Masterarbeit:
Regelung von permanentmagneterregten Synchronmaschinen

Roland Angerbauer

12. Februar 2013 in Graz

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

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

Graz, am 4.7.2013 ..... Angelika Rohrer
(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

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

Graz, 4.7.2013 ..... Angelika Rohrer
date (signature)

Kurzfassung

Diese Arbeit beschäftigt sich mit der Regelung von permanenterregten Synchronmaschinen (PMSM). Der Schwerpunkt liegt in der Implementierung der Regelung in einem Mikrocontroller. Es wird ein 32bit Mikrocontroller mit einem Cortex-M4 Kern der Firma STMicroelectronics verwendet. Für die PMSM wird die Modellbildung beschrieben. Als Regelverfahren wird eine feldorientierte Regelung verwendet. Für den Reglerentwurf wird das Frequenzkennlinienverfahren (FKL) für zeitdiskrete Systeme angewandt. Das entstandene Programm wurde komplett in C geschrieben. Es ist sehr modular gestaltet, um leichter unterschiedliche Hardware zu unterstützen. Die verwendete Hardware ist ein Developer-Board von STMicroelectronics. Mit dieser Plattform wurde das Programm entwickelt und der Regler getestet. Es besteht aus einem Evaluation-Board für den verwendeten Mikrocontroller, einer Power-Stage (drei Halbbrücken und Strommessung) und einem Motor. Der Regler wurde auch noch bei einer anderen Hardwareplattform eingesetzt. Er wird im „urban concept car“ (UCC) verwendet. Dieses Auto ist eine Selbstentwicklung von einem Studenten-Team (TERA) an der TU Graz.

Abstract

This thesis deals with the control of Permanent Magnet Synchronous Machines (PMSM). The focus is the implementation of the controller in a microcontroller. A 32 bit microcontroller with a Cortex-M4 core of the company STMicroelectronics is used. The formation of the model is described for the PMSM. The control method is: Field oriented control. The “Frequenzkennlinienverfahren” is used for the controller design. The resulting program was completely written in C. It is highly modular designed to facilitate easier different hardware. The hardware is a developer board from STMicroelectronics. With this platform, the program was developed and tested. It consists of an evaluation board for the used microcontroller, a power stage (three half-bridges and current sensing) and a motor. The controller was also used on a different hardware platform. He is used in the “urban concept car“ (UCC). This car is a self-development of a student team (TERA) at Graz University of Technology.

Inhaltsverzeichnis

1. Einleitung **1**

2. Permanenterregte Synchronmaschine (PMSM) **2**
2.1. Raumzeiger . . . 2
2.2. Koordinatensysteme . . . 4
2.3. Modell . . . 5
2.3.1. Spannungsgleichung . . . 5
2.3.2. Moment . . . 6
2.3.3. Bewegungsgleichung . . . 8
2.3.4. Zusammenfassung . . . 9

3. Regler **10**
3.1. Entwurf . . . 10
3.1.1. Entkopplung . . . 11
3.1.2. Stromregler . . . 11
3.1.3. Drehzahlregler . . . 12
3.2. Raumzeigermodulation . . . 12
3.2.1. Schaltmuster . . . 13
3.2.2. Bestimmen des Sektors . . . 14

4. Programm **17**
4.1. Mikrocontroller . . . 17
4.1.1. Allgemein . . . 17
4.1.2. Speicher . . . 17
4.1.3. Remapping . . . 19
4.1.4. FPU . . . 19
4.2. Bibliotheken . . . 20
4.2.1. Hersteller-Bibliotheken . . . 20
4.2.2. Eigene Bibliothek . . . 20
4.3. Regler . . . 22
4.3.1. Modell . . . 26
4.3.2. Wertedarstellung . . . 26
4.3.3. Controller . . . 28
4.3.4. Feedback . . . 29
4.3.5. SVPWM . . . 35
4.3.6. Rotorposition bestimmen . . . 36

4.4. Module . . . 37
4.4.1. Userinterface . . . 37
4.4.2. Fahrverhalten . . . 41
4.4.3. CAN-Protokoll . . . 41
4.4.4. syscall-stubs . . . 42
4.4.5. EEPROM-emulation . . . 43
4.4.6. RAM-Testprogramm . . . 43

5. Toolchain **45**
5.1. Übersicht . . . 45
5.2. Installation . . . 47
5.3. praktisches Arbeiten mit der Toolchain . . . 49
5.3.1. Eclipse . . . 50
5.3.2. udev-ruels . . . 52
5.3.3. MakeFile . . . 53
5.3.4. Linker-Script . . . 55

6. Hardware **59**
6.1. Prozessorboard . . . 59
6.1.1. Version 1 . . . 59
6.1.2. Version 2 . . . 62
6.2. EVAL-Board: STM3240G . . . 65
6.2.1. Peripherie . . . 66
6.2.2. Ports . . . 66
6.3. Power Stage: MB459-B01 . . . 71
6.4. STM-Motor-Board . . . 75
6.5. TERA Motorsteuerung . . . 75

A. Dateiübersicht **76**

Tabellenverzeichnis

3.1. Schaltmuster und Raumzeiger bei Sternschaltung 13
3.2. Schaltmuster und Raumzeiger bei Dreieckschaltung 13
3.3. Zeiten Vektoren bei Sternschaltung 14
3.4. Zeiten Vektoren bei Dreieckschaltung 15
3.5. Geradengleichungen zur Sektorbestimmung, Sternschaltung 16
3.6. Geradengleichungen zur Sektorbestimmung, Dreieckschaltung 16

4.1. Übersicht über die einzelnen Speicher 18
4.2. Statusbits des magnetischen Encoders ASS5145 30
4.3. Struktur einer CAN-Datennachricht 42
4.4. Struktur einer der CAN-ID 42
4.5. Struktur CAN Daten 42

5.1. übliche JTAG Steckerbelegung 46
5.2. Speicheraufteilung des Mikrocontrollers 57
5.3. Speicheraufteilung des Mikrocontrollers 58
5.4. Nutzung der Sektoren vom internen Flash des Mikrocontrollers 58

6.1. JTAG Steckerbelegung des Prozessorboards 2.0 63
6.2. Belegung der einzelnen Port-Pins beim STM3240G-EVAL Board 67
6.3. Parameter des Shinano Motors 72
6.4. Windungsanschluss des Shinano Motors 72
6.5. Encoder Anschluss des Shinano Motors 72

Abbildungsverzeichnis

2.1. Koordinatensysteme 4
2.2. elektrisches Ersatzschaltbild eines PMSM 5

3.1. Sektoren die von Raumzeigern gebildet werden 15

4.1. IQ-Regler simuliert in Matlab 23
4.2. IQ-Regler simuliert mit Drehzahlregler 24
4.3. Drehzahlregler simuliert 24
4.4. IQ-Regler, Messung vom Testaufbau 24
4.5. IQ-Regler mit Drehzahlvorgabe, Messung vom Testaufbau 25
4.6. Drehzahlregler, Messung vom Testaufbau 25
4.7. Simulink Modell der Regelung 26
4.8. Winkel in Festkommadarstellung $Q1.15$ 28
4.9. Duty cycle der Halbbrücken beim Auftreten des Fehlers 33
4.10. Phasenströme beim Auftreten des Fehlers 33
4.11. Rohdaten des rotary-encoders, Motor dreht sich 34
4.12. Rohdaten des rotary-encoders, Motor stillstand 34
4.13. schematischer Programmablauf Stromregler 35
4.14. Simulink Modell für die Rotorpositionsbestimmung 37
4.15. Bewegung des Rotors in Grad 38
4.16. Reglerausgang in Grad 39
4.17. Rotorbewegung in Encoderschritten 39
4.18. serielles VT100 Terminal 40

6.1. Schaltplan des Prozessorboards v1.0 60
6.2. Platinenlayout des Prozessorboards v1.0 61
6.3. Schaltplan des Prozessorboards Version 2.0 64
6.4. Bestückungsplan vom Prozessorboard 2.0 65
6.5. PlatinenlLayout vom Prozessorboard 2.0 65
6.6. Verstärkerschaltung für einen Shunt 73

Listings

4.1. Fehlermeldung, GCC keine FPU unterstützung 19

5.1. Befehle zum Starten von openOCD 48
5.2. Beispiel für eine openOCD Konfiguration 48
5.3. flash.script 48
5.4. Linux-Konsolen Befehl für die Standard-Defines 52
5.5. 10-jlink.rule 52
5.6. 11-prolific.rule 52
5.7. Ausschnitt der Ausgabe des Befehls lsusb 53
5.8. Beispiel für Variablendefinition und deren Segment 55
5.9. Benutzerdefinierte Zuweisung des Segmentes 56

1. Einleitung

Diese Arbeit beschäftigt sich mit der Implementierung einer Motorregelung in einem Mikrocontroller. Die Regelung soll in einem Elektroauto eingesetzt werden, das von TERA [1] entwickelt wird. TERA ist ein eigenständiger Verein an der TU Graz. Er besteht aus ungefähr 40 Studenten von allen Grazer Universitäten, großteils jedoch von Studenten der TU Graz. Der Verein existiert seit 2009 und hat schon in den Jahren 2010 und 2011 erfolgreich am Shell Eco Marathon [2] teilgenommen. In der Saison 2011/12 hatte sich das Team zum Ziel gesetzt, ein „urban concept car“ (UCC) zu entwickeln und zu bauen. Im Rahmen dieser Entwicklung entstand auch diese Diplomarbeit. Das Auto trägt den Namen *Panther* und konnte 2012 beim Bewerb den 12. Platz erreichen.

Die Motorsteuerung des Autos wurde komplett im Team entwickelt. Die Aufgabe dieser Diplomarbeit war die Implementierung der Software für die Motorregelung am Mikrocontroller. Ein Ziel war, die Software sehr modular zu programmieren, um die Unterstützung für unterschiedliche Hardware oder Sensoren einfacher zu gestalten. Außerdem sollte für den verwendeten Mikrocontroller eine Toolchain erstellt werden.

Am Ende der Arbeit sind zwei unterschiedliche Hardwareplattformen vollständig implementiert. Die erste Plattform ist ein Developer-Board von STMicroelectronics, das alle nötigen Baugruppen für eine Motorsteuerung enthält: Prozessorboard, Halbbrücken und Motor. Mit diesem Aufbau wurde die Software beziehungsweise der Regler entwickelt und getestet. Die andere Plattform ist die im Team entwickelte Motorsteuerung, welche im UCC eingesetzt wird.

Als Motor wird bei beiden Plattformen eine permanent erregte Synchronmaschine verwendet (PMSM). Dieser Typ wird wegen des guten Wirkungsgrades verwendet [9].

[1] TERA TU-Graz Verein für effiziente Fahrzeugtechnologien

[2] http://www.shell.com/global/environment-society/ecomarathon/events/europe.html

2. Permanenterregte Synchronmaschine (PMSM)

In diesem Kapitel wird das Erstellen und Berechnen eines Modells für eine permanenterregte Synchronmaschine beschrieben. Zusätzlich werden die nötigen Transformationen der physikalischen Größen angeführt.

Bei der Modellbildung werden folgende Vereinfachungen angenommen, um diese zu erleichtern:

- es wird nur die Grundwelle betrachtet
- der Aufbau der Maschine ist symmetrisch
- keine Nullkomponenten vorhanden $I_a(t) + I_b(t) + I_c(t) = 0$ (s.g. Nullbedingung).

2.1. Raumzeiger

Die Raumzeigerdarstellung ist eine einfache Schreibweise, um die drei Komponenten (A, B, C) in einem Zeiger (mit zwei Komponenten) darzustellen [9]. Dies ist auf Grund der Nullbedingung möglich. Es wird nur die Grundwelle in einem Raumzeiger zusammengefasst. Der Raumzeiger rotiert mit der Winkelgeschwindigkeit ω. Das ermöglicht die zeitlich und räumlich veränderlichen Größen in einem Raumzeiger zusammenzufassen. Der Raumzeiger ergibt sich aus der Überlagerung der einzelnen Komponenten I_a, I_b und I_c. Diese werden entsprechend der Wicklungsaufteilung mit einer Richtung versehen, über die Faktoren: 1, a und a^2.

$$I = \frac{2}{3}(I_a + a\, I_b + a^2\, I_c) \tag{2.1}$$

$$a = e^{j\frac{2\pi}{3}} = -\frac{1}{2} + j\frac{\sqrt{3}}{2}, \qquad a^2 = e^{j\frac{4\pi}{3}} = -\frac{1}{2} - j\frac{\sqrt{3}}{2}. \tag{2.2}$$

Durch den Faktor 2/3 wird die Amplitude des Raumzeigers an die Amplitude der Grundschwingung angepasst.

Die folgende Berechnung zeigt die Bestimmung des Faktors für die Anpassung der Amplitude. Eine dreiphasige Schwingung mit der Frequenz ω wird in einen Raumzeiger

transformiert:

$$
\begin{aligned}
i_a(t) &= \hat{I} \cdot \cos(\omega t) = \Re e\{\hat{I} e^{j\omega t}\} = \hat{I}\frac{1}{2}(e^{j\omega t} + e^{-j\omega t}) \\
i_b(t) &= \hat{I} \cdot \cos(\omega t - \frac{2\pi}{3}) = \Re e\{\hat{I} e^{j(\omega t - \frac{2\pi}{3})}\} = \hat{I}\frac{1}{2}(e^{j(\omega t - \frac{2\pi}{3})} + e^{-j(\omega t - \frac{2\pi}{3})}) \\
i_c(t) &= \hat{I} \cdot \cos(\omega t - \frac{4\pi}{3}) = \Re e\{\hat{I} e^{j(\omega t - \frac{4\pi}{3})}\} = \hat{I}\frac{1}{2}(e^{j(\omega t - \frac{4\pi}{3})} + e^{-j(\omega t - \frac{4\pi}{3})}).
\end{aligned} \tag{2.3}
$$

Diese Transformation

$$
I_s = \frac{2}{3}\begin{bmatrix} e^{j\,0} & e^{j\,\frac{2\pi}{3}} & e^{j\,\frac{4\pi}{3}} \end{bmatrix} \cdot \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} \tag{2.4}
$$

ist eine andere Schreibweise der Gleichung 2.1.

Der Rechenweg der Transformation lautet:

$$
\begin{aligned}
I_s &= \frac{2}{3}\begin{bmatrix} e^{j\,0} & e^{j\,\frac{2\pi}{3}} & e^{j\,\frac{4\pi}{3}} \end{bmatrix} \cdot \frac{\hat{I}}{2}\begin{bmatrix} e^{j\omega t} + e^{-j\omega t} \\ e^{j(\omega t - \frac{2\pi}{3})} + e^{-j(\omega t - \frac{2\pi}{3})} \\ e^{j(\omega t - \frac{4\pi}{3})} + e^{-j(\omega t - \frac{4\pi}{3})} \end{bmatrix} \\
I_s &= \frac{2}{3}\frac{3}{2}\,\hat{I}\,e^{j\omega t} + \frac{2}{3}\frac{\hat{I}}{2}\,e^{-j\omega t}\begin{bmatrix} e^{j\,0} & e^{j\,\frac{2\pi}{3}} & e^{j\,\frac{4\pi}{3}} \end{bmatrix} \\
I_s &= \hat{I}\,e^{j\omega t}
\end{aligned}
$$

Daraus ist ersichtlich, dass aufgrund des eingeführten Faktors die Amplitude des Raumzeigers und der Grundschwingung gleich ist. Außerdem hat der Raumzeiger dieselbe Phasenlage wie die Phase A.

Teilt man die Gleichung 2.1 in Real- und Imaginärteil auf,

$$
\begin{aligned}
I &= I_\alpha + j\,I_\beta \\
I_\alpha &= \Re e\{I\}, \qquad I_\beta = \Im m\{I\},
\end{aligned}
$$

erhält man die folgende Transformationsmatrix:

$$
\begin{pmatrix} I_\alpha \\ I_\beta \end{pmatrix} = \frac{2}{3}\begin{pmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{pmatrix}\begin{pmatrix} I_a \\ I_b \\ I_c \end{pmatrix}. \tag{2.5}
$$

Nimmt man noch die Nullbedingung hinzu, benötigt man nur noch zwei Ströme, um den Raumzeiger zu berechnen:

$$
\begin{pmatrix} I_\alpha \\ I_\beta \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \frac{1}{\sqrt{3}} & \frac{2}{\sqrt{3}} \end{pmatrix}\begin{pmatrix} I_a \\ I_b \end{pmatrix} \tag{2.6}
$$

$$
\begin{pmatrix} I_a \\ I_b \\ I_c \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2} \end{pmatrix}\begin{pmatrix} I_\alpha \\ I_\beta \end{pmatrix} \tag{2.7}
$$

Im Programm ergibt sich bei Verwendung von Raumzeigern eine Vereinfachung. Über die Raumzeigermodulation, beschrieben in 3.2, kann aus einem gewünschten Statorspannungsraumzeiger direkt das Schaltmuster für die Halbbrücken berechnet werden.

2.2. Koordinatensysteme

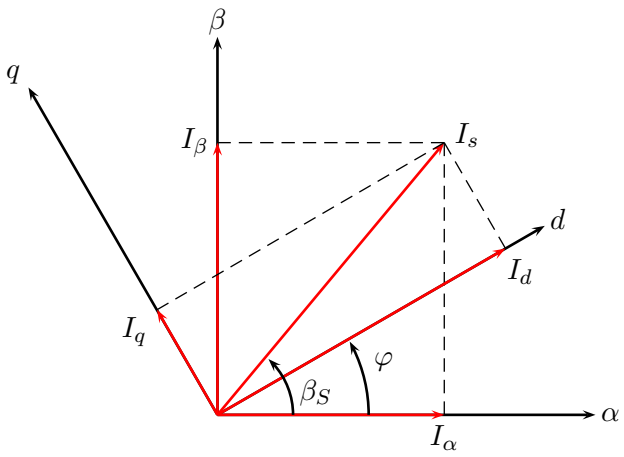


Abbildung 2.1.: Koordinatensysteme

Um die Berechnung des Modells eines permanent erregten Synchronmotors zu vereinfachen, werden für die elektrischen Größen unterschiedliche Koordinatensysteme verwendet. Bei Synchronmaschinen sind dies üblicherweise ein statorfestes sowie ein rotorfestes Koordinatensystem. Die gängigen Bezeichnungen der Achsen sind im statorfesten Koordinatensystem α und β, im rotorfesten Koordinatensystem d und q (für *direct* und *quadrature axis*). Die Zugehörigkeit einer Variable zu einem bestimmten Koordinatensystem wird durch einen hochgestellten Index dargestellt. Bei der Angabe von d/q oder α/β als gewöhnlichen Index wird der hochgestellte Index nicht verwendet, da die Zugehörigkeit zum jeweiligen Koordinatensystem bereits eindeutig ist. Um mathematische Berechnungen durchführen zu können, müssen die darin vorkommenden Variablen im selben Koordinatensystem ausgedrückt werden. Bei einer permanenterregten Synchronmaschine wird im Rotorkoordinatensystem die d-Achse an der Richtung des magnetischen Feldes vom Permanentmagneten ausgerichtet. Die α-Achse des statorfesten Koordinatensystems wird an der Wicklung der Phase A ausgerichtet.

In der Abbildung 2.1 sind beide Koordinatensysteme dargestellt. Der Stromraumzeiger kann folgendermaßen ausgedrückt werden:

$$I_S^S = \hat{I}\, e^{j\beta_S} \tag{2.8}$$

Die beiden Gleichungen:

$$X^R = X^S \cdot e^{-j\varphi} \tag{2.9}$$

$$X^S = X^R \cdot e^{j\varphi} \tag{2.10}$$

transformieren einen Raumzeiger in das jeweilig andere Koordinatensystem. Die Transformation kann auch in Matrixschreibweise dargestellt werden:

$$\begin{pmatrix} I_\alpha \\ I_\beta \end{pmatrix} = \begin{pmatrix} \cos(\varphi) & -\sin(\varphi) \\ \sin(\varphi) & \cos(\varphi) \end{pmatrix} \begin{pmatrix} I_d \\ I_q \end{pmatrix} \tag{2.11}$$

$$\begin{pmatrix} I_d \\ I_q \end{pmatrix} = \begin{pmatrix} \cos(\varphi) & \sin(\varphi) \\ -\sin(\varphi) & \cos(\varphi) \end{pmatrix} \begin{pmatrix} I_\alpha \\ I_\beta \end{pmatrix}. \tag{2.12}$$

Dabei wurde die Eulersche Formel ($e^{j\varphi} = \cos(\varphi) + j\sin(\varphi)$) angewandt und der Raumzeiger in seinen Real- und Imaginärteil aufgeteilt.

2.3. Modell

Das hier erstellte Modell der PMSM wird später für den Reglerentwurf und die Parameterbestimmung benötigt. Für die Modellbildung werden die elektrischen Spannungsgleichungen, die Leistungsbilanz und die Bewegungsgleichung benötigt. In Abbildung 2.2 ist das elektrische Ersatzschaltbild einer PMSM dargestellt.

Bei der Modellbildung muss das verwendete Regelverfahren berücksichtigt werden, in diesem Fall handelt es sich dabei um eine feldorientierte Regelung. Dabei werden die Ströme rechentechnisch in eine drehmoment- und flussbildende Komponente aufgeteilt. Beim Reglerentwurf werden die Gleichungen im Rotorkoordinatensystem verwendet.

