Hochschule
Augsburg University of
Applied Sciences

Fakultät für
Elektrotechnik

Bachelorarbeit

Elektrotechnik
Informations- und Kommunikationstechnik

Markus Müller
Mikrocontrollerbasierte Steuerung und Regelung eines Brushless DC Motors mit einem Atlas Digital Amplifier Steuermodul

Schwerpunktfach: Mikrocomputertechnik
Erstprüfer: Prof. Martin Bayer
Thema erhalten am: 18.04.2018

Verfasser der Bachelorarbeit
Markus Müller
Seitzstraße 5a
86154 Augsburg
Telefon 0157 581 69 179
Markus.mueller@hs-augsburg.de

Hochschule für angewandte
Wissenschaften Augsburg
University of Applied Sciences
An der Hochschule 1
D-86161 Augsburg
Telefon +49 821 55 86-0
Fax +49 821 55 86-3222
www.hs-augsburg.de
info@hs-augsburg.de

Kurzfassung

Das Ziel der Bachelorarbeit beim Deutschen Zentrum für Luft- und Raumfahrt (DLR) ist es, mit Hilfe des Mikrocontrollers STM32F726 Nucleo einen Brushless DC Motor über ein Atlas Digital Amplifier Steuermodul anzusteuern und zu regeln. Dabei soll der Atlas Digital Amplifier folgende Kriterien erfüllen: Der Mikrocontroller soll innerhalb einer Zykluszeit von 25 microsekunden dem Atlas Digital Amplifier einen Motorsteuerungsbefehl schicken und Informationen vom Atlas erhalten. Des Weiteren sollen die aktuellen Motorströme von dem Atlas Digital Amplifier ausgelesen werden und an den Mikrocontroller STM32F746 gesendet werden. Diese Informationen werden anschließend von dem Mikrocontroller verarbeitet und auf einem PC-Terminal dargestellt. Hauptsächlich soll untersucht werden, wie der Atlas Digital Amplifier funktioniert und wie mit ihm umgegangen werden muss. Besonders interessant ist die erreichbare Bandbreite bei der Kommandierung und dem Austausch der Sensordaten. Außerdem soll ein Regler entwickelt werden, der die Drehzahl des Motors reguliert. Dazu muss mit Hilfe eines Positionssensors die absolute Position bestimmt werden. Hierbei wird besonders auf das Verfahren zur Berechnung der Position und auf die bidirectional serial synchronous (BiSS) Schnittstelle des Sensors eingegangen.

Abstract

The aim of this bachelor thesis at the German Aerospace Center (DLR) is to operate and regulate a brushless DC motor with an Atlas Digital Amplifier based on the programming of a STM32F746 microcontroller. Therefore the Atlas Digital should meet the following criteria: The first criteria is that the microcontroller should send a command for motor operation to the Atlas Digital Amplifier and at the same time get some Information from it in a total cycle time of 25 microseconds. The second one is that the Atlas Digital Amplifier can measure the current of all phases of the motor and send them to the STM32F746. All the data from the Atlas Digital Amplifier has to be processed by the microcontroller and should be visualized on a computer display. The primary objective of this thesis is to analyze the operating principle of the Atlas Digital Amplifier and to examine how to work with it. Furthermore a PI Regulator should be developed to regulate the rotational speed of the brushless DC motor. In addition to that the absolute position of the motor has to be determined by a positioning sensor. Therefore the proceeding of the calculation of the position and the BiSS interface of the sensor is especially explained.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

BiSS	**Bidirectional Serial Synchronous**
CLK	**Clock**
CPHA	**Clock Phase**
CPOL	**Clock Polarity**
CS	**Chip Select**
DLR	**Deutsches Zentrum für Luft- und Raumfahrt**
FPGA	**Field Programmable Gate Array**
GPIO	**General Purpose Input Output**
GND	**Ground, Masse**
MISO	**Master In Slave Out**
MOSI	**Master Out Slave In**
NOP	**No Operation**
NVRAM	**Non-Volatile storage**
PWM	**Pulse-width modulation**
RAM	**Random-access memory**
SPI	**Serial Peripheral Interface**
UART	**Universal Asynchronous Receive Transmit**

1 Einleitung

Am Institut für Robotik und Mechatronik das Deutschen Zentrum für Luft- und Raumfahrt wird ein Rover entwickelt, der sowohl rollen als auch gehen können soll, um somit bisher unüberwindbare Hindernisse schneller und effektiver bezwingen zu können. Dies stellt die Entwickler vor neue Herausforderungen, insbesondere in der Regelung des Systems. Um eine effektive Regelung erzielen zu können, muss das entsprechende Regelungssystem möglichst nahe am zu regelnden Objekt angebracht werden. Dazu soll ein am DLR entwickeltes Motor Control Board zur Steuerung und Regelung der Motoren zu Einsatz kommen, das diese Eigenschaften erfüllen kann. Um dieses Board zu optimieren soll nun geprüft werden, ob der Atlas Digital Amplifier einen Teil ersetzen kann. In der Abbildung 1 ist der Aufbau des Motor Control Boards als Blockdiagramm dargestellt und der Teil, der durch den Atlas Digital Amplifier ersetzt werden soll, gekennzeichnet.

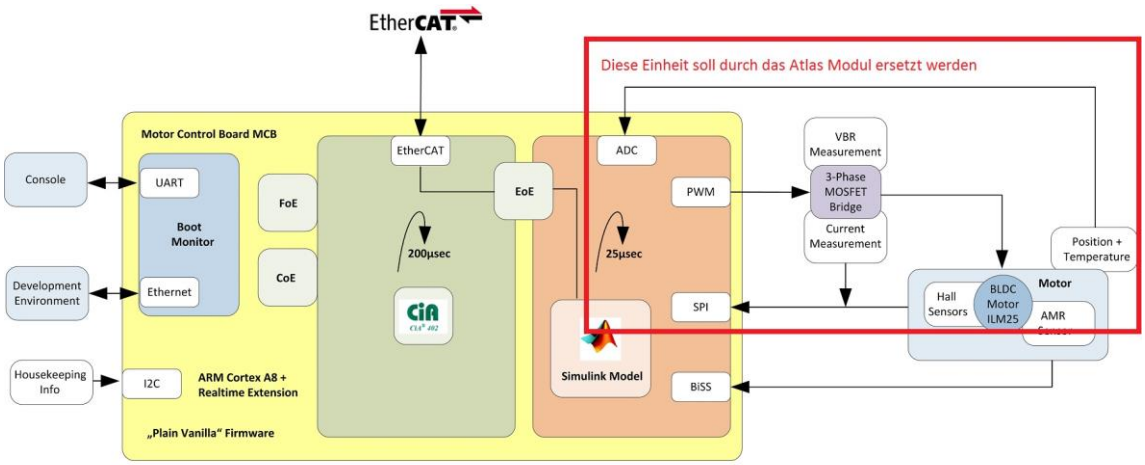


Abbildung 1: Blockdiagramm vom Motor Control Board der DLR
(Quelle: Reill Josef, Real-Time Development Interface Embedded in a Compact Motion Controller, S.2)

Mit der Bachelorarbeit soll daher überprüft werden, ob der Atlas Digital Amplifier für den Einsatz in der DLR geeignet ist. Dazu müssen gewisse Kriterien zur Zykluszeit und zur Strommessung genauer betrachtet werden.

Zu Beginn wird der Aufbau und die Funktionsweise des Brushless DC Motors erklärt und gezeigt, wie solch ein Motor anzusteuern ist. Anschließend wird in Kapitel 3 das Atlas Digital Amplifier Steuermodul genauer betrachtet und alles Wichtige zur Kommunikation mit dem Modul dargelegt. Dabei wird besonders auf das Datenprotokoll des Unternehmens Performance Motion Devices eingegangen und im Nachhinein ein Ausblick auf weitere Funktionen und Features des Atlas Digital Amplifier gegeben. In den nachfolgenden Kapiteln werden der Versuchsaufbau, die Vorgehensweise und die Erkenntnisse, die während der Arbeit bei der DLR gewonnen wurden, beschrieben. Dazu wird sowohl über die Anfertigung einer Adapterplatine für den STM32F746, als auch über die Programmierung desselbigen berichtet. Im abschließenden Fazit wird evaluiert, ob der Atlas Digital Amplifier für den Einsatz in der DLR geeignet ist.

2 Funktionsweise und Ansteuerung von Brushless DC Motoren

Ganz allgemein besteht der Aufbau eines Brushless DC Motor aus einem Stator und einem Rotor. Dabei bildet der Stator den äußeren Teil des Motors und wird aus mindestens drei Spulen mit Eisenkernen gebildet. Der Rotor sitzt in der Mitte des Motors und besteht aus mindestens einem Permanentmagneten. In Abbildung 2 wird der Aufbau eines Brushless DC Motors skizziert dargestellt.

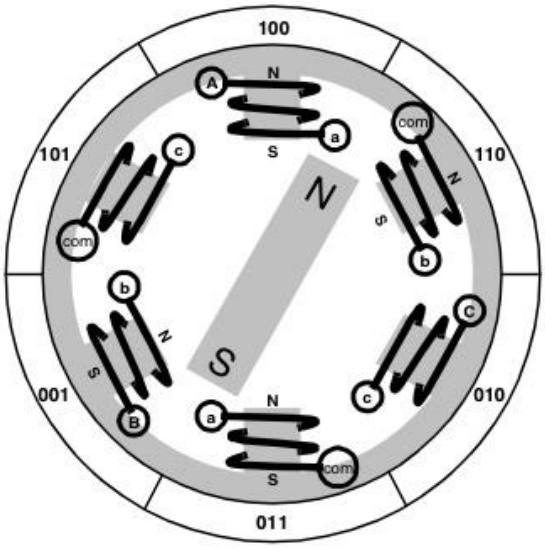


Abbildung 2: Skizzierter Aufbau eines Brushless DC Motors
(Quelle: http://bldc.wdfiles.com/local--files/bldc-and-8051/Three_phase_BLDC_internal_diagram.jpg)

Schließt man die Spulen des Stators nun an eine Stromquelle an, erzeugt jede ein eigenes Magnetfeld. Durch Überlagerung der einzelnen Felder entsteht in der Mitte des Motors ein Gesamtfeld. Je mehr Spulen an dem Motor angebracht sind und je geringer der Abstand der Spulen zueinander ist, desto homogener wird das Magnetfeld und desto stabiler wird die Rotationsbewegung. Die nächste Abbildung zeigt die Überlagerung der magnetischen Durchflutung aller Spulen am Stator.

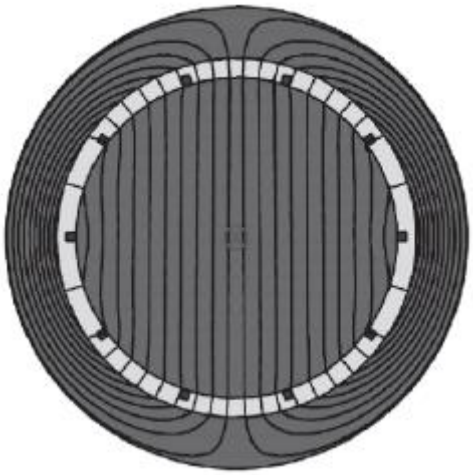
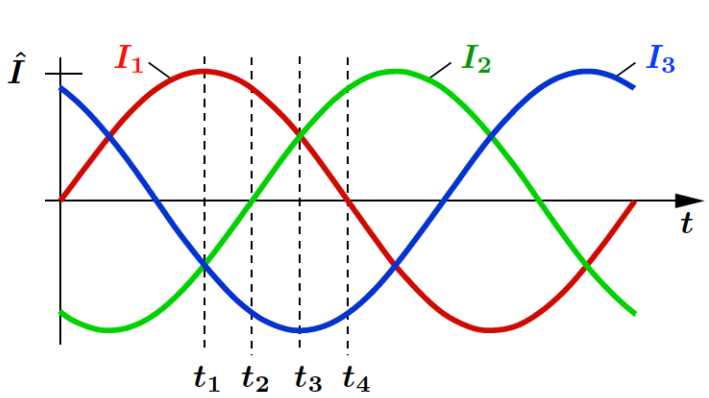


Abbildung 3: Darstellung des Magnetfeldes in einem Motor
(Quelle: Schröder Dierk, Elektrische Antriebe – Grundlagen, S.273)

Werden die Spulen mit einem sinusförmigen Strom gespeist, der jeweils um 120 Grad phasenverschoben ist, resultiert daraus, dass sich das Gesamtfeld mit konstanter Geschwindigkeit dreht. [1]
Dieses sogenannte Drehfeld ändert somit kontinuierlich seine Orientierung und zieht das Magnetfeld, welches durch den Permanentmagneten des Rotors erzeugt wird, durch den magnetischen Querdruck mit. Somit folgt der Rotor der Rotation des Drehfeldes und der Motor bewegt sich. [1]

Die Abbildung 4 zeigt den Zusammenhang zwischen dem Strom durch die einzelnen Spulen und der Drehung des dadurch erzeugten Magnetfeldes. Dabei sieht man, dass zum Zeitpunkt t_1 der Strom I_1 durch die Spule 1 maximal ist und die Ströme I_2 und I_3 jeweils nur halb so groß. Außerdem sind sie entgegengesetzt zum Strom I_1 gerichtet. Der Pfeil in den unteren Bildern stellt dabei die Ausrichtung des Drehfeldes dar. Ändert man nun den Strom durch alle Spulen, so beginnt das Magnetfeld sich zu drehen. Zum Zeitpunkt t_2 kann man erkennen, dass die Ströme I_1 und I_2 abgenommen haben und der Strom I_2 gleich Null ist. Dass nun durch die Spule 2 kein Strom fließt, hat den Grund, dass die Ausrichtung des Magnetfeldes nun genau in Phase mit dem Magnetfeld dieser Spule ist und sie somit keine Querkraft auf das Drehfeld erzeugen kann. Daher ist die Spule 2 zu diesem Zeitpunkt nicht von Nöten und kann abgeschaltet werden. Der Zeitpunkt t_3 ist analog zum Zeitpunkt t_1 zu betrachten und genauso stellt t_4 einen ähnlichen Zeitpunkt wie t_2 dar. Da ein Motor ein geschlossener Stromkreis ist, muss zu jedem Zeitpunkt die Summe aller Ströme gleich Null sein.



	I_1	I_2	I_3
t_1	$\hat{I}$	$-\frac{1}{2}\hat{I}$	$-\frac{1}{2}\hat{I}$
t_2	$\frac{\sqrt{3}}{2}\hat{I}$	0	$-\frac{\sqrt{3}}{2}\hat{I}$
t_3	$\frac{1}{2}\hat{I}$	$\frac{1}{2}\hat{I}$	$-\hat{I}$
t_4	0	$\frac{\sqrt{3}}{2}\hat{I}$	$-\frac{\sqrt{3}}{2}\hat{I}$

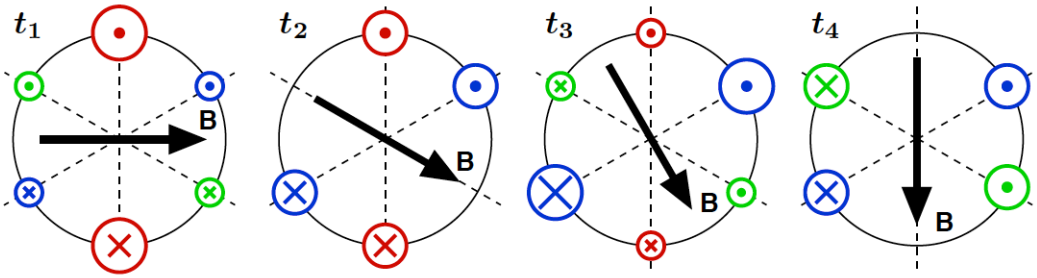


Abbildung 4: Zusammenhang zwischen Stromfluss und Drehfeld
(Quelle: https://proxy.duckduckgo.com/iu/?u=http%3A%2F%2Fhome.teleos-web.de%2Fvsteinkamp%2Fantriebe%2Fasynchronmotor%2Fdrehfeld.png&f=1)

Die Abbildung 4 veranschaulicht die Ansteuerung eines Brushless DC Motors mit einer Sinuskommutierung. Das heißt, dass die Motorspulen mit sinusförmigen Strömen gespeist werden. Alternative kann zur Steuerung des Motors auch die Blockkommutierung genutzt werden. Bei der Blockkommutierung oder auch 120 Grad Rechteckkommutierung fließt immer nur durch 2 der 3 Spulen ein Strom. Nach einer elektrischen Umdrehung von 60 Grad wird die nächste Spule abgeschaltet und die eben stromlose Spule wird nun eingeschaltet. Wird dies kontinuierlich fortgesetzt, so erhält man ebenfalls ein Drehfeld. Abbildung 5 zeigt, wie dann die Kurve der einzelnen Spulenströme aussieht.

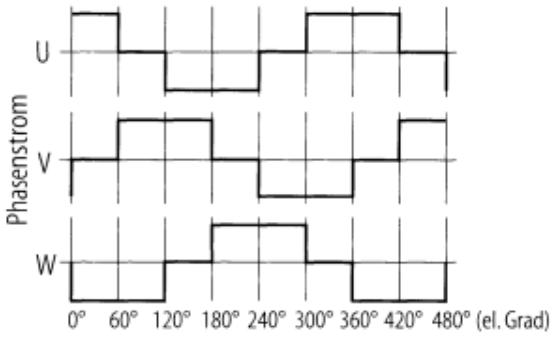


Abbildung 5: Stromkurven bei Blockkommutierung
(Quelle: Hering Ekbert, Handbuch der Elektrischen Anlagen und Maschinen, S.206)

Um diese Art des Stromflusses zu erhalten, wird der Brushless DC Motor mit einer dreiphasigen Brückenschaltung mit Endstufe beschaltet, wie die Abbildung 6 zeigt. Durch die MOSFETs kann man jede der drei Spulen individuell an- oder abschalten und der Stromfluss kann mittels Pulsweitenmodulation(PWM) gesteuert werden. [2]

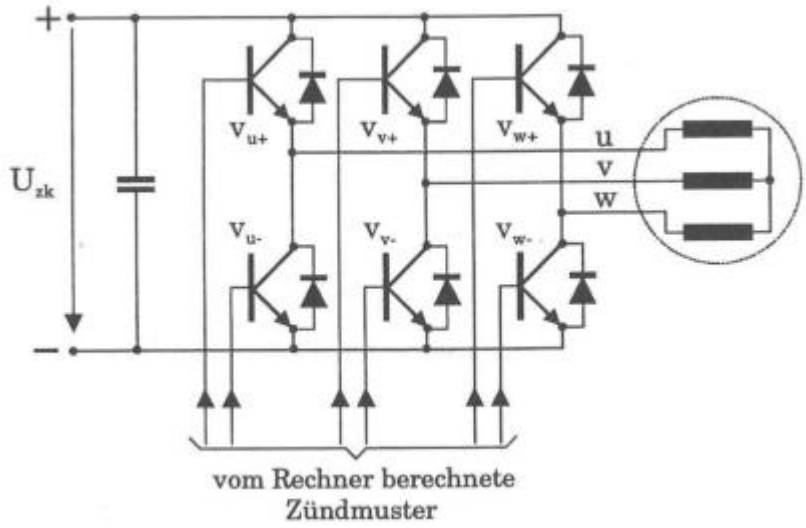


Abbildung 6: Dreiphasige Brückenschaltung mit Endstufe
(Quelle: Nguyen Phung Quang, Praxis der feldorientierten Drehstromantriebsregelung, S.11)

In der Praxis werden die PWM Signale für die MOSFETs oft mit dem Raumzeiger-Modulationsverfahren erstellt. Da der Atlas Digital Amplifier nur ein Kommando mit einem Spannungswert und einem Winkelwert zur Ansteuerung des Motors verwendet, nutzt auch er das Raumzeiger-Modulationsverfahren. Die nachfolgende Abbildung soll das Prinzip dieses Verfahrens veranschaulichen.

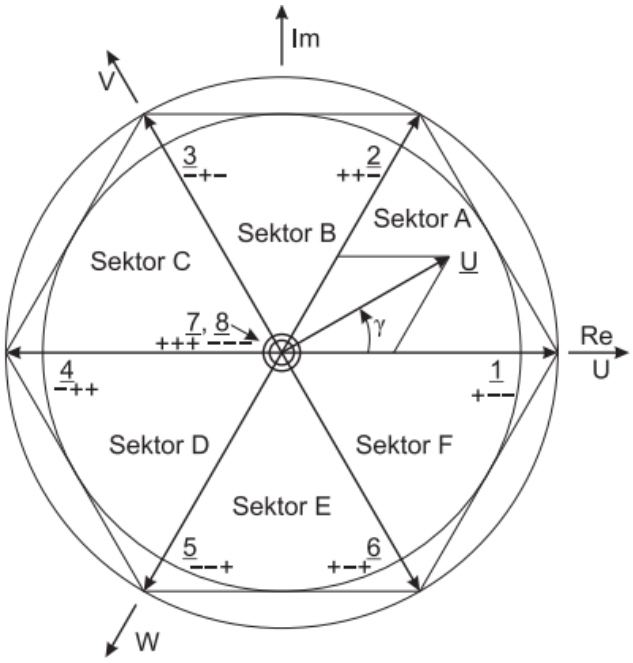


Abbildung 7:Prinzip der Raumzeigermodulation
(Quelle: Reill Josef, Lagegeberlose Regelung für ein accelermetergestütztes, hochdynamisches Roboterantriebssystem mit permanenterregtem Synchronmotor, S.15)

Mit den sechs MOSFETs zur Beschaltung des Motors, lassen sich acht Zustände definieren. Diese Zustände beschreiben welche Motorphase auf + und welche auf – gelegt wird und wie die MOSFETs dazu zu beschalten sind. Die Abbildung 8 zeigt den Zusammenhang von Spannungszeiger, Schaltzustand und Spannung an den Motorphasen

Spannugszeiger	Schaltzustand	$\frac{U_U}{U_{ZK}}$	$\frac{U_V}{U_{ZK}}$	$\frac{U_W}{U_{ZK}}$
1	+ − −	$+\frac{2}{3}$	$-\frac{1}{3}$	$-\frac{1}{3}$
2	+ + −	$+\frac{1}{3}$	$+\frac{1}{3}$	$-\frac{2}{3}$
3	− + −	$-\frac{1}{3}$	$+\frac{2}{3}$	$-\frac{1}{3}$
4	− + +	$-\frac{2}{3}$	$+\frac{1}{3}$	$+\frac{1}{3}$
5	− − +	$-\frac{1}{3}$	$-\frac{1}{3}$	$+\frac{2}{3}$
6	+ − +	$+\frac{1}{3}$	$-\frac{2}{3}$	$+\frac{1}{3}$
7	+ + +	0	0	0
8	− − −	0	0	0

Abbildung 8: Zusammenhang von Spannungszeiger, Schaltzustand und Spannung an den Motorphasen
(Quelle: Reill Josef, Lagegeberlose Regelung für ein accelermetergestütztes, hochdynamisches Roboterantriebssystem mit permanenterregtem Synchronmotor, S.16)

Im Zustand 7 werden alle MOSFETs auf der positiven Seite durchgeschaltet und in Zustand 8 alle auf der negativen Seite. In beiden Zuständen kommt somit kein Stromfluss zustande und sie werden daher als Nullzeiger bezeichnet. Da der Spannungszeiger aber nicht immer nur genau auf den Zuständen liegt sondern auch dazwischen, spannen, mit Ausnahme von Sektor 7 und 8, immer zwei Zustände einen Sektor auf. Um nun zum Beispiel einen Spannungszeiger in Sektor A zu erzeugen, muss zuerst der Zustand 1 und anschließend der Zustand 2 an der Brückenschaltung angelegt werden. Somit wird zuerst der Spannungszeiger auf den Zustand 1 gelegt und dann in Richtung Zustand 2 gedreht. Wie lang welcher Zustand angelegt werden muss, wird wie folgt berechnet:

$$t_1 = T_P \cdot \frac{|U|}{U_{ZK}} \cdot \sqrt{3} \cdot \sin(\frac{\pi}{3} - \gamma)$$

$$t_2 = T_P \cdot \frac{|U|}{U_{ZK}} \cdot \sqrt{3} \cdot \sin(\gamma)$$

$$t_0 = T_P - t_1 - t_2$$

Dabei steht T_P für die Pulsperiodendauer, |U| für den Betrag und γ für den Winkel des Spannungszeigers. Mit diesen Werten lassen sich dann die PWM Signale zur Ansteuerung der MOSFETs erstellen. [3]

3 Der Atlas Digital Amplifier

Der Atlas Digital Amplifier ist ein Steuermodul für Brushless DC Motoren, DC Brush Motoren und Schrittmotoren. Dabei bietet das Modul viele Funktionen an, wie eine digitale Stromsteuerung und Regelung, Strombegrenzung und Stromüberwachung und das Aufnehmen von Daten und Parametern direkt mit dem dafür eingebauten Speicher. Auf Grund dessen sind die Einsatzmöglichkeiten des Atlas Digital Amplifier breit gefächert. So kann er in medizinischem Equipment, bei der Automatisierung oder ganz allgemein zur einfachen Motorsteuerung eingesetzt werden. Der Atlas Digital Amplifier kann über eine Serial Peripheral Interface (SPI) Schnittstelle mit jedem herkömmlichen Mikrocontroller oder Field Programmable Gate Array (FPGA) kommunizieren. Für Schrittmotoren bietet der Atlas Digital Amplifier auch eine Pulse & Direction Schnittstelle an. [4]

Für den in dieser Bachelorarbeit verwendeten Versuchsaufbau wurde der Atlas Digital Amplifier zusammen mit einem Brushless DC Motor, einem PC, einem Mikrocontroller und einem Positionssensor verwendet. Abbildung 9 zeigt eine Skizze dieses Versuchsaufbaus.

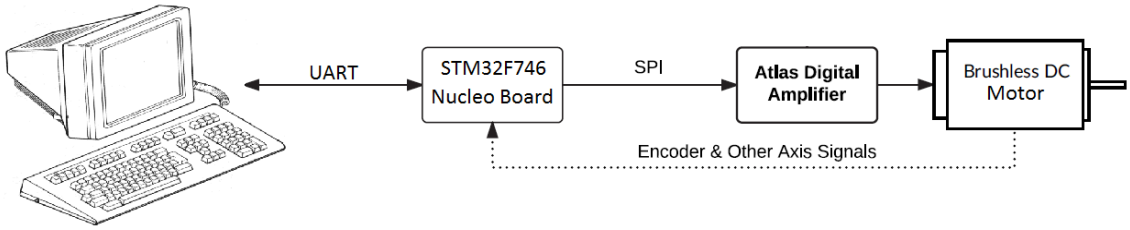


Abbildung 9: Skizze des Versuchsaufbaus
(Quelle: selbst erstellt)

3.1 Aufbau des Atlas Digital Amplifier

Der Atlas Digital Amplifier ist vermutlich nach der Von-Neumann-Architektur aufgebaut. Dabei ist das System in das Rechenwerk, das Steuerwerk, das Speicherwerk und das Ein- und Ausgabewerk aufgeteilt. Das Rechenwerk besteht aus dem Atlas Control Prozessor. Dieser verarbeitet die SPI Kommandos, steuert die Kommutierung sowie den Digitalen Regler und die Stromausgabe über die Power Stage. Das Ein- und Ausgabewerk besteht aus den Hardware Anschlüssen, wie zum Beispiel den SPI Pins, dem Enable oder FaultOut Pin, und den dazugehörigen Komponenten, wie zum Beispiel der Power Stage.

