Online; accessed 28-March-2015]

[Ost11b] Ostermann, Thorsten: *Unipolar oder Bipolar?* http://www.schrittmotor-blog.de/unipolar-oder-bipolar/, 2011. – [Online; Stand 23. March 2014]

[Ott08] Ottens, Manfred: *Grundlagen der Systemtheorie*. https://prof.beuth-hochschule.de/fileadmin/user/ottens/Skripte/Grundlagen_der_Systemtheorie.pdf, 2008

[Pap09] Papula, Lothar: *Mathematische Formelsammlung für Ingenieure und Naturwissenschaftler*. Vieweg+Teubner, 2009

[Pie07] Pierce, Eric: *Stepper Motor Illustration*. https://en.wikipedia.org/wiki/User:Wapcaplet, 2007

[Pry04] Prymak, John: *Ripple Current Capabilities*. http://www.newark.com/pdfs/techarticles/kemet/Ripple-Current-Capabilities-Technical-Update.pdf, 2004

[Rum07] Rummich, Erich: *Elektrische Schrittmotoren und -antriebe: Funktionsprinzip, Betriebseigenschaften, Meßtechnik*. Bd. 365. expert verlag, 2007

[Sch07] Schlienz, Ulrich: Leistungsschalter. In: *Schaltnetzteile und ihre Peripherie*. Vieweg, 2007, S. 127–142

[Sch10] Schon, Klaus: Übertragungsverhalten linearer Systeme und Faltung. In: *Stoßspannungs- und Stoßstrommesstechnik*. Springer Berlin Heidelberg, 2010, S. 57

[Sch11] Schaedel, Prof. Dr.-Ing. H. M.: *Entwurf von PI- und PID - Reglern nach dem Kriterium der gestuften Dämpfung*. https://www.f07.th-koeln.de/imperia/md/content/personen/lemke_monica_sanda/rt_skript.pdf, 2011

[SP10] Steffen, Paul ; Paul, Reinhold: *Grundlagen der Elektrotechnik und Elektronik 1*. Springer Berlin Heidelberg, 2010

[Ste14] Norm DIN IEC 60050-351 (IEV Nummer 351-47-02) 09 2014. *Arten der Regelung und Steuerung*

[STM00] STMicroelectronics: *L298*. http://www.st.com/web/en/resource/technical/document/datasheet/CD00000240.pdf, 2000

[STM01] STMicroelectronics: *L297*. http://www.st.com/web/en/resource/technical/document/datasheet/CD00000063.pdf, 2001

[Tas] Norm DIN EN 60469:2014-02 . *Übergänge, Impulse und zugehörige Schwingungsabbilder – Begriffe, Definitionen und Algorithmen*

[TM] The MathWorks, Inc.: *Control System Toolbox*. http://de.mathworks.com/products/control/,