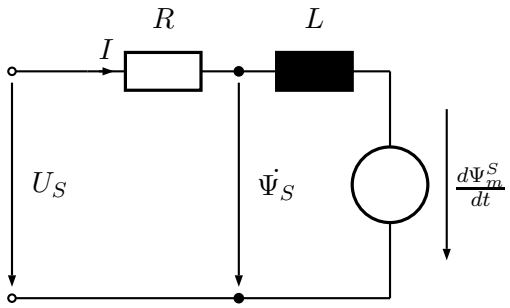


Abbildung 2.2.: elektrisches Ersatzschaltbild eines PMSM

2.3.1. Spannungsgleichung

Die Statorspannung im Statorkoordinatensystem lautet in Raumzeigerschreibweise:

$$U_S^S = R_S\, I_S^S + \frac{d\Psi_S^S}{dt}. \tag{2.13}$$

Der Term $R_S\, I_S^S$ berücksichtigt den ohmschen Spannungsabfall an den Statorwicklungen. Der Term $d\Psi_S^S/dt$ stellt die zeitliche Änderung des gesamten Statorflusses dar. Dieser

setzt sich aus der Selbstinduktion der Statorwicklungen und dem induzierten Fluss durch den Permanentmagneten Ψ_m zusammen:

$$\Psi_S^S = L\ I_S^S + \Psi_m^S. \tag{2.14}$$

Der Rotorfluss Ψ_R wird nur durch den Permanentmagneten gebildet. Durch die Annahme, dass das Rotorkoordiantensystem an diesem Fluss orientiert ist, hat der Rotorfluss nur eine Komponente in d-Richtung $\Psi_R^R = \Psi_{PM}$. Um die Beziehung in die Flussgleichung einsetzen zu können, muss sie in das Statorkoordinatensystem umgerechnet werden.

$$\begin{aligned} \Psi_m^S &= \Psi_{PM} \cdot e^{j\varphi} \\ \Psi_S^S &= L\ I_S^S + \Psi_{PM} \cdot e^{j\varphi} \end{aligned} \tag{2.15}$$

Im Rotorkoordiantensystem lauten die Flussgleichungen:

$$\begin{aligned} \Psi_d &= L_d\ I_d + \Psi_{PM}, \qquad \Psi_q = L_q\ I_q \\ \Psi_S^R &= \Psi_d + j\Psi_q. \end{aligned} \tag{2.16}$$

Zur Berechnung des Modells wird die Gleichung 2.13 ins Rotorkoordinatensystem transformiert:

$$U_S^R\ e^{j\varphi} = R\ I_S^R\ e^{j\varphi} + \frac{d(\Psi_S^R\ e^{j\varphi})}{dt}. \tag{2.17}$$

Beim Ausrechnen der zeitlichen Ableitung des Statorflusses muss berücksichtigt werden, dass sowohl der Raumzeiger Ψ_S^R als auch der Rotorwinkel φ zeitvariant sind. Daher muss die Produktregel angewandt werden:

$$U_S^R\ e^{j\varphi} = R\ I_S^R\ e^{j\varphi} + \frac{d\Psi_S^R}{dt}\ e^{j\varphi} + j\ \Psi_S^R\ \dot{\varphi}\ e^{j\varphi}. \tag{2.18}$$

Die Gleichung kann vereinfacht werden:

$$U_S^R = R\ I_S^R + \dot{\Psi_S^R} + j\ \Psi_S^R\ \dot{\varphi}. \tag{2.19}$$

Die einzelnen Raumzeiger in ihre Komponenten anschreiben und ω_l als Ableitung von φ einführen:

$$U_d + jU_q = R\ (I_d + jI_q) + (\Psi_d + \dot{j\Psi_q}) + j\ \omega_l\ (\Psi_d + j\Psi_q) \tag{2.20}$$

2.3.2. Moment

Die Gleichung für das innere Moment des Motors ist aus der Leistungsbilanz ersichtlich. Die anschließende hergeleitete Formel für das Moment wird ebenfalls im Rotorkoordinatensystem angegeben.

Die gesamte Leistung die der Maschine zugeführt wird, berechnet sich aus:

$$p_{el}(t) = u_a i_a + u_b i_b + u_c i_c \tag{2.21}$$

Strom und Spannung als Raumzeiger anschreiben. Mit T^{-1} wird die Transformationsmatrix 2.7 bezeichnet.

$$u_{abc} = T^{-1} u_s, \qquad i_{abc} = T^{-1} i_s \tag{2.22}$$

Die Multiplikation der einzelnen Spannungen und Ströme als Matrizenmultiplikation angeben:

$$\begin{aligned}
P_{el} &= u_s^T (T^{-1})^T T^{-1} i_s \\
P_{el} &= u_s^T \begin{pmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2} \end{pmatrix} i_s \\
P_{el} &= u_s^T \begin{pmatrix} \frac{3}{2} & 0 \\ 0 & \frac{3}{2} \end{pmatrix} i_s \\
P_{el} &= \frac{3}{2} u_s^T i_s
\end{aligned} \tag{2.23}$$

Der Faktor $\frac{3}{2}$ kommt von der Anpassung an die Amplitude der Grundwelle (siehe Raumzeigertransformation).

Die Leistung erhält man, indem man die Gleichung 2.20 mit den jeweiligen Strömen multipliziert:

Die Leistungsbilanz lautet:

$$P_{el} = \frac{3}{2} R (I_d^2 + I_q^2) + \frac{3}{2} (\dot{\Psi}_d \, I_d + \dot{\Psi}_q \, I_q) + \frac{3}{2} (-\omega_l \, \Psi_q \, I_d + \omega_l \, \Psi_d \, I_q). \tag{2.24}$$

Die einzelnen Terme zusammengefasst und übersichtlich angeschrieben, ergeben:

$$P_{Cu} = \frac{3}{2} (R \, I_d^2 + R \, I_q^2) \tag{2.25}$$

$$P_{mag} = \frac{3}{2} (\dot{\Psi}_d \, I_d + \dot{\Psi}_q \, I_q) \tag{2.26}$$

$$P_m = \frac{3}{2} \, \omega_l (\Psi_d \, I_q - \Psi_q \, I_d) \tag{2.27}$$

Die einzelnen Leistung bezeichnen

- P_{el}: die elektrische Leistung die der Maschine zugeführt wird
- P_{Cu}: die Verlustleistung im Kupfer der Statorwicklungen
- P_{mag}: magnetische Leistung in der Spule
- P_m: die mechanische Leistung

Eine mechanische Leistung kann allgemein über:

$$P_{mech} = \omega_m \cdot M \tag{2.28}$$

beschrieben werden.

Bei einer Synchronmaschine muss noch die Polpaarzahl Z_p berücksichtigt werden. Die elektrische Winkelgeschwindigkeit ω_l mit welcher sich der Statorstromraumzeiger dreht, ist um den Faktor Z_p größer als die mechanische Winkelgeschwindigkeit ω_m des Rotors.

$$\omega_l = Z_p \cdot \omega_m.$$

Die Gleichung für eine mechanische Leistung 2.28 mit jener Gleichung für P_m aus der Leistungsbilanz 2.27 gleichsetzten:

$$P_{mech} = P_m$$

Daraus erhält man eine Beziehung für das Moment:

$$\frac{\omega_l}{Z_p} \cdot M = \frac{3}{2}\,\omega_l\,(\Psi_d\,I_q - \Psi_q\,I_d)$$

Gleichung umformen auf M:

$$M = \frac{3}{2} Z_p (\Psi_d\,I_q - \Psi_q\,I_d)$$

Die Gleichung steht für das innere Moment, das von der PMSM erzeugt wird. Im weiteren Text wird es als M_i bezeichnet.

2.3.3. Bewegungsgleichung

Für einen Drehzahlreger wird auch noch ein Formalismus benötigt. Dazu wird die mechanische Grundgleichung für die Drehung (Rotation) eines starren Körpers um eine Achse verwendet (Satz von EULER):

$$J\alpha = \Sigma M \tag{2.29}$$

Das Trägheitsmoment J multipliziert mit der Winkelbeschleunigung α entspricht der Summe aller auftretenden Momente.

Als Vereinfachung werden beim Modell nur zwei Momente verwendet:

- M_i: das innere Moment das von der Maschine erzeugt wird
- M_l: ein Lastmoment, im Lastmoment wird die gesamte Belastung zusammengefasst, auch auftretende Verluste in der Maschine

ω_m wird als Integral von α eingeführt:

$$J\dot{\omega_m} = M_i - M_l \tag{2.30}$$

2.3.4. Zusammenfassung

Als Übersicht ist hier das zuvor hergeleitete Modell vollständig angeschrieben:

$$\begin{aligned} U_d &= R\ I_d + \dot{\Psi}_d - \omega_l\ \Psi_q \\ U_q &= R\ I_q + \dot{\Psi}_q + \omega_l\ \Psi_d \\ \Psi_d &= L_d\ I_d + \Psi_{PM} \\ \Psi_q &= L_q\ I_q \\ M_i &= \frac{3}{2} Z_p(\Psi_d\ I_q + \Psi_q\ I_d) \\ J\dot{\omega_m} &= M_i - M_l \end{aligned} \tag{2.31}$$

3. Regler

Wie bereits bei der Modellbildung erwähnt, wird ein feldorientierter Regler eingesetzt. Durch die rechentechnische Aufteilung in eine drehmoment- und flussbildende Komponente erreicht man ein ähnliches Verhalten wie bei der Gleichstrommaschine. Durch die Regelung im mitdrehenden Koordiantensystem sind die Ströme im stationären Zustand Gleichgrößen und nicht sinusförmig, was die Arbeit für den PI-Regler vereinfacht. Nach einer Entkopplung der Flussgleichungen, handelt es sich bei den Übertragungsfunktionen um Eingrößensysteme. Deshalb kann für den Reglerentwurf das FKL-Verfahren [1] angewendet werden. In Matlab ist das einfach über das `SISOTOOL` möglich.

3.1. Entwurf

Für den Entwurf wird das zuvor hergeleitete Modell verwendet:

$$\begin{aligned}
\dot{\Psi}_d &= U_d - RI_d + \omega_l \Psi_q \\
\dot{\Psi}_q &= U_q - RI_q - \omega_l \Psi_d \\
\Psi_d &= \Psi_{PM} + L_d I_d \\
\Psi_q &= L_q I_q \\
M_i &= \frac{3}{2} Z_p (\Psi_d I_q - \Psi_q I_d) \\
J\dot{\omega_m} &= M_i - M_l \\
\omega_l &= Z_p \, \omega_m
\end{aligned} \tag{3.1}$$

Es wird ein Motor mit an der Oberfläche montierten Magneten verwendet, das Bedeutet: $L_d = L_q = L$. Durch die Betriebsart kann das Modell vereinfacht werden. Es soll nur der Grundstellbereich (kein Feldschwächbetrieb $I_d < 0$ gefordert) verwendet werden. Dadurch kann I_d dauerhaft null gesetzt werden.

Die Momentengleichung vereinfacht sich deshalb zu:

$$M_i = \frac{3}{2} Z_p \Psi_{PM} I_q \tag{3.2}$$

Im Mikrocontroller ist ein Filter beim Interface zu den inkremental Encodern vorhanden. Dies wurde im Reglerentwurf nicht berücksichtigt. Als Parameter für den Motor wurden die Daten aus dem Datenblatt verwendet. Es wurde versucht mit dem Testaufbau die Parameter zu messen. Die Resultate waren nicht sehr genau. Durch die große Anzahl an Peripherie am Evaluation-Board waren anscheinend zu viele Störungen. Deswegen wurden wie bereits erwähnt die Daten aus dem Datenblatt verwendet.

[1] Frequenzkennlinienverfahren

3.1.1. Entkopplung

Die beiden Spannungsgleichungen bilden ein verkoppeltes System. Die Differentialgleichungen der Flüsse Ψ_d und Ψ_q enthalten einen Term des jeweils anderen Flusses. Das bedeutet mehr „Arbeitffür den Regler, dieser muss diesem Einfluss entgegenwirken. Als Abhilfe kann man die Differentialgleichungen in einen linearen und einen Kompensationsterm aufteilen. Der Kompensationsterm wird nach dem PI-Regler dazu addiert. Ein weiterer Vorteil, der sich daraus ergibt ist, dass die beiden Differentialgleichungen der zwei Stromregler nach der Entkopplung äquivalent sind. Der Unterschied besteht in den Koppel-Termen, diese müssen aus den aktuellen Messwerten berechnet werden.

$$\begin{aligned} u_d^* &= u_{d,k} + u_{d,lin} \\ u_q^* &= u_{q,k} + u_{q,lin} \\ L\frac{dI_d}{dt} &= -RI_d + u_{d,lin} \\ L\frac{dI_q}{dt} &= -RI_q + u_{q,lin} \\ u_{d,k} &= -\omega_l L I_q \\ u_{q,k} &= \omega_l L I_d + \omega_l \Psi_{PM} \end{aligned}$$

3.1.2. Stromregler

Die d- und q-Komponenten besitzen identische Übertragungsfunktionen. Das bedeutet, dass nur ein Regler ausgelegt werden muss. Die Übertragungsfunktion von Statorspannung zu Statorstrom lautet:

$$G_i(s) = \frac{\frac{1}{L}}{s + \frac{R}{L}}.$$

Unter Berücksichtigung des Abtast- und Halteglieds ergibt sich die zeitdiskrete Übertragungsfunktion:

$$G_i(z) = (1 - z^{-1})\mathcal{Z}\{\frac{G_i(s)}{s}\}.$$

Für den Entwurf wird noch mit z^{-1} der Pulswechselrichter berücksichtigt:

$$P_i(z) = z^{-1}G_i(z)$$

Diese Übertragungsfunktion muss noch in den q-Bereich transformiert werden, um das FKL-Verfahren für zeitdiskrete Systeme [7] verwenden zu können. Die Durchtrittsfrequenz soll ungefähr um dem Faktor zehn größer sein als die des überlagerten Drehzahlreglers.

3.1.3. Drehzahlregler

Die benötigte Differentialgleichung wird im Bildbereich angeschrieben:

$$G_\omega(s) = \frac{\frac{1}{J}\Psi_{PM} Z_p}{s}$$

Berücksichtigen des Abtast- und Halteglieds:

$$G_\omega(z) = (1 - z^{-1})\mathcal{Z}\{\frac{G_\omega(s)}{s}\}$$

Die gesamte Übertragungsfunktion des offenen Kreises ohne Drehzahlregler lautet mit Berücksichtigung des inneren Stromregelkreises:

$$P_\omega(z) = \frac{R_i(z) z^{-1} G_i(z) G_\omega(z)}{1 + R_i(z) P_i(z)}$$

Die Übertragungsfunktion der Strecke wurde vereinfacht (dabei wird nicht berücksichtigt, dass der "Übergang" vom Strom- zur Drehzahl im zeitkontinuierlichen Bereich des Systems stattfindet). Wieder die Übertragungsfunktion in den q-Bereich transformieren und das FKL-Verfahren für zeitdiskrete Systeme anwenden. Der Drehzahlregler würde nur für Testzwecke erstellt, der Drehmomentenregler war wichtiger.

3.2. Raumzeigermodulation

Die sogenannte Raumzeigermodulation wird verwendet, um aus dem Stator-Spannungsraumzeiger ein PWM-Muster für die Halbbrücken zu generieren [10, 8]. Mit drei Halbbrücken sind acht verschiedene Pulsmuster ($2^3 = 8$) möglich. Das bedeutet im orthogonalen Modell: sechs Spannungsraumzeiger im Abstand von 60° und zwei Nullspannungsraumzeiger (entweder Schalter ein oder aus). Bei einer Dreieck- oder Sternschaltung der Spulen ergeben sich unterschiedliche Spannungszeiger.In den Tabellen 3.1 und 3.2 sind diese Raumzeiger angegeben. Der Faktor U_{zk} wurde in den beiden Tabellen nicht angegeben, um die Lesbarkeit zu vereinfachen. Die Spannungen sind normiert.

Der resultierende Spannungszeiger ist eine Kombination aus den beiden Zeigern die den Sektor begrenzen und aus einem der beiden Nullzeiger.

Wichtig für das Programm ist die Berechnung der *duty-cycles* der einzelnen Halbbrücken aus einem Spannungsraumzeiger. Diese Berechnung ist weiter unten 3.2.1 beschrieben.

Die maximale Zeigerlänge ist gleich der Höhe des aufgespannten Dreiecks

$$|U| = \frac{\sqrt{3}}{2} v_i. \tag{3.3}$$

Bei der Dreieckschaltung beträgt die maximale Zeigerlänge daher:

$$|U| = \frac{\sqrt{3}}{2} \cdot \frac{2}{\sqrt{3}} = 1, \tag{3.4}$$

bei der Sternschaltung:

$$|U| = \frac{\sqrt{3}}{2} \cdot \frac{2}{3} = \frac{1}{\sqrt{3}}. \tag{3.5}$$

	S_A	S_B	S_C	U_A	U_B	U_C	u_α	u_β	U_S
v_0	0	0	0	0	0	0	0	0	0
v_1	1	0	0	$\frac{2}{3}$	$-\frac{1}{3}$	$-\frac{1}{3}$	$\frac{2}{3}$	0	$\frac{2}{3}$
v_2	1	1	0	$\frac{1}{3}$	$\frac{1}{3}$	$-\frac{2}{3}$	$\frac{1}{3}$	$\frac{1}{\sqrt{3}}$	$\frac{2}{3}e^{j\frac{\pi}{3}}$
v_3	0	1	0	$-\frac{1}{3}$	$\frac{2}{3}$	$-\frac{1}{3}$	$-\frac{1}{3}$	$\frac{1}{\sqrt{3}}$	$\frac{2}{3}e^{j\frac{2\pi}{3}}$
v_4	0	1	1	$-\frac{2}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$-\frac{2}{3}$	0	$\frac{2}{3}e^{j\pi}$
v_5	0	0	1	$-\frac{1}{3}$	$-\frac{1}{3}$	$\frac{2}{3}$	$-\frac{1}{3}$	$-\frac{1}{\sqrt{3}}$	$\frac{2}{3}e^{j\frac{4\pi}{3}}$
v_6	1	0	1	$\frac{1}{3}$	$-\frac{2}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$-\frac{1}{\sqrt{3}}$	$\frac{2}{3}e^{j\frac{5\pi}{3}}$
v_7	1	1	1	0	0	0	0	0	0

Tabelle 3.1.: Schaltmuster und Raumzeiger bei Sternschaltung

	S_A	S_B	S_C	U_{AB}	U_{BC}	U_{CA}	u_α	u_β	U_S
v_0	0	0	0	0	0	0	0	0	0
v_1	1	0	0	1	0	-1	1	$\frac{1}{\sqrt{3}}$	$\frac{2}{\sqrt{3}}e^{j\frac{\pi}{6}}$
v_2	1	1	0	0	1	-1	0	$\frac{2}{\sqrt{3}}$	$\frac{2}{\sqrt{3}}e^{j\frac{\pi}{2}}$
v_3	0	1	0	-1	1	0	-1	$\frac{1}{\sqrt{3}}$	$\frac{2}{\sqrt{3}}e^{j\frac{5\pi}{6}}$
v_4	0	1	1	-1	0	1	-1	$-\frac{1}{\sqrt{3}}$	$\frac{2}{\sqrt{3}}e^{j\frac{7\pi}{6}}$
v_5	0	0	1	0	-1	1	0	$-\frac{2}{\sqrt{3}}$	$\frac{2}{\sqrt{3}}e^{j\frac{3\pi}{2}}$
v_6	1	0	1	1	-1	0	1	$-\frac{1}{\sqrt{3}}$	$\frac{2}{\sqrt{3}}e^{j\frac{11\pi}{6}}$
v_7	1	1	1	0	0	0	0	0	0

Tabelle 3.2.: Schaltmuster und Raumzeiger bei Dreieckschaltung

3.2.1. Schaltmuster

Hier wird beschrieben wie man aus einem beliebigen Spannungsraumzeiger, die Dauer t_i der einzelnen Raumzeiger (v_i) bzw. Schaltmuster (siehe Tabellen 3.1) berechnen kann. Für die Berechnung werden normierte Zeiten verwendet:

$$\tau_i = \frac{t_i}{T} \tag{3.6}$$

Wie bereits erwähnt besteht der gewünschte Spannungsraumzeiger aus den beiden begrenzenden Zeiger des Sektors, diese werden mit v_l und v_r bezeichnet und einem Nullzeiger. Der benötigte Nullzeiger wird zu gleichen Teilen auf die beiden Zeiger: v_0 und v_7

aufgeteilt.

$$\tau_0 + \tau_7 + \tau_l + \tau_r = 1 \tag{3.7}$$

Die Zeiten werden symmetrisch aufgeteilt:

$$\frac{\tau_0}{2} + \frac{\tau_l}{2} + \frac{\tau_r}{2} + \frac{\tau_7}{2} + \frac{\tau_7}{2} + \frac{\tau_r}{2} + \frac{\tau_l}{2} + \frac{\tau_0}{2} = 1 \tag{3.8}$$

Den Raumzeiger U wird folgendermaßen beschrieben:

$$U = \hat{U}\, e^{j\gamma} = u_\alpha + j u_\beta \tag{3.9}$$

Im Programm ist es einfacher mit den Komponenten des Raumzeigers zu rechnen. Für alle Sektoren wird diese Gleichung angesetzt und die entsprechenden Spannungszeiger eingesetzt:

$$u_\alpha + j u_\beta = v_l \cdot \tau_l + v_r \cdot \tau_r \tag{3.10}$$

$$\tag{3.11}$$

Sie wird nach τ_l und τ_r aufgelöst. Die Zeiten für die Nullzeiger können wie folgt berechnet werden:

$$\tau_0 + \tau_7 = 1 - \tau_l - \tau_r \tag{3.12}$$

Für eine einfachere Darstellung wurden die Spannungszeiger normiert: $\frac{v_i}{U_{ZK}/2}$.