Des Weiteren wird ein Teil des Speicherwerks dem Benutzer des Atlas Digital Amplifier zur Verfügung gestellt. Mit dem non-volatile storage random-access memory (NVRAM) und dem Trace random-access memory (RAM) können motor- und atlasspezifische Daten abgespeichert werden. In der nachfolgenden Abbildung ist ein Modell der Schaltung im Inneren des Atlas Digital Amplifier dargestellt.

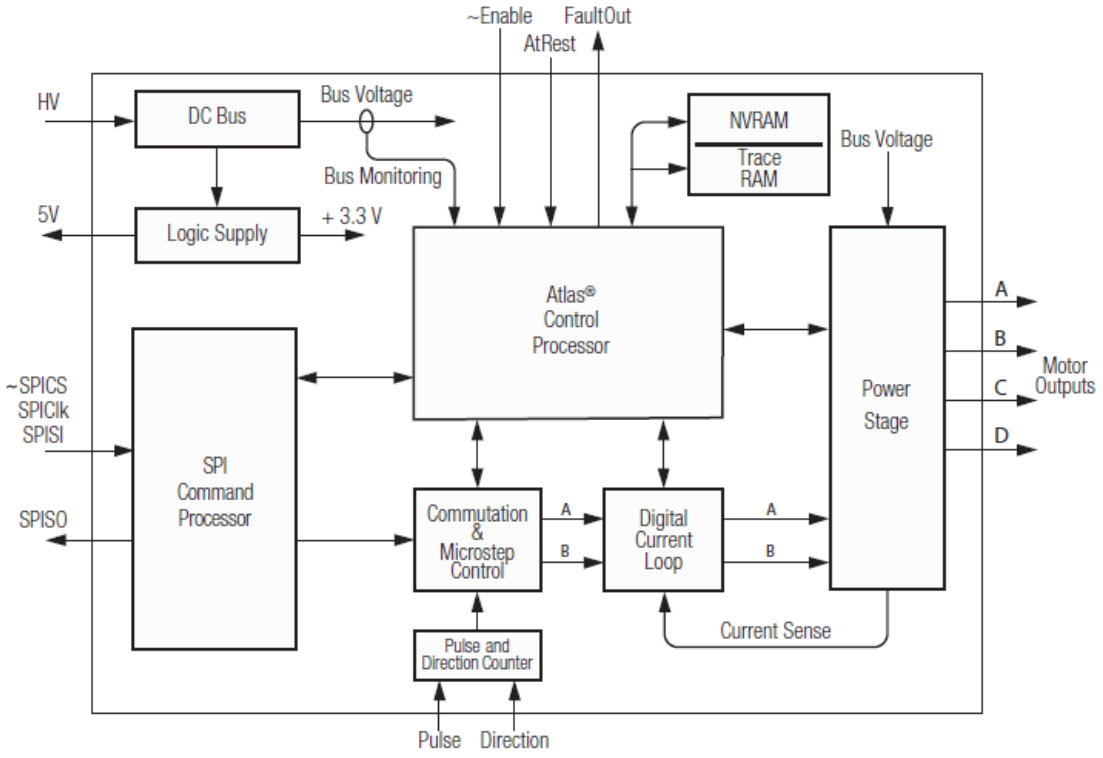


Abbildung 10: Modell des inneren Aufbaus des Atlas Digital Amplifier
(Quelle: Atlas Complete Technical Reference, S.40)

Die Spannungsversorgung erfolgt über die Spannungsquelle des Motors. Diese wird direkt am Atlas Digital Amplifier angeschlossen, sodass der Atlas Digital Amplifier die Spannungsversorgung des Motors übernimmt. Dabei kann die Spannung, abhängig vom verwendeten Motor, im Bereich von 12 bis 48 Volt liegen. [5]
Von dieser Spannung zweigt sich der Atlas Digital Amplifier 3,3V für die Versorgung der Logikschaltung und 5V für den 5V Output Pin ab. Die Motorspannung selbst wird im Atlas Digital Amplifier zu einer Power Stage geführt. Diese versorgt den Motor mit Strom und Spannung und schaltet die MOSFET Brücken zur präzisen Ansteuerung der Motorspulen. Außerdem findet in der Power Stage die Pulsweitenmodulation statt, wodurch Drehmoment und Drehzahl gesteuert werden.

Insgesamt besitzt der hier eingesetzte Atlas Digital Amplifier 18 Pins, von denen jedoch 4 keine Funktion haben. Diese 18 Pins sind:

- Versorgungsspannung (HV & Power Ground)
- Vier Output Pins für den Motor
- Ein Enable und ein FaultOut Pin
- Ein 5V Output Pin
- Ein Masse Pin
- Vier SPI Pins

Der Enable Pin aktiviert oder deaktiviert die Power Stage des Atlas Digital Amplifier. Der FaultOut Pin ist dazu da, Fehler beim Betrieb des Atlas Digitla Amplifier anzuzeigen. Diese Fehler und deren Beschreibungen sind im Kapitel 3.2 aufgelistet. [5] Der Atlas Digital Amplifier besitzt außerdem die Einheit Commutation and Microstep Control, in welcher die eingehenden Werte zur Motorsteuerung verarbeitet und die PWM Signale erstellt werden. Des Weiteren besitzt der Atlas Digital Amplifier eine Digital Current Loop. Dies ist der interne PI-Regler. Hier stehen dem Nutzer verschiedene Regelkreise zur Verfügung, die die Ströme des Motors regeln. Dabei sind die Parameter des Reglers frei wählbar. [5]

3.2 Statusregister des Atlas Digital Amplifier

Zur Sicherheit und Fehleranalyse besitzt der Atlas Digital Amplifier 5 Statusregister. Diese sind:

- Event Status
- Drive Status
- Signal Status
- SPI Status
- Drive Fault Status

Tritt ein Fehler auf, setzt der Atlas Digital Amplifier das entsprechende Bit im Register. Mit Hilfe des Event Status Registers können die Fehler: instruction error, overtemperature fault, drive exception und current foldback erkannt werden.
Das instruction Error Bit wird immer dann gesetzt, wenn bei der Kommunikation mit dem Mikrocontroller via SPI ein Fehler auftritt. Dies kann unterschiedliche Gründe haben. Zum Beispiel wenn die berechneten Prüfsummen nicht korrekt sind oder empfangene Werte nicht im Definitionsbereich liegen. Das overtemperature fault Bit wird gesetzt, sobald die Temperatur des Atlas Digital Amplifier ein einstellbares Limit überschreitet.

Tritt ein Fehler ein, der ein Bit im Drive Fault Status Register setzt, so wird auch das drive exception Bit im Event Status Register gesetzt. Zusätzlich wird mit diesem Bit auch ein Fehler angezeigt, sollte der Enable Pin des Atlas Digital Amplifier nicht mehr eingeschaltet sein. Um den Enable Pin zu aktivieren, muss eine logische Null an diesem Pin angelegt werden. Das current foldback Bit dient zur Sicherheit des Motors. Hierbei hat der Nutzer des Atlas Digital Amplifier die Möglichkeit ein Stromlimit einzustellen. Dieses Limit wird dann vom Atlas Digital Amplifier über die Zeit, während der Motor bestromt wird, addiert. Sollte dieser aufaddierte Wert dann einen weiteren Schwellenwert überschreiten, so wird das current foldback Bit gesetzt und der Motor ausgeschaltet. So ist es möglich kurzfristige Stromspitzen zuzulassen, den Motor, auf längere Zeit gesehen, aber nicht zu überlasten. Mit dem Befehl SPI GetEventStatus lassen sich die Bits des Event Stauts Registers auslesen. Ist eins dieser Bits einmal gesetzt, so löscht der Atlas Digital Amplifier dieses Bit nicht mehr von selbst. Nur ein externer Mikrocontroller kann die Bits dieses Registers mit dem SPI Befehl ResetEventStatus wieder zurücksetzten.

Im Gegenteil dazu kann der Atlas Digital Amplifier die Bits des Drive Status Registers jederzeit setzen und löschen. Im Drive Status Register stehen die Informationen in foldback, overtemperature, in holding, overvoltage, undervoltage, disabled, output clipped und drive not initialized. Das Bit in foldback wird gesetzt, wenn derselbe Fall eintritt, wie für das Current Foldback Bit im Event Status Register. Das Overtemperature Bit wird gesetzt, sobald die Temperatur des Atlas Digital Amplifier ein selbst wählbares Limit übersteigt. Das In Holding Bit kann nur gesetzt werden, wenn der Atlas Digital Amplifier im Puls & Direction mode für Schrittmotoren betrieben wird und ist im Zusammenhang mit einem Brushless DC Motor nicht von Interesse. Die Grenzen für Überspannung und Unterspannung des Motors können ebenfalls vom Benutzer gewählt werden. Sollten diese Werte dann über- bzw. unterschritten werden, so wird das Overvoltage oder Undervoltage Bit gesetzt. Das Disable Bit setzt man, wenn der Enable Pin des Atlas Digital Amplifier nicht aktiv geschaltet ist. Das Output Clipped Bit ist nur von Bedeutung, wenn ein PWM Limit gesetzt wird. Ein solches wurde in diesem Versuchsaufbau jedoch nicht verwendet und das Bit ist somit nicht von Interesse. Der Atlas Digital Amplifier benötigt eine gewisse Zeit um sich zu initialisieren, sobald er bestromt wird oder ein Reset Befehl geschickt wird. Möchte man in dieser Zeit mit dem Atlas Digital Amplifier kommunizieren, so wird das Drive Not Initialized Bit gesetzt. Der Status der Bits in diesem Register kann mit dem SPI Kommando GetDriveStatus abgefragt werden.

Das Signal Status Register besitzt nur zwei Bits: eines für den Enable Pin und eines für den FaultOut Pin. Das Enable Bit wird dann gesetzt, wenn an dem Enable Pin eine logische Eins vom Mikrocontroller angelegt wird und das FaultOut Bit, wenn der Atlas Digital Amplifier eine logische Eins am FaultOut Pin anlegt. Eine Eins am Enable Pin bedeutet, dass der Ausgang zum Motor deaktiviert werden soll. Sollte ein Bit im FaultOut Register gesetzt sein, so wird am FaultOut Pin eine Eins angelegt. Diese Bits können zusätzlich mit dem SPI Kommando GetSignalStatus abgefragt werden.

Das SPI Status Register wird zu Beginn jeglicher SPI Kommunikation an den Mikrocontroller gesendet. So erhält man immer ein Feedback über den Status des Atlas Digital Amplifier. Das SPI Status Register enthält Informationen über: in foldback, overtemperature, overvoltage, undervoltage, disabled, instruction error, overtemperature event, drive exception event, output clipped und foldback event. Alle Bits aus diesem Register spiegeln Bits aus den anderen Registern. Somit werden die wichtigsten Bits hier zusammengefasst und an den Mikrocontroller gesendet.

Das Letzte der 5 Register ist das Drive Fault Status Register. In diesem Register werden die Fehler overcurrent, operating mode mismatch, watchdog timeout, overvoltage, undervoltage, disabled, current foldback, overtemperature, SPI checksum error und watchdog fault angezeigt. Das Overcurrent Bit wird gesetzt, wenn der Atlas Digital Amplifier einen Kurzschluss an den Ausgängen zum Motor feststellt. Empfängt der Atlas Digital Amplifier ein SPI Kommando, welches den Motor ansprechen soll, die Motorausgabe aber noch nicht freigeschaltet sein, so wird das Operating Mode Mismatch Bit gesetzt. Das Watchdog Timeout Bit zeigt an, dass die Zykluszeit des watchdog überschritten wurde. Die Bits overvoltage und undervoltage werden gesetzt, wenn der Atlas Digital Amplifier feststellt, dass bereits die Versorgungsspannung zu hoch oder zu niedrig gewählt wurde, um einen stabilen Betrieb zu gewährleisten. Das Disable Bit wird gesetzt, sollte der Enable Pin nicht eingeschaltet sein und das Current Foldback Bit hat dieselbe Bedeutung wie im Event Status Register. Auch die Bits in diesem Register müssen von dem Benutzer gelöscht werden. Das Overtemperature Bit wird gesetzt, sobald ein vom Benutzer einstellbares Limit überschritten wird. Das SPI Checksum Error Bit wird vom Atlas Digital Amplifier immer dann gesetzt, wenn ein fehlerhaftes SPI Kommando empfangen wurde und somit die Berechnung der checksum für dieses Kommando inkorrekt ist. Das Wachtdog Fault Bit wird in der Dokumentation des Atlas Digital Amplifier nicht beschrieben, ist aber dennoch von Bedeutung. Dieses Bit zeigt einen Fehler mit dem watchdog an, wodurch der Motor gestoppt wird.
Wird einer dieser Fehler angezeigt, so sollte das Register erst mit dem SPI Befehl GetDriveFaultStatus ausgelesen und anschließend mit dem SPI Befehl ClearDriveFaultStatus zurückgesetzt werden. [5]

3.3 Kommunikation mit dem Atlas Digital Amplifier

Um mit dem Atlas Digital Amplifier kommunizieren zu können, muss eine Verbindung über die SPI Schnittstelle aufgebaut werden. Die Einstellungen der SPI Schnittstelle werden im Kapitel 5.1.3 erläutert. Für den Datenaustausch zwischen dem STM32F746 und dem Atlas wird von dem Atlas Digital Amplifier ein eigenes Protokoll verlangt. Dieses besteht aus einem Haupt- und einem optionalen Nebenkommando. Die Hauptkommandos bestehen immer aus 2 Wörtern. Dabei besteht jedes Wort aus 16 Bit. Die Anzahl an Wörtern der Nebenkommandos kann je nach Befehl variieren. Aber auch bei den Nebenkommandos muss jedes Wort aus 16 Bit bestehen.

Ein wichtiges Signal für die Kommunikation des Atlas Digital Amplifier mit einem Mikrocontroller, ist das Chipselect Signal (CS). Wie in Abbildung 11 zu sehen ist, beginnt das Protokoll mit einer fallenden Flanke auf der CS Leitung und endet mit einer steigenden Flanke. Innerhalb dieser Flanken muss der Datenaustausch stattfinden. Zur Veranschaulichung wurde hierfür eine Momentaufnahme am Oszilloskop gemacht.
[5]

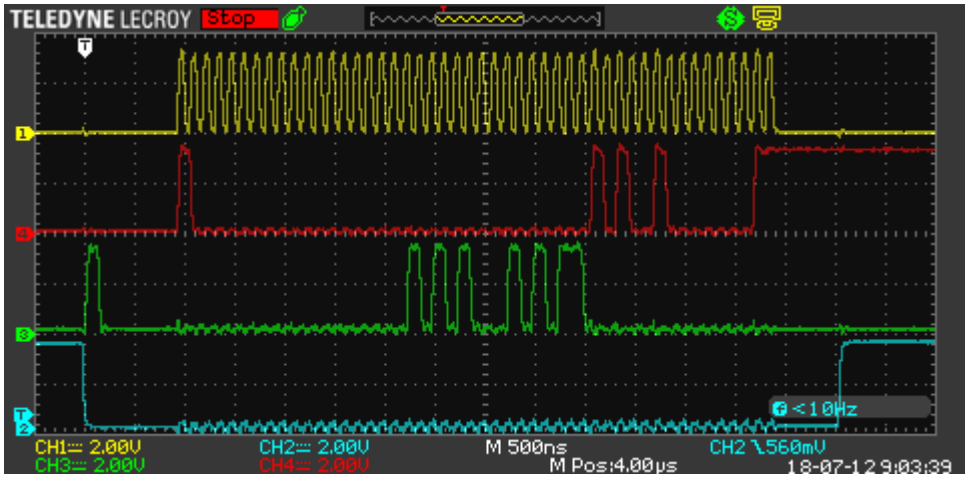


Abbildung 11: Oszilloskopaufnahme der SPI Kommunikation des Atlas Digital Amplifier mit dem STM32F746 Nucleo Development board
(Quelle: eigene Aufnahme)

In der Abbildung 11 sind vier Graphen in unterschiedlichen Farben dargestellt. Der gelbe Graph zeigt die Clock Leitung (CLK) der SPI Schnittstelle. Der rote die Datenleitung vom STM32F746 zum Atlas Digital Amplifier. Dies entspricht der Master Out Slave In (MOSI) Leitung. Der grüne Graph stellt die Leitung vom Atlas Digital Amplifier zum STM32F746 dar, also die MISO Leitung. Der letzte Graph, der blaue, entspricht der CS Leitung.

Außerdem ist zu erkennen, dass der Atlas Digital Amplifier, mit dem Setzen der CS Leitung auf einen Low Pegel, die MISO Leitung auf einen High Pegel setzt. Dies kann als eine Art busy state interpretiert werden. Erst nachdem diese Leitung vom Atlas Digital Amplifier wieder auf einen Low Pegel gesetzt wird, kann mit der Kommunikation begonnen werden.

3.3.1 Einführung in das Datenprotokoll von Performance Motion Devices

Für das Protokoll von Performance Motion Devices existieren drei Hauptkommandos. Diese sind:

1. Sending a Voltage or Torque Output Value
2. Sending an Amplifier Disable
3. Sending a No Operation (NOP)

Das erste dieser Kommandos dient dazu den Motor anzusprechen. Das zweite ist ein Software Notfall Ausschalter für den Atlas Digital Amplifier. Der dritte Befehl ist der No Operation Befehl und ist dafür da Informationen mit dem Atlas Digital Amplifier auszutauschen, ohne dabei den Motor anzusteuern. Dies ist für die Initialisierung des Atlas Digital Amplifier sehr nützlich.

Als nächstes wird die Grundstruktur für diese drei Kommandos erklärt. Jedes dieser Kommandos besteht aus einem Packet Header und zwei Header Daten.
Der Packet Header besteht aus vier Bits. Diese sind immer die Bits 12, 13, 14 und 15 im ersten gesendeten Wort. Das erste Bit, Bit Nummer 12, des Headers ist reserviert und enthält immer eine Null. Die anderen drei Bit sind Flags, die unterschiedliche Funktionen des Atlas Digital Amplifier aktivieren. Diese Flags sind: Das Update flag u, das Trace active flag t und das Torque data flag x. Das Update flag dient allein dazu die Werte des internen Reglers zu setzten. Das heißt, möchte man zum Beispiel den Proportionalanteil des PI-Reglers erhöhen, so schickt man dem Atlas Digital Amplifier zuerst den neuen Wert und anschließend muss ein weiteres Kommando gesendet werden, zum Beispiel ein NOP, in dem das Update flag gesetzt ist. Erst nach dem zweiten Kommando mit dem Update flag wird der neue Wert für den Regler übernommen.
Das Trace active flag funktioniert ähnlich. Möchte man die interne Aufzeichnung von Daten des Atlas Digital Amplifier nutzen, so muss zu Beginn und während der Dauer der Aufzeichnung das Trace active flag gesetzt sein.
Das Torque data flag signalisiert dem Atlas Digital Amplifier, ob das gerade gesendete Hauptkommando einen Motorsteuerungsbefehl enthält oder nicht. Möchte man, dass sich der Motor dreht, so darf dieses flag nicht gesetzt sein.

Dieser Packet Header dient dem Atlas Digital Amplifier als eine Art Vorsortierung. Anhand dieses Headers kann das Atlas Digital Amplifier Steuermodul bereits erkennen, welche Art von Kommando als nächstes folgen wird. Die untenstehende Abbildung veranschaulicht die Bits des Packet Headers. [5]

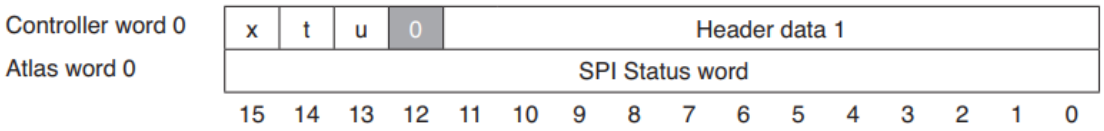


Abbildung 12: Veranschaulichung der Packet Header Bits
(Quelle: Atlas Complete Technical Reference 2.0, S. 85)

Um die Hauptkommandos zu vervollständigen müssen nun noch die Header Daten beschrieben werden. Im Kommando Sending a Voltage or Torque Output Value werden dem Atlas Digital Amplifier, für den Fall, dass ein Brushless DC Motor verwendet wird, der Stromphasenwinkel und das Drehmoment geschickt. Hierbei muss darauf geachtet werden, dass der Stromphasenwinkel, aufgrund des Packet Headers, nur ein 12 Bit unsigned integer ist. Der Wert des Stromphasenwinkels steht im Zusammenhang mit der Position des Motors. Mit diesem Wert wird die Ausrichtung des elektromagnetischen Feldes bestimmt. Die Änderung dieses Wertes bewirkt eine Drehung des Magnetfeldes und folglich die Drehung des Motors. Somit steht die Größe der Differenz zweier Stromphasenwinkelwerte im Zusammenhang mit der Rotationsgeschwindigkeit des Motors. Damit sich der Motor mit einer konstanten Geschwindigkeit dreht, muss man also den Wert des Stromphasenwinkels kontinuierlich erhöhen. Der Atlas Digital Amplifier nutzt dazu 4096 Werte für eine elektrische Umdrehung. Bei dieser Art von Motor entspricht das einer mechanischen Umdrehung von 36 Grad. Das heißt, dass der Atlas Digital Amplifier 40960 Werte für eine komplette mechanischen Umdrehung, also 360 Grad, zur Verfügung stellt. Möchte man zum Beispiel eine Drehzahl von circa $100\frac{U}{min}$, bei einer Zeit von 100µs zwischen zwei Sending a Voltage or Torque Kommandos, so wird der Phasenwinkelwert, den man an das Atlas Digital Amplifier schicken muss, folgendermaßen berechnet:

Zuerst muss man die Drehzahlberechnung rückwärts anwenden, um auf die Winkeldifferenz zu kommen, die in diesem Zeitabschnitt vom Motor zurückgelegt wird.

$$\frac{100\frac{U}{min}\cdot 360°}{60\cdot 10^4 s} = 0{,}06\frac{Grad}{100\mu s}$$

Anschließend muss man diese Winkeldifferenz noch in die 40960 Schritte des Atlas Digital Amplifier umwandeln. Über den Dreisatz erhält man somit:

$$360° \triangleq 40960$$

$$0{,}01° \triangleq \frac{40960}{36000°}$$

$$0{,}06° \triangleq \frac{40960}{36000°}\cdot 6$$

$$\frac{40960}{36000°}\cdot 6 \triangleq 6{,}827$$

Gerundet erhält man den Wert 7, um den der Phasenwinkel des Atlas Digital Amplifier kontinuierlich erhöht werden muss, um eine Drehzahl von circa $100\frac{U}{min}$ zu erhalten.

Der Wert des Drehmoments ist ein 16 Bit signed integer und steht in Relation zur Amplitude des Motorstroms. Der Atlas Digital Amplifier berechnet den Motorstrom indem er den Wert des Drehmoments mit dem konstanten Wert 1,526 $\frac{mA}{count}$ multipliziert. Möchte man zum Beispiel eine Amplitude von 1A einstellen, so berechnet sich der Wert n, den man als Drehmoment schicken muss, folgendermaßen:

$$n = \frac{1A}{1{,}526\ \cdot 10^{-3}A} = 655{,}308$$

Da als Drehmoment nur integer Werte gesendet werden können, werden bei dieser Berechnung die Nachkommastellen einfach abgeschnitten und man erhält den Wert 655. Umgewandelt in das hexadezimale System wird daraus 0x28F.

Die zwei nachfolgenden Abbildungen zeigen, wie das Kommando zur Steuerung des Motors aussehen muss, wenn man einen Motorstrom von 1A und eine Geschwindigkeit von circa 100 $\frac{U}{min}$ einstellen möchte.

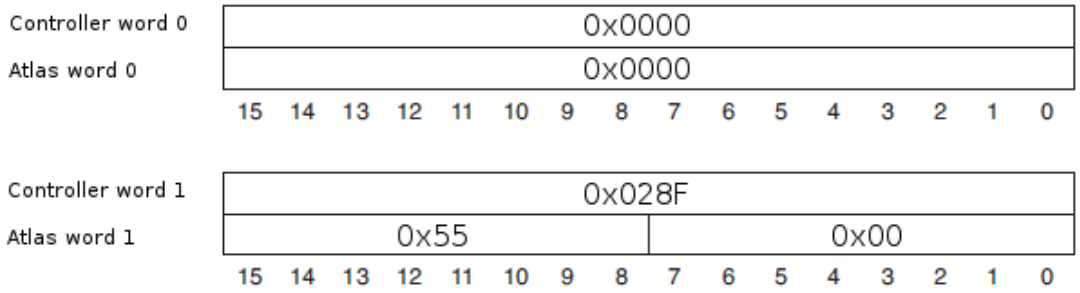


Abbildung 13: Torque Kommando mit 1A Motorstrom und Phasenwinkel 0
(Quelle: selbst erstellt)

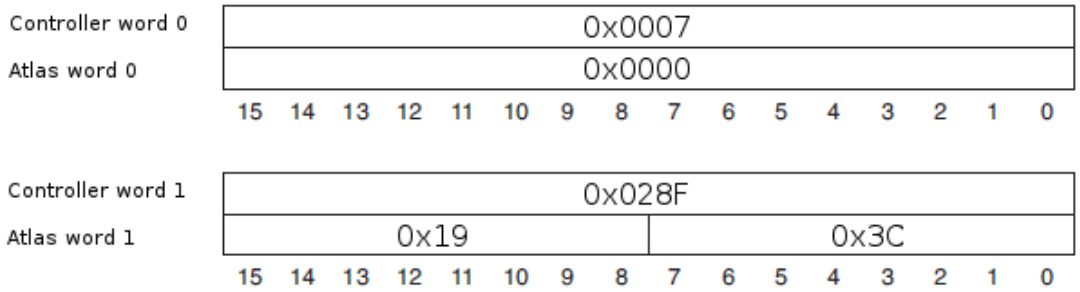


Abbildung 14: Torque Kommando mit 1A Motorstrom und Phasenwinkel 7
(Quelle: selbst erstellt)

Die Header Daten des Befehls Sending an Amplifier Disable bestehen nur aus einer 8 Bit großen checksum, welche die ersten 8 Bits des ersten Wortes belegt. Die Bits zwischen dieser checksum und dem Packet Header, also die Bits 8 bis 11, müssen laut Protokoll mit einer eins belegt werden. Des Weiteren schreibt das Protokoll vor, dass das zweite Wort komplett mit Nullen beschrieben sein muss und das Torque data flag im Packet Header gesetzt sein muss. Das Setzen des Torque data flags signalisiert dem Atlas Digital Amplifier, dass nun kein Motorsteuerungsbefehl gesendet wird. Die Berechnung der Checksum wird im Kapitel 3.3.2 erläutert. Die Abbildung 15 zeigt ein Beispiel für dieses Kommando.