Abbildungsverzeichnis

Anhang A Programmcode

```
#define F_CPU 16000000UL

#include <stdlib.h>
#include <stdio.h>
#include <avr/io.h>
#include <avr/interrupt.h>

#include "pid.h"
#include "config.h"

//globale Variablen
int16_t PI_sum1 = 0;    //Definition mit Startwert
int16_t PI_sum2 = 0;    //Definition mit Startwert
int16_t lastError1 = 0; //Definition mit Startwert
int16_t lastError2 = 0; //Definition mit Startwert
int16_t PI1_out = 0;    //Definition mit Startwert
int16_t PI2_out = 0;    //Definition mit Startwert

int16_t currentA = 0;   //Stromlimit Spule 1
int16_t currentB = 0;   //Stromlimit Spule 2

uint8_t stepIndex = 0;  //Schrittindex
int16_t lutMotor[16];   //globale LUT, auf Motor angepasst

uint16_t reduceCounter = 0; //Definition mit Startwert
//globale Variablen

void USART_Init();

int main()
{
    cli();  //globale Interrupts deaktivieren
#if DEBUG == 1
    USART_Init();
#endif

    //PWM Konfiguration
    DDRB |=  (1 << PB1) | (1 << PB2);   //Pin der PWM als Ausgang definieren
    TCCR1A = (1 << WGM10) | (1 << COM1A1) | (1 << COM1B1);  //Fast PWM 8-Bit
    ,nicht Invertierter Modus
    TCCR1B = (1 << WGM12) | (1 << CS11);    //Prescaler auf 8 setzen
    TIMSK = (1 << TOIE1);   //PWM Interrupt aktivieren

    //AD-Wandler Konfiguration
    ADMUX = (1 << REFS0) | (1 << MUX0); //Referenzspannung
    ADCSRA = (1 << ADPS2) | (1 << ADEN); //Frequenzteiler & ADC Aktivierung
    ADCSRA |= (1 << ADSC); //Initialisierungswandlung starten
    while (ADCSRA & (1 << ADSC))
        ; //Aktives Warten

    //Schrittsignal - externer Interrupt aktivieren
```

```
51 #if RISINGEDGE == 1 //steigende Flanke
52         MCUCR |= (1 << ISC01) | (1 << ISC00);
53 #else   //fallende Flanke
54         MCUCR |= (1 << ISC01);
55 #endif
56 
57     GICR |= (1 << INT0);    //Externen Interrupt aktivieren
58 
59     //Richtungssignal
60     DDRD &= ~(1 << PD3); //Pin D3 als Eingang definieren
61 
62     //Ausgaenge definieren fuer L298 Input
63     DDRB |= (1 << PB4) | (1 << PB5);    //Spule A
64     DDRD |= (1 << PD6) | (1 << PD7);    //Spule B
65 
66 
67     //Berechnung der Stromstaerken
68     int16_t curr = (MAXCURRENT * 1024 / 5) + 512; //Max. Strom von 512 bis
       1024
69 #if (STEPMODE == 1) //Mikroschritt
70     float lutSine[16] =
71     { 0, 0.2, 0.38, 0.56, 0.71, 0.83, 0.92, 0.98, 1, 0.98, 0.92, 0.83, 0.71,
        0.56, 0.38, 0.2 }; //Diskrete Sinuswerte 0-180 Grad
72 
73     for (int i = 0; i < 16; i++)
74         lutMotor[i] = (int16_t) (lutSine[i] * MAXCURRENT * 1024 / 5) + 512;
75 #elif (STEPMODE == 2 || STEPMODE == 4)  //Halbschritt und Wavedrive
76     lutMotor[4] = lutMotor[8] = lutMotor[12] = curr;
77 #elif (STEPMODE == 3)   //Vollschritt
78     lutMotor[0] = lutMotor[4] = lutMotor[8] = lutMotor[12] = curr;
79 #endif
80 
81     currentA = lutMotor[stepIndex & 15];
82     currentB = lutMotor[(stepIndex + 8) & 15];
83 
84     sei();
85     //globale Interrupts aktivieren
86 
87     while (1)
88         ;
89 }
90 
91 ISR(INT0_vect)  //externer Interrupt
92 {
93 #if (STEPMODE == 1)
94     if (PIND & (1 << PD3)) //falls Pin D3 High Pegel: Rechtsrotation
95         stepIndex += STEPRESOLUTION; //Schrittindex inkrementieren
96     else    //sonst Links-Rotation
97         stepIndex -= STEPRESOLUTION;
98 #elif (STEPMODE == 3 || STEPMODE == 4)  //Vollschritt und Wavedrive
99     if(PIND & (1 << PD3))   //falls Pin D3 High Pegel: Rechtsrotation
100         stepIndex += 8;    //Schrittindex inkrementieren
101     else    //sonst Links-Rotation
102         stepIndex -= 8;    //Schrittindex dekrementieren
```

```
#elif (STEPMODE == 2)   //Halbschritt
    if(PIND & (1 << PD3))   //falls Pin D3 High Pegel: Rechtsrotation
        stepIndex += 4;    //Schrittindex inkrementieren
    else    //sonst Links-Rotation
        stepIndex -= 4;    //Schrittindex dekrementieren
#endif

    currentA = lutMotor[stepIndex & 15];        //Strom Spule A (Sinus)
    currentB = lutMotor[(stepIndex + 8) & 15];  //Strom Spule B (Cosinus)

    //Polaritaet der Spulen
    if ((stepIndex & 0x10))
        PORTB = (PORTB | (1 << PB4)) & ~(1 << PB5);
    else
        PORTB = (PORTB | (1 << PB5)) & ~(1 << PB4);

    if (((stepIndex + 8) & 0x10))
        PORTD = (PORTD | (1 << PD7)) & ~(1 << PD6);
    else
        PORTD = (PORTD | (1 << PD6)) & ~(1 << PD7);

    reduceCounter = 1;      //Aktivieren/Zuruecksetzen des Zaehlers
}

ISR(TIMER1_OVF_vect)    //PWM Interrupt
{
    ADCSRA |= (1 << ADSC); //ADC-Messung starten Spule 1
    while (ADCSRA & (1 << ADSC))
        ; //Auf ADC warten ~13us
    uint16_t cur1 = ADCW; //Messwert speichern

    ADMUX &= ~(1 << MUX0); //Setzen des Multiplexers auf Eingang 1

    ADCSRA |= (1 << ADSC); //ADC-Messung starten Spule 2
    while (ADCSRA & (1 << ADSC))
        ;
    uint16_t cur2 = ADCW;

    ADMUX |= (1 << MUX0); //Setzen des Multiplexers auf Eingang 2

    OCR1A = CalcPI1(cur1, currentA); //Berechnung des Reglers
    OCR1B = CalcPI2(cur2, currentB); //Berechnung des Reglers

#if REDUCE_CURRENT == 1
    if (reduceCounter != 0)
    {
        reduceCounter++;
        if (reduceCounter > REDUCE_TIME)
        {
            reduceCounter = 0;
            currentA = ((currentA - 512) >> 1) + 512; //Spule 1 auf 50%
            currentB = ((currentB - 512) >> 1) + 512; //Spule 2 auf 50%
        }
    }
```

```
157 #endif
158 }
159
160 void USART_Init()   //Serielle Schnittstelle aktivieren
161 {
162     UBRRH = 0x00; //bei 16Mhz Taktrate Baudrate = 1.000.000
163     UBRRL = 0x00;
164
165     UCSRC = 0b10000110; //8 Datenbits
166     UCSRB = (1 << TXEN); //Transmitter aktivieren
167 }
```