Sektor	Aktive Vektoren	
I	$\tau_1 = \frac{3}{4}u_\alpha - \frac{\sqrt{3}}{4}u_\beta$	$\tau_2 = \frac{\sqrt{3}}{2}u_\beta$
II	$\tau_2 = \frac{3}{4}u_\alpha + \frac{\sqrt{3}}{4}u_\beta$	$\tau_3 = -\frac{3}{4}u_\alpha + \frac{\sqrt{3}}{4}u_\beta$
III	$\tau_3 = \frac{\sqrt{3}}{2}u_\beta$	$\tau_4 = -\frac{3}{4}u_\alpha - \frac{\sqrt{3}}{4}u_\beta$
IV	$\tau_4 = -\frac{3}{4}u_\alpha + \frac{\sqrt{3}}{4}u_\beta$	$\tau_5 = -\frac{\sqrt{3}}{2}u_\beta$
V	$\tau_5 = -\frac{3}{4}u_\alpha - \frac{\sqrt{3}}{4}u_\beta$	$\tau_6 = \frac{3}{4}u_\alpha - \frac{\sqrt{3}}{4}u_\beta$
VI	$\tau_6 = -\frac{\sqrt{3}}{2}u_\beta$	$\tau_1 = \frac{3}{4}u_\alpha + \frac{\sqrt{3}}{2}u_\beta$

Tabelle 3.3.: Zeiten Vektoren bei Sternschaltung

3.2.2. Bestimmen des Sektors

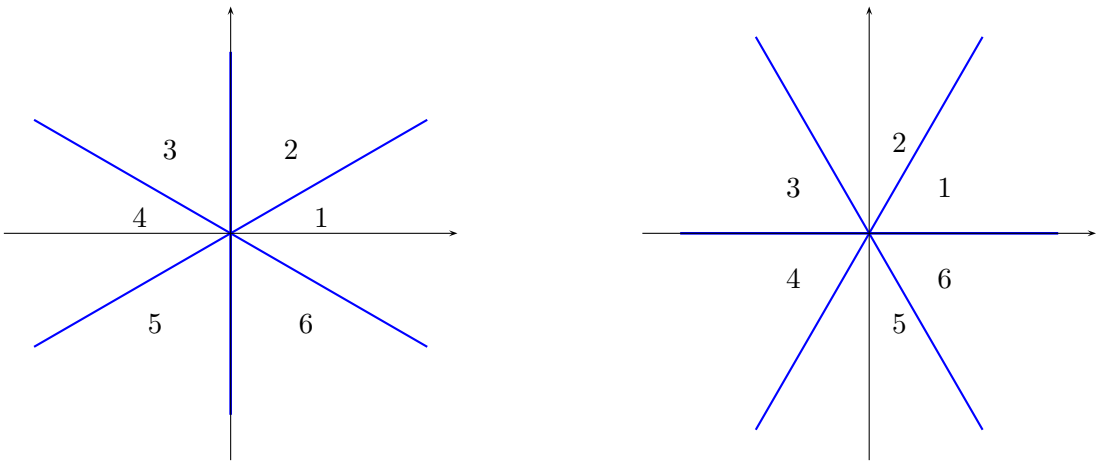
Um das gewünschte Schaltmuster aus einem Spannungsraumzeiger generieren zu können, muss man wissen in welchem Sektor sich der Zeiger befindet. Eine Möglichkeit ist es, den Winkel über den Arkustanges aus den beiden Komponenten (U_α, U_β) zu berechnen. Über den Winkel kann anschließend der Sektor bestimmt werden. Diese Operation ist auf einem Mikrocontroller aber sehr rechenintensiv.

Sektor	Aktive Vektoren	
I	$\tau_6 = \frac{1}{4}u_\alpha - \frac{\sqrt{3}}{4}u_\beta$	$\tau_1 = \frac{1}{4}u_\alpha + \frac{\sqrt{3}}{4}u_\beta$
II	$\tau_1 = \frac{1}{2}u_\alpha$	$\tau_2 = -\frac{1}{4}u_\alpha + \frac{\sqrt{3}}{4}u_\beta$
III	$\tau_2 = \frac{1}{4}u_\alpha + \frac{\sqrt{3}}{4}u_\beta$	$\tau_3 = -\frac{1}{2}u_\alpha$
IV	$\tau_3 = -\frac{1}{4}u_\alpha + \frac{\sqrt{3}}{4}u_\beta$	$\tau_4 = -\frac{1}{4}u_\alpha - \frac{\sqrt{3}}{4}u_\beta$
V	$\tau_4 = -\frac{1}{2}u_\alpha$	$\tau_5 = \frac{1}{4}u_\alpha - \frac{\sqrt{3}}{4}u_\beta$
VI	$\tau_5 = -\frac{1}{4}u_\alpha - \frac{\sqrt{3}}{4}u_\beta$	$\tau_6 = \frac{1}{2}u_\alpha$

Tabelle 3.4.: Zeiten Vektoren bei Dreieckschaltung

Eine weniger aufwändige Alternative ist, die Sektorgrenzen durch ihre Geradengleichungen zu beschreiben und mit dem Spannungsraumzeiger zu vergleichen. Die Überprüfung gestaltet sich sehr einfach, die Komponenten des Zeigers (U_α, U_β) in die Gleichungen (siehe Tabelle 3.5 und 3.6) einsetzten und das Ergebnis auf dessen Vorzeichen überprüfen. Der Sektor ergibt sich aus der Vorzeichenkombination.

Die Sektoren können mit drei Geraden beschrieben werden (siehe Abbildung 3.1). Eine Gerade fällt mit einer Achse zusammen. Die anderen beiden Geraden haben dieselbe absolute Steigung. Sie werden mit der Geradengleichung: $u_\beta = k \cdot u_\alpha$ angesetzt. Für die Berechnung der Steigung k werden die Punkte $P_S(\frac{1}{2}/\frac{\sqrt{3}}{2})$ und $P_D(\frac{\sqrt{3}}{2}/\frac{1}{2})$ (Stern- oder Dreieckschaltung) verwendet.



(a) Dreieckschaltung (b) Sternschaltung

Abbildung 3.1.: Sektoren die von den sechs möglichen Raumzeigern aufgespannt werden

Sektor	u_β	$\sqrt{3}u_\alpha - u_\beta$	$\sqrt{3}u_\alpha + u_\beta$
1	+	+	+
2	+	-	+
3	+	-	-
4	-	-	-
5	-	+	-
6	-	+	+

Tabelle 3.5.: Geradengleichungen zur Bestimmung des Sektors bei einer Sternschaltung der Motorwicklungen

Sektor	u_α	$u_\alpha - \sqrt{3}u_\beta$	$u_\alpha + \sqrt{3}u_\beta$
1	+	+	+
2	+	-	+
3	-	-	+
4	-	-	-
5	-	+	-
6	+	+	-

Tabelle 3.6.: Geradengleichungen zur Bestimmung des Sektors bei einer Dreieckschaltung der Motorwicklungen

4. Programm

Im folgenden Kapitel wird das am Mikrocontroller ausgeführte Programm beschrieben und dokumentiert. Der Code wurde großteils in C geschrieben, einzelne Teile des Startup-Codes oder spezielle Initialisierungen von Prozessoreigenschaften jedoch in Assembler.

Das Programm ist sehr modular aufgebaut. Um verschiedene Hardware zu unterstützen sind die dafür wichtigen Module sehr leicht auszutauschen. Beim Programmieren wurde versucht für die unterschiedliche Hardware ein einheitliches Interface bereitzustellen. Die Hauptaufgabe ist die Motorregelung auf beiden Platinen (STM-Motor-Board und TERA-Motorplatine). Die verwendete Hardware ist in Kapitel 6 beschrieben.

Für einfache Einstellungen und eine Benutzersteuerung wurde ein VT100-kompatibles Terminal über die serielle Schnittstelle implementiert. Über das Terminal ist eine einfache Anzeige und eine simple Menüführung realisiert. Weiters kann die Motorsteuerung noch über den CAN-Bus gesteuert werden.

Das Programm verwendet kein Betriebssystem, der zeitliche Ablauf ist interruptgesteuert. Ein Betriebssystem für den Mikrocontroller zu adaptieren wäre zu aufwändig gewesen. Die wichtigste Aufgabe, die Regelung des Motors, kann sehr leicht über eine Interruptsteuerung bewerkstelligt werden.

4.1. Mikrocontroller

4.1.1. Allgemein

Die ausgewählte Mikrocontrollerserie `STM32F40x` von STMicroelectronics verwendet als Prozessor einen IP-Core der Firma `ARM Limited` [1]. Der Prozessor ist eine für Mikrocontrolleranwendungen optimierte Version des `ARMv7-ME` Kernes, mit der Bezeichnung `Cortex-M4`. Dieser hat eine DSP-Erweiterung: den 16/32-bit MAC-Befehl [2] und die 8/16-Bit SIMD [3] Befehle. Außerdem hat der Prozessor noch einen integrierten FPU-Coprozessor.

4.1.2. Speicher

Nach außen hin (Programmiermodell) ist der `Cortex-M4` ein Von-Neumann-Modell, d.h. alle physikalischen Speicher und alle Register befinden sich in einem linearen Adressraum. Die physischen Speicher sind untereinander und mit dem Prozessor über eine

[1] http://www.arm.com/
[2] multiply and accumulate: ein Befehl multipliziert und addiert mehrere Variablen
[3] single instruction multiple data: Befehle, die mit einem Aufruf gleichzeitig mehrere gleichartige Datensätze bearbeiten können

Bus-Matrix verbunden. Im *reference manual* [3] zu der Microcontroller-Serie ist in Abbildung 1 die Bus-Matrix dargestellt. Im Datenblatt des Mikrocontrollers befindet sich eine übersichtliche *memory map* über den gesamten Speicherbereich und die Speicherbereiche der einzelnen Peripherien. Für den verwendeten `STM32f40x` [2], siehe Abbildung 16 (`STM32F40x` *memory map*) und Tabelle 9 (`STM32F40x` *register boundary addresses*).

Der `STM32F4xx` besitzt folgende Speicher:

- 1024kb Flash
- 128kb SRAM [4]
- 64kb core coupled SRAM (im weiteren CCRAM genannt)
- 4kb backup SRAM (im weiteren BKPSRAM genannt)

Die Adressbereiche der einzelnen Speicher sind in Tabelle 4.1.2 aufgelistet.

Speicher	Adressbereich	Größe [kb]
Flash	0x0800 0000 - 0x080F FFFF	1024
CCRAM	0x1000 0000 - 0x1000 FFFF	64
SRAM1	0x2000 0000 - 0x2001 BFFF	112
SRAM2	0x2001 C000 - 0x2001 FFFF	16
BKPSRAM	0x4002 4000 - 0x4002 4FFF	4

Tabelle 4.1.: Übersicht über die einzelnen Speicher

Flash Der interne Flash in den `STM32F4`-Mikrocontrollern besteht aus 12 Sektoren, aufsteigend nach Adresse sind das vier 16kb, ein 64kb und sieben 128kb Sektoren.

SRAM Die 128kb SRAM sind in einen 112kb und 16kb Bereich aufgeteilt. Diese beiden Bereiche sind eigenständig mit der Bus-Matrix verbunden. Das bringt den Vorteil, dass unterschiedliche Peripherien gleichzeitig auf je einen der getrennten Bereiche zugreifen können. Zum Beispiel liest/schreibt die CPU im 112kb SRAM während eine andere Peripherie (Ethernet, USB, ...) auf den 16kb SRAM liest/schreibt.

CCRAM Der CCRAM ist direkt (nicht wie andere Speicherbereiche über die Bus-Matrix) an die CPU angebunden. Nur die CPU (über standardmäßige Schreib-/Lesebefehle) kann auf diesen Speicherbereich zugreifen. Da der DMA-Controller über die Bus-Matrix auf die Speicherbereiche zugreift, kann der CCRAM nicht für DMA-Buffer verwendet werden. Er eignet sich aber sehr gut für Variablen (Daten), Stack oder den Heap.

BKPSRAM Der Mikrocontroller besitzt noch einen 4kb Backup-SRAM. Dieser kann über eine externe Batterie versorgt werden, der Mikrocontroller hat dazu einen extra Pin (VBAT), an den eine Batterie angeschlossen werden kann. Im Mikrocontroller wird bei

[4] statischer RAM, ist ein flüchtiger Speicher: nach Abschaltung sind die gespeicherten Daten verloren; die einzelnen Speicherzellen müssen aber nicht periodisch aufgefrischt werden

Fehlen der Standard-Versorgungsspannung oder in einem Standby-Modus automatisch intern auf die Batterie umgeschaltet. Bei einer dauernden Versorgung über VBAT ist der Speicherbereich wie ein internes EEPROM. Dieser Speicher ist, wie der CCRAM, nur von der CPU aus erreichbar. Um den Speicherbereich zu verwenden, muss der betreffende Clock aktiviert sein.

4.1.3. Remapping

Der Cortex-M4 Kern ist über drei Busse mit der Bus-Matrix verbunden. Davon wird ein Bus (I-Bus) speziell für Instruktionen verwendet. Dies entlastet die anderen beiden Busse vom Cortex-M4 Kern, die Ausführung ist performanter. Der I-Bus wird verwendet, wenn Befehle von der Adresse 0 geladen werden sollen (die Remappingfunktionalität wird verwendet). In dem Bereich von Adresse 0x00000 bis 0xFFFFF (die ersten 1024kb) können andere Speicherbereiche hinein gemappt werden; dies geschieht abhängig von den ersten beiden Bits im `SYSCFG_MEMRMP` Register. Es können entweder der SRAM, der interne Flash, ein externer Speicher (angebunden über den FSMC) oder das System-Memory (embedded bootloader) gemappt werden. Im Abschnitt 5.3.4 wird das Linker-Script beschrieben, das die Aufteilung des Codes in die Speicherbereiche tätigt. Eine mögliche Codeausführung im RAM ist in Abschnitt 5.3.4 beschrieben.

Nach einem Reset wird der Wert des `SYSCFG_MEMRMP`-Registers durch die beiden BOOT-Pins bestimmt. Dieses Register kann auch während dem Betrieb jederzeit geändert werden. Die Speicher sind trotz remapping weiterhin unter ihrer ursprünglichen Adresse erreichbar.

4.1.4. FPU

Der verwendete Cotex-M4 Kern hat einen FPU Coprozessor, im weiteren nur als FPU bezeichnet. Um diesen zu nutzen, muss der Coprozessor nach dem Reset des Mikrocontrollers aktiviert werden. Zusätzlich muss noch der Compiler die richtigen Befehle generieren. Die FPU ist eine Implementierung der *ARMv7-M floating point extension* (`FPv4-SP`) und ist kompatibel mit dem ANSI/IEEE Std 754-2008 *IEEE Standard for Binary Floating-Point Arithmetic*. Die FPU unterstützt folgende Rechenoperationen mit einfacher Genauigkeit: Addition, Subtraktion, Multiplikation, Division und Wurzelziehen. Außerdem erlaubt sie eine Umwandlung zwischen Festkomma- und Fließkomma-Typen.

Ist der Compiler und/oder die C-Bibliothek nicht für die Verwendung einer FPU compiliert, tritt beim Erstellen der Binärdatei der in Listing 4.1 angeführte Fehler auf. Zur besseren Darstellung wurden im Listing die Pfade vereinfacht, der Sinn der Fehlermeldung ist aber der Gleiche geblieben.

Listing 4.1: Fehlermeldung, GCC keine FPU unterstützung

```
.../codesourcery/arm-none-eabi/bin/ld: error: build/flash.elf
    uses VFP register arguments,
    .../codesourcery/arm-none-eabi/lib/thumb2/libc.a
    (lib_a-errno.o) does not
```

4.2. Bibliotheken

Für eine einfachere Verwendung des Mikrocontrollers wurden oft wiederkehrende Programmteile und häufige Arbeitsabläufe zu einfachen Modulen zusammengefasst. Für den Prozessor existieren von den Herstellern bereits sehr praktische Bibliotheken.

4.2.1. Hersteller-Bibliotheken

CMSIS

Die Herstellerfirma (ARM Limited) des Prozessorkernes (Cortex-M4) bietet eine Bibliothek an, die ein Interface zu den Funktionen und Registern des Prozessors bildet. Die Bibliothek heißt CMSIS [5], eine genaue Beschreibung findet man auf der Herstellerseite [6]. Sie stellt einfache C-Wrapperfunktionen für die Assemblerbefehle sowie Funktionen für die Konfiguration/Einstellung der CPU Peripherie (FPU, NVIC, Systick) zur Verfügung. Die Bibliothek kann einfach von der Herstellerseite heruntergeladen werden. Die beiden Funktionen: `__get_FPSCR` und `__set_FPSCR` in der Datei `core_cmfunc.h` wurden auskommentiert. Sie verursachen, da die FPU nicht verwendet wird, Compiler-Warnings.

Standard Peripheral Lib

Diese Bibliothek ist von STMicroelectronics. Sie deckt die komplette Peripherie des Mikrocontrollers ab, für diese bietet sie Funktionen mit denen die Register einfacher modifiziert werden können. Dadurch wird eine Art hardware-abstraction layer (HAL) der Peripherie gebildet, was das Benutzen des Mikrocontrollers sehr erleichtert. Ein weiterer Vorteil ist, dass diese Library für jede Peripherie ein paar Beispielprogramme enthält.

An dieser Bibliothek wurden folgende Änderungen vorgenommen:

- In der Datei `stm32f4xx_rcc.c`: bei der konstanten Variable `APBAHBPrescTable` wurde `volatile` zu `const` umgeändert, um Speicherplatz im SRAM zu sparen.
- In der Datei `stm32f4xx.h` wurde die eigene `config.h` includiert, um das Zusammenspiel mit der eigenen Projektstruktur zu erleichtern.

4.2.2. Eigene Bibliothek

Hardware Für Hardware am im Team entwickelten Prozessorboard oder am Developer-Board wurden einige Module erstellt. Für die am Developer-Board vorhandene und verwendete Hardware wurde der mitgelieferten Source-Code von STMicroelectronics verwendet und angepasst. Die Quelldateien sind in der Eclipse-Projektstruktur im Ordner `./src/board/*` gespeichert (siehe 5.3.1).

- button: Funktionen für die Initialisierung/Verwendung der Taster beim Eval-Board: STM3240G

[5] Cortex Microcontroller Software Interface Standard
[6] http://www.arm.com/cmsis/

- eval_fsmc_sram: Initialisierung des SRAM-Chip. Es ist ein 16MBit (2Mb) SRAM auf dem Developer-Board verbaut.
- fonts: Schriftarten für das LCD
- led: Funktionen für das Ein-/Ausschalten der LEDs, am Developer-Board sind vier LEDs, am eigenen Prozessorboard sind zwei LEDs vorhanden
- stm324xg_eval_lcd: Funktionen für das Schreiben von Text und Zahlen am LCDs

Für das Prozessorboard wird nur das Modul *led* verwendet, ansonsten ist keine Hardware am Prozessorboard verbaut.

Peripherie/Module In der folgenden Aufzählung sind Module für allgemein verwendete Peripherie oder für projektübergreifende Softwaremodule (Protokolle, ...) aufgelistet. Die Module sind in der Eclipse-Projektstruktur im Ordner `./src/modules/*` gespeichert (siehe 5.3.1).

bkpsram: Die bereitgestellte Initialisierungsfunktion muss aufgerufen werden, um den BKPSRAM verwenden zu können. Wird eine 3,3V Versorgung mit dem VBAT Pin des Mikrocontrollers verbunden, geht der Inhalt des Speicherbereichs nicht verloren, solange die Versorgung verbunden ist.

can: Implementiert das verwendete CAN-Protokoll, eine detaillierte Beschreibung ist unter 4.4.3 zu finden.

dac_output: Gedacht für die Ausgabe von globalen Variablen über den Digital-Analog-Wandler. Der nutzbare Wertebereich beträgt 0 - 4096, dieser wird abgebildet auf den Spannungsbereich 0 - 3,3V. Somit kann man mit einem Oszilloskop gleichzeitig Spannungsverläufe auf der Platine und interne Variablen des Mikrocontrollers darstellen. Der Mikrocontroller besitzt zwei DAC-Ausgänge, das heißt es können maximal zwei Variablen gleichzeitig dargestellt werden.

delay: Implementiert *busy-waiting*, wird von den Display-Funktionen und von einigen selbst geschriebenen Initialisierungsfunktionen benötigt.

eeprom: Realisiert einen nicht flüchtigen Speicher im Flash-Speicher des Mikrocontrollers. Dieser kann dazu verwendet werden, um Einstellungen dauerhaft zu speichern. Die Funktionalität des Speichers ist im Abschnitt 4.4.5 genau beschrieben.

fifo: Eine Byte-FIFO, wird von dem "usart"-Modul benötigt.

mem_copy: Stellt eine Funktion zur Verfügung, die Speicherbereiche im SRAM über einen DMA-Stream kopiert.

stdio: Stellt eine standard Aus- und Eingabe über die serielle Schnittstelle zur Verfügung. Nutzt dazu die syscall-Funktionen `read` und `write`. Es ist eine formatierte Ausgabe über die serielle Schnittstelle mit den `printf`-Funktionen möglich.

syscall: Implementiert die in der C Standard-Bibliothek definierten "syscall stubs" um Funktionen der Standardbibliothek zu unterstützen die auf die Hardware zugreifen müssen.

systick: Funktionen für die Konfiguration des Systick.

terminal: Stellt Kommandos für ein VT100-kompatibles Terminal zur Verfügung.

timing: Stellt einen 32bit-Timer zur Verfügung, mit dem Zeitabstände gemessen werden können. Die Auflösung beträgt 250ns.

usart: Implementiert eine byte-orientierte asynchrone serielle Schnittstelle. Es existieren einfache Funktionen, um Zeichenketten zu senden.