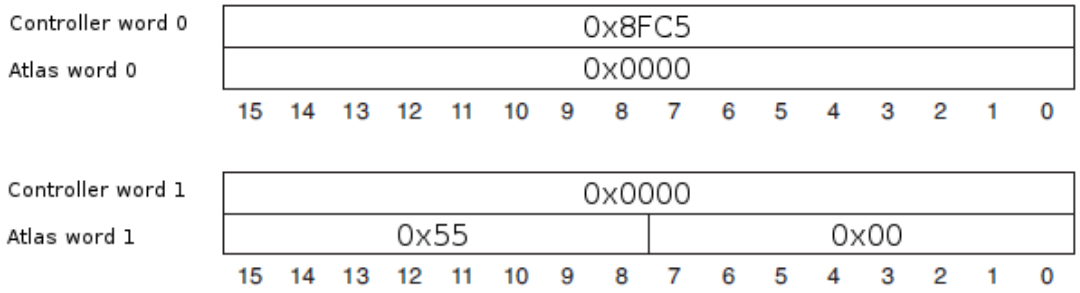


Abbildung 15: Beispiel für das Hauptkommando Sending an Amplifier Disable
(Quelle: selbst erstellt)

Im Sending a NOP Kommando stehen keine Informationen für das Atlas Digital Amplifier Steuermodul. Somit können in den ersten 8 Bit des ersten Wortes und im ganzen zweiten Wort beliebige Werte stehen. Die Bits 8 bis 11 des ersten Wortes müssen jedoch mit Nullen beschrieben werden und ebenso wie bei dem Befehl Sending a Amplifier Disable muss auch bei dem Kommando Sending a NOP das Torque data flag gesetzt sein. Die Abbildung 16 zeigt ein Beispiel für das Kommando Sending a NOP.

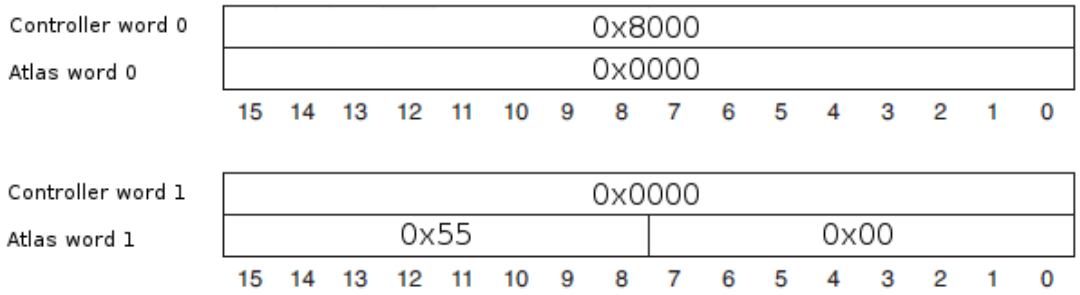


Abbildung 16: Zahlenbeispiel des Kommandos Sending a NOP
(Quelle: selbst erstellt)

Wie bereits in der vorherigen Abbildung zu sehen ist, erhält man zu jedem gesendeten Wort immer eine Antwort. Bei Hauptkommandos ist diese Antwort immer gleich. Sie besteht aus einem SPI Status word, einer Atlas checksum und einer Controller checksum. Das erste Wort, welches der Atlas Digital Amplifier sendet, ist das SPI status word. Die Informationen aus dem SPI Status word sind im Kapitel 3.2 beschrieben. Im zweiten Wort steht in den Bits 0 bis 7 die Controller checksum und in den Bits 8 bis 15 die Atlas checksum. Die Atlas checksum wird vom Atlas Digital Amplifier berechnet und dient dazu dem Mikrocontroller die Möglichkeit zu geben die empfangenen Pakete auf ihre Vollständigkeit und Korrektheit zu überprüfen.

Mit der Controller checkusm kann kontrolliert werden, ob der Atlas Digital Amplifier das zuletzt gesendete Hauptkommando richtig empfangen hat. Anhand der Abbildung 11 lassen sich die einzelnen Wörter gut erkennen. Das Kommando aus der Abbildung besteht aus dem Hauptkommando Sending a NOP und dem Nebenkommando GetMotorType. Das Kommando NOP erkennt man daran, dass am roten Graphen das erste Bit gesetzt ist, gefolgt von 31 Nullen. Damit ist das Hauptkommando komplett und es folgt das Nebenkommando. Als Antwort sieht man, dass der Atlas Digital Amplifier keine Bits im SPI Status word gesetzt hat, da die ersten 16 Bit des grünen Graphens Nullen sind. Anschließend folgen die Atlas und die Controller checksum. Auf das Nebenkommando antwortet der Atlas Digital Amplifier mit Nullen.

An den Atlas Digital Amplifier wird nur von dem Kommando Sending Amplifier Disable, sowie den Nebenkommandos eine checksum gesendet. Dabei wird jede checksum unterschiedlich berechnet. [5]

3.3.2 Berechnung der unterschiedlichen checksums

Die Berechnung der checksums erfolgt immer Byteweise. Das heißt, dass Daten wie das SPI status word in ein Highbyte und eine Lowbyte aufgeteilt werden müssen. Das Lowbyte enthält dabei die Bits 0 bis 7 und das Highbyte die Bits 8 bis 15.

Für die Berechnung der Atlas checksum müssen das Highbyte und das Lowbyte des SPI Status word mit der empfangenen Atlas checksum, der empfangenen Controller checksum und einem konstanten Wert von 0xAA addiert werden. Ergibt die Addition 0xFF so ist die empfangene Atlas checksum korrekt. Der konstante Wert 0xAA stellt sicher, dass keine der Datenleitungen dauerhaft mit einer Eins oder Null beschrieben wird. Dadurch können Schäden an der Hardware ausgeschlossen beziehungsweise schnell erkannt werden. In nachfolgender Tabelle ist diese Berechnung anhand der Antwort des Atlas Digital Amplifier auf ein Torque Kommando als Zahlenbeispiel dargestellt. Zur Veranschaulichung wurden hierfür die Werte aus Abbildung 14 genommen.

Tabelle 1: Beispiel für die Berechnung der Atlas checksum

	0x00	Highbyte des SPI status word
	0x00	Lowbyte des SPI status word
	0x19	empfangene Atlas checksum
	0x3C	empfangene Controller checksum
+	0xAA	Konstante
	0xFF	Ergebnis

(Quelle: selbst erstellt)

Zur Umsetzung dieser Berechnung wurde die Funktion CalculateAtlasChecksum implementiert. Dieser Funktion muss das SPI status word, sowie die Atlas checksum und die Controller checksum übergeben werden. Die Funktion führt dann die Addition wie in der Tabelle 1 dargestellt aus und überprüft anschließend, ob der Definitionsbereich von einem Byte überschritten wurde und somit ein Überhang entstanden ist. Danach wird das Ergebnis der Addition überprüft. Bei einem falschen Ergebnis wird eine Bildschirmausgabe erzeugt, die den Anwender darauf aufmerksam machen soll.
Mit dieser Funktion als Vorlage wurden auch alle anderen Funktionen zur Berechnung der verschiedenen checksums programmiert. Die nachfolgenden Zeilen zeigen ein Beispiel wie der Programmcode für so eine Berechnung aussehen könnte.

```
char CalculateAtlasChecksum( Data SPI_Status_word, Data
Atlas_and_Controller_checksum )
{
   unsigned short sum=0;
   int row, column;

   sum += SPI_Status_word.EightBit[HighByte] +
     SPI_Status_word.EightBit[LowByte];
   sum += Atlas_and_Controller_checksum.EightBit[HighByte] +
     Atlas_and_Controller_checksum.EightBit[LowByte];
   sum += 0xAA;
   CarryAddingOne(&sum);

   if( sum != 0xFF )
   {
     GetCursorPosition( &row, &column );
     JumpPosition( 18, 1 );
     SendTOTerminal( "\x1b[31matlas checksum error !!\x1b[30m" );
     JumpPosition( row, column );
     return SPI_STATUS_ERROR;
   }
   else
     return SPI_STATUS_OK;
}
```

Bei der Summation kann es vorkommen, dass der Definitionsbereich von einem Byte überschritten wird. Tritt dieser Fall ein, so wird der Überhang ausmaskiert, um 8 Bits nach rechts geschoben und wieder auf das Ergebnis aufaddiert. Dabei muss das Ergebnis dann inklusive addiertem Überhang 0xFF ergeben. Die selbst programmierte Funktion CarryAddingOne erfüllt diese Aufgabe. Dazu muss ihr nur ein Zeiger auf das Ergebnis der vorherigen Addition übergeben werden. Da es theoretisch auch möglich ist, dass die Funktion CarryAddingOne wieder einen Überhang erzeugt, wurde eine while Schleife eingebaut, die die Ausmaskierung, Verschiebung und Addition solange ausführt, bis kein Überhang mehr entstanden ist. Die nachfolgenden Zeilen zeigen den dafür erstellten Quellcode.

```
void CarryAddingOne( unsigned short *sum )
{
   int mask = 0xFF;
   int maskI = ~mask;
   int adding;
   while( (*sum) & (maskI))
   {
      adding = (*sum) & maskI;
      adding = adding >> 8;
      *sum = (*sum) & mask;
      *sum += adding;
   }
}
```

Die Berechnung der Controller checksum funktioniert ähnlich wie die Berechnung der Atlas checksum. Zu beachten ist hierbei, dass die Controller checksum für das aktuelle Kommando immer erst ein Kommando später vom Atlas Digital Amplifier gesendet wird. Um die Controller checksum zu erhalten, müssen das High- und Lowbyte beider Wörter des Hauptkommandos und der Konstanten 0xAA addiert werden. Ein Beispiel dieser Berechnung, mit den Werten aus Abbildung 14, befindet sich in der Tabelle 2.

Tabelle 2: Beispiel für die Berechnung der Controller checksum

	0x02	Highbyte des ersten Wortes
	0x8F	Lowbyte des ersten Wortes
	0x00	Highbyte des zweiten Wortes
	0x07	Lowbyte des zweiten Wortes
+	0xAA	Konstante
	0x3C	Controller checksum

(Quelle: selbst erstellt)

Anschließend muss auch nach dieser Berechnung wieder die Funktion CarryAddingOne aufgerufen werden, um einen möglichen Überhang zu behandeln. Danach muss diese berechnete Controller checksum mit der, vom Atlas Digital Amplifier gesendeten Controller checksum, verglichen werden. Stimmen diese checksums überein, so wurde das letzte gesendete Hauptkommando richtig vom Atlas Digital Amplifier empfangen. [5]

Die Berechnung der checksum des Befehls Sending an Amplifier Disable ist sehr ähnlich wie die Berechnung der Controller checksum. Genauso wie bei der Controller checksum müssen auch hier die High- und Lowbytes beider Wörter des Hauptkommandos und die Konstante 0xAA addiert werden. Um nun aber die checksum des Befehls Sending an Amplifier Disable zu bekommen, muss das Ergebnis der Addition noch von dem Wert 0xFF abgezogen werden. Die Tabelle 3 veranschaulicht diese Rechnung noch einmal.

Tabelle 3: Beispiel für die Berechnung der checksum für das Hauptkommando Sending an Amplifier Disable

	0x8F	Highbyte des ersten Wortes
	0xAA	Konstante
+	0x3A	Zwischenergebnis
	0xFF	Endergebnis bei Addition aller Werte
-	0x3A	Zwischenergebnis
	0xC5	Sending an Amplifier Disable checksum

(Quelle: selbst erstellt)

Jedes Nebenkommando enthält ebenfalls eine checksum. Zur Berechnung dieser checksum müssen alle High- und Lowbytes, aller im Nebenkommando enthaltenen Wörter, addiert und anschließend von 0xFF subtrahiert werden. Die Subtraktion von 0xFF erklärt sich daher, dass nach dem Protokoll bei der Addition aller Werte inklusive der checksum das Ergebnis 0xFF ergeben muss. Um nun daraus die checksum berechnen zu können, wird diese Addition nach dem gesuchten Wert der checksum, umgestellt. Die anschließende Tabelle zeigt anhand des Befehls SetOperatingMode ein selbst erstelltes Beispiel zur Berechnung der checksum.

Tabelle 4: Beispiel für die checksum Berechnung der Nebenkommandos

	0x65	opcode des Nebenkommandos
	0x06	mode Einstellung des Befehls SetOperatingMode
+	0xAA	Konstante
	0x16	Zwischenergebnis
	0xFF	Endergebnis bei Addition aller Werte
-	0x16	Zwischenergebnis
	0xE9	Nebenkommando checksum

(Quelle: selbst erstellt)

Auch hier muss, bei der Addition, wieder auf einen möglichen Überhang geachtet werden. Für ein besseres Verständnis dieser checksum Berechnung wird in Tabelle 5 noch einmal die Addition aller Werte mit der bereits berechneten checksum dargestellt.

Tabelle 5: Addition aller Werte des Nebenkommandos

	0x65	opcode des Nebenkommandos
	0x06	mode Einstellung des Befehls SetOperatingMode
	0xE9	Checksum des Nebenkommandos SetOperatingMode
+	0xAA	Konstante
	0xFF	Endergebnis bei Addition aller Werte

(Quelle: selbst erstellt)

Nach jedem Nebenkommando erhält man eine Antwort. In dieser Antwort kommt ebenfalls eine checksum vor. Diese wird exakt genauso berechnet wie die checksum des Nebenkommandos. [5]

3.3.3 Beschreibung der Nebenkommandos des Atlas Digital Amplifier

Möchte man nun die Einstellungen des Atlas Digital Amplifier ändern oder Informationen aus selbigem auslesen, so muss man dafür die Nebenkommandos verwenden. Diese bestehen aus folgender Struktur: Einer checksum, einem opcode Wert und eventuell weiteren befehlsabhängigen Werten. Der opcode dient zur Individualisierung und somit zur Identifizierung des jeweiligen Befehls. Die checksum ist zur Überprüfung der Daten auf ihre Vollständigkeit. Alle weiteren Werte sind Informationen für den Atlas Digital Amplifier. Nebenkommandos können nicht alleine an den Atlas Digital Amplifier gesendet werden. Sie müssen immer zusammen mit einem Hauptkommando geschickt werden. Dazu werden die Informationen des Nebenkommandos einfach nach den Informationen des Hauptkommandos angehängt. Bei Verwendung der Nebenkommandos muss man darauf achten, dass man immer zwei Befehle, einen Schreib- und einen Lesebefehl, schickt. Der Schreibbefehl besteht aus einem Hauptkommando und einem Nebenkommando. Der Lesebefehl muss immer im Anschluss an einen Schreibbefehl gesendet werden, auch wenn man keine Informationen als Antwort erwartet. Dieser Lesebefehl besteht aus einem Hauptkommando gefolgt von Nullen. Die Anzahl der Nuller muss dieselbe sein, wie die Anzahl an Wörtern, die man als Antwort erwartet. Die Abbildung 17 stellt einen kompletten Schreibbefehl und Abbildung 18 einen vollständigen Lesebefehl, anhand des Nebenkommandos SetOperatingMode, dar.

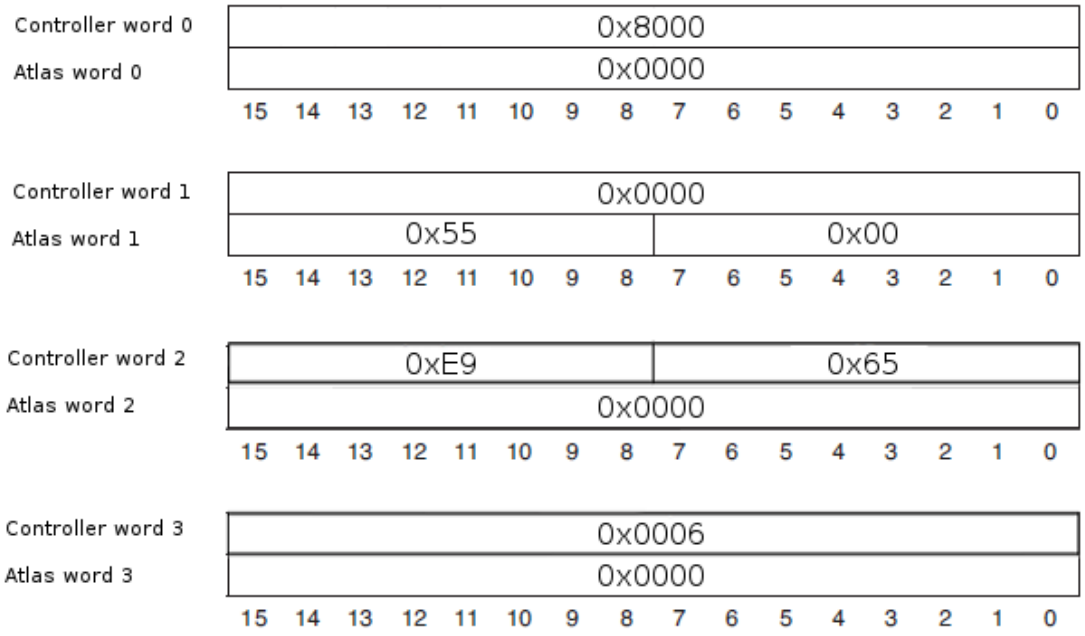


Abbildung 17: Beispiel eines Schreibbefehls mit den Werten des Nebenkommandos SetOperatingMode
(Quelle: selbst erstellt)

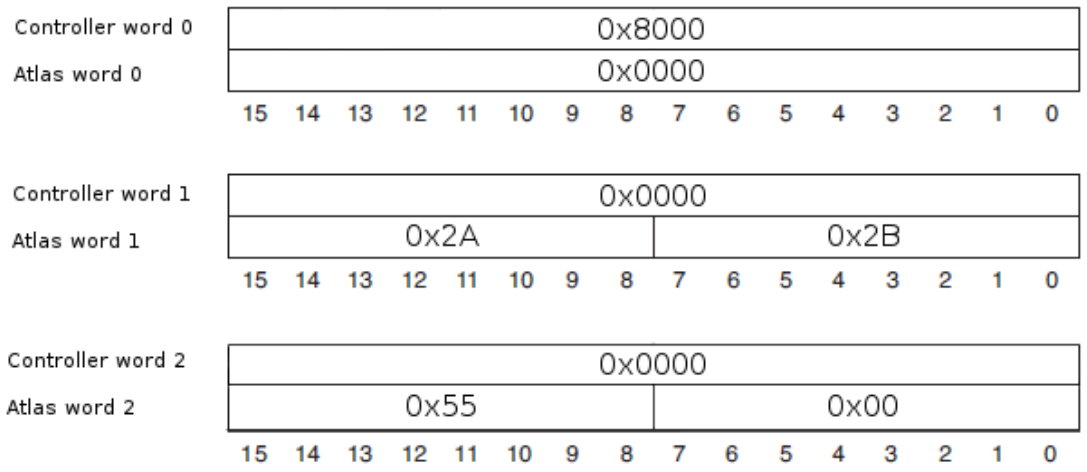


Abbildung 18: Beispiel eines Lesebefehls nach dem Schreibbefehl SetOperatingMode (Quelle: selbst erstellt)

Als Antwort auf den Nebenkommandoteil des Schreibbefehls, schickt der Atlas Digital Amplifier nur Nullen. Beim Lesebefehl wird im ersten Wort eine checksum und ein result word vom Atlas Digital Amplifier gesendet. Im result status steht die Anzahl der Wörter, die der Atlas Digital Amplifier noch senden wird. Danach folgt, je nach Nebenkommando, die Anzahl an Wörtern mit den jeweiligen Informationen.
Die checksum und der result status werden vom Atlas Digital Amplifier immer als Antwort auf ein Nebenkommando geschickt. Aus diesem Grund müssen immer ein Schreib- und ein Lesebefehl geschickt werden. [5]

Des Weiteren wurde experimentell festgestellt, dass man zwischen einem Lesebefehl und einem Schreibbefehl mindestens 30µs warten muss. Wird diese Wartezeit nicht eingehalten, so erhält man unverständliche oder falsche Antworten vom Atlas Digital Amplifier. Die Entdeckung dieser Wartezeit wird in Kapitel 5.4 genauer erklärt. Eine Zykluszeit von maximal 25µs für eine Anfrage und eine Antwort ist so nicht möglich und das erste Kriterium, welches an den Atlas Digital Amplifier gestellt wurde kann nicht erfüllt werden.

Um das zweite Kriterium zu erfüllen, würde man ein Nebenkommando benötigen, dass die Motorströme misst und diese an den externen Mikrocontroller sendet. Der Atlas Digital Amplifier ist in der Lage die Motorströme zu messen und diese, mit Kommandos wie GetFOCValue oder GetTraceValue, an den Mikrocontroller zu senden. Dafür müsste der Mikrocontroller, wie bei jedem SPI Nebenkommando, einen Schreib- und einen Lesebefehl an den Atlas Digital Amplifier schicken.

Für den Anwendungsfall in der DLR nimmt dies jedoch zu viel Zeit in Anspruch, weshalb diese Kommandos dafür nicht zu gebrauchen sind. Eine Möglichkeit an die Motorströme ohne ein SPI Kommando zu kommen gibt es beim Atlas Digital Amplifier nicht.

Außerdem wurde bei dieser Art der Kommunikation festgestellt, dass hier sehr viele Nuller und NOPs gesendet werden, um an die gewünschten Daten zu kommen. Zur Initialisierung des Atlas Digital Amplifier müssen NOPs als Hauptkommando verwendet werden. Somit werden bei Kommandos, wie zum Beispiel dem SetOperatingMode aus Abbildung 17 und 18, zwei NOPs gesendet, die jeweils 31 Bit nur mit Nullen enthalten. Des Weiteren antwortet der Atlas Digital Amplifier auf den Nebenkommandoteil des Schreibkommandos immer mit Nullen. Zuletzt muss man noch ein Lesekommando schicken, um das result word und die checksum abzuholen. Dieses Lesekommando enthält daher keine Nutzdaten für den Entwickler. Alles zusammengenommen besteht dieser Befehl nur aus 32 Bit an Nutzdaten von insgesamt 112 Bit, die an den Atlas Digital Amplifier gesendet werden. Als Antwort erhält man, abgesehen von dem SPI status word und den checksums, gar keine Nutzdaten. Somit hat man beim Senden an den Atlas Digital einen Durchsatz von 28,6% und beim Empfangen 0%. Während des Betriebs des Motors kann der Durchsatz, je nach Kommando, auf über 80% beim Senden und circa 70% beim Empfangen erhöht werden, allerdings nur auf Grund dessen, dass das NOP Hauptkommando durch das Sending a Voltage or Torque Kommando ausgetauscht wurde. Der Durchsatz der Nutzdaten bei den Nebenkommandos kann dadurch jedoch nicht erhöht werden. Dieser bleibt beim Senden bei 66% und beim Empfangen bei 33%. Eine Verbesserungsmöglichkeit wäre hier die Kommunikation im Vorfeld festzulegen, wodurch das Senden der Nuller überflüssig wird.
Ein weiterer negativer Aspekt ist, dass nach jedem Kommando das Chipselect Signal wieder zurückgesetzt werden muss. Dies bedeutet, dass zum Beispiel die Zeit zwischen einem Schreib- und einem Lesebefehl verloren geht. Diese Zeit könnte bei einer Optimierung wegfallen, oder es könnten teilweise sogar ganze Lesekommandos wegfallen.

3.4 Wichtige Kommandos zur Initialisierung des Steuermoduls

Um mit dem Atlas Digital Amplifier einen Motor steuern zu können, ist es notwendig einige Einstellungen des Atlas Digital Amplifier zu ändern. So muss dem Atlas Digital Amplifier zum Beispiel der Motortyp mitgeteilt werden und die Ausgabe der Signale an den Motor freigeschaltet werden. Zu diesem Zweck wurde eine Initialisierungsfunktion geschrieben, die alle Voreinstellungen und Abfragen auf Betriebsbereitschaft enthält. Die Erklärung der SPI Kommandos erfolgt in der Reihenfolge, wie sie auch im Programm implementiert wurden.

Zuerst wurde mit dem Kommando SetDriveFaultParameter der watchdog timer deaktiviert, da er in diesem Versuchsaufbau nicht benötigt wird. Der Atlas Digital Amplifier schaltet diesen nämlich nach dem Anlegen der Betriebsspannung ein. Weil der watchdog nicht zyklisch bedient wird, führt dies zu einem Fehler und das entsprechende Bit wird im Drive Fault Status Register setzt. Daher müssen anschließend die Bits des Drive Fault Status Registers zurückgesetzt werden. Dies geschieht mit dem ClearDriveFaultStatus Kommando. Mit dem Kommando SetDriveFaultParameter können Limits für die Parameter overvoltage, undervoltage, watchdog und temperature gesetzt oder deaktiviert werden. Außerdem kann mit diesem Kommando der recovery mode und eine temperature hysteresis eingestellt werden. Diese Werte werden von dem Atlas Digital Amplifier überwacht und im Fall einer Über- bzw. Unterschreitung wird das entsprechende Bit in einem der Status Register gesetzt. Im Fall des watchdog timers befindet sich dieses Bit im Drive Fault Status Register. Das Kommando SetDriveFaultParameter besteht aus drei 16 Bit langen Wörtern. Das erste ist für alle Nebenkommandos gleich aufgebaut. Es beinhaltet eine checksum und einen opcode. Mit Hilfe des opcodes identifiziert der Atlas Digital Amplifier das Kommando. Mit der checksum wird am Ende überprüft, ob alle Daten korrekt empfangen wurden. Mit dem zweiten Wort wählt man einen der oben genannten Parameter. In das dritte Wort wird dann der Wert geschrieben, den man als Limit setzen möchte. [5]
Die genaue Erklärung des Aufbaus solcher Kommandos und wie diese an den Atlas Digital Amplifier gesendet werden müssen, befindet sich im Kapitel 3.3. Das Kommando ClearDriveFaultStatus besteht nur aus einem 16 Bit langem Wort. Hier wird an den Atlas Digital Amplifier nur die checksum und der opcode geschickt, da man weder Informationen schicken noch welche empfangen möchte. [5]
Als nächstes muss dem Atlas Digital Amplifier mitgeteilt werden, was für eine Art von Motor angeschlossen ist. Dies geschieht mit dem Kommando SetMotorType. Das Kommando besteht nur aus zwei Wörtern. Das erste Wort ist wieder die checksum und der opcode. Im zweiten Wort steht der Motortyp. Dabei können an den Atlas Digital Amplifier drei Arten von Motoren angeschlossen werden.