Listing 16: main.c

```
1  #ifndef CONSTANTS_H_
2  #define CONSTANTS_H_
3
4  #include <stdio.h>
5
6  //Konfiguration
7  #define MAXCURRENT 0.43     //max. Strom des Motors nach Datenblatt in
       Ampere
8  #define STEPRESOLUTION  1   //Mikroschrittaufloesung: 1 -> 1/8 | 2 -> 1/4
9  #define STEPMODE        1   //Betriebsarten
10                             //Mikroschritt: 1
11                             //Halbschritt:  2
12                             //Vollschritt:  3
13                             //Wavedrive:    4
14
15 #define REDUCE_CURRENT 1    //bei Stillstand Stromreduzierung, 0 = disable
16 #define REDUCE_TIME 7813    //Interrupts bis Stromreduzierung, 1 = 128us
17
18 #define RISINGEDGE 1        //Schrittsignal bei steigender Flanke
19
20 #define MOTOR_L 0.205       //Motor Induktivitaet
21 #define MOTOR_R 82.5        //Motor Widerstand
22
23 #define DC 30               //Versorgungsspannung
24
25 #define DEBUG  0            //Werte ausgeben ueber UART
26
27 //Interne Berechnungen
28 #define SAMPLE_TIME 0.000128    //Abtastzeit, Reziproke PWM Frequenz
29
30 #define RISE_TIME MOTOR_L/MOTOR_R   //Anstiegszeit
31
32 #define PI_GAIN 3*MOTOR_R/(RISE_TIME * DC)          //Verstaerkungsfaktor
33 //Konstanten Ka und Kb
34 #define PI1 (int32_t)(PI_GAIN*(MOTOR_L/MOTOR_R + SAMPLE_TIME/2) * 16) *
       255*5
35 #define PI2 (int32_t)(PI_GAIN*(MOTOR_L/MOTOR_R - SAMPLE_TIME/2) * 16) *
       255*5
36
37 #define MIN_PWM 4               //entspricht ca. 2 Mikrosekunden
```

```
38
39 //Deklarationen
40 extern int16_t PI_sum1;     //Ergebnis der Differenzengl. Regler 1
41 extern int16_t PI_sum2;     //Ergebnis der Differenzengl. Regler 2
42
43 extern int16_t lastError1;  //Abweichung zurueckliegende Messung Regler 1
44 extern int16_t lastError2;  //Abweichung zurueckliegende Messung Regler 2
45
46 extern int16_t PI1_out;     //Ausgabe des PI-Reglers 1
47 extern int16_t PI2_out;     //Ausgabe des PI-Reglers 2
48
49 #endif /* CONSTANTS_H_ */
```