Die aufgeführten Module können über die `config.h` einzeln eingebunden werden (siehe Abschnitt 5.3.1).

4.3. Regler

In diesem Abschnitt wird die Implementierung des Reglers beschrieben. Die Module, aus denen sich der Regler zusammensetzt, sind in der folgenden Auflistung angeführt.

advanced_timer: Funktionen für den Timerbaustein, der verwendet wird, um die drei Halbbrücken anzusteuern.

controller: Implementiert einen PI-Regler und die Regelschleifen für Strom- und Drehzahlregler.

feedback: Funktionen für das Auslesen des Rotorwinkelsensors und die Berechnung der Drehzahl. Das Modul unterstützt aktuell drei verschiedene Sensortypen.

memory: Zum Mitloggen von Werten. Die aufgezeichneten Werte werden im Speicher welcher am Evaluation-Board vorhanden ist gespeichert.

svpwm: Strommessung und Berechnung der Raumzeigermodulation. Es werden zwei Strommessverfahren unterstützt: direkte Messung des Phasenstroms oder Messung des Stromes durch den low-side MOSFET.

timebase: Timer für periodische Abläufe.

transformation: Implementiert die nötigen Transformationen: das Umrechnen in die verschiedenen Koordinatensysteme und in Raumzeiger.

ui: Benutzerinterface über ein serielles Terminal.

Da die FPU nicht verwendet werden konnte, muss eine andere Möglichkeit für die Darstellung der Werte gewählt werden. Es wurde eine Festkommadarstellung gewählt (siehe 4.3.2). Für die internen Regelgrößen sowie Strom- und Spannungen werden normierte Werte verwendet. Dies kann sehr komfortabel mit den Festkommazahlen realisiert werden (auf eins normierte Werte). Die Ausgangsspannung des Reglers wird auf U_{ZK} normiert. Der Strom wird auf den Messbereich ($\pm 2.3A$) normiert. Die gemessene Drehzahl wird nicht normiert, sie wird in Zehntel Hz gemessen.

Das Programm ist, wie bereits erwähnt, interruptgesteuert, weil kein Betriebssystem verwendet wird. Die Stromregler (I_d, I_q) werden vom ADC-Interrupt aufgerufen, siehe 4.3.3. Der Drehzahlregler wird von einem separaten Timer-Interrupt aufgerufen. Die restlichen Programmteile werden entweder von einem niedrig priorisierten Timer aufgerufen, oder in der `main-loop` abgehandelt.

Wie in 5.3.4 beschrieben, kann der Binärcode vom RAM ausgeführt werden. Das soll einen schnellere Ausführung gegenüber dem internen Flash bringen. Dieser Vorteil soll für den Regler genutzt werden. Um zu überprüfen, wie groß der Unterschied in der

Ausführungszeit ist, wird ein Port-Pin auf eins gesetzt solange der Stromregler arbeitet. Mit einem Oszilloskop wurde die Dauer gemessen. Bei einer Ausführung aus dem Flash haben die Berechnungen $t_F = 10,18\mu s$ gedauert, bei der Ausführung aus dem RAM dauerten die Berechnungen $t_R = 10,157\mu s$. Diese Werte sind ein Durchschnitt aus 10'000 Messungen. Die Differenz beträg nur: $\delta t = 23ns$. Dieser Unterschied ist kleiner als erhofft. Die Ausführung im RAM bringt eigentlich keinen Vorteil.

Die Ansteuerung der Halbbrücken erfolgt über einen sogenannten Advanced-Timer. Das ist die Bezeichnung einer speziellen Timer-Peripherie im Mikrocontroller. Dieser Timer besitzt drei *capture-compare* Kanäle zur Generierung einer PWM. Die Kanäle verfügen jeweils über einen komplementären Ausgang für die Ansteuerung einer Halbbrücke. Diese Ausgangseinheit hat auch einen integrierten Totzeitgenerator.

In der Datei `drive_config.h` werden die nötigen Einstellungen vorgenommen. Die folgenden Komponenten werden konfiguriert: verwendete Ports für Aus- und Eingänge, Motorparameter, Motortyp, Advanced-Timer und verwendete Hardware. Die Einstellungen werden über Präprozessoranweisungen getätigt, nach einer Änderung muss das Projekt vollständig kompiliert werden. Ansonsten werden die Änderungen nicht übernommen.

In den folgenden Abbildungen, sind Simulationsergebnisse und Messungen mit dem Testaufbau dargestellt. Die Abbildungen 4.1, 4.2 und 4.3 sind Simulationsergebnisse.

Die Abbildungen 4.4, 4.5 und 4.6 Messungen mit dem Testaufbau.

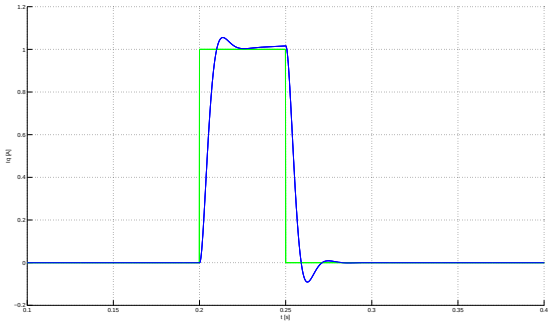


Abbildung 4.1.: IQ-Regler simuliert in Matlab

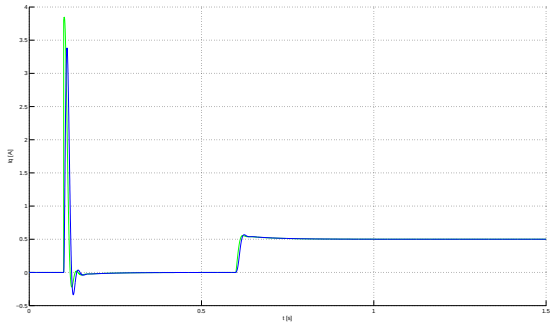


Abbildung 4.2.: IQ-Regler simuliert mit Drehzahlregler

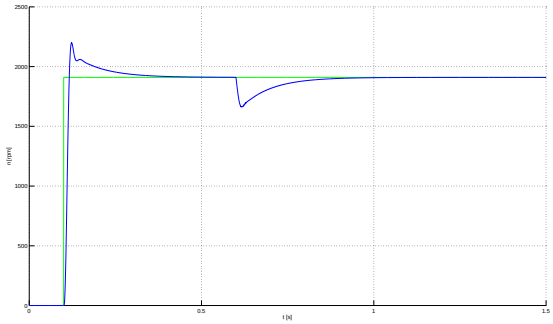


Abbildung 4.3.: Drehzahlregler simuliert

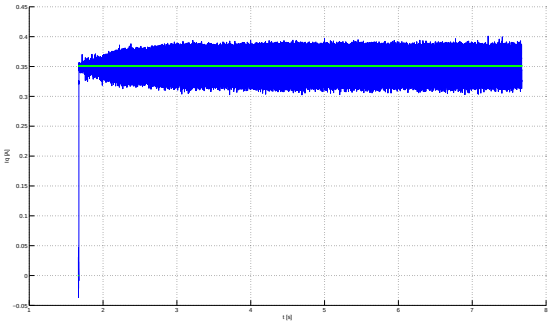


Abbildung 4.4.: IQ-Regler, Messung vom Testaufbau

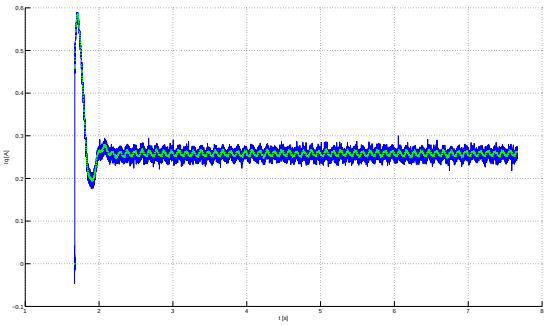


Abbildung 4.5.: IQ-Regler mit Drehzahlvorgabe, Messung vom Testaufbau

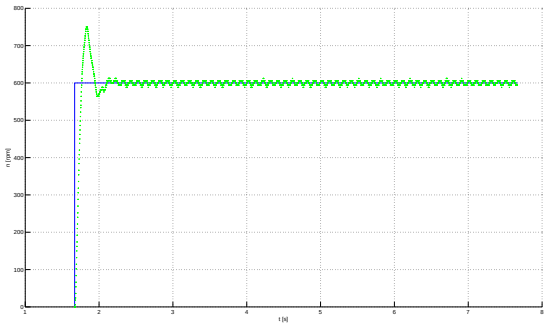


Abbildung 4.6.: Drehzahlregler, Messung vom Testaufbau

4.3.1. Modell

In Abbildung 4.7 ist ein Modell der verwendeten Regelung dargestellt.

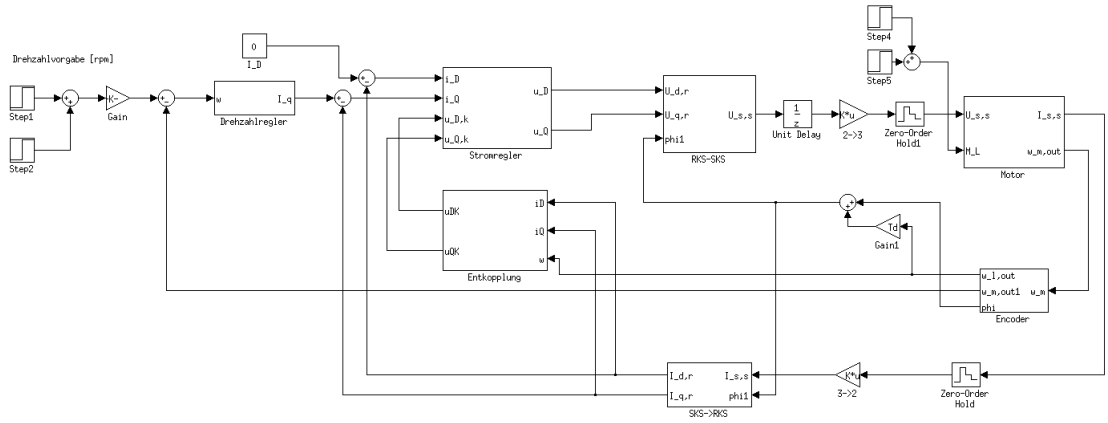


Abbildung 4.7.: Simulink Modell der Regelung

In der Auflistung wird erläutert welche Programmmodule für welchen Abschnitt der Regelung zuständig sind.

- Der Drehzahlregler wird von einem Timer-Interrupt laufend aufgerufen und stellt den Sollwert für den Stromregler bereit.
- Stromregler: wird, nachdem der Messwert vom ADC bereitsteht, aufgerufen (Synchron mit der PWM-Frequenz), verwendet das SVPWM Modul, um die Stellgröße auszugeben.
- Transformationen: die Transformationen sind einfache Funktionen.
- Encoder: wird vom Modul Feedback ausgewertet, stellt den aktuellen Rotorwinkel (elektrisch und mechanisch) über Funktionen zur Verfügung

4.3.2. Wertedarstellung

Beschreibung wie im Programm die unterschiedlichen Größen (Stellgröße, ...) dargestellt werden. Da die FPU nicht verwendet werden kann, wäre es nicht effizient für die Berechnungen Gleitkommadatentypen zu verwenden. Ist keine FPU vorhanden, werden die Gleitkomma-Rechenoperationen in c-lib–Funktionen implementiert, welche eine ungleich längere Ausführungszeit haben. Um das zu umgehen, kann man die Festkommadarstellung verwenden.

Festkommazahl

Bei einer Zahl in dieser Darstellung ist die Position des Kommas fest vorgegeben. Das heißt die Anzahl an Vor- und Nachkommastellen ist im Vorhinein bekannt und kann nicht mehr verändert werden. Diese Darstellung hat einen enormen Geschwindigkeitsvorteil.

Die Zahlen können in Integer-Datentypen gespeichert werden und für die standard Rechenoperationen (plus, minus, multiplizieren, dividieren) können die Operationen für ganze Zahlen verwendet werden.

Im Prinzip bedeutet das, dass die zu speichernde Kommazahl mit einer Zahl multipliziert wird, die ein Vielfaches der Basis des verwendeten Zahlensystems darstellt, um das Komma zu verschieben. Bei Mikrocontrollern ist es vorteilhaft, Festkommazahlen der Basis zwei zu nehmen. Das vereinfacht die Operationen, da eine Multiplikation einem bit shift nach links und eine Division einem bit-shift nach rechts entspricht. Der Kommapunkt wird nicht gespeichert, sondern die Position wird als fix angenommen, das Programm bzw. der Programmierer muss die Zahlen korrekt behandeln.

Eine übliche Bezeichnung für den verwendeten Fixkommatyp ist "**Q**`i.f`". Dabei gibt `i` die Anzahl der Vorkommastellen und `f` die Anzahl der Nachkommastellen an. In mancher Literatur findet man auch die Bezeichnung "**Q**`f`", hier wird mit `f` nur die Anzahl der Nachkommastellen angegeben.

Die Umrechnung zwischen Gleit- und Festkommazahlen erfolgt folgendermaßen:

$$\begin{aligned} \text{fixed-point} &= \text{float-point} \cdot 2^f \\ \text{float-point} &= \text{fixed-point} \cdot \frac{1}{2^f}. \end{aligned} \tag{4.1}$$

Im Gegensatz zu den Fließkommazahlen ist die Auflösung bei Festkommazahlen konstant (2^{-f}). Der Wertebereich beträgt:

$$-2^{i-1} \leq n \leq 2^{i-1} - 2^{-f}. \tag{4.2}$$

Rechenoperationen

Wie bei der normalen Verwendung von Integer-Datentypen muss man bei jeder Rechenoperation auf einen Überlauf achten. Addieren und Subtrahieren ist einfach möglich. Bei Multiplikationen und Divisionen muss das Ergebnis nachbearbeitet werden. Hier ist exemplarisch eine Multiplikation und Division angeführt:

$$\begin{aligned} (\text{float} \cdot 2^f) \times (\text{float} \cdot 2^f) &= (\text{float} \times \text{float}) \cdot (2^f \times 2^f) \\ (\text{float} \cdot 2^f) \div (\text{float} \cdot 2^f) &= \text{float} \div \text{float} \end{aligned}$$

Nach einer Multiplikation muss das Ergebnis durch 2^f dividiert werden. Bei einer Division ist es ähnlich, das Ergebnis muss mit 2^f multipliziert werden. Bei einer Multiplikation muss für das Ergebnis ein größerer Datentyp verwendet werden, das Ergebnis hat $2{*}i.2{*}f$ Stellen!! Beim Dividieren kann man den Zähler (Dividend) vorher mit dem Faktor 2^f multiplizieren, man erhöht so die Genauigkeit und spart sich das Nachbearbeiten des Ergebnisses.

Darstellung

Im Programm werden für die Reglergrößen Festkommavariablen des Typs $Q1.15$ verwendet. Das ergibt einen Wertebereich von $-1 \leq n \leq 0,999969$ und eine Auflösung von $30 \cdot 10^{-6}$.

Winkeldarstellung

Für die Darstellung von Winkeln werden auch Integer Zahlen verwendet. Der Wertebereich $-32768 \leq \alpha \leq 32767$ wird abgebildet auf: $-180° \leq \alpha \leq 180°$. Der Vorteil bei dieser Darstellung ist, dass, wenn zwei Winkel addiert werden, das Ergebnis trotz eines Überlaufes korrekt ist. Die Abbildung 4.8 zeigt den Zusammenhang zwischen Winkel und Integer Zahl. Auf der x-Achse ist der Winkel in Grad und auf der y-Achse die entsprechende ganze Zahl angegeben. Die beiden strichlierten Linien zeigen den Wertebereich der verwendeten Integer Zahl an.

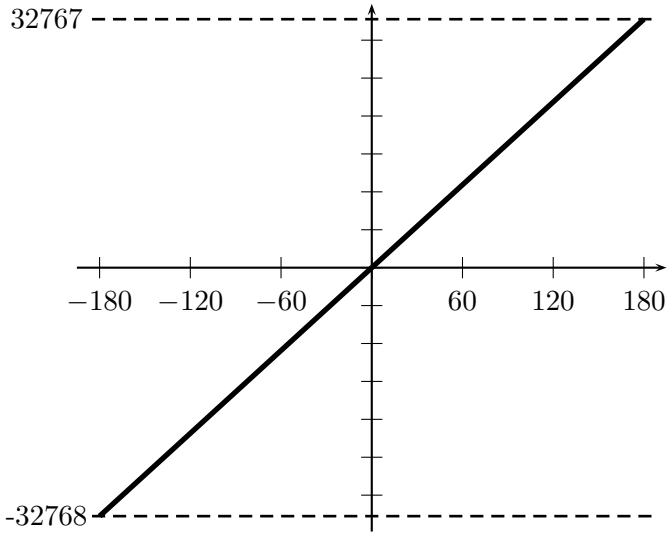


Abbildung 4.8.: Winkel in Festkommadarstellung $Q1.15$

4.3.3. Controller

Für die Regelung werden PI-Regler verwendet:

$$u_n = K_p \cdot e_n + K_i \sum_{i=0}^{n} e_i \tag{4.3}$$

Für den I-Anteil der Regler wurde eine einfache Anti-Windup Methode gewählt. Es wird einfach der I-Anteil begrenzt. Dies wird durch die gewählte Darstellung realisiert. Als Stellgröße werden normierte Werte verwendet. Für die Berechnung innerhalb der Reglerfunktion werden für die Zwischenergebnisse ausreichend große Datentypen verwendet (siehe Wertedarstellung 4.3.2).

Die Regelparameter für die beiden Stromregler: $k_p = 0.977$, $k_i = 1$. Für den Drehzahlregler lauten die Regelparameter: $k_p = 62.5$, $k_i = 2.7$.

Regular Sampling

Der Stromregler wird synchron zur PWM-Frequenz aufgerufen [11, Zusatz]. Ein häufigeres Aufrufen des Reglers ergibt keinen Sinn, da die Stellgrößen keine Auswirkung auf den Duty-Cycle der PWM haben. Außerdem ist der Abtastzeitpunkt für eine Strommessung in der Mitte der PWM, das heißt pro PWM-Periode steht ein Stromwert zur Verfügung. Im Programm ist es überdies leichter dieses Verhalten zu implementieren.

4.3.4. Feedback

Für eine feldorientierte Regelung benötigt man die Orientierung des Rotorflusses. Bei einer permanent erregten Synchronmaschine ist das einfach möglich, es muss nur die Rotorlage gemessen werden. Im Programm ist dafür das Modul `feedback` zuständig.

Im Programm wird der Sensorwert durch eine Variable repräsentiert, die einen vordefinierten Wertebereich haben muss. In diesem Wertebereich muss eine gesamte Umdrehung des Rotors abgebildet werden. Die Implementierung für den Sensor muss diesen Wert laufend aktualisieren, zumindest bevor der Stromregler aufgerufen wird.

Im Programm werden drei verschiedene Sensortypen unterstützt: simulierter Winkel, inkrementaler Sensor und absoluter Sensor.

Simulierter Winkel Bei dem simulierten Winkel wird ein Timerbaustein als Zähler verwendet. Der aktuelle Zählerstand des Timers wird als Winkel interpretiert. Durch die Änderung des Taktes vom Timerbaustein kann die Geschwindigkeit der Winkeländerung eingestellt werden.

Inkrementalgeber Bei inkrementalen Sensoren (üblicherweise *incremental encoder* oder nur Encoder genannt) werden zwei um 90° phasenverschobene Signale generiert. Damit kann die Drehrichtung und die Drehzahl gemessen werden. Als Anbindung an den Mikrocontroller wird ein Timerbaustein verwendet. Diese Bausteine bieten ein Interface für derartige Sensoren. Je nach Phasenverschiebung zwischen den beiden Kanälen, wird der Zählwert des Timers hinauf- oder hinunter gezählt. Es wird jede Änderung der beiden Signale mitgezählt, das heißt die Strichanzahl muss mit 4 multipliziert werden. Dadurch hat man die Position des Rotors. Durch eine solche Messung ist aber der Anfangswert unbekannt. Der Rotorwinkel wird dadurch bestimmt, dass ein bekannter Stromraumzeiger an den Motor angelegt wird. Da der Motor sich frei drehen kann, ist das problemlos möglich. Der Inkrementalgeber wird nur in Verbindung mit dem Testboard verwendet.

Rotary Encoder Bei dem absoluten Sensor wird ein magnetischer Sensor von Austia Microsystems `AS5145` [7] verwendet. Dieser Sensor misst die Ausrichtung eines Magneten, der sich über oder unter der Chipfläche befindet. Dieser Chip hat ein SPI-ähnliches Interface zum Auslesen des 12 Bit Wertes. Die verwendeten Daten vom `AS5145` beziehen sich alle auf das Datenblatt in der Version 1.15 (siehe Herstellerseite).