Ein Brushless DC Motor, ein DC Brush Motor und ein Schrittmotor. Nachdem man den richtigen Motor ausgewählt und eingestellt hat, kann man mit dem Befehl GetMotor-Type diese Einstellung überprüfen. Dieser Befehl besteht nur aus einem Wort. Als Antwort erhält man den Motortyp der im Atlas Digital Amplifier eingestellt ist.
[5]
Das nächste Kommando bei der Initialisierung ist das ResetEventStatus Kommando. Dieses Kommando braucht man, um das Drive Exception Bit im Event Status Register zurückzusetzten. Das Drive Exception Bit wird aufgrund des Fehlers des watchdog timers gesetzt. Hierbei wurde herausgefunden, dass das Kommando nicht direkt nach dem Befehl ClearDriveFaultStatus geschickt werden kann, da der Atlas Digital Amplifier etwas Zeit benötigt um alle Zusammenhänge mit den gelöschten Bits im Drive Fault Status zu bearbeiten. Dabei werden mit diesem Befehl nicht alle Bits des Event Status Registers zurückgesetzt. Der Benutzer muss im zweiten Wort des Befehls das Bit, welches man zurücksetzen möchte, angeben. Des Weiteren können auch nur die Bits instruction error, overtemperature fault, drive exception und current foldback gelöscht werden. [5]
Danach sollte man die Ausgabe an den Motor mit dem Kommando SetOperatingMode freischalten. Bei diesem Kommando stehen einem zwei Modi zur Verfügung. Der erste ist der Motor output enabled, der zweite der current control enabled. Mit dem Motor output enabled wird der Motorausgang des Atlas Digital Amplifier freigegeben. Der current control enabled aktiviert den internen PI-Regler des Atlas Digital Amplifier. Welchen Modus man aktivieren möchte, muss im zweiten Wort des Kommandos stehen. Anschließend kann man mit dem Befehl GetOperatingMode die Einstellungen der Modi im Atlas Digital Amplifier prüfen. Dafür sendet man dem Atlas Digital Amplifier in einem Wort die entsprechende checksum und den opcode und als Antwort erhält man den eingestellten Modus. [5]
Arbeitet man mit dem internen PI-Regler, so müssen mit dem Kommando SetFOC die Werte des Reglers eingestellt werden. Das Kommando besteht aus drei Wörtern. Im ersten steht, wie gewohnt, die checksum und der opcode. Das zweite Wort besteht aus zwei Teilen. Der erste Teil besteht aus den Bits 0 bis 7. In diesem Teil wird der Regelparameter ausgewählt. Zur Auswahl stehen Proportional Gain, Integral Gain und Integral Sum Limit. Für jeden Wert, den man einem dieser Parameter zuweisen möchte, muss das Kommando SetFOC erneut geschickt werden. Das heißt, möchte man den Parametern Proportional Gain und Integral Sum Limit einen unterschiedlichen Wert zuordnen, so muss das Kommando SetFOC zweimal an den Atlas Digital Amplifier gesendet werden. Der zweite Teil des Wortes besteht aus den Bits 8 bis 11 und ist nur bei der feldorientierten Regelung von Interesse. Mit diesem Teil wählt man den Parametern direct oder quadrature.

Es ist auch möglich die gewünschte Reglereinstellung für beide Parameter gleichzeitig vorzunehmen. Anschließend wird mit dem dritten Wort der Wert für die Reglereinstellung festgelegt. [5]
Zuletzt sollte bei der Initialisierung noch einmal das Drive Fault Status Register mit dem Kommando GetDrvieFaultStatus überprüft werden. Sollte in diesem Register wieder ein Bit gesetzt worden sein während der Initialisierung, so deaktiviert der Atlas Digital Amplifier den Motorausgang und es ist kein Betrieb des Motors möglich. In diesem Fall müssen die gewählten Einstellungen nochmal geprüft werden. Im Normalfall sollte am Ende der Initialisierung kein Bit in diesem Register gesetzt sein und der Motor ist betriebsbereit.

3.5 Ausblick auf weitere Features des Atlas Digital Amplifier

3.5.1 Die Trace Funktion

Der Atlas Digital Amplifier besitzt noch zusätzliche Funktionen, wie die Trace Funktion oder den non-volatile storage (NVRAM). Mit der Trace Funktion können Parameter und Registereinträge des Atlas Digital Amplifier aufgezeichnet werden und anschließend von einem Mikrocontroller heruntergeladen werden. Dies kann sehr nützlich sein, wenn man zum Beispiel den internen Regler optimieren möchte oder Graphen erstellen möchte, bei denen eine präzise zeitbasierte Aufnahme der Werte wichtig ist. Bevor man jedoch eine Aufnahme starten kann, müssen einige Parameter der Trace Funktion eingestellt werden. Dazu gehören der Trace Puffer, die Trace Periode, die Trace Variablen, der Trace mode, der Trigger mode und der Trace Start/Stop. Für den Trace Puffer muss der Nutzer eine gültige Startadresse im RAM des Atlas Digital Amplifier und die Länge des Puffers angeben. Das Trace System unterstützt eine, vom Benutzer einstellbare, Abtastfrequenz. Diese lässt sich mit dem Trace Periode Parameter einstellen. Dabei ist zu beachten, dass sich die Trace Periode nur einstellen lässt, wenn der Trigger mode auf internal eingestellt ist. Nur dann kann der Atlas Digital Amplifier triggern. Alternative kann der Trigger mode auch auf extern gestellt werden. Dann muss der Mikrocontroller das Triggern übernehmen. Die Trace Variablen spezifizieren den Parameter, der aufgenommen werden soll. Der Atlas Digital Amplifier kann bis zu vier Parameter gleichzeitig aufnehmen. In der nächsten Abbildung ist ein Ausschnitt der möglichen Trace Variablen zu sehen.

Current Loop		
66	Phase A Reference	The current loop reference for Phase A
67	Phase B Reference	The current loop reference for Phase B
30	Phase A Error	The current loop error for Phase A
35	Phase B Error	The current loop error for Phase B
31	Phase A Actual Current	The current loop actual current for Phase A
36	Phase B Actual Current	The current loop actual current for Phase B
33	Phase A Integrator Contribution	The current loop integrator contribution for Phase A
38	Phase B Integrator Contribution	The current loop integrator contribution for Phase B
34	Phase A Current Loop Output	The current loop output for Phase A
39	Phase B Current Loop Output	The current loop output for Phase B
Field Oriented Control		
40	d Reference	The FOC reference for d (direct) loop
46	q Reference	The FOC reference for q (quadrature) loop
41	d Error	The FOC d (direct) loop error
47	q Error	The FOC q (quadrature) loop error
42	d Feedback	The d (direct) feedback current
48	q Feedback	The q (quadrature) feedback current
44	d Integrator Contribution	The FOC integrator contribution for d (direct)
50	q Integrator Contribution	The FOC integrator contribution for q (quadrature)
45	d Output	The FOC output for d (direct)
51	q Output	The FOC output for q (quadrature)
52	FOC phase A Output	The FOC output for phase A
53	FOC phase B Output	The FOC output for phase B
73	Alpha Current	The FOC α current component (stationary frame)
74	Beta Current	The FOC β current component (stationary frame)
31	Phase A Actual Current	The FOC actual current for phase A
36	Phase B Actual Current	The FOC actual current for phase B

Abbildung 19: Beispiel für mögliche Trace Variablen
(Quelle: Atlas Complete Technical Reference)

In der Abbildung sieht man auf der linken Seite eine Nummer. Dies ist die ID für die entsprechende Variable. In der Mitte steht der Variablenname und rechts eine Kurzbeschreibung der Variablen. Der Trace mode bestimmt die Handlungsweise, sollte die Aufnahme an das Ende des Trace Puffers gelangen. In diesem Fall stehen zwei Optionen zur Verfügung. Wird die one-time Methode ausgewählt, so wird die Aufnahme am Ende des Puffers beendet. Alternative dazu kann man die rolling-buffer Methode auswählen, bei der die Aufnahme erneut bei der Startadresse des Puffers beginnt, sollte das Ende des Puffers erreicht werden. Die vorherigen Aufnahmen werden dann überschrieben. Mit der rolling-buffer Methode wird kein Ende der Aufnahme erreicht. Die Aufnahme kann in diesem Fall nur vom Benutzer über den externen Mikrocontroller beendet werden.

Der Trace Start/Stop Parameter dient dazu den Anfang bzw. das Ende einer Aufnahme zu definieren. Hier stehen dem Benutzer zwei Möglichkeiten zur Auswahl. Bei der ersten reagiert der Atlas Digital Amplifier auf die SPI Kommandos SetTraceStart und SetTraceStop. Empfängt der Atlas Digital Amplifier diese Kommandos, so startet oder beendet er die Aufnahme. Bei der zweiten Möglichkeit kann der Anfang und das Ende der Aufnahme über das trace flag im SPI Packet Header gestartet bzw. beendet werden. [5]

3.5.2 Der non-volatile storage

Der Atlas Digital Amplifier bietet neben der Trace Funktion auch einen nicht flüchtigen Speicher. Auf diesen Speicher kann mit einem Mikrocontroller zugegriffen werden. Der Hauptgrund dieses Speicherelements ist, eine Initialisierungssequenz speichern zu können, die dann automatisch beim Start des Atals Digital Amplifier ausgeführt wird. Der NVRAM steht dem Benutzer jedoch frei zur Verfügung und muss nicht zwingend für die Initialisierung genutzt werden. Möchte man den NVRAM nutzen, so muss man sich an das vom Hersteller entwickelte PMD structured data storage format halten. Für das beschreiben des NVRAM steht das SPI Kommando DriveNVRAM zur Verfügung. Bei der Verwendung des DriveNVRAM muss eine bestimmte Reihenfolge an Befehlen beachtet werden, die ebenfalls vom Hersteller vorgegeben ist. Zuerst muss man den NVRAM Modus des Atlas Digital Amplifier aktivieren. Anschließend muss der komplette Speicherbereich gelöscht werden. Das Löschen oder Überschreiben von einzelnen Speichersegmenten ist nicht möglich. Danach kann der Speicher beschrieben werden. Zum Auslesen des NVRAM gibt es das SPI Kommando ReadBuffer16. Mit diesem Kommando kann immer ein 16 Bit langes Wort aus dem Speicher ausgelesen werden. Möchte man mehrere Wörter auslesen, so muss das Kommando, entsprechend der gewünschten Anzahl an zu lesenden Wörtern, wiederholt werden. [5]

Auf das Format des NVRAM wird in dieser Bachelorarbeit nicht weiter eingegangen, da dieser Speicher nicht verwendet wurde.

4 Entwurf und Anfertigung einer Adapterplatine für den STM32F746 Mikrocontroller

Um das STM32F746 Nucleo mit dem Atlas Digital Amplifier sowie dem Computer und dem Positionssensor zu verbinden, wurde mit einer Lochrasterplatine ein Steckaufsatz für das Entwicklungsboard hergestellt. Somit ist es möglich nur die notwendigen Pins des STM herauszuführen und mit den entsprechenden Steckern zu verlöten.

Für die Verbindung der SPI Schnittstelle des STM mit der SPI Schnittstelle des Atlas Moduls ist ein 9 poliger D-Sub Stecker notwendig. Er besitzt alle für die SPI Kommunikation notwendigen Leitungen. Diese sind:

- Clock (CLK)
- Master In Slave Out (MISO)
- Master Out Slave In (MOSI)
- Chip Select (CS)

Da dieser D-Sub Stecker über 9 Leitungen verfügt, für eine SPI Schnittstelle jedoch nur 4 Leitungen notwendig sind, besitzt der Stecker mehrere Chip Select Leitungen. Somit kann mit bis zu 4 Atlas Digital Amplifier über die gleichen SPI Leitungen gesprochen werden. Weil hier nur ein Atlas Modul zum Einsatz kommt und die CS Leitungen mit einer logischen Null (Low-active) aktiviert werden, muss an den Leitungen CS2, CS3 und CS4 eine logische Eins angelegt werden. Die übrigen 2 Leitungen des D-Sub Steckers sind eine Ground (GND) Leitung und eine Abschirmungsleitung. Letztere wurde in diesem Versuchsaufbau nicht verwendet.
Das Steuermodul benötigt noch zwei weitere Leitungen: eine für den Enable Pin und eine für den FaultOut Pin. Damit der Digital Amplifier das Ausgangssignal an den Motor schicken kann, ist es erforderlich eine logische Null am Enable Pin anzulegen. Der FaultOut Pin dient dazu, schnellstmöglich einen Fehler anzuzeigen. Welcher Fehler eingetreten ist, muss anschließend im DriveFaultStatus Register nachgesehen werden.

Da das Steuermodul an diesen Pins 5 Volt (V) Pegel anlegt, das Nucleo Board aber nur 3,3V Pegel an seinen Pins verträgt, wurde der Pegelwandelchip TXB0108 von Texas Instruments auf der Lochrasterplatine verbaut. Dieser kann die Amplitude der Datenbits sowohl von 3,3V auf 5V als auch von 5V auf 3,3V anheben bzw. absenken. Die Kommunikation mit dem Computerterminal erfolgt über die Universal Asynchronous Receiver Transmitter (UART) Schnittstelle des STM32F746. Hierfür wurde eine, eigens im DLR entwickelte, Adapterplatine auf dem Steckaufsatz angebracht.

Diese Adapterplatine kann die Daten von der Transmit- und Receiveleitung auf das USB Protokoll Communications Device Class (CDC) umsetzen und somit einen USB Port am Computer als Schnittstelle für den Mikrocontroller benutzbar machen.
Als nächstes muss noch der Positionssensor mit dem Nucleo verbunden werden. Der Sensor ist direkt am Motor angebracht und benötigt ebenfalls eine Adapterplatine. Diese besitzt einen 15 poligen D-Sub Stecker, über den mit dem Sensor kommuniziert wird. Der Sensor selbst muss über das BiSS Protokoll angesprochen werden, welches am Nucleo über eine SPI Schnittstelle gesendet werden kann. Bei diesem D-Sub Stecker sind nur 6 Leitungen von Bedeutung. Er besitzt eine Leitung für die Versorgungsspannung, sowie eine GND Leitung. Für die BiSS Schnittstelle sind nur 2 Leitungen notwendig eine Clock Leitung und eine Datenleitung, auf der nur die Daten vom Sensor an den Mikrocontroller gesendet werden können. Die anderen 2 Leitungen sind Sensorselect Leitungen. Diese sind für die Adapterplatine des Positionssensors wichtig, weil diese für mehrere Sensoren ausgelegt ist. Da die Sensorselect Leitung, für den in diesem Versuchsaufbau verwendeten Sensor, Low-active ist und die andere Sensorselect Leitung High-active sind, können beide Leitungen auf GND gelegt werden. Die nachfolgenden Abbildungen zeigen den für die Adapterplatine entworfenen Schaltplan. Die räumliche Anordnung orientiert sich an der Einteilung der Pins des STM32F746 Nucleo Boards.

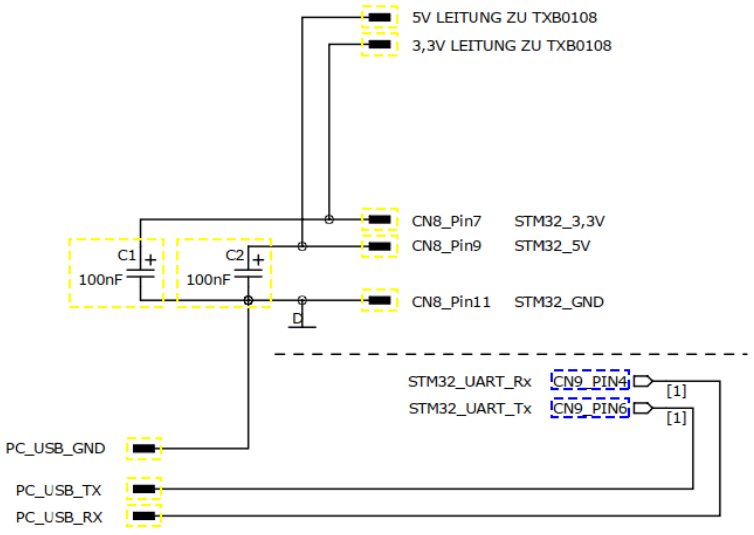


Abbildung 20: Schaltplan für Adapterplatine USB Anschluss und Versorgungsleitungen (Quelle: selbst erstellt)

In der Abbildung 20 sieht man, dass zwischen den Versorgungsspannungen und der Masse zwei Kapazitäten C1 und C2 eingebaut sind. Dies sind Bypass Kondensatoren. Mit ihnen werden hochfrequente Störungen unterdrückt und Störungen auf den Versorgungsspannungen minimiert. Es empfiehlt sich immer Bypass Kondensatoren zu verwenden, wenn man mit Motoren arbeitet, da diese Störungen auf den Leitungen verursachen können.

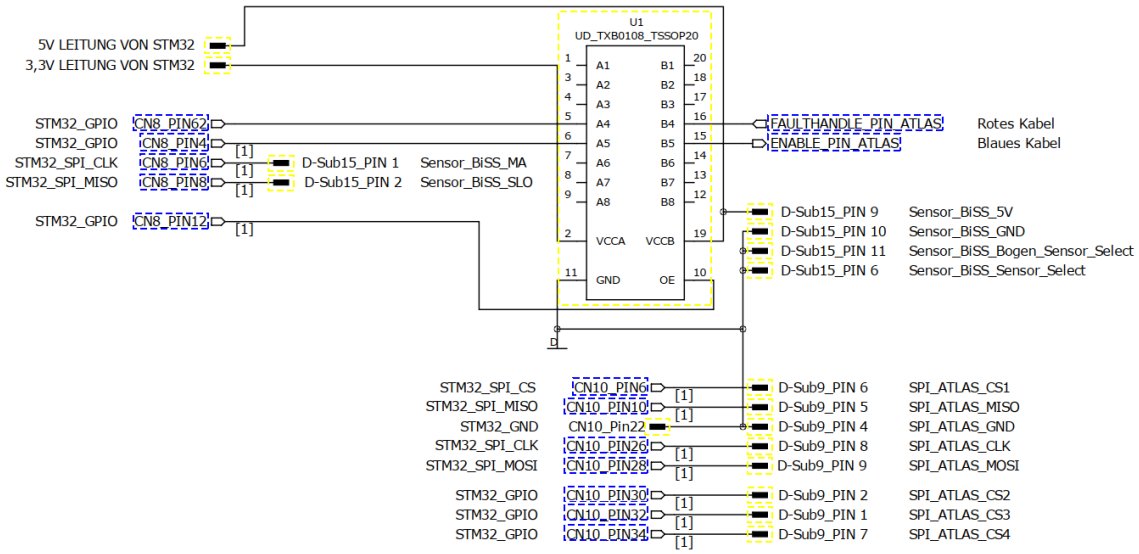


Abbildung 21: Schaltplan für Adapterplatine Pegelwandelchip und D-Sub Stecker (Quelle: selbst erstellt)

Damit auch die richtigen General Purpose Input Output (GPIO) Register programmiert werden, wurde auch hierfür ein Schaltplan erstellt. Die Abbildungen 22 und 23 zeigen, welche GPIOs Register zu den Pins des Nucleo gehören und für welche Schnittstelle sie eingesetzt wurden.

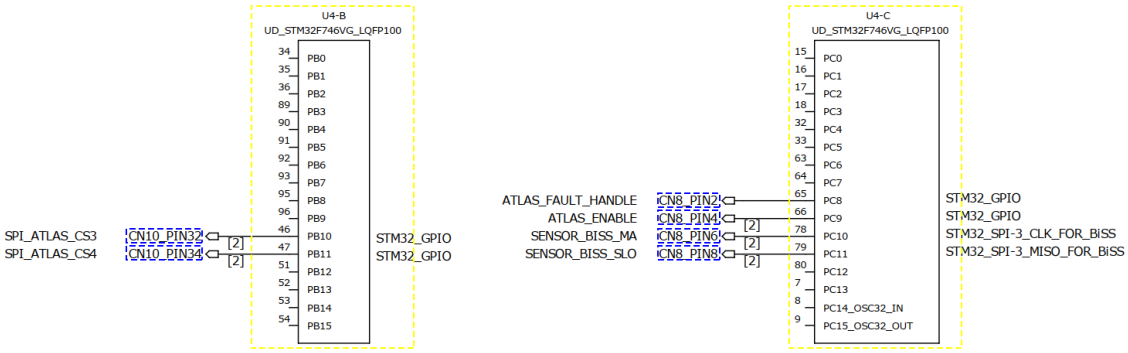


Abbildung 22: Schaltplan der GPIO Register B und C
(Quelle: selbst erstellt)

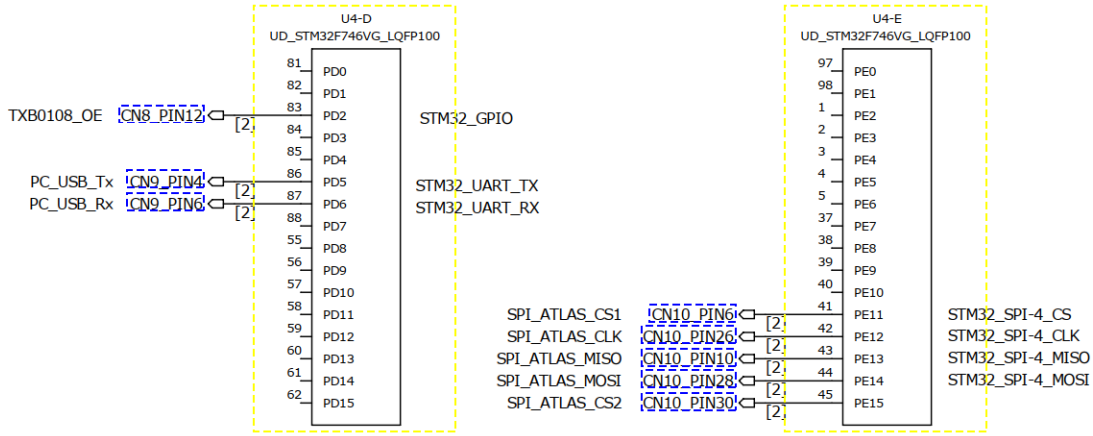


Abbildung 23: Schaltplan der Register D und E
(Quelle: selbst erstellt)

In Abbildung 23 kann man erkennen, dass am GPIO PD2 die Leitung TXB0180_OE anliegt. Diese Leitung ist die Enable Leitung für den Pegelwandelchip. Erst wenn an dieser Leitung eine logische Eins anliegt, beginnt der Chip mit der Umsetzung der Signale.

Als nächstes wurde ein Layout für den Steckaufsatz entworfen, welches als Vorlage zum Verlöten der Bauteile auf der Lochrasterplatine diente. Da sich der Schaltplan bereits an den räumlichen Gegebenheiten des STM32F746 Nucleo Boards orientiert, musste auf dem Layout nur noch die Platzierung der Leiterbahnen beachtet werden.

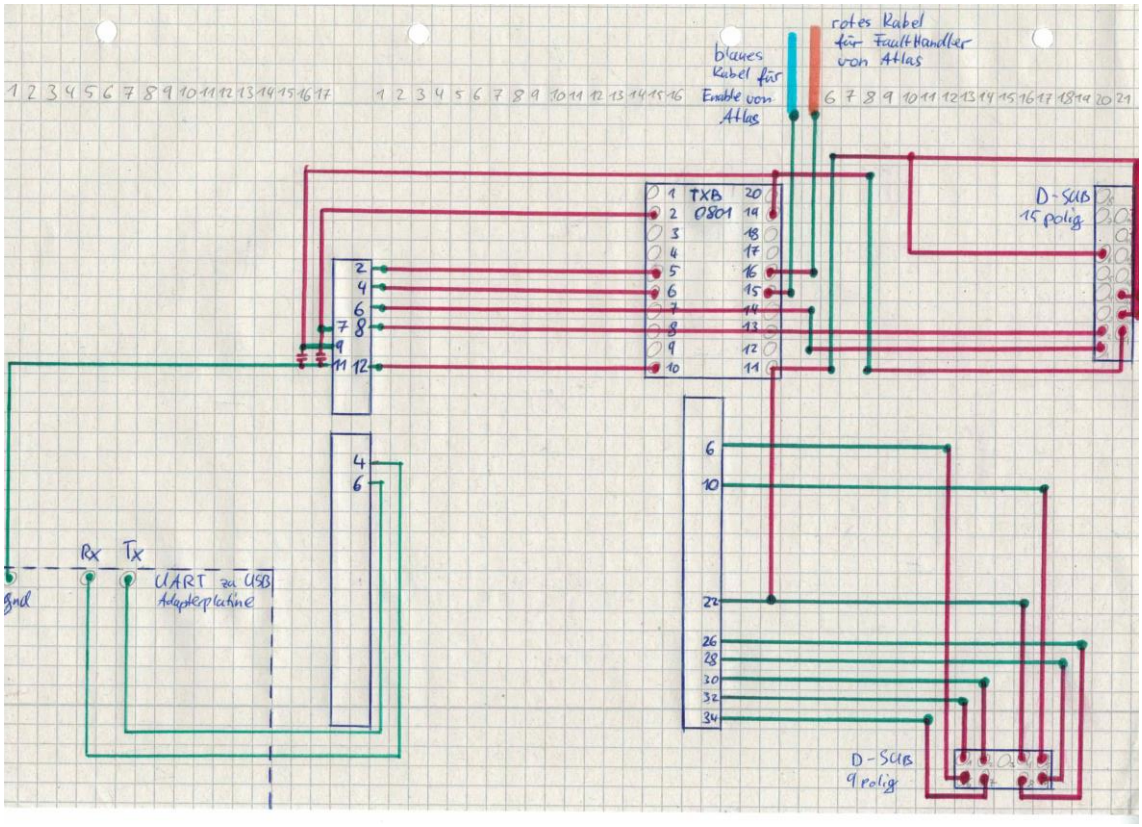


Abbildung 24: Lötvorlage für den Steckaufsatz
(Quelle: selbst erstellt)

In der Abbildung 24 sind Leitungen in unterschiedlichen Farben dargestellt. Grün bedeutet, dass diese Leitung auf der Oberseite der Platine verläuft und rot auf der Unterseite. Bei der Erstellung des Layouts wurde besonders darauf geachtet nicht zu viele Biegungen in die Leitungen machen zu müssen, da dies das Löten erschweren würde. Des Weiteren wurden die Stecker für die Kabel möglichst nahe am Rand der Platine platziert, um eine leichte Handhabung der Steckverbindungen zu ermöglichen.

Die nächsten Abbildungen zeigen die Ober- und Unterseite des fertiggestellten Steckaufsatzes, sowie den vollständigen Versuchsaufbau.