Listing 17: config.h

```
1 #ifndef PID_H_
2 #define PID_H_
3
4 extern int16_t CalcPI1(uint16_t currADC, int16_t currRef)
5 extern int16_t CalcPI2(uint16_t currADC, int16_t currRef)
6
7 #endif /* PID_H_ */
```

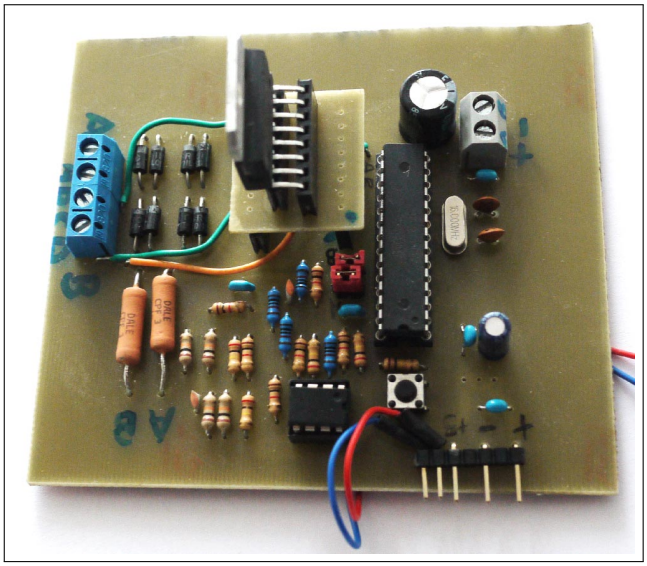
Listing 18: pid.h

```
1  #include "config.h"
2  #include <avr/io.h>
3  #include <stdlib.h>
4
5  char buffer2[10];
6
7  extern inline int16_t CalcPI1(uint16_t currADC, int16_t currRef)
8  {
9      //Berechnung der Differenz des Spulenstromes (Regelabweichung)
10     int16_t currentError = currRef - (int16_t) currADC;
11
12     //temporaere Variable
13     int32_t temp_PI_sum = (PI1 * (int32_t) currentError - PI2 * (int32_t)
       lastError1) >> 14;
14
15     //Differenzengleichung
16     PI_sum1 = PI_sum1 + (int16_t)temp_PI_sum - ((PI_sum1 - PI1_out) >> 4);
17
18     //Speichern der zurueckliegenden Messung
19     lastError1 = currentError;
20
21 #if DEBUG == 1                          //Sende Werte, falls Debug aktiviert
22     //ADC Messwerte ausgeben
23     while (!(UCSR0A & (1 << UDRE0)))    //auf UART warten
24         ;
25     UDR0 = currADC >> 8;        //obere 8 Bit senden
26     while (!(UCSR0A & (1 << UDRE0)))    //auf UART warten
27         ;
28     UDR0 = currADC & (0xFF);    //untere 8 Bit senden
```