[7] `www.austriamicrosystems.com/AS5145`

Der Chip überträgt insgesamt 18 Bit: den Winkel (12 Bit) und sechs Statusbit. Die Statusbits sind in Tabelle 4.2 aufgelistet. Der Wert wird mit einer SPI-Schnittstelle vom Mikrocontroller ausgelesen. Die SPI-Schnittstelle kann nur eine ganzzahlige Anzahl von Bytes lesen. Deshalb werden 24 Bit ausgelesen. Dieses Verfahren hat zu Beginn gut funktioniert. Eine andere Möglichkeit ist, das Protokoll mittels Bit-Banging nachzuempfinden. In diesem Fall wird das Protokoll ohne der Unterstützung einer Peripherie selbst implementiert. Diese Option wurde nicht in Betracht gezogen. Ein derartiges Verfahren wäre zu rechenintensiv gewesen. Es sollte ja während jeder Periode ein aktueller Winkelwert zur Verfügung stehen.

bit	Name	Beschreibung
0	Parität	gerade Parität über die 12 Datenbits und 5 Statusbits
1	mag dec	Feld wird zu klein, Magnet bewegt sich vom Sensor zu weit weg
2	mag inc	Feld wird zu groß, Magnet bewegt sich auf den Sensor zu
3	lin	Linearitätsfehler, Daten können wahrscheinlich fehlerhaft sein
4	COF	Abstandsfehler, ist das Bit gesetzt, sind die Daten ungültig
5	OCF	Logisch 1 zeigt an, dass der “offset compensation algoritm” abgearbeitet ist

Tabelle 4.2.: Statusbits des magnetischen Encoders ASS5145

Struktur im Programm

In der Hauptdatei des Moduls sind die Funktionen, welche von jeder Implementierung verwendet werden. Sie dienen zum Berechnen der Drehzahl und zur Umrechnung des Sensorwertes in einen elektrischen oder mechanischen Winkel.

- FEEDBACK_clearSpeedBuffer
- FEEDBACK_calcAverageSpeed
- FEEDBACK_getElectricalAngle
- FEEDBACK_getMechanicalAngle

Die folgenden Funktionen müssen für den Sensor implementiert werden:

- _init
- _resetValue
- _initValue
- _getValue
- _getOffset
- _setOffset

Alle genannten Funktionen sind für die zwei Sensoren implementiert. Die Funktion `FEEDBACK_calcAverageSpeed` ist verantwortlich für die Drehzahlberechnung (siehe Gleichung 4.4). Diese Funktion soll periodisch aufgerufen werden. In der Motorsteuerung

wird sie alle $500\mu s$ aufgerufen.

$$n = \Delta \cdot f_{Sample} \cdot \frac{1}{PPR} \tag{4.4}$$

- PPR "pulses per revolution", Auflösung des Sensors (z.B. Strichanzahl eines Encoders)
- Δ Winkeldifferenz
- f_{Sample} Frequenz mit der der Wert vom Sensor gelesen wird

Bei der Drehzahlmessung muss bei der Berechnung der Differenz auf den Übergang von 360° auf 0° geachtet werden. Hier kann die Differenz fehlerhaft sein.

Wird die Differenz der beiden Winkelwerte zu einem Zeitpunkt berechnet, zu welchem ein Sprung von 360° auf 0° oder umgekehrt stattgefunden hat, ist sie falsch. Es findet ein "Überlauf" statt. Durch entsprechendes Addieren oder Subtrahieren der PPR kann die Differenz berichtigt werden.

Eine einfache Möglichkeit diesen Fall zu bemerken ist zu testen, ob für die Differenz Δ gilt: $|\Delta| \leq k$. Ist Δ negativ, muss der Wert PPR dazu addiert werden, ist Δ positiv, muss PPR subtrahiert werden. Die Grenze k muss entsprechend der maximal auftretenden Drehzahl im System gewählt werden. Ist k zu klein, wird die Drehzahl falsch berechnet, da ein Erkennen des Überlaufes nicht mehr korrekt funktioniert.

Probleme mit dem Absolut Encoder

Beim UCC wird, wie bereits erwähnt, ein Sensor von AMS eingesetzt. Bei Testfahrten ist ein sich wiederholendes Klopfgeräusch aufgefallen, das anscheinend vom Motor oder Lager verursacht wird. Nach einigen Tests konnte ein mechanischer Fehler ausgeschlossen werden. Also war anzunehmen, dass der Fehler in der Ansteuerung der Motoren liegt.

Um mögliche Ursachen einzugrenzen, wurden verschiedene Versuche durchgeführt. Bei Tests des Radnabenmotors mit einer zugekauften Steuerung ist kein Fehler aufgefallen. Beim Versuch mit der eigenen Steuerung konnte bei einer sehr langsamen Drehbewegung (ca. 6rpm) der Fehler reproduziert werden. Er macht sich bemerkbar durch ein periodisches Geräusch: es sind fünf Piepser schnell hintereinander, diese wiederholen sich ungefähr alle fünf Sekunden.

Im Auto wird durch das Carbon-Monocoque das Geräusch anscheinend verstärkt und verzerrt, sodass es sich wie ein mechanischer Fehler anhört. Der Fehler tritt auch im Stillstand des Motors auf. Sobald der Stromregler im Programm aktiv ist, tritt der Fehler auf. Die weiteren Tests wurden im Stillstand durchgeführt, um das Messen zu erleichtern. Am Oszilloskop wurde der Motorstrom dargestellt, dort konnte man ungewöhnliche Spitzen im Stromsignal erkennen. In einem Testprogramm wurde auf die Spitzen im Stromsignal getriggert und die Werte aufgezeichnet. In der Abbildung 4.10 sieht man den Phasenstrom aller drei Phasen. Im Diagramm sind im Abstand von $50ms$ fünf ungewöhnliche Spitzen zu erkennen. Die anscheinend das Geräusch verursachen. Diese Gruppe aus fünf fehlerhaften Werten, wiederholt sich zirka alle fünf Sekunden.

In der Abbildungen 4.9 sind die berechneten Duty-Cycles dargestellt. Hier kann wieder das selbe Fehlverhalten beobachtet werden. Die ebenfalls fehlerhaften Duty-Cycles sind

ein Hinweis, dass die Berechnung bzw. der Regelalgorithmus nicht korrekt ist. Weiters ist das periodische Auftreten und der exakten Abstand der falschen Werte ein Hinweis auf die falsche Berechnung.

Mögliche Ursachen sind dabei ein unglückliches Zusammenspiel der Interrupt-Serviceroutinen, die eine Berechnung zu weit verzögern um den Wert rechtzeitig zu aktualisieren, oder ein Indexüberlauf in einem Array, das die betreffenden Register überschreibt.

Alle Berechnungen wurden auf ihre Korrektheit überprüft. Die Ursache war der verwendete Rotary-Encoder, dieser lieferte falsche Winkelwerte. In den Abbildungen 4.11 und 4.12 sind die Rohdaten dargestellt. Die Werte von 1 - 5 sind zu dem betreffenden Zeitpunkt der Zustand, der einzelnen Status-Bits. Ist in der Abbildung ein Punkt eingezeichnet, ist das jeweilige Statusbit eins und zeigt einen Fehler an. Die Bedeutung der Bits ist in Tabelle 4.2 aufgelistet.

Die Behebung im Programm war ein einfacher Workarround. Da sich bei den Messungen gezeigt hat, dass bei jedem falschen Wert ein Fehlerbit gesetzt ist, wird in so einem Fall dieser Wert ignoriert und linear extrapoliert. Für die lineare Extrapolation werden die beiden vorangegangenen Werte gespeichert.

Die Ursache für das Fehlverhalten des Chips konnte nicht gefunden werden. Eine der folgenden zwei Eigenschaften vom Chip kann vielleicht den Fehler verursachen:

- Der IC liest seine Hall-Sensoren mit einer Frequenz von $f_s = 10,52kHz$ aus. Der Stromregler holt sich über das digitale Interface die Daten mit einer Rate von $f_{ra} = 10kHz$. Durch den knappen Unterschied der beiden Frequenzen kann vielleicht ein falscher Wert ausgelesen werden.
- Beim Auslesen des absoluten Wertes werden zu viele Taktsignale angelegt: 24 anstatt der benötigten 18. Das liegt daran, dass die SPI-Peripherie vom Mikrocontroller nur ganze Bytes auslesen kann.

  Der IC hat einen Daisy-Chain Modus. In dieser Betriebsart können mehrere Sensoren hintereinander geschaltet werden. Die Daten der einzelnen Sensoren werden seriell ausgelesen. Dazu muss an jedem Sensor derselbe Takt anliegen. Die Datenleitungen werden von Chip zu Chip verbunden.

  Werden nun mehr Takte generiert als nötig, befindet sich der Chip im Daisy-Chain Modul und will Daten von einem nicht angeschlossen Chip weitersenden.

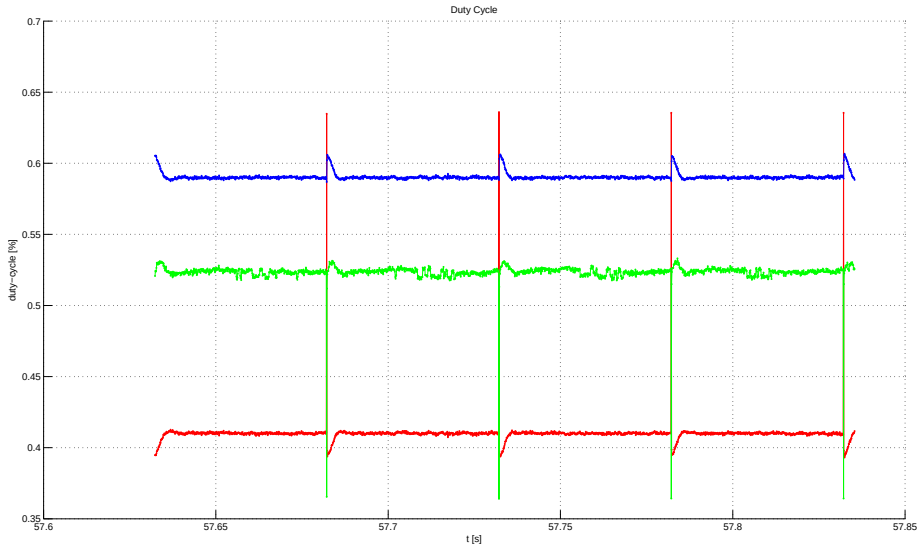


Abbildung 4.9.: Duty cycle der Halbbrücken beim Auftreten des Fehlers

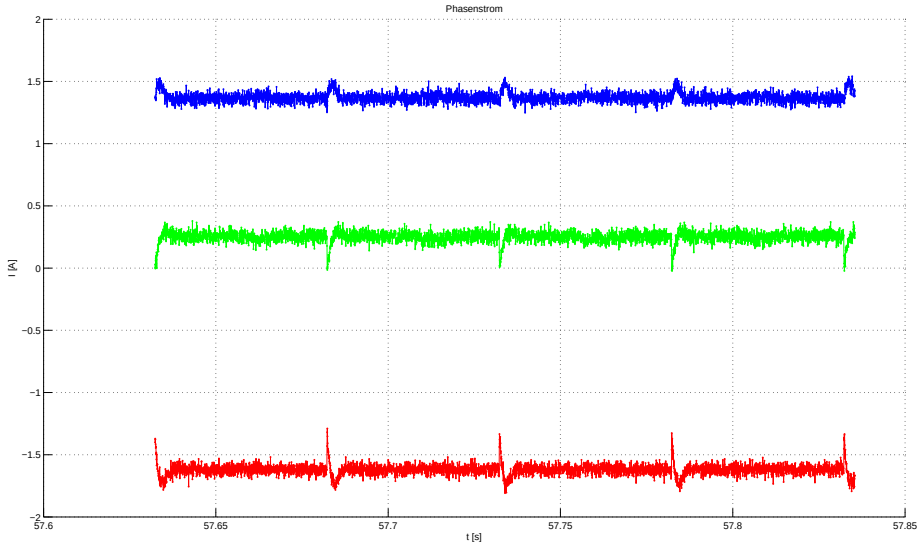


Abbildung 4.10.: Phasenströme beim Auftreten des Fehlers

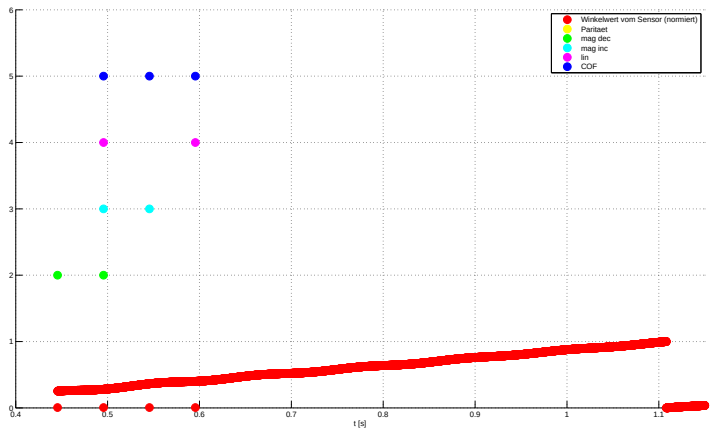


Abbildung 4.11.: Rohdaten des rotary-encoders, Motor dreht sich

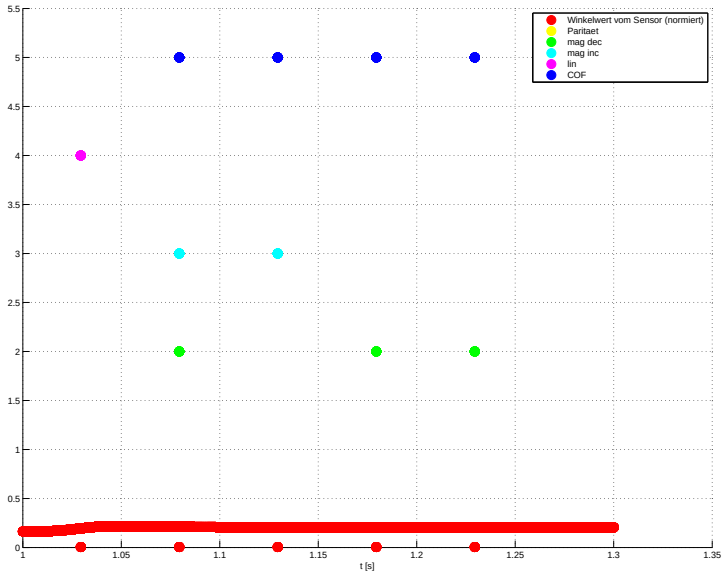


Abbildung 4.12.: Rohdaten des rotary-encoders, Motor stillstand

4.3.5. SVPWM

Dieses Modul ist die Schnittstelle zum Motor. Es realisiert einerseits die Raumzeigermodulation (siehe Abschnitt 3.2), steuert die Halbbrücken und ist verantwortlich für die Strommessung. Diese Aufgaben sind im selben Modul weil sie Programmtechnisch sehr stark voneinander abhängig sind. Die wichtigste Funktion für die Raumzeigermodulation ist die Funktion: `SVPWM_calcDutyCycles(`U_s^s`)`. Diese Rechnet aus der gewünschten Statorspannung die CCR Werte für den Timer an welchem die Halbbrücken angeschlossen sind. Der genaue Ablauf der Strommessung ist weiter unten beschrieben. Es werden zwei unterschiedliche Strommessungen unterstützt.

Ablauf

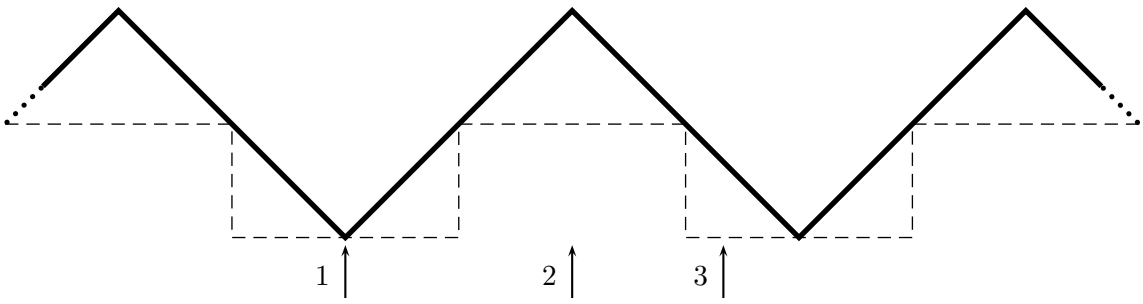


Abbildung 4.13.: schematischer Programmablauf Stromregler

Die dicke Linie stellt den Zählerwert des Timers dar. Die strichlierte Linie stellt die PWM dar.

1. zu Beginn jeder Periode wird der ADC-Trigger eingestellt
2. ADC Messung ist beendet, Stromregler wird aufgerufen
3. neue Duty-Cycles werden berechnet und vom Timer übernommen

Strommessung

Für die Strommessung werden die drei ADC-Bausteine vom Mikrocontroller verwendet. Vor dem Starten des Motors müssen die Offsetspannungen gemessen werden. Pro PWM-Periode wird ein Wert abgetastet. Durch die induktive Last (Motor) entspricht der abgetastete Stromwert dem zeitlichen Mittelwert (siehe 4.3.3).

Bei der Berechnung der Duty-Cycle wird keine Rücksicht auf die Strommessung genommen. Es gibt keine Begrenzung auf minimale oder maximale Duty-Ccles. Im Betrieb hat das keine Auswirkungen, da der Regler nie in die Stellgrößenbeschränkung gekommen ist.

three-shunt Hier wird der Strom über Shunts an den Halbbrücken gegen Masse gemessen. Dieses Verfahren wird vom STM-Motor-Board verwendet (siehe Abschnitt 6.3). Der Abtastzeitpunkt für die Strommessung muss anhand der eingestellten Duty-Cycles zu Beginn jeder Periode berechnet werden. Dadurch soll sichergestellt werden, dass der low-side MOSFET eingeschaltet ist und Strom durch den MOSFET fließt. Als Trigger dient der vierte Kanal vom Advanced-Timer.

Phasenstrom Hier wird direkt der Phasenstrom gemessen. Zur Messung werden Hall Sensoren von Allegro MicroSystems [8] verwendet. Der Strom wird in der Mitte abgetastet. Als Trigger wird ein eigener Timer verwendet der mit dem Advanced-Timer synchronisiert wird. Da alle drei Phasenströme gemessen werden kann der Nullstrom ausgerechnet werden. Dieser wird von den Phasenströmen abgezogen um den Gleichanteil zu beseitigen.

4.3.6. Rotorposition bestimmen

Ein beim Startvorgang für permanenterreger Synchronmaschinen sehr wichtiger Parameter ist der aktuelle Rotorwinkel. Eine einfache Möglichkeit ist es, einen absoluten Positionsgeber zu verwenden. Wird ein inkremental Encoder verwendet hat man aber keine Information über den aktuellen Winkel. Kann sich der Rotor frei bewegen und ist eine Bewegung des Motors ungefährlich schaltet man einfach einen Stromzeiger auf. Dieser richtet den Rotor aus und man weiß den aktuellen Winkel. Will man aber, dass sich der Rotor nicht bewegt, weil es äußere Umstände nicht zulassen benötigt man ein anderes Verfahren.

Die Bestimmung des Rotorwinkels mit einem inkrementalen Encoder, bei möglichst geringer Bewegung des Rotors ist für die Autos die im TERA entwickelt werden sehr wichtig. Es sollte der Chip von AMS verwendet werden, weil er einfach zu montieren ist und es schon Erfahrungswerte zu ihm gibt. Die serielle Kommunikation ist wie oben Beschrieben fehlerhaft (siehe 4.3.4). Der Sensor hat ein inkrementales Interface, wird dieses verwendet kann man nicht mehr auf das absolute zurückschalten. Außerdem können inkrementale Sensoren einfach mit einem Timer-Baustein im Mikrocontroller verwendet werden.

Als Anhaltspunkt für die Implementierung dient das Paper [6]. Bei diesem Verfahren wird der Strom über eine Rampe erhöht. Die Stromregelung erfolgt in einem willkürlichen Koordinatensystem (WKS). Die Lage dieses willkürlichen Koordinatensystems gegenüber einem ständerfesten Koordinatensysem, ist die Ausgangsgröße eines PI-Reglers. Dieser PI-Regler wird als Lageregler verwendet. Er soll immer auf Lage 0 ausregeln. Durch das Erhöhen des Stromes bewegt sich der Rotor. Der Regler versucht dem entgegenzuwirken und den Winkel des Stromzeigers zu verändern sodass sich der Rotor nicht mehr bewegt. Das Ergebnis ist wie im ersten Absatz erwähnt, der ausgerichtete Zustand im unbelasteten Fall.

[8] http://www.allegromicro.com/

In Matlab/Simulink wurde dazu ein Modell erstellt (siehe Abbildung 4.14). Als Regler wird wie bereits erwähnt ein PI-Regler verwendet. Die Koeffizienten für den PI-Regler sind allerdings durch empirische Versuche eingestellt. Der Fehlerwinkel des Rotors beträgt 10°. In Abbildung 4.15 ist die Bewegung des Rotors dargestellt. In Abbildung 4.16 die Stellgröße des Reglers.