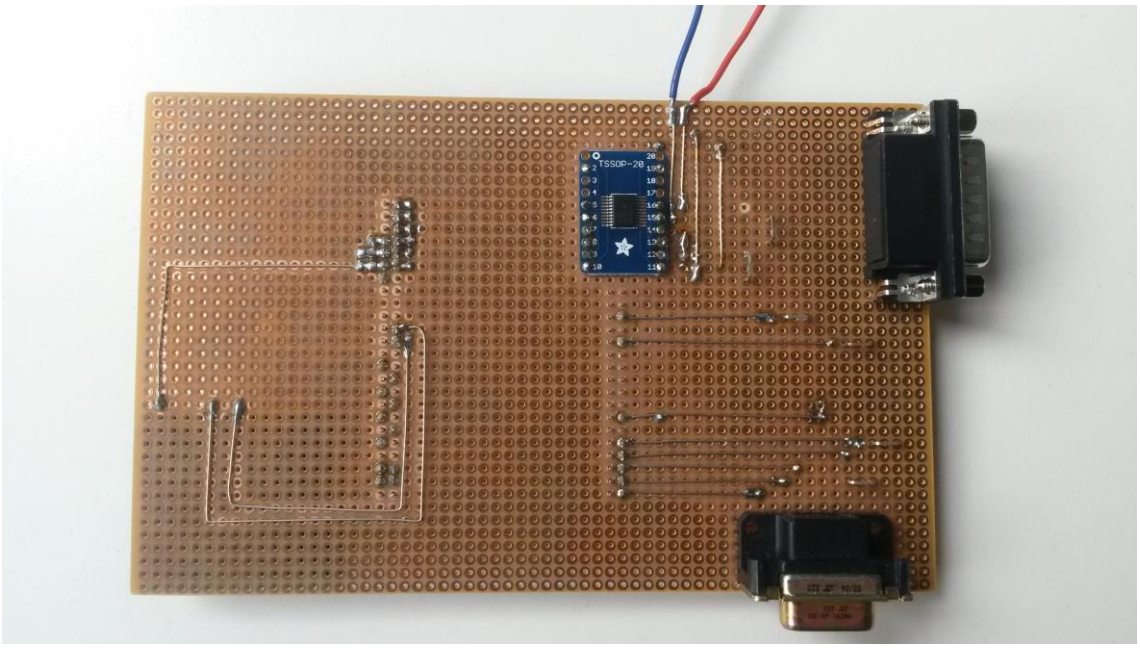


Abbildung 25: Steckaufsatz Oberseite
(Quelle: eigene Aufnahme)

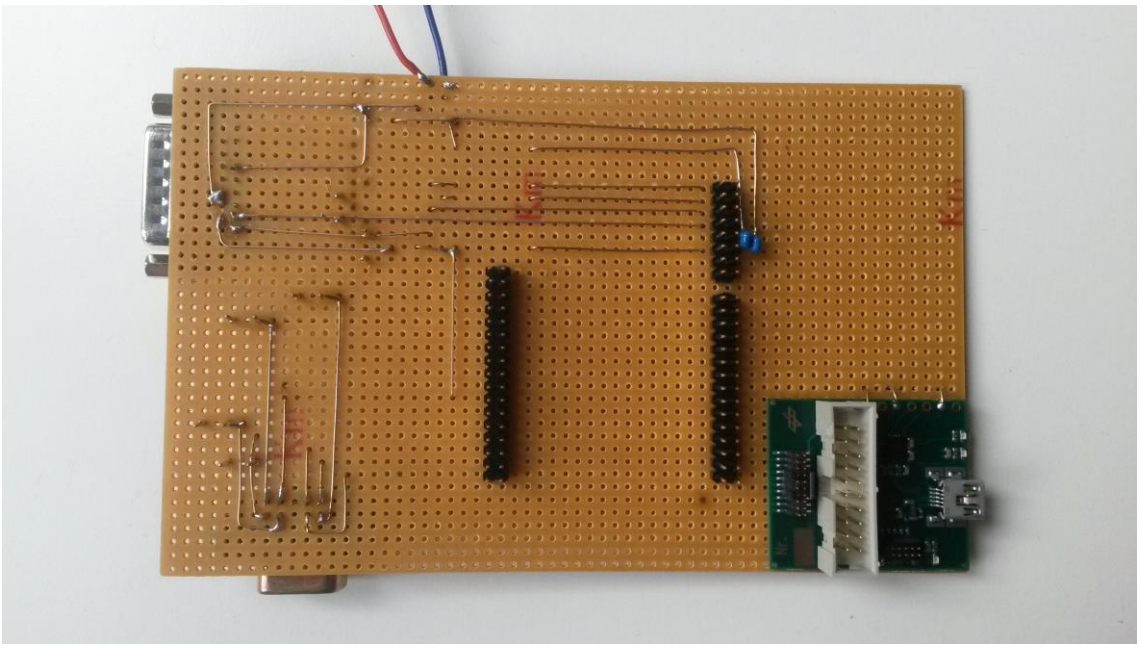


Abbildung 26: Steckaufsatz Unterseite
(Quelle: eigene Aufnahme)

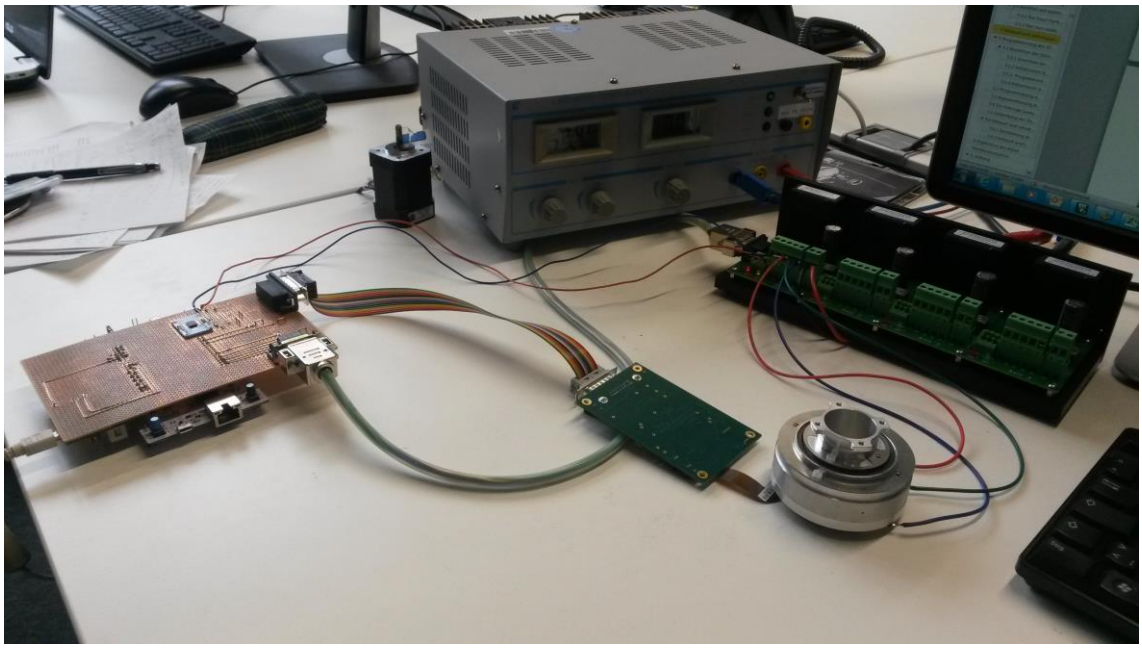


Abbildung 27: Versuchsaufbau für den Atlas Digital Amplifier in der DLR
(Quelle: eigene Aufnahme)

In der Abbildung 27 sieht man den Versuchsaufbau, um mit dem Atlas Digital Amplifier zu arbeiten. Links im Bild ist der STM32F746 mit dem Steckaufsatz zu sehen. Rechts befindet sich der brushless DC Motor mit der Adapterplatine für den Positionssensor. Hinter dem Motor stehen vier Atlas Digital Amplifier Steuermodule, von denen nur das ganz links in Betrieb genommen wurde. Im Hintergrund ist noch ein Netzteil zu sehen, dass den Atlas Digital Amplifier und den Motor mit Strom versorgt.

5 Programmierung des STM32F746 Nucleo

Für die Bachelorarbeit wurde ein STM32F746 Nucleo mit einem STM32F746ZG Prozessor verwendet. Dieses Board eignet sich besonders gut, da zu einem früheren Zeitpunkt bereits mit diesem gearbeitet wurde. Außerdem bietet es mehrere Pins für dieselben Schnittstellen und kann somit flexibel an Stecker und Adapter angepasst werden. Ein weiterer Vorteil ist, dass bei diesem Mikrocomputer mit der Entwicklerumgebung Keil Vision 5 gearbeitet werden kann. Das Programm Vision wurde bereits im Studium kennengelernt und bei der Arbeit im praktischen Semester verwendet. Für die Visualisierung der Daten auf dem Computer wurde das Programm Hyperterminal benutzt. Das Hyperterminal ist ein einfaches Ein- und Ausgabegerät, welches über eine UART Schnittstelle angesprochen werden kann. Auch dieses Programm wurde früher bereits eingesetzt und war somit schnell und einfach zu handhaben.

5.1 Einrichten der Schnittstellen des STM32

Damit überhaupt eine Art der Kommunikation möglich ist, mussten als erstes die Schnittstellen des Mikrocomputers programmiert werden. Dazu wurde der in der Entwicklungsumgebung bereits vorhandene Hardware Abstraction Layer (HAL) verwendet. Der HAL ist eine Schicht, die die Software von der Hardware abstrahiert. Damit ist es nicht mehr unbedingt notwendig die Details der Hardware Implementierung und ihre Verwendung über Registerzugriffe zu kennen. Diese Schicht bietet in einer aufbereiteten Art und Weise den Zugriff auf die Hardware, wie er durch eine Hochsprache wie C leicht möglich ist. Hauptelemente der HAL Bibliothek sind Strukturen, Funktionen, Konstanten und Definitionen, die für jede Hardwarekomponente des Mikrocontrollers existieren. Zusätzlich kann die bereitgestellte HAL Bibliothek einfach auf Mikrocontroller der gleichen Produktfamilie übertragen werden und bietet damit dem Entwickler eine gleichartige Schnittstelle zur Firmware Entwicklung. [6]

5.1.1 Einstellen der General Purpose Input Output Pins

Der STM32F746 gehört zur ARM Familie und ist stromsparend ausgelegt. Aus diesem Grund lassen sich einzelne Komponenten des Prozessors, wie den GPIOs, ein- bzw. ausschalten. Damit man mit den GPIO Port Pins arbeiten kann, muss man also, für jeden Port, als erstes die Clock aktivieren. Im Beispiel der UART Schnittstelle wurde der Port D gewählt.

Damit dieser Port nun mit der clock-Frequenz versorgt wird, kann aus der HAL Bibliothek die Funktion __HAL_RCC_GPIOD_CLK_ENABLE genutzt werden. Dies gilt analog für alle anderen Ports gleichermaßen.

Zur Programmierung der GPIOs stellt die HAL Bibliothek die Struktur GPIO_InitStruct zur Verfügung. In dieser Struktur können die Werte Pin, Mode, Pull, Speed und Alternate für jeweils einen Pin beschrieben werden. Für die UART Schnittstelle wurden der Pin PD5 für die Sendeleitung und der Pin PD6 für die Empfangsleitung ausgewählt. Somit muss die Struktur hier zweimal angelegt werden. Im Beispiel des Pins PD5 wurden folgende Einstellungen vorgenommen:

- GPIO_PIN_5
- GPIO_MODE_AF_PP
- GPIO_NOPULL
- GPIO_SPEED_HIGH
- GPIO_AF7_USART2

Im ersten Wert wird die Nummer des Pins gespeichert. Der zweite Wert besagt, dass dieser GPIO weder ein Input noch ein Output Pin ist, sondern eine alternative Funktion besitzt. GPIO_NOPULL bedeutet, dass keine pull up oder pull down Widerstände verwendet werden sollen. Der Wert GPIO_SPEED_HIGH beschreibt die Flankensteilheit am Pin und der letzte Wert spezifiziert die alternative Funktion. In diesem Fall ist es die Schnittstelle USART2. Zu beachten ist hierbei, dass USART2 und UART2 dieselben Funktionen teilen. Der Unterschied besteht darin, dass eine USART Schnittstelle zusätzlich zur UART Schnittstelle einen synchronen Teil besitzt. Zur Synchronisierung ist dann auch eine CLK Leitung notwendig.
Hat man alle Werte beschrieben, so muss man die Funktion HAL_GPIO_Init aufrufen. Dieser muss der Port und die Struktur, die man gerade mit den gewünschten Werten gefüllt hat, übergeben werden. Die Funktion schreibt die Werte dann an die richtigen Stellen in den jeweiligen Registern. [6]
Für die Programmierung der anderen Schnittstellen wurden dieselben Einstellungen vorgenommen mit einzelnen Ausnahmen. So kann man zum Beispiel die Chip Select Signale der SPI Schnittstelle auf GPIO_MODE_OUTPUT_PP setzten, damit der Benutzer diese selbst schalten kann. Der gleiche Wert muss auch für die Programmierung des Atlas Enable Pins und des Pegelwandelchip Enable Pins verwendet werden.
An dem Pin für den FaultHandle wird der Zustand des Signals eingelesen. Dazu muss man den Wert GPIO_MODE_INPUT_PP einstellen.

5.1.2 Initialisieren der UART Schnittstelle für die Kommunikation mit dem Computerterminal

Für die UART Schnittstelle, sowie für alle weiteren Schnittstellen auch, muss genauso wie bei den GPIO Pins zuerst die Clock aktiviert werden. Für den UART gibt es dazu die Funktion __USART2_CLK_ENABLE. Für die Konfiguration der UART Schnittstelle existiert ebenfalls eine Struktur wie bei der Programmierung der GPIOs. In der HAL Bibliothek lässt sich hierfür die Struktur UART_InitTypeDef finden. In dieser Struktur können folgende Werte festgelegt werden:

- BaudRate
- WordLength
- StopBits
- Parity
- Mode
- HwFlowCtl
- OverSampling
- OneBitSampling

[6]

Sowohl die Parity und die HwFlowCtl, als auch das OverSampling und das OneBitSampling sind für diese Anwendung uninteressant und wurden nicht verwendet. Die Baudrate wurde, aufgrund der schnellen Zykluszeiten, auf 921600 Bits pro Sekunde eingestellt. Dies ist die höchste Baudrate, die möglich ist. Da das Transmit data register nur 8 Bit besitzt, wurde als WordLength der maximale Wert von 8 Bit eingestellt. Des Weiteren wurde für den Wert StopBits nur ein Stop Bit und als Mode Senden und Empfangen gewählt. Hat man alle gewünschten Parameter eingestellt so muss auch hier, wie bei der Programmierung der GPIOs, eine Initialisierungsfunktion aufgerufen werden. In der HAL Bibliothek steht dazu die Funktion HAL_UART_Init zur Verfügung. Die Funktion liefert einen Integer Wert zurück. Mit diesem Wert kann überprüft werden, ob das Initialisieren erfolgreich war oder nicht.

Für diese Überprüfung wurde die Funktion assert aus der C Bibliothek verwendet. Der assert Funktion muss eine Bedingung übergeben werden. Sollte die Bedingungen erfüllt sein (TRUE) kehrt die Funktion assert unmittelbar zurück und das Programm wird weiter ausgeführt. Im anderen Fall, die Bedingung ist FALSE, erfolgt eine Ausgabe am Terminal und die Fortführung des Programms wird beendet. Damit kann die Gültigkeit einer Annahme geprüft werden und im Fall einer falschen Annahme, verweist die Ausgabe auf die Zeile im Quelltext. Das unterstützt die Entwicklung von Programmen, weil falsche Annahmen nicht übersehen werden und damit eventuell unentdeckt bleiben.
[7]

5.1.3 Programmieren der SPI Schnittstelle zur Kommunikation mit dem Atlas Digital Amplifier

Für die Kommunikation mit dem Atlas Digital Amplifier ist es erforderlich die SPI Schnittstelle zu programmieren. Dazu kann man wieder die Hilfe der HAL Bibliothek heranziehen. Dort findet man die Struktur SPI_InitTypeDef, in der folgende Parameter festgelegt werden können:

- Mode
- Direction
- DataSize
- CLKPolarity
- CLKPhase
- NSS
- BaudRatePrescaler
- FirstBit
- TIMode
- CRCCalulation
- CRCPolynomial

[6]

In diesem speziellen Fall fanden die letzten drei Parameter keine Verwendung und wurden gleich Null gesetzt. Als Mode wurde SPI_MODE_MASTER gewählt, da das Developmentboard als Master und das Steuermodul als Slave fungieren soll. Damit sowohl der Master als auch der Slave Daten senden und Daten empfangen können, muss man der Direction den Wert SPI_DIRECTION_2LINES zuweisen. Die Länge der gesendeten Daten wird im Parameter DataSize festgelegt. Wie bereits im Kapitel 3.4 erwähnt, schreibt das SPI Protokoll der Firma Performance Motion Devices vor, dass immer nur Datenpakete mit einer Größe von 16 Bit gesendet werden dürfen.

Auch muss man die Einstellung der CLKPolarity und die Einstellung der CLKPhase auf dieses Protokoll abstimmen. Diese beiden Parameter dienen dazu die Abtastung der empfangenen Daten des Mikroprozessors einzustellen. Laut SPI Protokoll muss auf die erste fallende Flanke der SPI clock abgetastet werden. Somit muss man die CLKPolarity mit dem Wert 0 und die CLKPhase mit dem Wert 1 beschreiben.

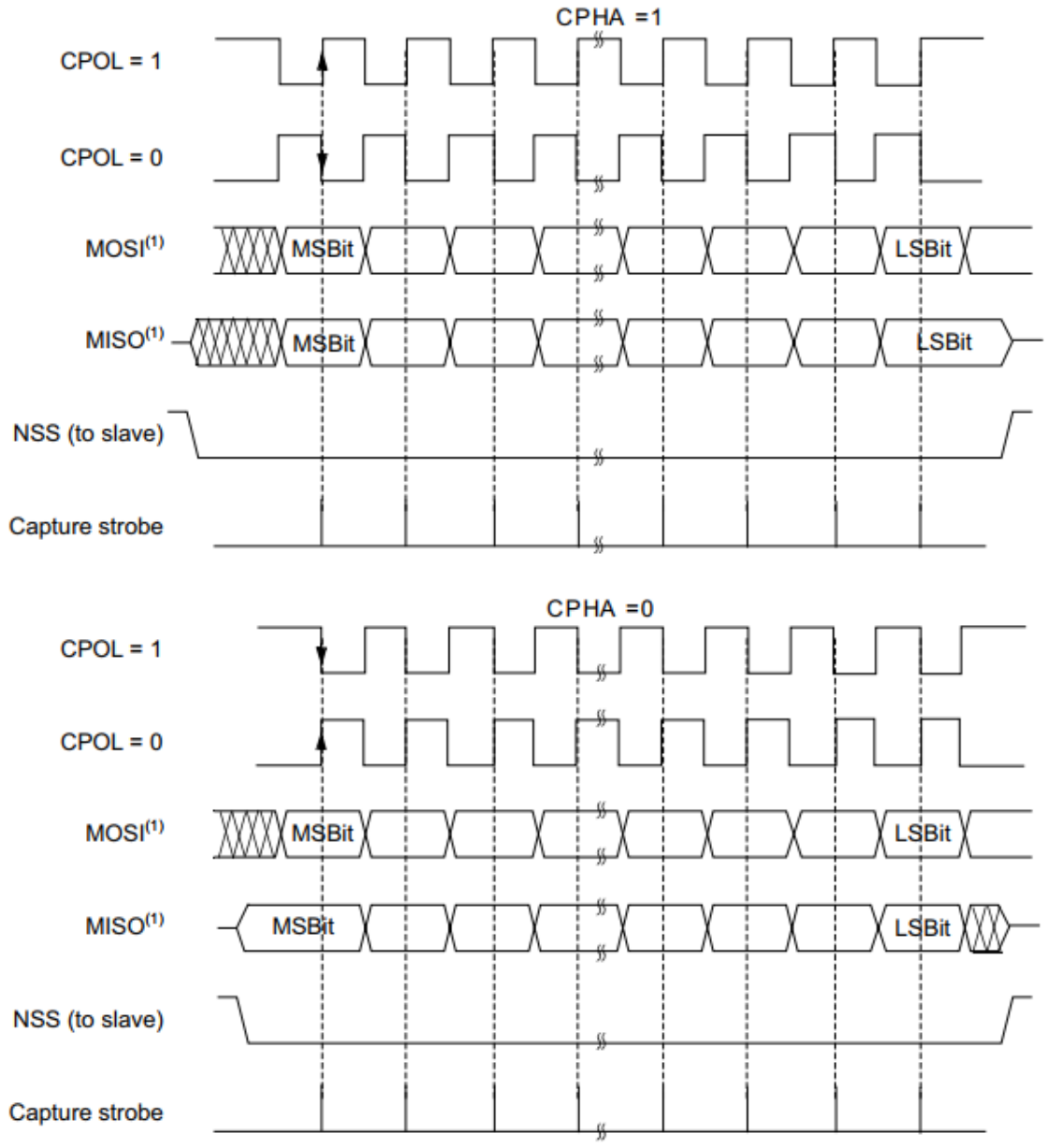


Abbildung 28: Graphik zur Einstellung der CLKPolarity und der CLKPhase (Quelle: STM32F746ZG Reference Manual, S.1058)

In der Abbildung 28 sieht man mehrere Graphen, die den Unterschied zwischen den einzelnen Clock Phasen (CPHA) und Polaritäten (CPOL) verdeutlichen sollen. Hierbei ist gut zu erkennen, dass es sehr wichtig ist die Phase und die Polarität richtig einzustellen, um immer in der Mitte des Signals abzutasten. Dass die in diesem Fall vorgenommene Einstellung von CPHA gleich 1 und CPOL gleich 0 sinnvoll ist, zeigt die nächste Abbildung. Hier wurden diese Einstellungen am Oszilloskop nachgemessen. Die weiße Linie im Bild markiert die erste fallende Flanke des Clock Signals.

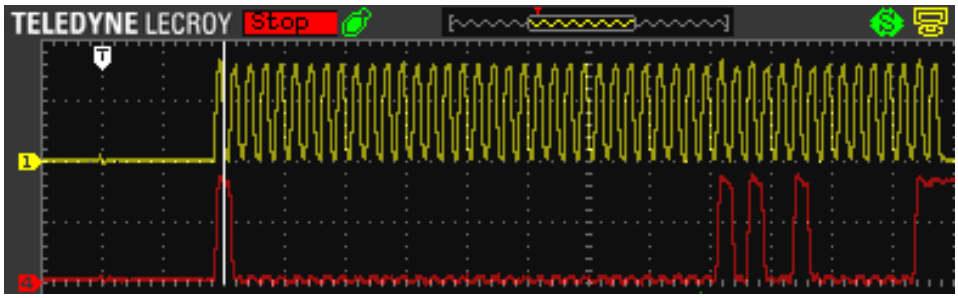


Abbildung 29: Oszilloskopbild zur Messung von CPHA und CPOL
(Quelle: eigene Aufnahme)

Die Chipselect Leitung, hier NSS, wird auf SPI_NSS_SOFT gesetzt. Damit kann der Entwickler das Chipselect Signal selbst aktivieren und deaktivieren. Dies ist Notwendig, da der Atlas Digital Amplifier, mit dem Setzten des CS Signals auf einen Low Pegel, in einen busy state wechselt und dieses auf der MISO Leitung mitteilt (siehe Abbildung 11). Erst wenn der Atlas Digital Amplifier den busy state wieder verlässt, kann mit der Kommunikation begonnen werden. Würde man das CS Signal von dem Mikrocontroller selbst setzten lassen, so würde er direkt nach dem Umschalten des CS Signals mit der Kommunikation beginnen. Der Atlas Digital Amplifier würde jedoch erst später reagieren und die Kommunikation wäre unvollständig und somit für beide Teilnehmer unverständlich.

Der BaudRatePrescaler stellt die Übertragungsgeschwindigkeit der Schnittstelle ein. Die maximale Geschwindigkeit, die der Atlas Digital Amplifier an der SPI Schnittstelle verarbeiten kann, beträgt 8MHz. Die Berechnung des dafür benötigten BaudRatePrescaler Wertes ergibt sich folgendermaßen:

$$\text{BaudRatePrescaler} = \frac{\frac{f_{CLK}}{2}}{f_{CLK\_SPI}} \Rightarrow \frac{\frac{128MHz}{2}}{8MHz} = 8$$

Möchte man also eine Frequenz von 8MHz an der SPI Schnittstelle und besitzt einen Prozessortakt von 128MHz so muss für den Prescaler der Wert 8 eingestellt werden. Die Prozessorfrequenz wurde hierfür, mit der Funktion SystemClock_Config, auf einen Wert von 128MHz herabgesetzt, damit, mit dem ganzzahligen Teiler des BaudRate-Prescalers, die maximale Frequenz von 8MHz an der SPI Schnittstelle erreicht werden kann. Mit der maximalen Prozessorfrequenz von 216MHz wären bestenfalls nur 6,75MHz möglich gewesen.
Zum Schluss muss noch der Parameter FirstBit beschrieben werden. Dieser Wert beschreibt den Startpunkt beim Lesen der Daten. Er entscheidet, ob bei dem Most Significant Bit (MSB) oder bei dem Least Significant Bit (LSB) angefangen wird.
In diesem Fall muss mit dem MSB begonnen werden und dem FirstBit der Wert SPI_FIRSTBIT_MSB zugeordnet werden.

5.1.4 Initialisieren der SPI Schnittstelle zum Empfangen der Positionssensordaten

Zur Kommunikation mit dem Positionssensor wird ebenfalls eine SPI Schnittstelle am STM32F746 benötigt. Die Programmierung dieser Schnittstelle ist nahezu identisch mit der Programmierung der SPI Schnittstelle zur Kommunikation mit dem Atlas Digital Amplifier. Aufgrund dessen, dass der ausschließlich über das BiSS Protokoll angesprochen werden muss, existieren einige Unterschiede. So werden hier lediglich zwei der vier SPI Leitungen benötigt. Diese sind die CLK Leitung und die MISO Leitung. Auch bei der Einstellung der Parameter für die Struktur aus der HAL Bibliothek gibt es wenige Änderungen. So ist es zum Beispiel notwendig die CLKPolarity auf High und die CLKPhase auf SPI_PHASE_2EDGE einzustellen. Somit beginnt die Clock mit einer fallenden Flanke und die Abtastung mit der ersten steigenden Flanke. Im Gegensatz zur Kommunikation des STM32F746 mit dem Atlas Digital Amplifier ist das Chipselect Signal hier von der Software der Schnittstelle gesetzt und auch wieder zurückgesetzt. Der letzte Unterschied ist die Übertragungsgeschwindigkeit. Hierbei wurde die schnellstmögliche Geschwindigkeit des Positionssensors eingestellt, welche ebenfalls 8MHz beträgt. Jedoch wurde für die Kommunikation mit dem Positionssensor eine SPI Schnittstelle gewählt, die für die CLK Frequenz einen zusätzlichen Teiler von 2 verwendet. Daher ergibt sich für den BaudRatePrescaler, nach derselben Berechnung wie im vorhergehenden Kapitel, ein Wert von 4. Die restlichen Parameter beinhalten dieselben Werte wie bei der Einstellung der SPI Schnittstelle zur Kommunikation mit dem Atlas Digital Amplifier.