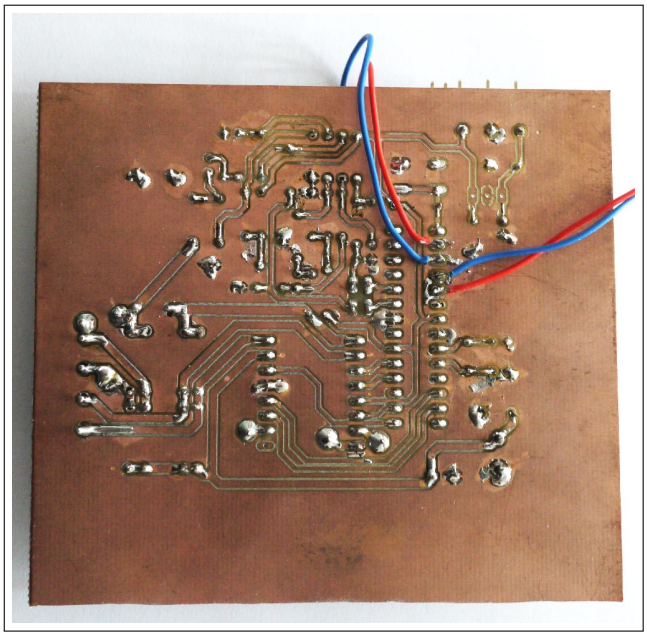
```
29      while (!(UCSR0A & (1 << UDRE0)))    //auf UART warten
30          ;
31      UDR0 = ',';                         //Komma senden
32      itoa(PI_sum1, buffer, 10);          //PI Summe zu String wandeln
33      for (uint8_t i = 0; buffer[i] != 0; i++)    //durch Buffer iterieren
34      {
35          while (!(UCSR0A & (1 << UDRE0)))//auf UART warten
36              ;
37          UDR0 = buffer[i];               //Zeichen aus String senden
38      }
39      while (!(UCSR0A & (1 << UDRE0)))    //auf UART warten
40          ;
41      UDR0 = '\n';                        //Zeilenumbruch senden
42  #endif
43
44      //Limitierung der PWM
45      if (PI_sum1 > 255)
46          PI1_out = 255;
47      else if (PI_sum1 < MIN_PWM)
48          PI1_out = MIN_PWM;
49      else
50          PI1_out = PI_sum1;
51
52      return PI1_out;
53  }
54
55
56  extern inline int16_t CalcPI2(uint16_t currADC, int16_t currRef)
57  {
58      //Berechnung der Differenz des Spulenstromes (Regelabweichung)
59      int16_t currentError = currRef - (int16_t) currADC;
60
61      //temporaere Variable
62      int32_t temp_PI_sum = (PI1 * (int32_t) currentError - PI2 * (int32_t)
        lastError2) >> 14;
63
64      //Differenzengleichung
65      PI_sum2 = PI_sum2 + (int16_t)temp_PI_sum - ((PI_sum2 - PI2_out) >> 4);
66
67      //Speichern der zurueckliegenden Messung
68      lastError2 = currentError;
69
70      //Limitierung der PWM
71      if (PI_sum2 > 255)
72          PI2_out = 255;
73      else if (PI_sum2 < MIN_PWM)
74          PI2_out = MIN_PWM;
75      else
76          PI2_out = PI_sum2;
77
78      return PI2_out;
79  }
```

Listing 19: pid.c

Anhang B Fotos der Platine



Oberseite der Platine. Die Anschlüsse für Schritt- und Richtungssignal wurden hier noch direkt an die Pins des Mikrocontrollers gelötet. Ein 5 V Spannungsregler wurde hier nicht verbaut.



Unterseite der Platine mit. Die 4 zusätzlichen Kabel sind für den UART und das Schritt- und Richtungssignal.