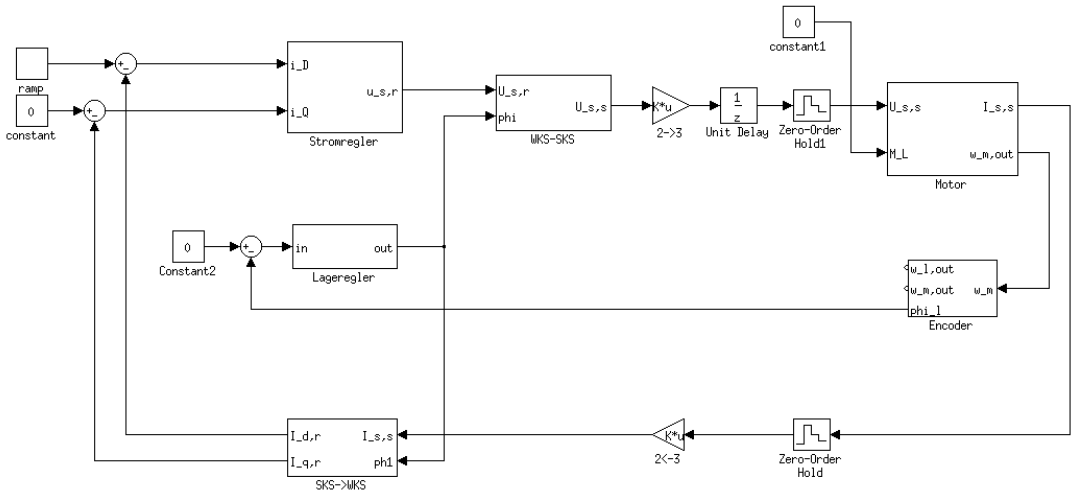


Abbildung 4.14.: Simulink Modell für die Rotorpositionsbestimmung

Im entwickelten Programm wurde das oben geschilderte Prinzip implementiert. Die verwendete Hardware war der Testaufbau von STMicroelectronics (siehe 6.4).

Das erwartete Verhalten vom Programm war nicht wie in der Simulation. Durch das sehr geringe Trägheitsmoment des Motors ist der Rotor dem Winkel des Stromzeigers zu schnell gefolgt. Ein Regeln des Winkels war nicht möglich/sinnlos. Hatte der Strom ca. die Hälfte des Nennstroms erreicht, hat der Rotor den vom Regler vorgegebenen Winkel übernommen. Als Abhilfe wurde der Endstrom der Rampe erniedrigt und die Zeit auf 6 Sekunden erhöht. Das ergibt eine Steigung von $116mA/s$. Der maximale Strom beträgt nur 15% des Nennstroms vom Motor. Mit diesen Werten hat das Ermitteln des Rotorwinkels besser funktioniert. In der Abbildung 4.17 ist die Bewegung des Rotors in Encoderschritten dargestellt. Ein Encoderschritt entspricht: $0,225°$.

Zur Zeit ist der Code noch nicht praxistauglich. Der Lageregler funktioniert nur, wenn kein Lastmoment vorhanden ist.

4.4. Module

4.4.1. Userinterface

Für die Steuerung des Programms wurde über die serielle Schnittstelle ein VT100 kompatibles Terminal implementiert. Der VT100 Befehlssatz ist nötig um Werte oder Texte an eine bestimmte Position im Terminal Fenster zu schreiben. Ansonsten würden sich

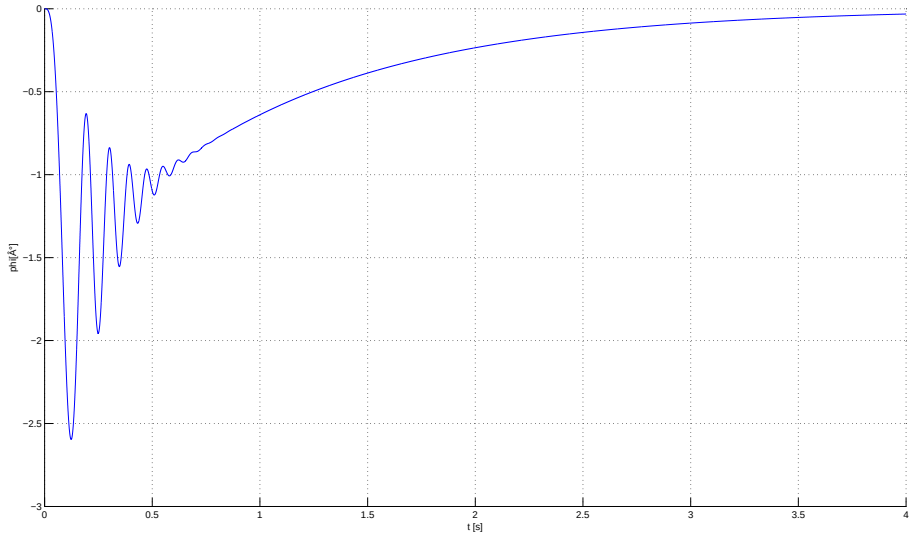


Abbildung 4.15.: Bewegung des Rotors in Grad

bei einer seriellen Kommunikation die Werte oder Texte laufend überschreiben. In der Abbildung 4.18 ist das User Interface zu sehen.

Mit den Bild hoch/runter Tasten ist eine einfache Menüführung realisiert. Über die Funktionstasten können vorher definierte Abläufe gestartet werden. Im mittleren Bereich werden wichtige Variablen ausgegeben, dieser Bereich wird alle $500ms$ aktualisiert.

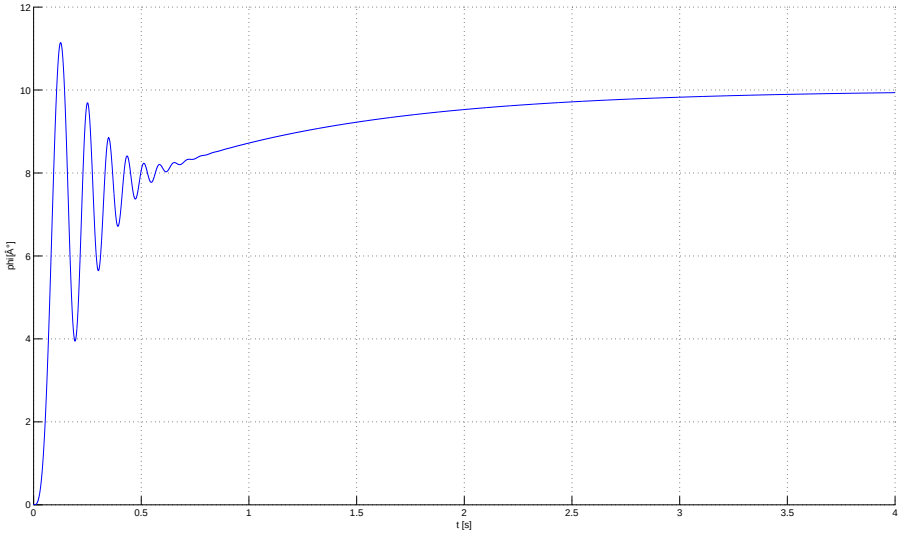


Abbildung 4.16.: Reglerausgang in Grad

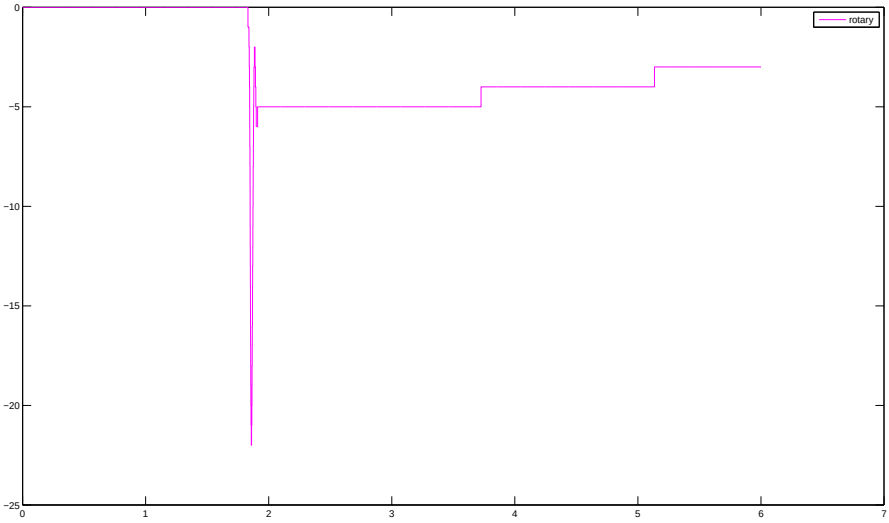


Abbildung 4.17.: Rotorbewegung in Encoderschritten

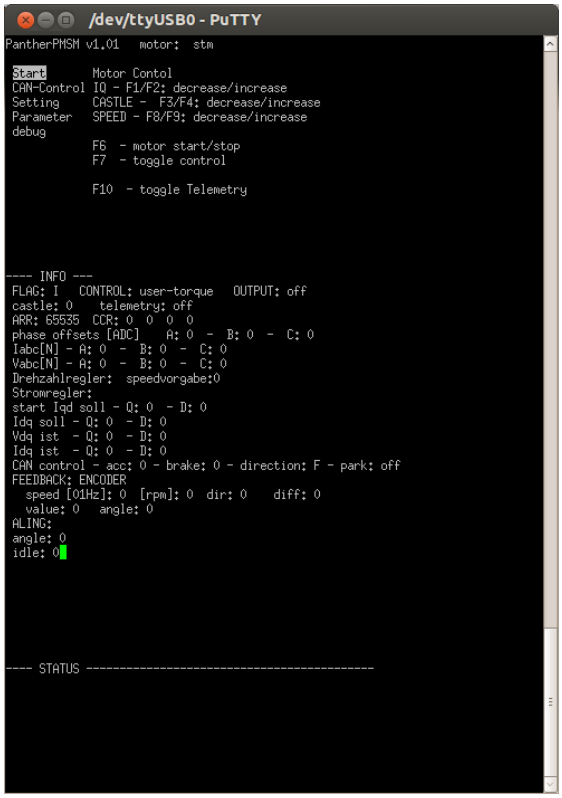


Abbildung 4.18.: serielles VT100 Terminal

4.4.2. Fahrverhalten

Wie sich die Motorplatine im Auto auf die Vorgaben, durch die beiden Pedale (Brems- und Gaspedal) und die beiden Knöpfe am Lenkrad (Park- und Rückwärtsknopf) verhält. Durch den Rückwärtsknopf kann die Fahrtrichtung bestimmt werden, die Fahrtrichtung wird nur gewechselt wenn die Geschwindigkeit des Autos null oder annähernd null ist. Der Parkknopf dient als eine sehr einfache Feststellbremse, wird er betätigt werden die drei Motorphasen gegen Masse kurzgeschlossen. Das soll ein unbeabsichtigtes Wegrollen erschweren. In der Praxis hat sich gezeigt, dass der Parkmodus nicht den gewünschten Effekt hat. Durch das langsame Anrollen wird zu wenig Spannung induziert. Der Code der die Vorgaben für den Regler tätigt ist in der Funktion `FOC_canCurrentLoop()` in der Datei `controller.c`.

Der Code muss sehr sicher gestaltet sein, da im fahrenden Auto ein plötzliches Bremsen oder unabsichtliches Rückwärts fahren bei hohen Geschwindigkeiten hohe Ströme verursacht was den Motor unnötig erwärmt. Der Bordcomputer ist verantwortlich für die Auswertung der beiden Knöpfe und des Gas- und Bremspedals. Er aktualisiert die Daten über den CAN-Bus. Für die beiden Pedale wird ein Wertebereich von 0-65535 verwendet. Der gesamte Pedalweg ist in diesem Wertebereich abgebildet. Da im Auto zwei Motorplatinen vorhanden sind, ist es nicht ratsam die einzelne Motorplatine entscheiden zu lassen ob sie jetzt z.B. die Fahrtrichtung ändern oder nicht. Hier sollte die Entscheidung eine Instanz höher getroffen werden. Der Bordcomputer überprüft die Drehzahlen beider Motorplatinen, wenn beide null sind wird eine Richtungsänderung oder der Parkmodus akzeptiert und weiter gesendet. Zusätzlich überprüft auch noch die Motorplatine ob die Drehzahl null ist. Weiters sollte auch noch sichergestellt werden ob der Bordcomputer noch vorhanden ist. Jede Platine sendet einen Heart-Beat, die Controller Funktion überprüft dadurch ob der Bordcomputer noch am CAN-Bus aktiv ist. Sendet der Bordcompputer keinen Heart-Beat mehr, werden die Halbbrücken ausgeschaltet.

Die möglichen Zustände der Motorsteuerung zusammengefasst:

- idle: die Halbbrücken sind ausgeschaltet, sobald der Gaswert unter einen bestimmten Wert sinkt
- park: die Halbbrücken sind gegen Masse kurzgeschlossen, wenn der Parkknopf betätigt ist
- Regler: der Regler gibt die Duty-Cycles für die Halbbrücken vor, sobald das Gaspedal betätigt wird

4.4.3. CAN-Protokoll

Zur Kommunikation im UCC zwischen den einzelnen Platinen wird ein CAN-Bus verwendet, der Standard CAN2.0B wird eingesetzt. Das Protokoll wurde selbst entwickelt. In der Praxis hat sich gezeigt, dass nur Variablen einer Anwendung (Platine) von einer oder mehreren Anderen benötigt werden. Darauf hin wurde das Protokoll optimiert. Es gibt zwei Arten von Variablen: normale-Variablen (maximal 32) und Status-Variablen, hier nur eine pro Platine. Der Zweck bei Status-Variablen soll sein, dass sie periodisch

gesendet werden. Die programmierte Empfangsroutine misst die Zeit zwischen zwei Statusnachrichten. Die Zeit dient als eine Art Heart-Beat der einzelnen Platinen und soll zeigen ob das Programm noch aktiv ist. Bindet man das CAN-Modul ein, ist automatisch Speicherbereich für eine Statusvariable und 32 normale Variablen reserviert.

Als Nachricht kann nur eine 32bit Variable gesendet werden. Die Variable ist gekennzeichnet mit einer Board-ID und einer Variablen-ID. Eine Daten-Nachricht kann mit der Funktion `CAN_sendDataMessage` gesendet werden, zum Senden einer Statusnachricht die Funktion `CAN_sendStatusMessage` verwenden. Alle Platinen am CAN-Bus empfangen die Nachricht und schreiben die darin enthaltene Variable in den Speicher. Die Empfangsfunktion greift einfach auf diesen Speicher zu um den Inhalt der Nachricht zu lesen.

Eine CAN-Bus Datennachricht ist folgendermaßen strukturiert (siehe Tabelle 4.4.3).

ID (29bit)	DLC (8bit)	Daten <DLC> Byte

Tabelle 4.3.: Struktur einer CAN-Datennachricht

In den Tabellen 4.4.3 und 4.4.3 ist die Struktur der Datennachricht abgebildet. Die Nachricht hat immer eine Länge von 6 Bytes.

CAN-ID (29bit)			
3bit reserviert	6bit Priorität	4bit Typ	16bit Sender

Tabelle 4.4.: Struktur einer der CAN-ID

CAN-Daten 6 Byte		
1 Byte	1 Byte	4 Byte's
Board-ID	Variablen-ID	4 Byte Variable

Tabelle 4.5.: Struktur CAN Daten

4.4.4. syscall-stubs

Die C Standard Bibliothek verwendet sogenannte syscall-stubs um ihren Funktionen den Zugriff auf die Hardware/Betriebssystem zu ermöglichen.

- `int _kill(int pid, int sig);`
- `void _exit(int status);`
- `int _getpid();`
- `caddr_t _sbrk(int incr);`
- `int _close(int file);`
- `int _fstat(int file, struct stat *st);`
- `int _isatty(int file);`
- `int _lseek(int file, int ptr, int dir);`

- `int _read(int file, char *ptr, int len);`
- `int _write(int file, char *ptr, int len);`

Es wird nicht der vollständige Umfang benötigt, es sind nur drei Funktionen ausprogrammiert. Die Funktion `_sbrk` für dynamische Speicherverwaltung (malloc, free). Für printf und scanf, C-Standard Funktionen zur formatierten Ein- und Ausgabe, sind noch die Funktionen `_read` und `_write` implementiert.

4.4.5. EEPROM-emulation

Um die Einstellungen des Programms dauerhaft speichern zu können wird ein nicht flüchtiger Speicher benötigt. Ein derartiger Speicher ist am einfachsten mit einem EEPROM realisierbar. Viele Mikrocontroller besitzen einen internen EEPROM was den Zugriff stark vereinfacht, außerdem spart man sich ein Bauteil auf der Platine. Ansonsten kann man externe EEPROMs verwenden, die über eine entsprechende Peripherie des Mikrocontrollers angebunden werden, normalerweise wird dafür der I^2C Bus verwendet.

Das interne Flash kann dafür verwendet werden einen EEPROM zu simulieren. Die Implementierung hat sich an das Application Note [1] von STMicroelectronics angelehnt. Ein Unterschied zu der Implementierung aus dem Application Note ist, die Speicheradressen der Variablen müssen nicht im Vorhinein feststehen. Ein praktischer Vorteil. Das bedeutet aber einen größeren Aufwand bei der Speicherung der einzelnen Variablen.

Bei der Implementierung müssen ein paar Eigenheiten des Flash berücksichtigt werden. Es können nur Daten auf einen gelöschten Flash-Sektor geschrieben werden. D.h. man kann einen bereits beschriebenen Bereich ohne vorheriges Löschen nicht wieder beschreiben. Änderungen in Variablen müssen daher nacheinander in den Flash geschrieben werden. Es müssen immer Dateninhalt und zugehörige Adresse gespeichert werden, das heißt die Variable mit der höchsten Speicheradresse im Flash ist die zuletzt geänderte und aktuellste. Das bedeutet das der reservierte Sektor irgendwann voll ist und keine Änderungen mehr möglich sind.

Das bedeutet es müssen mindestens zwei Sektoren verwendet werden. Ist ein Sektor voll, müssen die gültigen Variablen in einen neuen (zuvor gelöschten) Sektor transferiert werden. Es ist sinnvoll, dass die virtuelle Größe des EEPROMs kleiner ist als der Flash-Sektoren. In der Implementierung werden drei Sektoren verwendet, um eine gleichmäßigere Benutzung zu gewährleisten. Um den Zugriff auf das EEPROM zu reduzieren, sollten beim Starten die gespeicherten Variablen in den SRAM kopiert werden.

Da der erste 16kb Sektor für den Startup-Code reserviert ist, werden die nachfolgenden drei Sektoren für die Speicherung verwendet. In Tabelle 5.4 ist die Aufteilung des internen Flash dargestellt.

4.4.6. RAM-Testprogramm

Um sicherzustellen, dass das Remapping und die Ausführung aus dem SRAM funktioniert, wurde ein eigenes Testprogramm ”ramspot_test“ für das Evaluation-Board erstellt. Das Programm verändert den Interruptvektor eines externen Interrupt. Der

externe Interrupt wird über einen Taster aufgerufen. Dazu wird im SRAM an der entsprechenden Stelle eine neue Funktionsadresse geschrieben. Wenn das remapping funktioniert, wird nach drücken des Wakeup-Tasters eine andere Funktion ausgeführt. Das Programm hat wie erwartet funktioniert.

5. Toolchain

Als Toolchain wird eine Kombination aus Programmen genannt, die aus dem Source-Code ein Binär-Code erstellt. Die nötigen Einstellungen werden hier beschrieben. Die Anleitung ist auf das verwendete Programmiergerät [1] und der Mikrocontroller-Serie (`STM32F4xx`) zugeschnitten. Es ist aber möglich, die Toolchain für andere Programmiergeräte oder Cortex-M4 Mikrocontroller zu adaptieren.

5.1. Übersicht

Die Toolchain besteht aus folgenden Komponenten:

- Programmierumgebung: Eclipse IDE for C/C++ Developers
- Compiler: GCC [2] (gnu compiler collection), dieser muss für die ARM-Architektur kompiliert werden und das ARM-EABI verwenden. EABI: embedded-application binary interface. EABI ist eine Spezifikation in der Dateiformat, Registerbenutzung, Stack verwendung, Funktionszeiger und Parameterübergabe festgelegt sind. Das erlaubt auch das Linken von object-files, welche von anderen Compilern erstellt wurden.
- newlib [3]: eine C Standard Bibliothek, die für den Einsatz in embedded systems optimiert ist
- C/C++ GDB Hardware Debugging (Eclipse plugin)
- GDB-Server: ist im zuvor erwähnten GCC beinhaltet, dieser wird für das Debuggen der Software im Mikrocontroller benötigt. Der GDB-Server wird auch zum Flashen [4] verwendet.
- Schnittstelle/Treiber für das Programmiergerät zur Anbindung an den GDB-Server Das ist nötig, da der GDB-Server Zugriff auf dem Mikrocontroller benötigt. Hier existieren zwei Möglichkeiten für die Anbindung. Die populärste ist die open-source Software "openOCD". Für das im TERA verwendete Programmiergerät wird von der Herstellerfirma direkt ein GDB-Server angeboten.
  - openOCD: es unterstützt sehr viele Programmiergeräte, darunter auch Eigenbau-Geräte. Eine Liste mit unterstützen Geräten befindet sich auf der Homepa-

[1] Segger J-Link, http://www.segger.com/jlink.html
[2] http://gcc.gnu.org/
[3] http://sourceware.org/newlib/
[4] umgangssprachliche Bezeichnung für das Schreiben des kompilierten Source-Code (in Form einer Binär-Datei) in den Flash des Mikrocontroller

ge [5]. Unter Linux lässt es sich einfach installieren. Will man es unter Windows verwenden wird CygWin benötigt. Die praktische Handhabung ist etwas umständlich. Beim Debuggen muss ein OpenOCD im Server-Modus im Hintergrund laufen. Will man sein Programm flashen, muss das laufende openOCD beendet werden. Über ein separates Script wird wieder ein openOCD gestartet, welches sich nicht im Server-Modus befinden darf. Dieses openOCD flasht den erstellten Binär-Code.