5.2 Programmierung der Bildschirmausgabe

Um Werte des Atlas Digital Amplifier zu visualisieren und ein externes Eingabegerät einzubinden, wurde, über die UART Schnittstelle des STM32F746, eine Kommunikation mit einem Computer aufgebaut. Zur Ausgabe und Eingabe von Daten auf dem Computer wurde das HyperTerminal verwendet. Für die Ausgabe von Daten auf dem Terminal wurde eine Funktion geschrieben, die sich genauso wie die Funktion printf aus der C-Standartbibliothek benutzten lässt. Um dies zu ermöglichen wurde eine variable argument list eingesetzt. Mit Hilfe dieser Liste können der Funktion beliebig viele Parameter übergeben werden. Mit der Funktion vsprintf kann dann aus der Liste ein String erzeugt werden. [8]
Anschließend werden die einzelnen Zeichen des Strings nacheinander in das transmit data register der UART Schnittstelle geschrieben, bis das Ende des Strings, welches mit 0x00 markiert ist, erreicht wird. Die nachfolgenden Zeilen zeigen, wie solch eine Funktion aussehen könnte.

```
void SendTOTerminal( const char *format, ... )
{
  static char TransmitBuffer[1024];
  int i=0;
  va_list argptr;
  va_start( argptr, format );
  vsprintf( TransmitBuffer, format, argptr );
  va_end( argptr );

  while( TransmitBuffer[i] != 0x00 )
  {
    while ( 1 )
    {
      if (( UARTB.Instance->ISR & USART_ISR_TXE ) && (
UARTB.Instance->ISR & USART_ISR_TC ))
      break;
    }
  UARTB.Instance->TDR = TransmitBuffer[i];
    i++;
  }
}
```

Die if Abfrage dient dazu die Verfügbarkeit der UART Schnittstelle zu prüfen. Nur wenn das transmit Register leer ist und die vorherige Transaktion des Zeichens abgeschlossen ist, kann das nächste Zeichen in das transmit Register geschrieben werden. Das Empfangen von Zeichen am Mikrocontroller funktioniert auf eine ähnliche Weise. Hier wird jedoch keine variable argument list benötigt.

Die empfangenen Zeichen werden in einem Puffer zwischengespeichert, bis ein Ende der Zeichenkette empfangen wird. Das Ende der Zeichenkette wird mit dem Wert 0x0D (carriage return) gekennzeichnet. Auch das Löschen eines Zeichens mit dem backspace Wert 0x08 wurde berücksichtigt. Ist eine Zeichenkette vollständig am Mikrocontroller angekommen, so wird im nächsten Schritt untersucht, um was es sich bei der Zeichenkette handelt. Hierbei wird in zwei Kategorien unterschieden. Die erste Kategorie behandelt Befehle, die sich auf den Atlas Digital Amplifier beziehen. So kann über das Terminal zum Beispiel die Drehgeschwindigkeit oder das Drehmoment des Motors erhöht oder reduziert werden. Die zweite Kategorie behandelt Befehle, die sich an das Terminal selbst richten, um zum Beispiel den angezeigten Text auf dem Terminal zu löschen oder die Einstellungen des Displays wieder auf die Standarteinstellungen zurückzusetzten. Für diesen Fall wurden die VT100 codes implementiert, mit deren Hilfe ein Terminalfenster nach den Wünschen des Benutzers eingerichtet werden kann. So ist es möglich zum Beispiel die Farbe des Textes oder des Hintergrundes zu ändern. Des Weiteren kann die Cursor Position bestimmt und verändert werden, was sehr nützlich ist, um einen strukturierten Text zu erzeugen oder einzelne Werte im Text zu aktualisieren. Außerdem wurde eine help Funktion implementiert, die dem Benutzer anzeigt, welche Funktionen ihm zur Verfügung stehen und wie diese zu benutzen sind. In der folgenden Abbildung sieht man das HyperTerminal mit benutzerdefinierten Einstellungen.

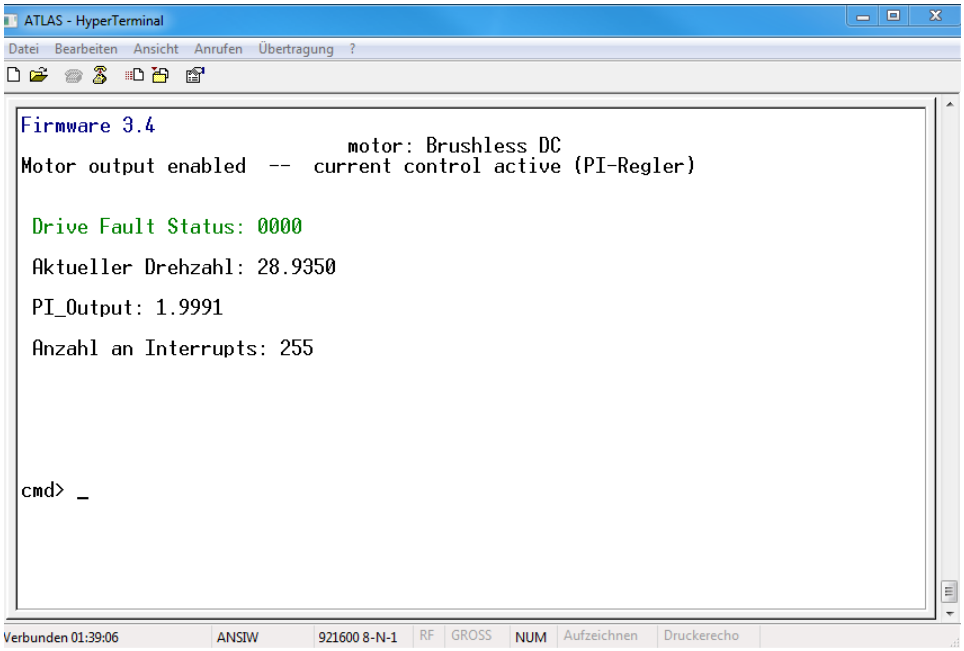


Abbildung 30: Ausgabe von Informationen auf dem HyperTerminal
(Quelle: eigene Aufnahme)

5.3 Implementierung des SPI Protokolls

Um mit dem Atlas Digital Amplifier kommunizieren zu können, muss nun das SPI Protokoll in den Mikrocontroller eingebunden werden. Dazu wird die Funktion USE_SPI_Interface erstellt, die die Informationen in die Form des Protokolls bringt. Sie sendet, überprüft die checksums und verarbeitet die empfangenen Daten. Außerdem kann diese Funktion unterscheiden, um welches Hauptkommando es sich handelt und ob es sich, bei Verwendung eines Nebenkommandos, um ein Lese- oder Schreibkommandos handelt. Somit können alle Arten von Kommandos mit nur einer Funktion an den Atlas Digital Amplifier gesendet werden. Damit dies möglich ist, muss der Funktion folgende Parameter übergeben werden:

1. Torque_OR_NOP
2. Phase_angle
3. VoltageORCurrent
4. Command_OR_Read
5. Command_opcode
6. Parameter_Array
7. Transmit_Buffer
8. Receive_Buffer
9. Number_OF_Datas_to_send

Zuerst überprüft die Funktion den selbst erstellten Sendepuffer und den selbst erstellten Empfangspuffer. Nur wenn die beiden Puffer leer sind, ist gewährleistet, dass bei der letzten Datenübertragung alle Daten gesendet wurden bzw. alle empfangenen Daten verarbeitet wurden.

Anschließend wird in der Funktion, mit dem Parameter Torque_OR_NOP, entschieden, welches Hauptkommando gesendet werden soll und dieses gleich im Sendepuffer abgespeichert. Bei einem Torque Kommando werden nun die Parameter Phase_angle und VoltageORCurrent in die Struktur des Protokolls eingefügt. Anderenfalls besitzen diese Parameter keine Bedeutung. Da bei der Berechnung der controller checksum immer das zuletzt gesendete Hauptkommando benötigt wird, wird das Hauptkommando hier zusätzlich in einem globalen Puffer gespeichert.

Möchte man eines der Nebenkommandos an den Atlas Digital Amplifier schicken, muss als nächstes, mit dem Parameter Command_OR_Read, entschieden werden, ob ein Schreib- oder Lesebefehl gesendet wird. Handelt es sich um einen Schreibbefehl, so werden nun die Daten des Nebenkommandos hinter das Hauptkommando in den Sendepuffer geschrieben. Diese Daten bestehen aus dem Command_opcode und dem Parameter_Array.

Für die Variable Command_opcode wurde, mit Hilfe von enum, eine Aufzählung erstellt, welche jedem opcode der Nebenkommandos ihren Nebenkommandonamen zuordnet. Dadurch kann zu einem späteren Zeitpunkt das jeweilige Nebenkommando wieder leichter identifiziert werden. In das Parameter_Array werden alle Zusatzinformationen eines Nebenkommandos geschrieben. Das heißt, alle Daten außer dem opcode und der checksum. So muss zum Beispiel bei dem Kommando SetMotorType der Typ des Motors in einem weiteren Wort an den Atlas Digital Amplifier geschickt werden. Diese Information steht in dem Parameter_Array und wird von der Funktion GetCommandTransmitData an die richtige Stelle im SPI Protokoll geschrieben. Als Alternative zu einem Schreibbefehl kann auch ein Lesebefehl gesendet werden. In diesem Fall müssen dann so viele Nullen an das Hauptkommando angehängt werden, wie in dem Parameter Number_OF_Datas_to_send steht.

Als nächstes findet der Datenaustausch mit dem Atlas Digital Amplifier statt. Dafür wird die Funktion SPI_Communication aufgerufen. Diese Funktion setzt das Chipselect Signal, beschreibt das Datenregister der SPI Schnittstelle und setzt anschließend das CS Signal wieder zurück. Mit Hilfe eines selbst erstellten Sendepuffers kann innerhalb der zwei CS Flanken ein vollständiges Lese- oder Schreibkommando oder auch ein alleinstehendes Hauptkommando an den Atlas Digital Amplifier gesendet werden. Gleichzeitig werden die vom Mikrocontroller empfangenen Daten in einem Empfangspuffer gespeichert.

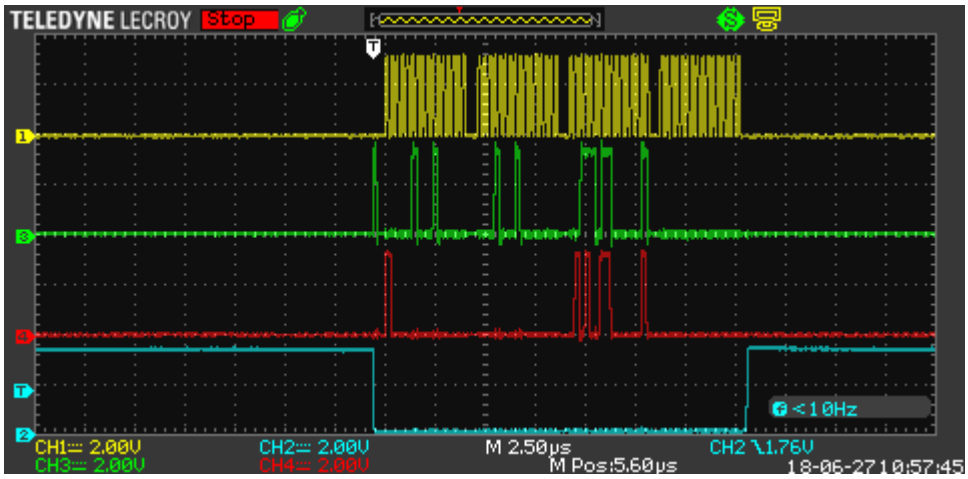


Abbildung 31: Senden eines Nebenkommandos innerhalb zweier CS Flanken (Quelle: eigene Aufnahme)

Die Abbildung 31 zeigt das Senden eines Nebenkommandos. Hierbei ist zu erkennen, dass in dem gelben Graphen, der CLK Frequenz, drei Lücken von circa 1µs sind. Diese Lücken entstehen, da der bereits vorhandene 32 Bit große Transmitbuffer der SPI Schnittstelle nur mit 16 Bit gefüllt wird, dieser dann, durch das Senden der 16 Bit an den Atlas Digital Amplifier, geleert wird und erst im Anschluss daran die nächsten 16 Bit in den Transmitbuffer geschrieben werden. Indem man den Transmitbuffer zu Beginn der Kommunikation mit zwei Wörtern, also 32 Bit füllt, kann dies optimiert werden, wie in der folgenden Abbildung zu sehen ist.

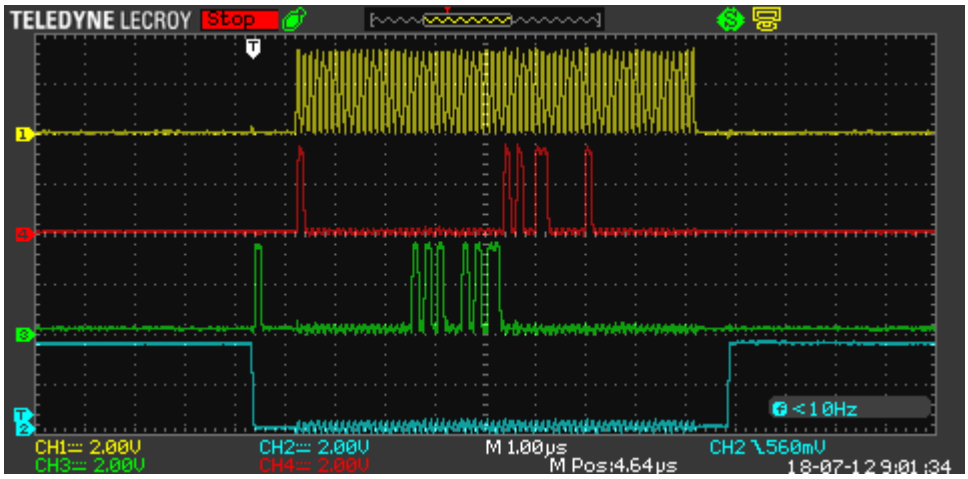


Abbildung 32: Optimierter Sendevorgang
(Quelle: eigene Aufnahme)

Somit muss der STM32F746 nach dem Senden von 16 Bit nicht erst wieder warten bis weitere 16 Bit in seinem Transmitbuffer stehen. Im Anschluss an das Senden wird er wieder auf 32 Bit aufgefüllt, bis das zu sendende Kommando komplett im Transmitbuffer steht.

Nach dem Datenaustausch wird als erstes das SPI Status word und im Anschluss die verschiedenen checksums geprüft. Die einzelnen Berechnungen der checksums und ein Beispiel für die Implementierung stehen im Kapitel 3.3.2. Stimmt eine der berechneten checksums nicht mit der empfangenen überein oder ist ein Fehlerbit in dem SPI Status word gesetzt, wird auf dem Computerterminal ein Fehler ausgegeben. Sind alle checksums korrekt und es ist kein Bit im SPI Status word gesetzt, so werden im nächsten Schritt die empfangenen Daten verarbeitet und ebenfalls auf den Computerterminal angezeigt.

5.4 Die Interrupt Service Routine des General-purpose timers

Für die eigentliche Steuerung des Motors wurde einer der general-purpose timer des STM32F746 verwendet. Mit diesem timer wurde das Kriterium zur Einhaltung einer 25µs Zykluszeit untersucht. Der timer sollte daher so eingestellt werden, dass immer nach 25µs ein Interrupt ausgelöst wird. Dazu mussten die Einstellungen des timers wie folgt programmiert werden. Der CounterMode wurde auf TIM_COUNTERMODE_UP gesetzt. Das bedeutet, dass der timer so lange hochzählt, bis er den Wert erreicht, den man in dem Parameter Period einstellt. Der Prescaler des timers gibt an, wie schnell hochgezählt wird. Der Wert wurde so eingestellt, dass ein Schritt beim Hochzählen genau einer Mikrosekunde entspricht. Der Wert wird folgendermaßen berechnet:

$$Prescaler = \frac{f_{CLK\_internal}}{f_{CLK\_TIM}}$$

Da der timer TIM2 für die internal clock auf den APB1CLK Wert aus dem RCC Register zugreift, ist der Wert nur halb so groß wie der eigentliche System clock Wert. Darum ergibt sich für den Prescaler folgender Wert:

$$\frac{(64MHz)}{1MHz} = 64$$

Mit diesem Prescalerwert kann man nun dem Period Parameter den Wert 25 übergeben und man erhält eine Zykluszeit von 25µs. Um den Start des timers besser kontrollieren zu können, wurde für diesen eine eigene Funktion geschrieben. Sobald der timer den Period Wert erreicht, wird ein Interrupt flag gesetzt und in die Interrupt Service Routine des timers gesprungen. Anschließend wird der timer wieder zurückgesetzt und beginnt erneut hochzuzählen.

Durch die Variation des Parameters Period konnte festgestellt werden, dass der Atlas Digital Amplifier mindestens eine Zykluszeit von 30µs benötigt, um seine Daten verarbeiten zu können und wieder für das nächste Kommando empfangsbereit zu sein. Somit kann der Atlas Digital Amplifier das erste Kriterium nicht erfüllen. Damit jedoch weiterhin noch mit dem Atlas Digital Amplifier gearbeitet werden kann, wurde eine Zykluszeit von 100µs eingestellt.

In der Interrupt Service Routine wird dann die eigentliche Motorsteuerung ausgeführt. Um den Motor mit einer konstanten Drehzahl drehen zu lassen, muss der phase angle Wert des Torque Kommandos kontinuierlich um denselben Wert erhöht werden. Bei einem gleichbleibendem Wert für das Drehmoment erhält man somit schon eine Drehung des Motors. Für eine präzise Steuerung wurde im nächsten Schritt die Interrupt Service Routine um die Messung der Position des Motors und damit auch um die Berechnung der Drehzahl des Motors erweitert. Anschließend wurde noch der PI-Regler in die Interrupt Service Routine mit aufgenommen. Der Aufbau und die Implementierung des Positionssensors, der Drehzahlberechnung und des Reglers werden in den nachfolgenden Kapiteln erklärt.

5.5 Einbindung des Positionssensor

Für die Messung der Position wurden zwei ARM Sensoren von Sensitec verwendet, welche jeweils über einen iC-NQ Chip vom Hersteller iC-Haus ausgelesen und ausgewertet werden. Des Weiteren werden zwei Magnetringe, M und N, mit alternierenden Magneten benötigt. Dadurch, dass einer der beiden Ringe mehr Magnete besitzt als der andere, existiert eine Grob- und eine Feinspur. Der Zusammenhang zwischen der Anzahl der Magnete beider Spuren ergibt sich wie folgt:

$$N = M + 1$$

Jeder dieser Sensoren liefert eine absolute Position innerhalb eines Magneten. Daraus resultieren zwei Messwerte, die nach dem Noniusverfahren miteinander verrechnet werden können.

Für das Noniusverfahren ist es notwendig die Anzahl an Magneten innerhalb einer elektrischen Umdrehung zu kennen. Daher wurde eine Reihe von Positionswerten gemessen und ein Graph erstellt, der diese Werte visualisiert.

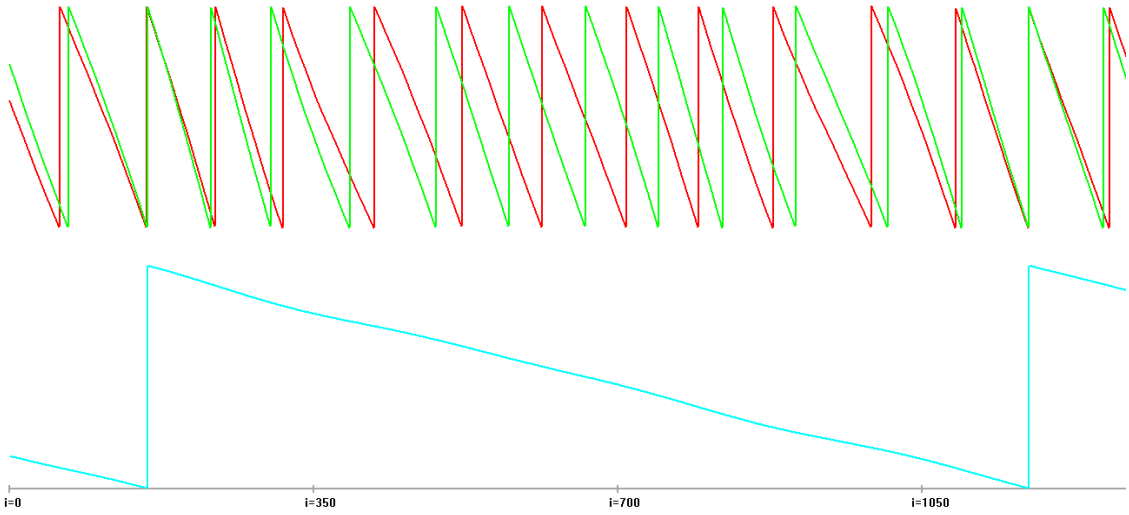


Abbildung 33: Bestimmung der Anzahl an Magneten pro elektrischer Phasendrehung (Quelle: selbst erstellt)

In der Abbildung 33 sind drei Graphen zu erkennen. Der grüne Graph zeigt die Werte der Feinspur N und der rote die der Grobspur M. Dabei steht jeder Sägezahn für einen Magneten. Der blaue Graph zeigt den absoluten Positionswinkel des Motors innerhalb einer elektrischen Phasendrehung. Um nun die Anzahl der Magneten innerhalb einer elektrischen Umdrehung zu bestimmen, muss also die Anzahl an Sägezähnen gezählt werden. Gezählt wird ab da, wo der rote und der grüne Graph exakt übereinander liegen bis zu der nächsten Überschneidung der beiden Graphen. Jede dieser Überschneidungen kennzeichnet eine vollständige Phasendrehung. Daraus ergibt sich für die Feinspur eine Anzahl von 12 und für die Grobspur eine Anzahl von 11 Magneten.

Zu beachten ist hierbei, dass die Graphen von Natur aus nicht genau übereinander liegen und dies per Software nachjustiert werden muss. Dazu sucht man den geringsten Abstand der beiden Werte und verwendet diesen Abstand als konstanten Offset, um eine der beiden Spuren zu justieren. Dadurch wird erreicht, dass beide Spuren einen definierten Anfang und Ende kennen und das Noniusverfahren die bestmöglichen Positionswerte liefert. Beachtet man dies nicht, so wird die Berechnung der Position fehlerhaft, wie in der folgenden Abbildung zu sehen ist.

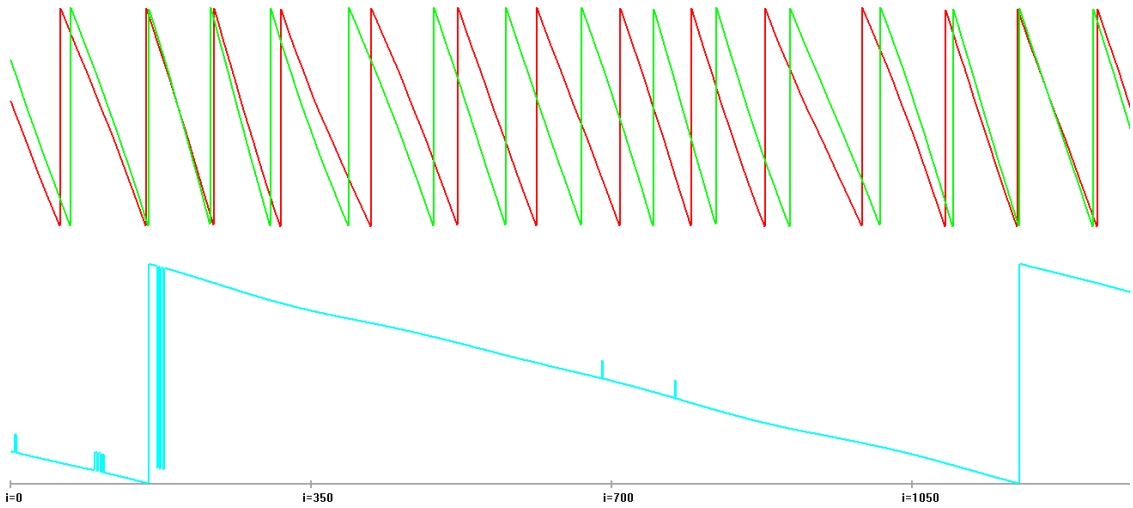


Abbildung 34: Bestimmung der Anzahl an Magneten pro elektrischer Phasendrehung ohne Offset
(Quelle: selbst erstellt)

Anhand dieser Messreihen konnte ebenfalls festgestellt werden, dass die Rotation des Motors und damit die Positionsmessung erheblich gleichmäßiger wird, je höher das Drehmoment des Motors ist. Da dies Drehmoment im Zusammenhang mit dem Motorstrom steht, wurden zwei Messreihen mit unterschiedlichen Motorströmen bei konstanter Rotationsgeschwindigkeit aufgenommen. Dabei wurden die Werte in Abbildung 33 und 34 mit einem Motorstrom von 3A und die Werte aus Abbildung 35 mit einem Strom von 1A aufgenommen.

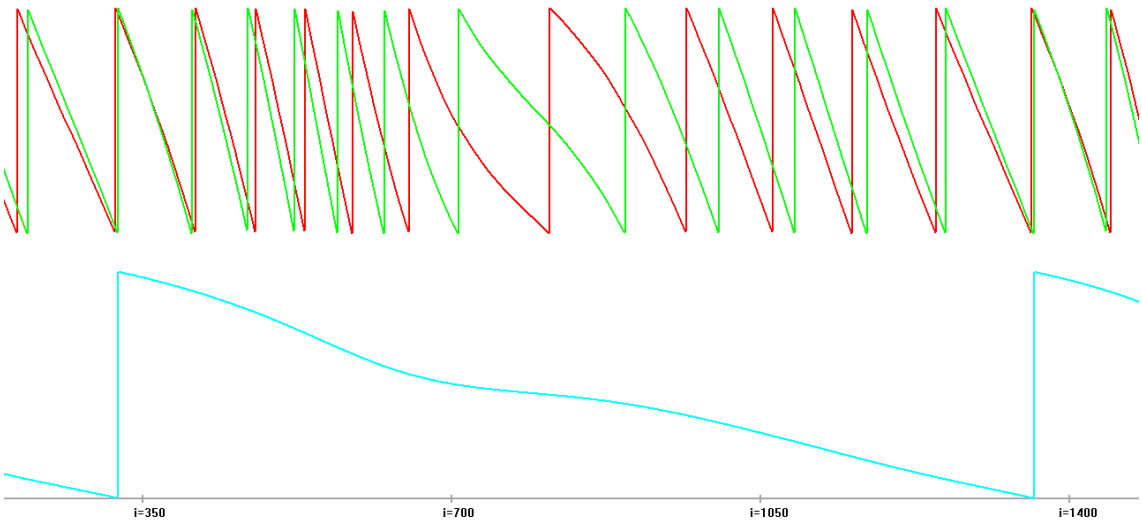


Abbildung 35: Positionsmessung mit 1A Motorstrom
(Quelle: selbst erstellt)

Vergleicht man die blaue Kurve der Abbildungen miteinander, so ist zu erkennen, dass die Kurve bei der Messung mit 1A Motorstrom deutlich geschwungener ist als die mit 3A Motorstrom. Daraus lässt sich schließen, dass an dem Motor, für einen gleichmäßigen Betrieb, ein möglichst hoher Strom angelegt werden sollte.

Um die Sensordaten aus dem iC-NQ Chip auslesen zu können, sollte das von der Firma iC-Haus herstellereigene BiSS Protokoll verwendet werden. Das Auslesen über eine SPI Schnittstelle wäre ebenfalls möglich, ist aber mit einer wesentlich niedrigeren Übertragungsrate verbunden.

Das BiSS Protokoll ermöglicht ein gleichzeitiges Senden der Sensordaten von allen angeschlossenen Slaves an einen Master. In diesem Fall gibt es einen Master, den Mikrocontroller, und zwei in Kette geschaltete Slaves, die beiden iC-NQ Chips. Diese Konstellation wird point-to-point configuration genannt. Die Kommunikation der Teilnehmer beginnt mit der Übertragung der CLK Frequenz vom Master an den Slave. Dieser startet dann mit der ersten steigenden Flanke der CLK Frequenz die Messung. Mit der zweiten steigenden Flanke sendet der Slave mindestens ein Acknowledge Bit, indem er seine Slave Out Leitung auf 0 setzt. Benötigt einer der beiden Slaves mehr Zeit, zum Beispiel für die Analog-Digital-Konvertierung oder den Speicherzugriff, so sendet er mehrere Acknowledge Bits. Anschließend folgt eine 1 als Startbit und eine 0 als Zerobit. Danach kommen die Sensordaten als zwei Bytewerte. Nach den Daten setzt der Sensor seine Output Leitung wieder auf eine 1 und ist bereit für die nächste Abfrage. Die nachfolgende Abbildung soll die Bitreihenfolge des BiSS Protokolls noch einmal veranschaulichen. [9]

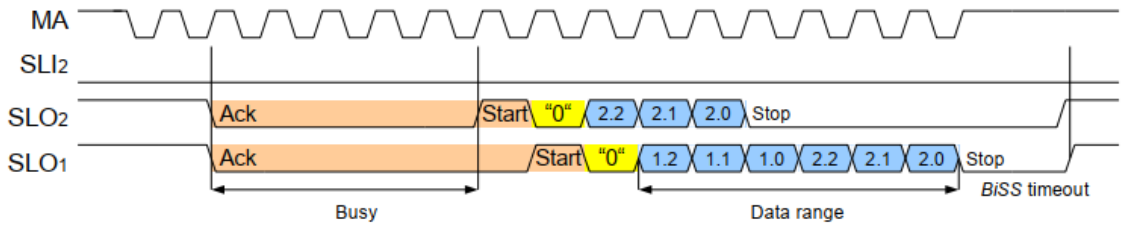


Abbildung 36: Bitreihenfolge des BiSS Protokolls
(Quelle: BiSS C unidirectional protocol, S.3)

In der Abbildung ist zusätzlich zu der CLK und der Slave Output Leitung noch eine Slave Input Leitung (SLI) zu sehen. Bei der point-to-point configuration der beiden verketteten Slaves, wird die SLI Leitung beim letzten BiSS Teilnehmer auf Masse gelegt. Das markiert das Ende der Kette.

Dies ist eine allgemeine Beschreibung des BiSS Protokolls. Der genaue Aufbau der übertragenen Daten wird im iC-NQ Chip festgelegt. Insgesamt senden beide Chips zusammen mindestens 36 Bits in der Reihenfolge, wie sie in der Abbildung 37 zu sehen ist. Außerdem kann man in dieser Abbildung erkennen, dass die Null nach dem Start Bit fehlt. Dies liegt daran, dass der iC-NQ Chip das BiSS B Protokoll verwendet und nicht das BiSS C. Der Rest ist in den beiden Protokollarten gleich.

IDLE	ACK	START	ANGLE1	ERR1	CRC1	ZERO1	ANGLE2	ERR2	CRC2	ZERO2	IDLE
		1	8	2	5	1	8	2	5	1	

Abbildung 37: Bitreihenfolge des Positionssensors
(Quelle: selbst erstellt)

Da dies zu viele Bits sind um sie in einer integer Variablen zu speichern, werden die Daten zunächst in zwei integern gespeichert. Um mit nur einer Variablen arbeiten zu können, wird die erste Variable solange nach links geschoben, bis alle IDLE und ACK Bits, sowie das Start Bit nicht mehr in der Variablen stehen. Dadurch entsteht ausreichend Platz, um die restlichen Bits aus der zweiten Variablen mit einer bitweisen ODER Operation in die erste Variable zu übernehmen. Somit stehen alle wichtigen Daten nun in einer Variablen. Als nächstes wird mit dem CRC25 Verfahren untersucht, ob die Prüfsumme korrekt ist. Die CRC25 Berechnung bezieht sich dabei jeweils auf die 8 Bits von ANGLE und die 2 Bits von ERR. Anschließend werden die Bytewerte ANGLE1 und ANGLE2 ermittelt und an die Nonius-Funktion für die Berechnung der Position übergeben.

5.6 Entwurf und Umsetzung einer Drehzahlregelung

5.6.1 Bestimmung der Übertragungsfunktion

Für die Reglungstechnik wird ein System mit einer Übertragungsfunktion beschrieben. Diese beinhaltet alle Parameter des offenen Regelkreises. In dieser Bachelorarbeit besteht der offene Regelkreis aus dem Regler selbst, dem Atlas Digital Amplifier und dem Brushless DC Motor. In Abbildung 38 ist dieser Schematisch dargestellt.

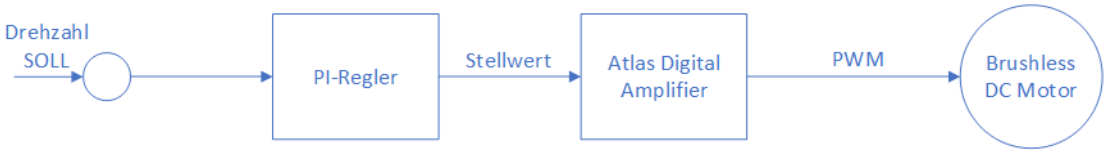


Abbildung 38: offener Regelkreis
(Quelle: selbst erstellt)

Um nun aus diesem Regelkreis die Übertragungsfunktion aufstellen zu können, müssen zuerst die Gleichungen der einzelnen Blöcke bestimmt werden. Die Übertragungsfunktion des gesamten Systems ergibt sich dann aus dem Produkt der Funktionen der einzelnen Blöcke:

$$G_{ges} = G_{Regler} \cdot G_{Atlas} \cdot G_{Motor}$$

Für den PI-Regler gilt die allgemeingültige Gleichung:

$$G_{Regler} = K_R + \frac{K_R}{T_R \cdot s}$$

Der Atlas Digital Amplifier und dessen Motorsteuerung verzögert die Strecke und hat sonst keinen weiteren Einfluss. Daher kann er, wie ein Totzeitglied, mit folgender Funktion betrachtet werden:

$$G_{Atlas} = \frac{1}{1 + T_{tot} \cdot s}$$

Die Funktion des Brushless DC Motors lässt sich aus seinem Ersatzschaltbild ableiten. Dieses besteht aus einer Sternschaltung mit drei Verzweigungen, bei der pro Zweig jeweils ein Widerstand, eine Spule und eine Spannungsquelle in Reihe geschaltet sind, wie in Abbildung 39 zu sehen ist.

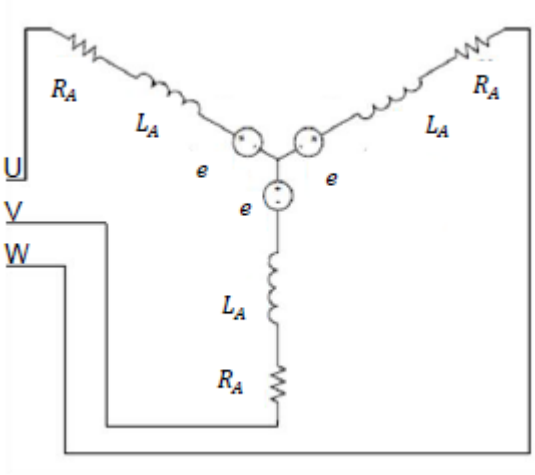


Abbildung 39: Ersatzschaltbild des Brushless DC Motors
(Quelle: Kasper Johanna, Sensorgestützte Ansteuerung eines BLDC-Motors, S.43)

Die Funktion des Motors lässt sich nun bestimmen, indem man die Eingangsgröße, die Motorspannung, in Bezug zur Ausgangsgröße, dem Motorstrom, betrachtet. Dadurch erhält man die Gleichung:

$$G_{Motor} = \frac{K_M}{1 + T_M \cdot s}$$

mit den Motorparametern:

$$K_M = \frac{1}{R_M}$$

und

$$T_M = \frac{L_M}{R_M}$$

Da man nun die Funktionen aller Blöcke bestimmt hat, lässt sich jetzt die Übertragungsfunktion des Gesamtsystems aufstellen. Diese lautet:

$$G_{ges} = \frac{K_R \cdot (T_R \cdot s + 1) \cdot K_M}{(T_R \cdot s) \cdot (1 + T_{tot} \cdot s) \cdot (1 + T_M \cdot s)}$$

Mit dieser Übertragungsfunktion kann man nun einen Regler für die Drehzahl auslegen. [10]

5.6.2 Entwurf und Implementierung des Drehzahlreglers

Für den Entwurf eines Drehzahlreglers, muss zunächst die Drehzahl berechnet werden. Dazu muss zu Beginn des Programms die Interrupt Service Routine zweimal durchlaufen werden, um die Positionswerte aufzunehmen. Danach wird immer der zuletzt gemessene und der aktuell gemessene Wert genommen. Von dem aktuellen Wert wird dann der ältere abgezogen und nach der Zeit abgeleitet. Da die Zeit von einem Positionswert zum nächsten immer exakt der Zeit zwischen zwei Interrupts entspricht, lässt sich diese leicht über den Period Wert des timers bestimmen. Der daraus resultierende Wert beschreibt die Drehzahl des Motors und dient als Eingabewert für den PI-Regler. Im Anschluss daran wird der Regler programmiert und in die Interrupt Service Routine eingebunden.

Als erstes berechnet man dazu, wie bei jedem Regler, die Regeldifferenz. Danach wurde die Differenz mit dem Verstärkungsfaktor des Proportionalteils multipliziert und in einer Variablen gespeichert. Diese Variable teilt man anschließend durch den Integralanteil des Reglers und addiert sie auf eine, mit Null initialisierte, globale Variable. Zuletzt müssen der P Anteil und der I Anteil noch miteinander addiert werden und man erhält einen neuen Drehzahlwert. Der neue Wert wird an den Atlas Digital Amplifier gesendet, die Drehzahl wird erneut gemessen und wieder an die Reglerfunktion übergeben. Somit wird der Regelkreis geschlossen. Die Abbildung 40 zeigt eine Visualisierung des Regelkreises.

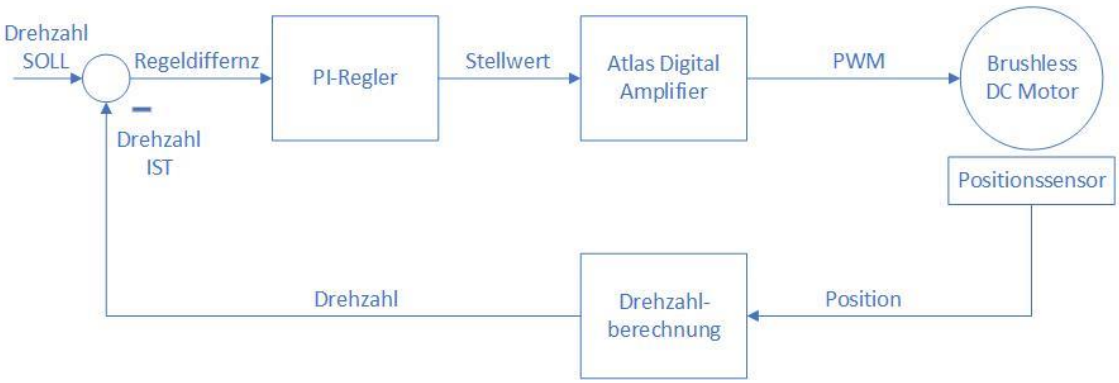


Abbildung 40: geschlossener Regelkreis zur Drehzahlregelung
(Quelle: selbst erstellt)

Da die Parameter des Motors, also die Widerstands- und Spulenwerte des Motors, nicht bekannt sind, kann der Regler nicht mit der Übertragungsfunktion des offenen Regelkreises ausgelegt werden und es muss auf ein praktisch orientiertes Verfahren zurückgegriffen werden.

Zur Einstellung der Regelparameter wurde daher nach dem Ziegler/Nichols Verfahren vorgegangen. Als erstes wird dazu nur der Verstärkungsfaktor K_p betrachtet. Dieser muss solange erhöht werden, bis eine Dauerschwingung des Systems erreicht wird. Der somit ermittelte Verstärkungsfaktor wird K_{p_krit} genannt. Anschließend wird die Periodendauer der Dauerschwingung gemessen und man erhält den Wert T_{krit}. Damit hat man die Regelparameter bestimmt, die an der Stabilitätsgrenze des Regelkreises arbeiten. Um nun die Parameter für einen stabilen Regler zu erhalten, müssen die neuen Parameter nach folgender Tabelle berechnet werden. [11]

Tabelle 6: Regelparameterauslegung nach Ziegler/Nichols

Parameter	P-Regler	PI-Regler	PID-Regler
K_{PR}	$0{,}5 \cdot K_{PRkr}$	$0{,}45 \cdot K_{PRkr}$	$0{,}6 \cdot K_{PRkr}$
T_n	-	$0{,}83 \cdot T_{kr}$	$0{,}5 \cdot T_{kr}$
T_v	-	-	$0{,}125 \cdot T_{kr}$

(Quelle: Reuter Manfred und Zacher Serge, Regelungstechnik für Ingenieure, S.224)

Aus Sicherheitsgründen wurde der Regler zuerst in Excel berechnet und so die theoretischen Werte bestimmt. Dazu wurde für den Verstärkungsfaktor des Reglers mit einem Wert von 1 begonnen. Dabei erhielt man bereits ein sehr gutes Dauerschwingungsverhalten, wie die nachfolgende Abbildung zeigt.

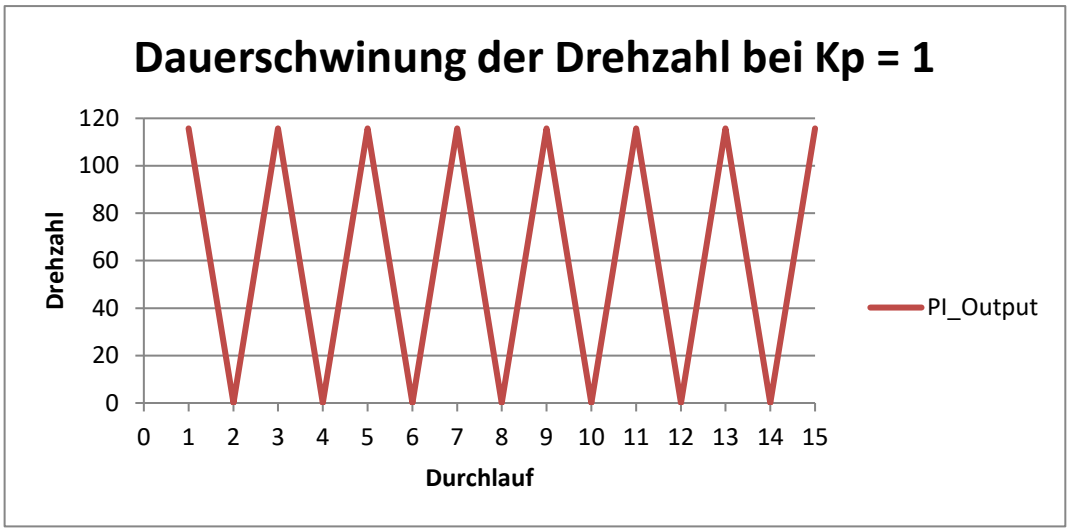


Abbildung 41: Dauerschwingung bei einem Wert von $K_p = 1$
(Quelle: selbst erstellt)

Dadurch, dass die die Einstellung der Drehzahl in der Interrupt Service Routine stattfindet, beträgt die Zeit von einem Durchlauf zum nächsten genau die Periodendauer des timers und somit 100µs. Die Periodendauer der Dauerschwingung lässt sich daher leicht bestimmen. Sie beträgt genau zwei Durchläufe der Interrupt Service Routine und man erhält für T_{krit} einen Wert von 200µs. Verwendet man nun die Formeln zur Bestimmung der Regelparameter nach Ziegler/Nichols, so ergeben sich folgende Werte:

$$K_p = 1 \cdot 0{,}45 = 0{,}45$$

$$T_n = 0{,}83 \cdot 200 \cdot 10^{-6} = 1{,}66 \cdot 10^{-4}$$

Mit diesen Werten konnte ein Regler erstellt werden, der den Soll-Wert nach circa 30 Durchläufen erreicht, wie in der Abbildung 42 zu sehen ist.

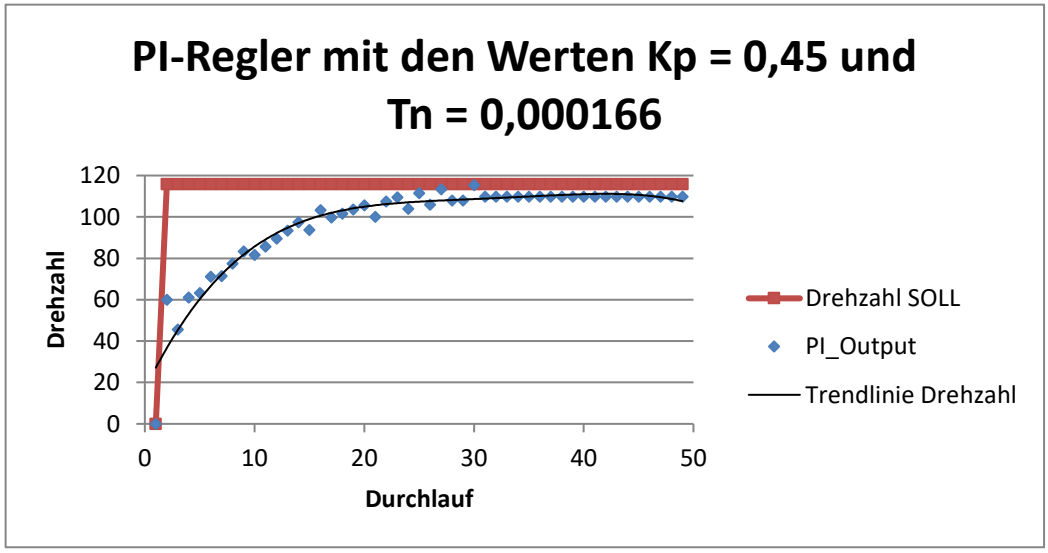


Abbildung 42: PI-Regler ausgelegt nach Ziegler/Nichols
(Quelle: selbst erstellt)

Da bei 30 Durchläufen der Soll-Wert erst nach 3ms erreicht wird, wurde der Regler, ausgehend von den bereits berechneten Werten, weiter optimiert. Dabei wurden die Parameter des Reglers solange variiert, bis sich ein möglichst schnelles Einstellen des Soll-Wertes zeigte. So ergaben sich für den Proportionalanteil ein Wert von K_p = 0,5 und für den Integralanteil ein Wert von $T_n = 25 \cdot 10^{-6}$. Das optimierte Verhalten wird in der Tabelle 7 noch einmal dargestellt und in Abbildung 43 visualisiert.

Tabelle 7: Optimierung der Regelparameter

Durchlauf	PI_diff	PI_P	PI_I	PI_Out	Motospeed out PI	Motorspeed out Atlas	Drehzahl
0	0	0	0	0	0	0	0
1	101,2746	50,6373	50,6373	101,2746	7	7	101,2746
2	14,4678	7,2339	108,5085	115,7424	8	8	115,7424
3	0	0	115,7424	115,7424	8	8	115,7424
4	0	0	115,7424	115,7424	8	8	115,7424
5	0	0	115,7424	115,7424	8	8	115,7424
6	0	0	115,7424	115,7424	8	8	0
7	115,7424	57,8712	173,6136	231,4848	16	16	231,4848
8	-115,7424	-57,8712	173,6136	115,7424	8	8	115,7424
9	0	0	115,7424	115,7424	8	8	115,7424
10	0	0	115,7424	115,7424	8	8	50
11	65,7424	32,8712	148,6136	181,4848	12,54404954	13	188,0814
12	-72,339	-36,1695	145,3153	109,1458	7,544049545	8	115,7424
13	0	0	109,1458	109,1458	7,544049545	8	115,7424
14	0	0	109,1458	109,1458	7,544049545	8	115,7424
15	0	0	109,1458	109,1458	7,544049545	8	115,7424

(Quelle: selbst erstellt)

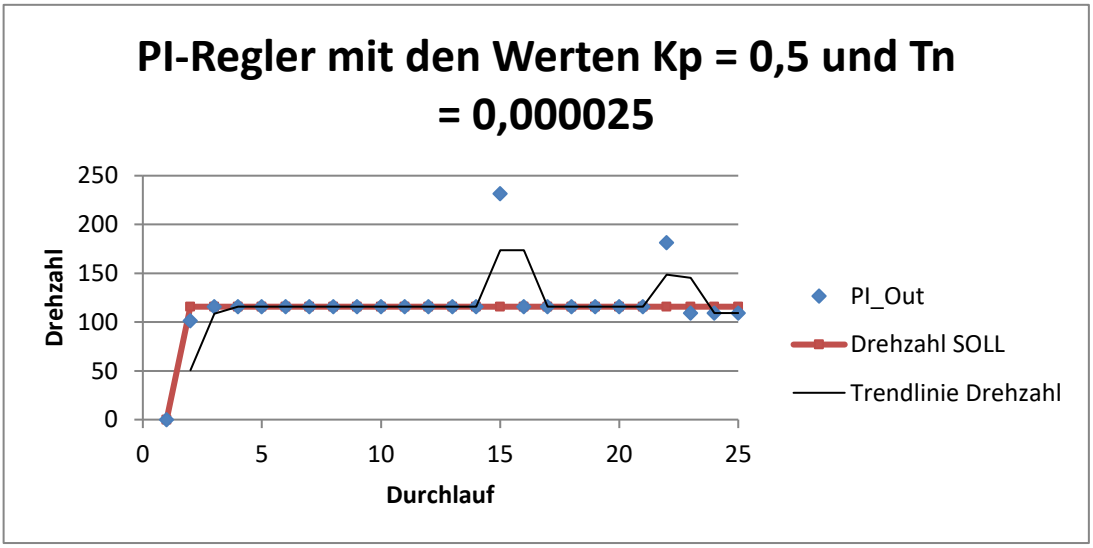


Abbildung 43: Visualisierung des optimierten PI-Reglers

(Quelle: selbst erstellt)

Neben dem schnellen Einstellen des gewünschten Wertes sieht man in der obigen Tabelle, dass Störeinflüsse, wie ein fehlerhafter oder ganz fehlender Drehzahlwert, schnell von dem PI-Regler korrigiert werden können. Der Nachteil daran ist, dass die Drehzahl kurzfristig sehr stark ansteigt. Möchte man, dass die Drehzahl bei Störungen nicht so stark ansteigt, so muss der Integralanteil vergrößert werden. Dies hat dann wiederrum zur Folge, dass das System verlangsamt wird und der Regler länger braucht um den Soll-Wert zu erreichen, wie es in Abbildung 42 der Fall war.

6 Ergebnisse der Arbeit

Abschließend muss man feststellen, dass der Atlas Digital Amplifier für den Gebrauch in der DLR nicht geeignet ist, da er weder das Kriterium der Zykluszeit noch das der Strommessung in dem Maße erfüllen konnte, in dem es erforderlich wäre. Der Atlas Digital Amplifier bietet jedoch viele Möglichkeiten und eine leichte Handhabung der Motorsteuerung. Daher lautet das Fazit, dass der Atlas Digital Amplifier vermutlich nicht besonders gut für einen speziellen Einsatz in der Erforschung von neuen Antriebssystemen zu gebrauchen ist, durch die leichte Handhabung aber für den Industriebereich oder allgemeinere Anwendungsfälle möglicherweise interessant sein könnte. Die Eigenschaften des Atlas Digital Amplifier bieten eine Vielfalt an Möglichkeiten, für die man sonst spezielle Hardware brauchen würde. Somit kann der Atlas Digital Amplifier dem Entwickler den Aufbau von zum Beispiel Treiberstufen oder Strommessungen abnehmen. Außerdem ist der Atlas Digital Amplifier durch die SPI Anbindung leicht in eigene Anwendungen integrierbar.
Auch wenn das Übertragungsprotokoll des Atlas Digital Amplifier nicht im Fokus des Herstellers ist, stellt es doch einen besonders wichtigen Punkt für den Einsatz des Moduls dar. Hier kann man dem Hersteller nur dringend empfehlen, die Dokumentation der Software Schnittstelle wesentlich zu überarbeiten und damit verständlicher zu gestalten. Insgesamt scheint das Protokoll noch Raum für Verbesserungen zu bieten. Beispielsweiße indem die Reihenfolge der Kommandos des Atlas Digital Amplifier von vornherein feststeht und die Antwort auf Kommandos, von denen man keine Antwort erwartet, wegfällt.

Daher wird als nächster Schritt in der DLR selbst ein Steuermodul entwickelt, welches die Anforderungen erfüllen kann. Die Erfahrungen und Ergebnisse dieser Bachelorarbeit sollen für die Neuentwicklung des Steuermoduls als Leitmotiv dienen und einige Funktionen, wie zum Beispiel die Ansteuerung über eine SPI Schnittstelle, sollen mit eingebunden werden.