  - JLinkGDB-Server: kann nur mit Programmiergeräten der Firma Segger verwendet werden, es existiert eine Linux und eine Windows-Version. Dieser muss vorm flashen/debuggen im Hintergrund gestartet werden.

- Mikrocontroller flashen: Möglichkeit den Flash zu beschreiben, wie bereits erwähnt über GDB und Treiber vom Programmiergerät möglich.

JTAG JTAG ist die Schnittstelle des Mikrocontroller's über die er einfach programmiert (geflasht) werden kann. Jedes JTAG fähige Programmiergerät kann dazu verwendet werden. In Tabelle 5.1 ist die weit verbreitete Belegung eines JTAG-Steckers angegeben, ein Standard konnte nicht gefunden werden. Normalerweise wird ein 20 poliger Stecker verwendet (zwei-reihig, 2,54mm Rastermaß).

Pin	JTAG-Funktion	STM32F Pin
Pins 1, 2	—	μC VCC, 3V3
Pins 4, 6, 8, 10, 12, 14, 16, 18, 20	—	GND
Pin 3	JNTRST	PB4, 10k pull-up
Pin 5	JTDI	PA15, 10k pull-up
Pin 7	JTMS	PA13, 10k pull-up
Pin 9	JTCK	PA14, 10k pull-down
Pin 11	RTCK	
Pin 13	JTDO	PB3, 10k pull-up
Pin 15	RESET	NRST
Pin 17	—	n.c.
Pin 19	—	n.c.

Tabelle 5.1.: übliche JTAG Steckerbelegung

Beim neuen Prozessorboard wir wegen der Größe ein eigener Stecker verwendet (8polige Stiftleiste, Rastermaß 2,54mm), Belegung siehe Tabelle 6.1.2, bei der ersten Prozessorboard Version war ein Standard JTAG-Stecker verbaut.

Bei einem `STM32F4` Mikrocontroller werden für die JTAG Signale keine pull-up/down Widerstände benötigt. Der Chip hat interne pull-up/down Widerstände, welche standardmäßig aktiviert sind.

[5] http://openocd.sourceforge.net/supported-jtag-interfaces

5.2. Installation

Es ist nur die Installation unter Linux beschrieben. Von allen benötigten Paketen existiert auch eine Windows Version und die Installationsschritte sind in Windows sehr ähnlich.

- Eclipse CDT
  - "Eclipse IDE for C/C++ Developers" herunterladen, [6] Version: Indigo oder höher
  - das Eclipse Paket muss nur entpackt werden und ist sofort startbar
  - für das Debuggen auf Mikrocontroller benötigt man noch das Eclipse-Plugin: "C/C++ GDB Hardware Debugging", das wie folgt beschrieben, installiert werden kann
    - im Eclipse Menüpunkt "Help ⇒ Install New Software..." auswählen
    - beim Feld: "Work with:" "– all available sites –" auswählen
    - in dem Textfeld "type filter text" den Begriff "gdb" eingeben und auf das Ergebnis warten, die Suche kann einige Zeit in Anspruch nehmen
    - anschließend das Plugin auswählen, zu finden als Unterpunkt von "Mobile and Device Development"
    - ausgewähltes Plugin installieren
- GCC
  - man kann den GCC entweder selbst kompilieren oder ein fertiges Paket verwenden
  - das Paket von Codesourcery beinhaltet alle benötigten Komponenten für die Entwicklung. Das Paket kann direkt von der Homepage [7] heruntergeladen werden.
  - für eine reibungslose Verwendung in Verbindung mit Eclipse und dem Makefile sollte man den bin-Pfad der GCC-Installation der System-PATH Variable hinzufügen
  - unter Linux am einfachsten möglich wenn man den entsprechenden Pfad der Datei "/etc/environment" hinzufügt
  - im Windows unter den Systemeigenschaften, Erweitert, Umgebungsvariablen
- newlib, das codesourcery Paket enthält bereits die "newlib", will man den GCC selbst kompilieren, muss natürlich auch die C-Library eigens kompiliert werden
- GDB-Server für das Programmiergerät installieren, hier existieren zwei Möglichkeiten: openOCD oder JLink-GDBServer
- openOCD
  - üblicherweise existiert für die gängigen Linux Distributionen ein vorgefertigtes

[6] http://www.eclipse.org/downloads/
[7] http://go.mentor.com/212s8

Paket, wird das Paket eigens kompiliert hat man den Vorteil, dass es genauer konfiguriert werden kann und man verwendet die neueste Version

- Paket herunterladbar von der Projekthomepage [8]
- vor dem compilieren sicherstellen, dass die beiden Pakete auf dem System installiert sind: `libusb` und `libusb-dev`
- openOCD wird mit folgenden Befehlen compiliert, hier wird nur das Programmiergerät J-Link aktiviert, bei Bedarf kann natürlich auch ein anderes oder alle unterstützten Programmiergeräte aktiviert werden.

```
./configure --enable-jlink
make all
sudo make install
# - oder
sudo checkinstall
```

Checkinstall erstellt ein Debian-Paket und installiert dieses, so ist es möglich das selbst kompilierte OpenOCD wieder üben den Paketmanager zu deinstallieren. Wer dies nicht will (oder kein Debian-basiertes Paketsystem hat), kann natürlich auch mit "sudo make install" installieren. Die Ubuntu Paketquellen enthalten auch ein openOCD-Paket, bei diesem sind alle möglichen Programmiergeräte aktiviert, aufgrund der Ubuntu-Releases ist das Paket meist veraltet. OpenOCD kann jetzt verwendet werden.

- Scripte und Befehle für den Betrieb von openOCD

Listing 5.1: Befehle zum Starten von openOCD

```
/usr/bin/xterm -e sudo openocd --file openocd.cfg -c
    init
/usr/bin/xterm -e sudo openocd --file openocd.cfg -c
    init -c "script flash.script"
```

Das "xterm -e" ist nötig um einfach ein Passwort eingeben zu können.

Listing 5.2: Beispiel für eine openOCD Konfiguration

```
source [find interface/jlink.cfg]
source [find chip/st/stm32/stm32.tcl]
source [find target/stm32.cfg]
```

Listing 5.3: flash.script

```
#hardware reset processor with halt
reset halt
#check target state
poll
#unprotect flash for writing
```

[8] http://openocd.sourceforge.net/

```
stm32x mass_erase 0
flash write_image <hexfile.hex> 0x00 ihex

reset run
shutdown
```

- JLink-GDBServer
  - Paket von der Homepage [9] herunterladen
  - in Windows einfach die Setup Datei ausführen und das Programm wird passend eingerichtet
  - unter Linux muss man die mitgelieferten Shared-Libraries selbstständig installieren, dazu einfach die Dateien in das Library-Verzeichnis kopieren und die Shared-Libraries neu laden
  - folgender Shell-code erledigt das

    ```
    sudo cp -d libjlinkarm.so.* /usr/lib/
    sudo ldconfig
    ```

  - nun einfach den GDB-Server mittels dem Befehl “JLinkGDBServer” starten
  - Es wurde die Version “4.36j“ verwendet. Jeder Versuch eine neuere Version zu verwenden ist fehlgeschlagen. Die Kommunikation mit dem Programmiergerät war anscheinend bei neueren Versionen fehlerhaft.
- die Verwendete Schnittstelle zum Porgrammiergerät muss vor dem Debuggen oder Flashen im Hintergrund laufen

Das openOCD wurde nur zu Beginn verwendet. Die Variante mit dem JLinkGDBServer war weitaus komfortabler.

5.3. praktisches Arbeiten mit der Toolchain

Hier werden noch einige Tipps/Hinweise zur Toolchain und nützliche Einstellungen zu benötigten Programmen erwähnt.

Der gesamte Source-Code wird über ein Eclipse-Projekt ”verwaltet“. Die wichtigsten Dateien abgesehen vom Source-Code sind: Linker-Script und MakeFile bzw. das dazugehörige Makefile.config. Für das Erstellen des Binaries wird das Build-Management System ”make“ verwendet, dieses führt alle dazu benötigten Kommandos in der richtigen Reihenfolge aus. Die nötige Konfigurationsdatei (”Makefile.config“) ist unter 5.3.3 beschrieben. Das Linker-Script (siehe 5.3.4) ist verantwortlich für das Zusammensetzten der Object-Files und verwendeten Library zu der fertigen Binär-Datei.

[9] http://www.segger.com/jlink-software.html

5.3.1. Eclipse

Die Ordnerstruktur des Eclipse-Projektes ist darauf ausgelegt um mehrere verschiedene Projekte zu verwalten. Der gesamte Source-Code eines Projektes muss in einem Ordner im project-Ordner sein. Das ist nötig um dem Makefile zu ermöglichen nur das gewünschte Projekt zu kompilieren.

Ordnerstruktur

Ein Überblick über die Ordnerstruktur des Eclipse-Projekts, die Ordner sind Kursiv dargestellt, wichtige Source-Dateien sind Fett dargestellt.

- *build*: hier werden die im Buildprozess erstellten Dateien abgelegt
- *firmware*: Bibliotheken von den Herstellern, eine genaue Übersicht und Beschreibung ist in Abschnitt 4.2 zu finden.
  - *CMSIS*: wird direkt von ARM zur Verfügung gestellt, ein Hardware-Abstraction-Layer für den Prozessor, bietet Funktionen für die prozessorinterne Peripherie
  - *lwip*: TCP/IP-Stack implementierung für Mikrocontroller ausgelegt (geringer Resourcenverbrauch)
  - *STM32F4xx_StdPeriph_Driver*: Interface für die Peripherie am Mikrocontroller von STMicroelectronics
- *src*
  - *board*: Interface für die am Developer-Board vorhandene Beschaltung/Peripherie
  - *common*: für alle Projekte benötigte Header
  - *modules*: Module die Funktionalität in Form einer einfachen API zur Verfügung stellen
  - *project*: Ordner für die einzelnen Projekte
    * *motor*
    * *etherbridge*
    * ...
- *assert_stm.c*: Stellt eine ASSERT [10] Funktion zur Verfügung. Die Funktion hat nur einen Parameter, ist dieser logisch `WAHR` wird nichts unternommen. Ist der Ausdruck jedoch logisch `FALSCH` wird eine Fehlermeldung ausgegeben und das Programm wird angehalten. Dieses Verhalten wird verwendet um Parameter im Programmablauf auf deren Richtigkeit zu überprüfen.

  In der "Standard Peripheral Library" werden in jeder Funktion die Parameter mittels der ASSERT-Funktion überprüft.
- *assert_stm.h*: dazugehörige Header-Datei
- *gxx_std_includes.h*: wird benötigt für die bessere Source-Code Darstellung im Eclipse, genaue Beschreibung in Abschnitt 5.3.1

[10] assert - englisches Wort für versichern bzw. feststellen

- *startup.c*: Startup Code, initialisiert den Clock, die globalen Variablen und wenn nötig den Speicher zur Ausführung im SRAM, ruft anschließend die `main()`-Funktion auf.
- *startup.h*: dazugehörige Header-Datei
- *stm32f4xx_it.c*: stellt die Funktionen für die Exception-Handler zur Verfügung.
- *stm32f4xx_it.h*: dazugehörige Header-Datei

Wird der Projektbaum neu in ein Eclipse-Projekt eingefügt, muss man das Eclipse-Projekt mit folgenden Einstellungen erstellen: "new Project" ⇒ C-Project ⇒ Projekttyp: "Makefile Project (Empty Poject)" / "– Other Toolchain –".

config.h

Im root-Verzeichnis des Projektes befindet sich die Include-Datei `config_template.h`. Diese Datei muss bei jedem Projekt als `config.h` vorhanden sein. In dieser Datei sind nötige Einstellungen für die Standard-Peripheral-Lib und es können die selbst geschriebenen Module (Aufgelistet unter 4.2) eingebunden werden. Standardmäßig wird kein einziges Modul verwendet, um keinen unnötigen Source-Code mitzukompilieren und Codegröße zu sparen.

Source-Code Darstellung

Für die korrekte Darstellung der Warnings und Errors und der Code Auto Completion benötigt das Eclipse Informationen über den verwendeten GCC. Diese Informationen sind in den Header Dateien des verwendeten GCC verfügbar. Eclipse holt sich die Header-Dateien mit den unter "Discovery Options" (Projekteinstellungen unter C/C++ Build) sogenannten Scanner. Bei der Verwendung eines externen GCC (nicht der Standard System GCC), hat dieser Scanner immer Probleme verursacht. Die vorhandenen Scanner-Profile waren nicht ausreichend, deshalb wurde er standardmäßig deaktiviert (durch Auswahl "– Other Toolchain –" bei der Projekterstellung). Die benötigten Einstellungen müssen selbstständig getätigt werden. Die Einstellungen oder das Fehlen ebendieser hat keinen Einfluss auf das Compilieren oder die erstellte Binät-Datei, es betrifft nur die Darstellung im Eclipse.

Es müssen folgende Include Verzeichnisse angegeben werden:

- .../codesourcery/arm-none-eabi/include
- .../codesourcery/lib/gcc/arm-none-eabi/<GCC-Version>/include
- .../codesourcery/lib/gcc/arm-none-eabi/<GCC-Version>/include-fixed

Viele Darstellungen, z.B. korrekte Datentypen, unbenutzter Code, beziehen sich auf #defines die im GCC definiert sind. Das Problem ist, diese #defines stehen in keiner #include-Datei, sondern werden erst beim Compilieren an den Compiler mit übergeben. Um diese #defines auch im Eclipse verfügbar zu haben, kann man über den Befehl (siehe Listing 5.4) die Standard-Defines in eine Datei schreiben und diese zu dem Eclipse-Projekt hinzufügen. Man muss aber beachten, dass diese Datei nicht mit compiliert

wird. Ansonsten hat man während dem Kompilieren einige Warnings, weil alle #defines doppelt vorhanden sind.

Listing 5.4: Linux-Konsolen Befehl für die Standard-Defines

```
echo | arm-none-eabi-gcc -E -dD - > gcc\_std\_includes.h
```

5.3.2. udev-ruels

Folgendes gilt nur für Linux-basierte Systeme, auf Windows Rechnern ist das nicht nötig. Zum Ausführen von "openOCD" oder dem "JLinkGDBServer" werden root-Rechte benötigt. Grund, ist der Zugriff auf das USB-Programmiergerät, welches die erhöhten Rechte benötigt. Das kann man umgehen, indem eine udev-rule erstellt wird, die es dem Benutzer erlaubt ohne root-Rechte auf ein bestimmtes oder mehrere USB-Geräte zugreifen zu können.

Installiert man openOCD aus den Ubuntu - Paketquellen wird bereits eine passende udev-rule für alle unterstützten Programmiergeräte erstellt (40-openocd.rules). Der Benutzer muss, wie bei der folgenden Methode, Mitglied der Gruppe "plugdev" sein.

Der Inhalt einer udev-Rule um die verwendeten JLink-Programmiergeräte ohne root-Rechte zu benutzen, ist in dem Listing 5.5 dargestellt. Die Product- und Vendor-ID muss mit dem verwendeten USB-Gerät übereinstimmen. Jeder Benutzer der auf dem System in der Gruppe "plugdev" ist hat, lese und schreib-Rechte für das angegebene USB-Gerät.

Listing 5.5: 10-jlink.rule

```
SUBSYSTEMS=="usb", ATTRS{idProduct}=="0101",
    ATTRS{idVendor}=="1366", MODE="666", GROUP="plugdev"
```

Diese Datei muss in den Ordner "/lib/udev/rules.d" kopiert werden. Der Dateiname ist nicht zufällig gewählt und entspricht dem Muster: <Zahl>-<Name>.rule. Die Zahl gibt an, in welcher Reihenfolge die Dateien geladen werden, der Name ist beliebig und die Endung muss ".rule" sein.

Der Befehl um die Regeln neu zu laden lautet *sudo service udev reload*, normalerweise sollte das genügen und man kann sofort starten. In der Praxis hat sich aber gezeigt, dass die neuen Regeln erst nach einem Neustart mit Sicherheit übernommen werden.

Die Technik kann auf beliebige andere USB-Geräte angewandt werden, man muss nur Product- und Vendor-ID entsprechend ändern. Auch nützlich bei USB-seriell Wandler, manchmal benötigt man auch root-Rechte um die Geräte zu verwenden, abhängig vom Treiber. In Listing 5.6 ist die udev-rule für einen Prolific USB-seriell Wandler.

Listing 5.6: 11-prolific.rule

```
SUBSYSTEMS=="usb", ATTRS{idProduct}=="2303",
    ATTRS{idVendor}=="067b", MODE="666", GROUP="plugdev"
```

Die IDs können mit dem Befehl "lsusb" (siehe Listing 5.7) angezeigt werden. Die interessante Ausgabe sind die beiden Zahlen nach "ID". Der erste Teil (vor dem Doppelpunkt) ist die Vendor-ID der zweite Teil ist die Product-ID.

Listing 5.7: Ausschnitt der Ausgabe des Befehls lsusb

```
Bus 002 Device 006: ID 1366:0101 SEGGER J-Link ARM
Bus 006 Device 004: ID 067b:2303 Prolific Technology, Inc.
    PL2303 Serial Port
```

Das beschriebene Verfahren wurde auf verschiedenen Ubuntu Versionen 32 und 64bit getestet und hat hier einwandfrei funktioniert. Auf anderen Linux-Systemen kann es sein, dass die Pfade anders sind, die Vorgehensweise sollte aber sehr ähnlich sein.

Alle benötigten Linux-Befehle noch einmal zusammengefasst:

- *groups <USERNAME>* zeigt an in welchen Gruppen der angegebene Benutzername ist
- *sudo useradd -G <GRUPPE> <USERNAME>* Benutzer der Gruppe hinzufügen
- *lsusb* zeigt alle vorhandenen USB-Geräte an
- *sudo service udev reload* alle udev-rules neu laden, nach dem ändern wichtig

5.3.3. MakeFile

Diese Datei enthält die Konfiguration für den Build-Prozess. Das make-System verfügt über einen kleinen Befehlssatz, der es ihm ermöglicht das Kompilieren und den Ablauf zu steuern. Ein Vorteil bei der Verwendung von Makefiles ist, es werden nur geänderte Source-Code Dateien neu kompiliert.

Aufgrund der im TERA großen Anzahl an unterschiedlichen Software-Projekten, müsste das Make-File oft geändert werden. Das ist in Verbindung mit einer Speicherung in einem SVN-Repository problematisch. Das Einchecken der Datei wäre sehr umständlich, weil immer nur die selben Zeilen geändert sind. Als Abhilfe, wurde das Makefile in eine Konfigurationsdatei und Script-Datei aufgeteilt. In der Script-Datei ist der Ablauf für den Build-Prozess enthalten, dieser greift auf die Konfigurationsdatei zu. Die Konfigurationsdatei enthält alle Einstellungen, die betreffenden Variablen können nach belieben gesetzt werden. Diese Datei wird im Make-File inkludiert. Die Konfigurationsdatei ist nicht im SVN-Repository um Konflikte im Vorhinein zu vermeiden.

Das Make-File enthält sogenannte ”targets“ die festgelegte Befehle hintereinander ausführen. Beim verwendeten Make-File sind folgende Targets verfügbar:

- version: zeigt die verwendete GCC Version an
- size: zeigt die Größe der einzelnen Sections an
- all: alle geänderten Dateien werden neu kompiliert
- clean: löscht den gesamten Build-Ordner
- clean_firmware: löscht den gesamten Firmware-Ordner, die `STM32F4`-SPL, das CMSIS und LWIP (falls mit kompiliert)
- clean_files: löscht nur die erstellte Binär-Datei
- clean_user: löscht Object-files die vom Benutzer erstellt wurden
- program_jlink: das erstellte Binary wird in den Mikrocontroller programmiert, unter Verwendung des JLinkGDB-Servers

- program_openocd: das erstellte Binary wird in den Mikrocontroller programmiert, unter Verwendung von openOCD

Im Eclipse gibt es eine "Make Target"Ansicht. Diese listet alle im Makefile vorhandenen Targets auf, die Targets können durch einen Doppelklick gestartet werden. Die View kann in Eclipse unter dem Menüpunkt: "Window ⇒ Show View → Make Target" eingeschaltet werden.

Das Make-System bemerkt nur Änderungen an Source-Code Dateien. Wird eine Header-Datei geändert, werden die davon abhängigen Source-Code Dateien nicht neu kompiliert. Für das Make-System hat sich die Source-Code Datei nicht verändert. Das ist auch von Bedeutung, wenn die Konfigurationsdatei geändert wird. Eine derartige Änderungen wird vom Make-System nicht bemerkt. Die einzige Abhilfe ist, den gesamten Source-Code neu zu kompilieren. Dazu die beiden Make-Targets: `clean_all` und `all` hintereinander ausführen.