Literaturverzeichnis

[1] Schröder Dierk (2009): Elektrische Antriebe – Grundlagen, 4. Auflage: Springer-Verlag Berlin Heidelberg. ISBN 978-3-642-02989-9

[2] Hering Ekbert; Vogt Alois; Bressler Klaus (1999): Handbuch der Elektrischen Anlagen und Maschinen, 1. Auflage: Springer-Verlag Berlin Heidelberg. ISBN 978-3-642-63592-2

[3] Reill Josef (2010): Lagegeberlose Regelung für ein accelermetergestütztes, hochdynamisches Roboterantriebssystem mit permanenterregtem Synchronmotor, Dessertation an der Technischen Fakultät der Universität Erlangen-Nürnberg.

[4] Performance Motion Devices Inc. Atlas+Family+datasheet+3.15.17.pdf. Online Verfügbar unter: https://www.pmdcorp.com/products/modules, zuletzt geprüft am 26.09.2018.

[5] Performance Motion Devices Inc. Atlas+Complete+Technical+Reference+2_0.pdf. Online Verfügbar unter: https://www.pmdcorp.com/products/modules, zuletzt geprüft am 26.09.2018.

[6] STMicroelectronics N.V. UM1725: Description of STM32F4 HAL and LL drivers.pdf. Online Verfügbar unter: https://www.st.com/en/embedded-software/stm32cubef4.html, zuletzt geprüft am 26.09.2018.

[7] Cplusplus.com. assert. Online Verfügbar unter: http://www.cplusplus.com/reference/cassert/assert/, zuletzt geprüft am 26.09.2018.

[8] Cplusplus.com. va_list. Online Verfügbar unter: http://www.cplusplus.com/reference/cstdarg/va_list/, zuletzt geprüft am 26.09.2018.

[9] iC-Haus GmbH. BiSS C (unidirectional) Protocol Description.pdf. Online Verfügbar unter: https://www.ichaus.de/BiSS%20Interface, zuletzt geprüft am 26.09.2018.

[10] Dr.-Ing. Josef Reill (2012): Vortrag zur Regelung von Permanenterregten Synchronmaschinen feldorientierte Regelung und geberlose Regelung, Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR) Institut für Robotik und Mechatronik

[11] Reuter Manfred; Zacher Serge (2008): Regelungstechnik für Ingenieure, 12. Auflage: Vieweg + Teubner Wiesbaden. ISBN 978-3-8348-0018-3

Erklärung zur Abschlussarbeit

Hiermit versichere ich, die eingereichte Abschlussarbeit selbständig verfasst und keine andere als die von mir angegebenen Quellen und Hilfsmittel benutzt zu haben. Wörtlich oder inhaltlich verwendete Quellen wurden entsprechend den anerkannten Regeln wissenschaftlichen Arbeitens zitiert. Ich erkläre weiterhin, dass die vorliegende Arbeit noch nicht anderweitig als Abschlussarbeit eingereicht wurde.

Das Merkblatt zum Täuschungsverbot im Prüfungsverfahren der Hochschule Augsburg habe ich gelesen und zur Kenntnis genommen. Ich versichere, dass die von mir abgegebene Arbeit keinerlei Plagiate, Texte oder Bilder umfasst, die durch von mir beauftragte Dritte erstellt wurden.

_____ _____

Ort, Datum Unterschrift des/der Studierenden

A. Anhang

A.1 Datenblatt vom Atlas Digital Amplifier

PERFORMANCE MOTION DEVICES
MOTION CONTROL AT ITS CORE

DATASHEET

ATLAS® Digital Amplifiers

ATLAS® Digital Amplifiers

are compact single-axis amplifiers that provide high performance torque control for DC brush, brushless DC, and step motors. They are packaged in a compact, solderable module and are ideal for use in positioning motion control, velocity control, and precision force control applications.

High Performance in an Ultra Compact Package

ATLAS Digital Amplifiers are used for direct control of motor torque, or in conjunction with higher level motion controllers. Their very compact size and high power output make them ideally suited for applications such as medical equipment, laboratory automation, scientific instruments, general purpose motion control, force feedback, and actuator controls. ATLAS Amplifiers are provided in vertical and horizontal mounting configurations, with three power levels, and two package sizes.

Advanced Technology

ATLAS Digital Amplifiers utilize PMD's proprietary digital current control and switching technology for exceptional efficiency and quiet motor operation. Control features include user-programmable gain parameters, performance trace, field oriented control, and I²t current management. Atlas amplifiers are internally powered from a single motor supply voltage, and provide automatic protection from overcurrent, undervoltage, overvoltage, overtemperature, and short circuit faults.

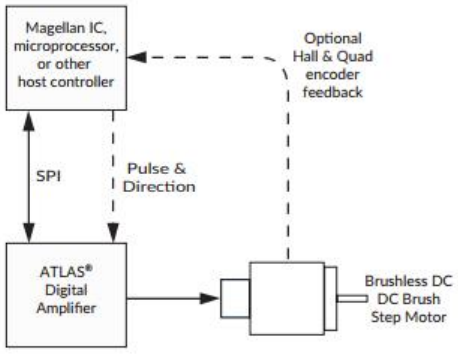
Flexibility

The ATLAS family has been designed to work seamlessly with PMD's Magellan motion control ICs. Alternatively, they can be used with dedicated FPGAs, digital signal processors, or general purpose microprocessors. Communication is via SPI (Serial Peripheral Interface) using a simple, packet-oriented protocol. For step motors, in addition to the SPI format a pulse & direction input mode is provided.

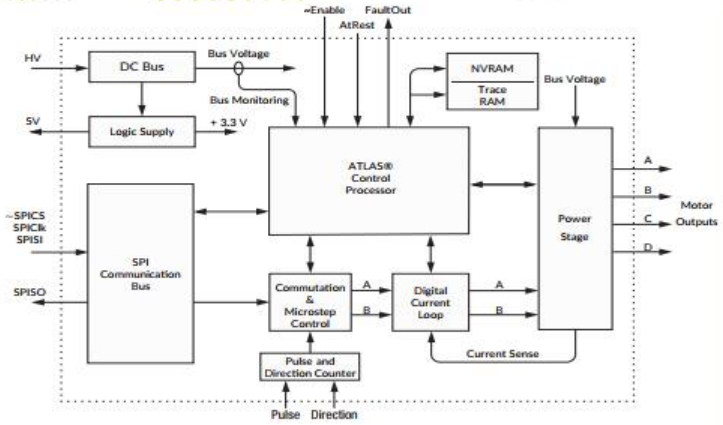
> FEATURES

- Ultra efficient all digital solderable power amplifier
- Controls brushless DC, step, and DC brush motors
- Available in 75 W, 250 W, and 500 W power levels
- Operating supply voltage range of 12 V to 56 V
- Field oriented control
- Overcurrent, overvoltage, and undervoltage protection
- Single supply operation from motor bus voltage
- Fully digital current control
- I²t current foldback limiting
- On-board performance trace and motor parameter storage in NVRAM
- Multi-motor version allows motor type to be programmed by user
- SPI (Serial Peripheral Interface) eliminates analog +/- 10 V torque signals
- Standalone pulse & direction step motor operation
- Internal temperature monitor
- Two different package sizes available
- Enable input and Fault output safety interlocks
- Works with Magellan® ICs, FPGAs or microprocessor-based controllers
- Comes in horizontal and vertical mount configurations
- Digital SPI torque command with checksum

> CONFIGURATION

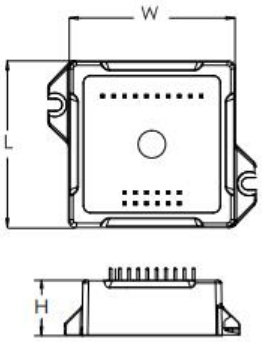


TechnicalOverview

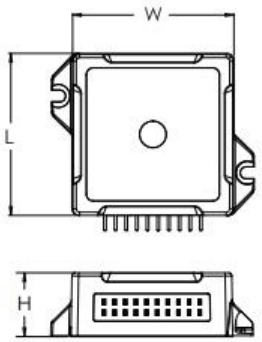


> MECHANICAL DIMENSIONS

Horizontal Configuration



Vertical Configuration



> ATLAS FAMILY SPECIFICATIONS

Parameter	Value
Supported Motor Types	brushless DC, step motor, DC brush, and multi-motor
PWM frequency	20, 40, 80, 120 kHz
Current Loop rate	20 kHz
Microstepping resolution	256 microsteps per full step
User Programmability:	Non-volatile RAM user configuration storage
Trace Memory:	2 KB
I/Os:	FaultOut, Enable
Safety:	Short Circuit, OverCurrent, I²t Current Foldback, SPI Watchdog, Overvoltage, Undervoltage
Operating Temperature:	0° - 40° C
Compliance:	RoHs, CE LVD:EN60204-1, EMC-D: EN61000-6-1, EN61000-6-3, EN55011
UL:	Designed to UL508C, UL840, and EN60204-1

> ATLAS MODEL SPECIFICATIONS

Model	Voltage Input	Peak Current	Continuous Current	Package
Low Power, brushless DC	12-48V	3.8 Amps	1.5 Arms	Ultra Compact
Low Power, step motor	12-48V	3.8 Amps	1.5 Arms	Ultra Compact
Low Power, DC brush	12-48V	3.8 Amps	1.5 ADC	Ultra Compact
Medium Power, brushless DC	12-48V	12.5 Amps	5.0 Arms	Ultra Compact
Medium Power, step motor	12-48V	12.5 Amps	4.5 Arms	Ultra Compact
Medium Power, DC brush	12-48V	12.5 Amps	7.0 ADC	Ultra Compact
High Power, brushless DC	12-56V	25.0 Amps	10.0 Arms	Compact
High Power, step motor	12-56V	25.0 Amps	9.0 Arms	Compact
High Power, DC brush	12-56V	25.0 Amps	14.0 ADC	Compact

Model	Length (L)	Width (W)	Height (H)
Ultra Compact Vertical	1.054 (in) 26.8 (mm)	1.051 (in) 26.7 (mm)	0.526 (in) 13.4 (mm)
Ultra Compact Horizontal	1.054 (in) 26.8 (mm)	1.051 (in) 26.7 (mm)	0.526 (in) 13.4 (mm)
Compact Vertical	1.520 (in) 38.6 (mm)	1.517 (in) 38.5 (mm)	0.600 (in) 15.2 (mm)
Compact Horizontal	1.520 (in) 38.6 (mm)	1.517 (in) 38.5 (mm)	0.600 (in) 15.2 (mm)

DevelopmentTools

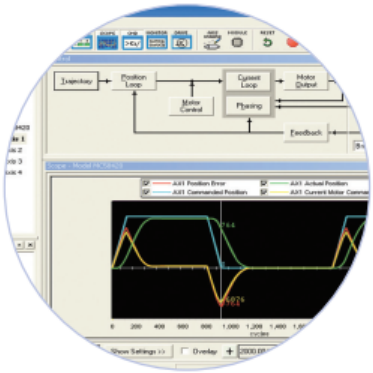
1 EASY START-UP

Atlas Developer's Kit board

1 or 4 axis configuration supports all Atlas unit types

Includes

- Atlas Developer's Kit board
- Pro-Motion CD and User's Guide
- Comes installed with rugged L-bracket hardware for easy heat sinking
- Complete manual set
- Complete cable & prototyping connector set



2 TUNE & CONFIGURE

Pro-Motion® GUI

Pro-Motion is a sophisticated, easy-to-use Windows-based exerciser program for use with PMD amplifiers, motion control ICs, modules, and boards.

Features

- Motion oscilloscope graphically displays parameters in real-time
- Autotuning of control parameters
- Ability to save and load configuration parameters in NVRAM
- Easy motor setup with Axis wizard
- Advanced Bode frequency machine analysis.
- Trace capability for analizing motor behavior.

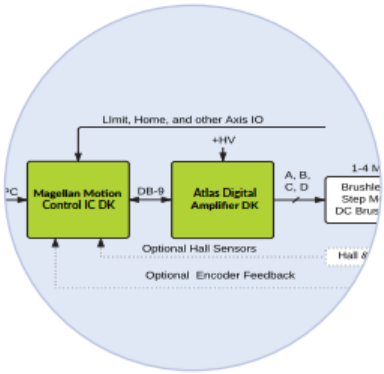
3 BUILD THE APPLICATION

Magellan® Motion Control IC DKs

Atlas Digital Amplifiers may be used in conjunction with PMD's Magellan family of motion control ICs. To build your complete motion controller add a Magellan IC DK to your Atlas DK.

Magellan IC Developer's Kit Features

- Connects to encoder feedback signals, limit switches, and other motion peripherals
- Control and exercise your entire machine
- Develop embeddable C/C++ application code
- Communicate to the PC via serial, CAN, or SPI communications
- Store NVRAM parameters



> PMD PRODUCT OVERVIEW

	VELOCITY & TORQUE CONTROL ICs	MAGELLAN® MOTION CONTROL ICs	ATLAS® DIGITAL AMPLIFIERS	PRODIGY® MOTION BOARDS	ION® DIGITAL DRIVES
					
No. Axes	1	1, 2, 3, 4	1	1, 2, 3, 4	1
Format	• 64-pin TQFP	• 144-pin TQFP • 100-pin TQFP	• Compact: 20-pin solderable module • Ultra Compact: 19-pin solderable module	• PCI • PC/104 • Standalone • Machine Controller	• Fully enclosed module
Voltage	3.3 V	3.3 V	12 - 56 V	PCI, PC/104, Standalone: 5 V Machine Controller: 12 - 56 V	12 - 56 V / 20 - 195 V
Features	• Velocity control • Commutation • Torque/current control • Field-oriented control	• Position control • Commutation • Network communications • Torque/current control • Field oriented control • Profile generation • Multi-motor support	• Torque/current control • Field oriented control • Trace buffer • Pulse & direction input • Multi-motor support • SPI Interface • MOSFET amplifier	• Position control • Commutation • Network communications • Torque/current control • Field oriented control • Profile generation • Multi-motor support • PWM output • Analog output • Trace buffer • Programmable • Signal conditioning • General purpose user I/Os	• Position control • Commutation • Network communications • Torque/current control • Field oriented control • Profile generation • Trace buffer • MOSFET amplifier • Pulse & direction input • Programmable (ION/CME only) • General purpose user I/Os (ION/CME only)
Motor Types	• Brushless DC	• DC brush • Brushless DC • Step Motor	• DC brush • Brushless DC • Step Motor	• DC brush • Brushless DC • Step Motor	• DC brush • Brushless DC • Step Motor
Communication	• Standalone • RS232/485	• Parallel • RS232/485 • CANbus • SPI	• SPI	• Ethernet • RS232/485 • CANbus • PCI and PC/104 bus	• Ethernet • RS232/485 • CANbus
Loop Rate	20 kHz – current 10 kHz – velocity	50 – 75 µsec/axis	20 kHz – current	50 – 150 µsec/axis	20 kHz – current 10 kHz – position

> FOR ORDERING ATLAS

MD 1 3 1 056/25 HB

- Atlas Body Type: 1 Compact; 2 Ultra Compact
- # of Axis: 1 Default
- Atlas Mounting: VB Vertical; HB Horizontal
- Motor Type: 1 DC Brush; 3 BLDC; 4 Step; 8 Multi-motor
- Power Selection: 056/25 Atlas 500W; 048/05 Atlas 250W; 048/02 Atlas 75W

> FOR ORDERING ATLAS DKS

MDK 1 LI XXXX

- Number of Axis: 1 or 4
- Atlas Configuration*: 0 No Atlas Installed; 4 Multi-Motor, 500 Watt, Vertical; 6 Multi-Motor, 500 Watt, Horizontal; C Multi-Motor, 250 Watt, Vertical; I Multi-Motor, 250 Watt, Horizontal; S Multi-Motor, 75 Watt, Vertical; U Multi-Motor, 75 Watt, Horizontal

*Contact PMD for motor-specific options (BLDC, DC Brush, Step)

To place an order or for additional information and questions, contact PMD customer service.

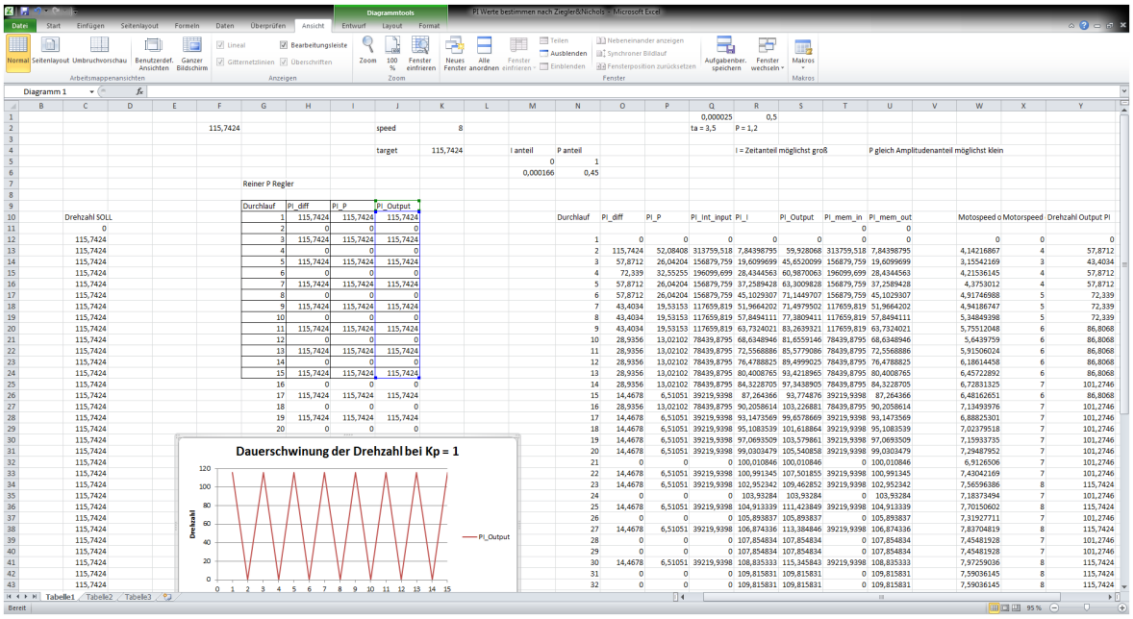
PERFORMANCE MOTION DEVICES
MOTION CONTROL AT ITS CORE

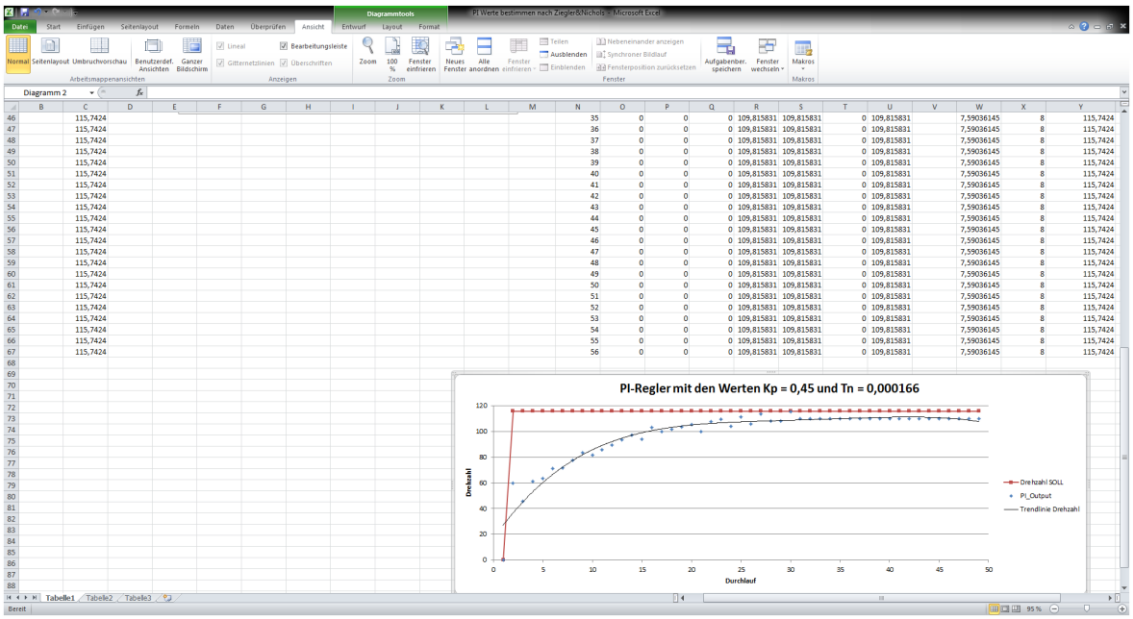
1 Technology Park Dr, Westford, MA 01886
Tel: 978.266.1210 Fax: 978.266.1211
e-mail: info@pmdcorp.com
www.pmdcorp.com

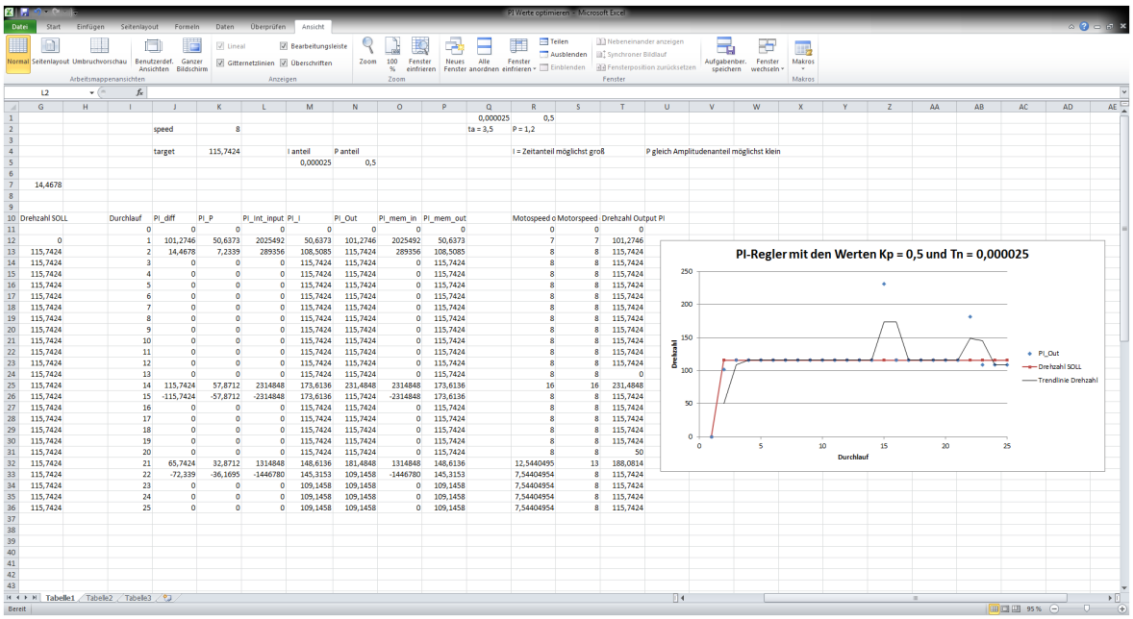
About Performance Motion Devices

Performance Motion Devices (PMD) is a worldwide leader in motion control ICs, boards and amplifiers. Dedicated to providing cost-effective, high performance motion systems to OEM customers, PMD utilizes extensive in-house expertise to minimize time-to-market and maximize customer satisfaction.

A.2 Reglerberechnungstabellen







	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE
1											0,000025	0,5													
2				speed	8						ta = 3,5	P = 1,2													
3																									
4				target	115,7424		I anteil	P anteil				I = Zeitanteil möglichst groß			P gleich Amplitudenanteil möglichst klein										
5							0,000025	0,5																	
6																									
7	14,4678																								
8																									
9																									
10	Drehzahl SOLL		Durchlauf	PI_diff	PI_P	PI_Int_input	PI_I	PI_Out	PI_mem_in	PI_mem_out		Motospeed o	Motorspeed	Drehzahl Output PI											
11			0	0	0	0	0	0	0	0		0	0	0											
12	0		1	101,2746	50,6373	2025492	50,6373	101,2746	2025492	50,6373		7	7	101,2746											
13	115,7424		2	14,4678	7,2339	289356	108,5085	115,7424	289356	108,5085		8	8	115,7424											
14	115,7424		3	0	0	0	115,7424	115,7424	0	115,7424		8	8	115,7424											
15	115,7424		4	0	0	0	115,7424	115,7424	0	115,7424		8	8	115,7424											
16	115,7424		5	0	0	0	115,7424	115,7424	0	115,7424		8	8	115,7424											
17	115,7424		6	0	0	0	115,7424	115,7424	0	115,7424		8	8	115,7424											
18	115,7424		7	0	0	0	115,7424	115,7424	0	115,7424		8	8	115,7424											
19	115,7424		8	0	0	0	115,7424	115,7424	0	115,7424		8	8	115,7424											
20	115,7424		9	0	0	0	115,7424	115,7424	0	115,7424		8	8	115,7424											
21	115,7424		10	0	0	0	115,7424	115,7424	0	115,7424		8	8	115,7424											
22	115,7424		11	0	0	0	115,7424	115,7424	0	115,7424		8	8	115,7424											
23	115,7424		12	0	0	0	115,7424	115,7424	0	115,7424		8	8	115,7424											
24	115,7424		13	0	0	0	115,7424	115,7424	0	115,7424		8	8	0											
25	115,7424		14	115,7424	57,8712	2314848	173,6136	231,4848	2314848	173,6136		16	16	231,4848											
26	115,7424		15	-115,7424	-57,8712	-2314848	173,6136	115,7424	-2314848	173,6136		8	8	115,7424											
27	115,7424		16	0	0	0	115,7424	115,7424	0	115,7424		8	8	115,7424											
28	115,7424		17	0	0	0	115,7424	115,7424	0	115,7424		8	8	115,7424											
29	115,7424		18	0	0	0	115,7424	115,7424	0	115,7424		8	8	115,7424											
30	115,7424		19	0	0	0	115,7424	115,7424	0	115,7424		8	8	115,7424											
31	115,7424		20	0	0	0	115,7424	115,7424	0	115,7424		8	8	50											
32	115,7424		21	65,7424	32,8712	1314848	148,6136	181,4848	1314848	148,6136		12,5440495	13	188,0814											
33	115,7424		22	-72,339	-36,1695	-1446780	145,3153	109,1458	-1446780	145,3153		7,54404954	8	115,7424											
34	115,7424		23	0	0	0	109,1458	109,1458	0	109,1458		7,54404954	8	115,7424											
35	115,7424		24	0	0	0	109,1458	109,1458	0	109,1458		7,54404954	8	115,7424											
36	115,7424		25	0	0	0	109,1458	109,1458	0	109,1458		7,54404954	8	115,7424											
37																									
38																									
39																									
40																									
41																									
42																									
43																									