MakeFile-Konfuguration

Mögliche Einstellungen in der MakeFile-Konfiguration:

- `DEVICE`: genaue Typenbezeichnung des benutzten Mikrocontrollers, in der folgenden Auflistung sind die Bezeichnungen der im TERA verwendeten Mikrocontroller zusammengefasst
  - TERA-Prozessorboard: STM32F405RG
  - STM3240G-EVAL: STM32F407IG
  - STM32F4DISCOVERY: STM32F407VG
- `CODE_SPOT` *RAM* oder *FLASH*, gibt an wo der ausführbare Code liegt (interner Flash oder RAM), eine genaue Beschreibung ist in Abschnitt 5.3.4 zu finden
- `PROJECT_NAME` der Projektname muss gleich lauten wie der Ordnername im Verzeichnis ./src/project/, das ist nötig damit das Makefile die Source-Code Dateien
- `CLOCK_SRC` *HSE* oder *HSI* es wird entweder der interne RC-Oszillator oder ein externer Quarz verwendet
- `MCU` *coretex-m4*, gibt den verwendeten Cortex-Kern an, prinzipiell wäre auch *cortex-m3* möglich
- `FPU` *false* zur Zeit kann keine FPU verwendet werden, weil der verwendete GCC [11] ohne FPU Unterstüzung kompiliert wurde, siehe Abschnitt 4.1.4 für eine genaue Beschreibung
- `TARGET` Name der erstellten Binär-Datei
- `USER_DEFINES` Benutzerdefiniert, es können noch gewünschte defines/macros angegeben werden, als Trennzeichen wird ein Leerzeichen verwendet. Wird bei dem define nur ein Name angegeben, wird es mit "1" definiert. Will man den Text/Wert selbst bestimmen, muss die Definition wie folgt aussehen: "name=definition".

[11] codesourcery

- `LIBS` zusätzliche Bibliotheken die mit gelinkt werden sollen, zum Beispiel "-lm" für die math-Library
- `GCC_PATH` Installationspfad der verwendeten GCC,
- `GCC_VERSION` Angabe der GCC-Version, wichtig für die System-Includes
- `OPTIMIZATION` Angabe welche Optimierungen während des Kompilierens angewendet werden sollen, mögliche Werte: 0, 1, 2, 3, s. Eine genaue Beschreibung ist in der Konfigurationsdatei vorhanden.

Die wichtigste Einstellungen ist der `GCC_PATH` in Verbindung mit der `GCC_VERSION`, sie bestimmen den "system include Path" `C_INCLUDE_PATH`. Das sind die `#include` Direktiven mit den Spitzklammern. Ohne diese Information ist ein Kompilieren nicht möglich, bitte überprüfen ob dieser Pfad korrekt ist. Der übliche Pfad wird mit den beiden vorher genannten Einstellungen zusammengebaut, er kann aber je nach Installation unterschiedlich sein.

5.3.4. Linker-Script

Der Linker ist dafür zuständig die, vom Compiler aus dem Source-Code generierten Object-Dateien, zu der endgültigen Binär-Datei zusammenzufügen. Eventuell werden noch benötigte Bibliotheken hinzugefügt (z.B. math-Library). Die Object-Dateien sind in einzelne Segmente unterteilt, diese Segmente müssen bestimmten Speicherbereichen zugeordnet werden. Dafür ist das Linker-Script zuständig. In diesem Script ist auch die Information über die vorhandenen Speicherbereiche im Mikrocontroller enthalten.

Laut der Dokumentation des verwendeten Compilers, sind die Object-Dateien in folgende Segmente unterteilt (dies entspricht den üblichen Aufteilung):

- *.text*: Programmcode
- *.rodata*: Konstanten
- *.data*: globale Variablen, werden initialisiert
- *.bss*: globale Variablen, mit 0 initialisiert

Die beiden Segmente .data und .bss werden für Variablen verwendet und müssen daher einem SRAM zugewiesen werden. Das .data Segment muss zusätzlich noch im Flash gespeichert werden, es enthält die Initialisierungen der Variablen. Der startup-Code ist dafür zuständig die Werte vom Flash an die entsprechenden Adressen der Variablen zu schreiben.

Das zu verwendete Segment von Variablen oder Funktionen wird normalerweise automatisch zugewiesen, es kann aber auch händisch festgelegt werden. Im nächsten Listing 5.8 sind Beispiele angeführt, welchem Segment Variablen automatisch zugewiesen werden.

Listing 5.8: Beispiel für Variablendefinition und deren Segment

```
const uint32_t test1 = 1;   // .rodata
uint32_t test2 = 2;         // .data
uint32_t test3;             // .bss
```

```
const uint32_t test4;          // .bss
```

Will man das Segment selbst festlegen, muss die Variable oder Funktion folgendermaßen definiert werden:

Listing 5.9: Benutzerdefinierte Zuweisung des Segmentes

```
__attribute__ ((section (".user_section"))) uint32_t test5;
    // .user_section muss im linker script definiert sein
```

Für lokale Variablen werden entweder Register verwendet oder sie sind am Stack gespeichert.

math-Library

Werden im Source-Code mathematische Funktionen verwendet (sin, ceil, ...) muss die `math.h` inkludiert werden. Weiters muss die math-Library mit dem Projekt "mitgelinkt" werden. Das kann man über die Variable `LIBS` in der Makefile.config machen.

Für die drei möglichen floating-point Datentypen: `float`, `double` oder `long double`, gibt es für jeder Funktion eine entsprechende Version. Dies wird durch den Funktionsnamen gekennzeichnet. Hier ein Beispiel für die Sinus-Funktion:

- `sinf` für float Werte
- `sin` für double Werte
- `sinl` fur long double Werte

Codeausfühung

Der verwendete Mikrocontroller kann Code vom internen Flash, SRAM oder einem externen Speicher (angebunden über den FSMC [12]) ausführen. Code vom RAM auszuführen sollte einen Geschwindigkeitsvorteil bringen. Beim Flash ist die Zugriffszeit von der CPU-Frequenz abhängig. Bei der maximalen CPU-Frequenz von 168MHz muss der Flash-Zugriff über 5 wait-states erfolgen. Das bedeutet das die CPU 5 Zyklen auf die Daten warten muss. Der Mikrocontroller versucht das zu umgehen, STM hat einen ART accelerator [13] verbaut. Dieser besitzt eine "instruction prefetch queue" und einen "branch cache" um 0 Wait-States zu ermöglichen. Diese können aber nicht garantiert werden und sind eher als gemittelte Werte zu verstehen. Ein zusätzlicher Vorteil bei der Ausführung aus dem RAM ist, im Betrieb können die Interrupt-Vectoren geändert werden.

Realisiert wird die Codeausführung über die Remap-Funktionalität des Prozessors (siehe 4.1.3). Diese Eigenschaft des Prozessors vereinfacht die Umsetzung sehr. Die Remap-Funktionalität bedeutet, dass an die Adresse 0x00 einer der folgenden Speicherbereiche gemappt werden: internes Flash, internen SRAM, System-Memory (embedded bootloader) oder ein externer Speicher.

[12] flexible static memory controller
[13] Adaptive real-time memory accelerator

Die Vorgangsweise ist wie folgt, der Code wird, wie üblich, im Flash gespeichert. Nach einem Reset wird vom Startup-Code der gesamte Code in den RAM kopiert. Anschließend wird das Speicher-Mapping geändert und der Prozessor führt den Code aus dem RAM aus. Es muss sichergestellt sein dass sich im Flash und RAM beim Starten der selbe Code befinden, ansonsten schlägt das Remapping fehl. Deshalb muss sich im ersten Flash-Sektor der Startup-Code befinden.

Die zwei Linker-Scripte: *stm32f4_flash.ld* und *stm32f4_ram.ld* sind für die unterschiedlichen Speicherbereiche in denen der Code ausgeführt wird. Das Linker-Script wird in der Makefile.config ausgewählt. Nach einem Wechsel unbedingt ein "clean-all" machen. Es ändern sich die Speicherbereiche und es müssen alle Dateien neu kompiliert werden. Nicht nur die, in denen eine Textänderung stattgefunden hat. Um sicherzustellen, dass die Ausführung aus dem SRAM funktioniert, wurde ein Testprogramm erstellt. Dieses Programm ist unter 4.4.6 beschrieben.

Speicher-Map

In den Tabellen 5.2 und 5.3 wird dargestellt, wie die einzelnen Segmente, in den unterschiedlichen Ausführungsmodi, auf den Speicher aufgeteilt sind. Die vorhandenen Speicher deren Größe und Adressbereich ist in Abschnitt 4.1 beschrieben.

FLASH	.startup_code .text .rodata
CCRAM	.data .bss .heap .stack
SRAM1	eigene Daten, DMA-Buffer, ...
SRAM2	eigene Daten, DMA-Buffer, ...
BKPSRAM	eigene Verwendung

Tabelle 5.2.: Aufteilung des Speichers, wenn der Flash für die Ausführung verwendet wird

In der Tabelle 5.4 ist die Aufteilung des Flashs dargestellt. Der Statup-Code muss wegen der Remapping-Funktion im ersten Sektor (an der Adresse null) sein. Der Speicherbereich für den EEPROM ist gleich dahinter platziert, um noch die kleineren Sektoren nutzen zu können.

FLASH	.startup_code .text .rodata
CCRAM	.data .bss .heap .stack
SRAM1	.startup_code .text .rodata
SRAM2	eigene Daten, DMA-Buffer
BKPSRAM	eigene Verwendung

Tabelle 5.3.: Aufteilung des Speichers, wenn der SRAM für die Ausführung verwendet wird

	Größe	Nutzung
Sektor 0	16kb	Startup-Code
Sektor 1	16kb	EEPROM
Sektor 2	16kb	
Sektor 3	16kb	
Sektor 4	64kb	Code
Sektor 5	128kb	
Sektor 6	128kb	
Sektor 7	128kb	
Sektor 8	128kb	
Sektor 9	128kb	
Sektor 10	128kb	
Sektor 11	128kb	

Tabelle 5.4.: Nutzung der Sektoren vom internen Flash des Mikrocontrollers

6. Hardware

6.1. Prozessorboard

Da der gewählte Mikrocontroller auf jeder Platine im Auto verwendet wird, haben wir uns im Team dazu entschieden ein Prozessorboard zu fertigen. Auf dem Prozessorboard soll nur die notwendigste Hardware verbaut werden. Trotzdem soll es möglich sein, das Modul ohne Zusatzbeschaltung in Betrieb zu nehmen. Der Einsatz eines eigenen Moduls für den Mikrocontroller hat mehrere Vorteile. Der Layoutaufwand für eine Anwendung reduziert sich, da die Grundbeschaltung des Mikrocontrollers schon existiert, man muss nur einen Stecker auf der Platine vorsehen. Bei noch in der Entwicklung stehende Platinen ist das Prozessorboard leicht wiederverwendbar. Durch die Modularisierung kann im Fehlerfall das Trägerboard (Anwendung) oder das Prozessorboard einfach getauscht werden.

Auf dem Prozessorboard ist folgende Hardware verbaut:

- notwendige Zusatzbeschaltung für den Mikrocontroller: Quarz, Kondensatoren, pull-up/down Widerstände
- Reset Taster
- Stecker (Stiftleisten) für JTAG, RS232 und I2C
- LED's
- CAN-Bus Transceiver + TVS-Diode

6.1.1. Version 1

Bei der ersten Version wurde nur der Schaltplan von mir erstellt. Dabei wurden die Empfehlungen und Vorgaben vom Datenblatt verwendet. Das Schematic gestaltet sich sehr einfach: es wurden die nötigen Stütz- und Koppelkondensatoren verbaut, vorgeschriebene Widerstände und ein CAN-Transceiver. Um einige Peripherien/Schnittstellen leichter zu erreichen wurden für die serielle Schnittstelle, den CAN-Bus, I2C und die Versorgung Stiftleisten vorgesehen. Die Verbindung zum Trägerboard wurde ebenfalls über Stiftleisten realisiert. Das Layout wurde von einer anderen Person im Team erstellt. Es wurde das open source Programm KiCAD [1] verwendet.

[1] http://www.kicad-pcb.org/

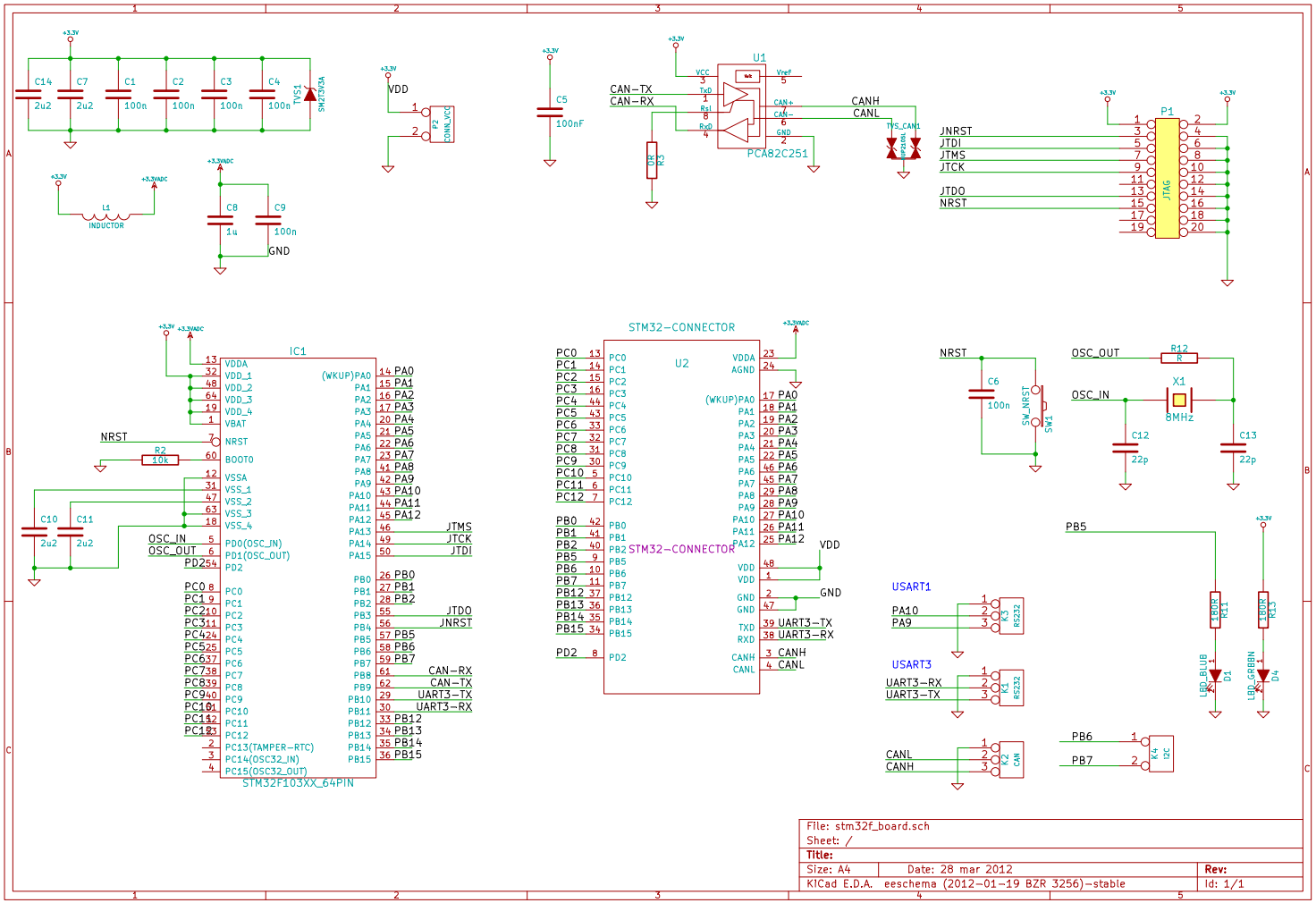


Abbildung 6.1.: Schaltplan des Prozessorboards v1.0

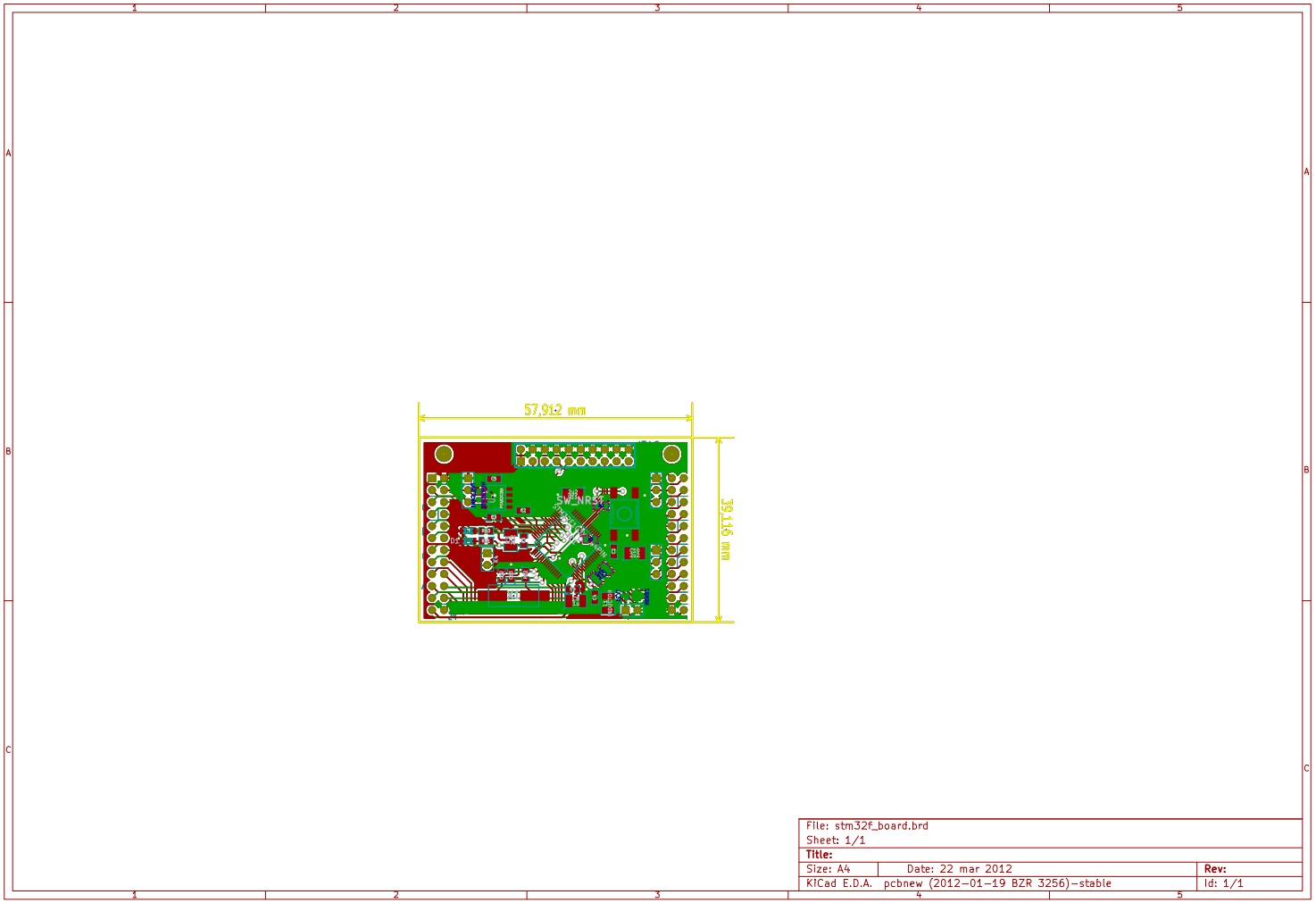


Abbildung 6.2.: Platinenlayout des Prozessorboards v1.0

6.1.2. Version 2

Anhand der gesammelten Erfahrungen im praktischen Einsatz mit der ersten Version des Prozessorboards konnten einige Unzulänglichkeiten gefunden werden. Bei der zweiten Version wurde versucht diese auszubessern. Eine andere große Änderung, war die Umstellung auf ein neues Layoutprogramm. Nun wird das Programm Eagle von der Firma Cadsoft [2] verwendet.

Folgende Änderungen wurden durchgeführt:

- Die Stecker wurden geändert. Der alte Steckertyp war sehr unhandlich und dadurch gestaltete sich das Ein-/Ausstecken sehr schwierig. Dabei wurde das Prozessorboard stark gebogen und verformt.
- Die Platine kann jetzt mit 4 Schrauben am Trägerbord befestigt werden, das soll einen besseren und gleichmäßigeren Halt auf dem Trägerboard gewährleisten.
- Es wurden mehr Stiftleisten eingesetzt, um mehr Peripherie während dem Entwicklungsprozess einfacher verwenden zu können. Folgende Peripherie ist über Stiftleisten erreichbar: RS232, CAN, I2C, SPI, DAC, JTAG.
- Bei der ADC-Peripherie des Mikrocontrollers kann über einen Lötjumper zwischen zwei unterschiedlichen Spannungsversorgungen gewählt werden: entweder eine Referenzspannungsquelle oder die Standardversorgung über eine Ferrit-Spule. Durch die Referenzspannungsquelle ist der ADC weniger Störungen ausgesetzt und sollte bessere Messergebnisse liefern.
- VBAT des Mikrocontrollers wurde an den Stecker geführt. So kann der BKPSRAM mit Strom versorgt werden wenn am Trägerboard eine Batterie verbaut ist.
- Es sind alle Ports des Mikrocontrollers an die Stecker geführt.
- Die beiden Ground-Potentiale: Digital und Analog wurden räumlich getrennt. Das soll Störungen im ADC verringern.
- Durch die kleine Bauweise musste der JTAG-Stecker geändert werden (siehe Tabelle 6.1.2)
- Es wurden mehr LED's verbaut.

Der in der Aufzählung mehrmals erwähnte Stecker ist die Verbindung zum Trägerboard. Hierbei sind immer (alte oder neue Version) zwei Stecker gleichen Typs verbaut.

[2] http://www.cadsoft.de/

Pin	Funktion
Pin 1	GND
Pin 2	3,3V
Pin 3	JTMS
Pin 4	$\overline{SRST}$
Pin 5	JTCK
Pin 6	JTDI
Pin 7	JTDO
Pin 8	JNRST

Tabelle 6.1.: JTAG Steckerbelegung des Prozessorboards 2.0

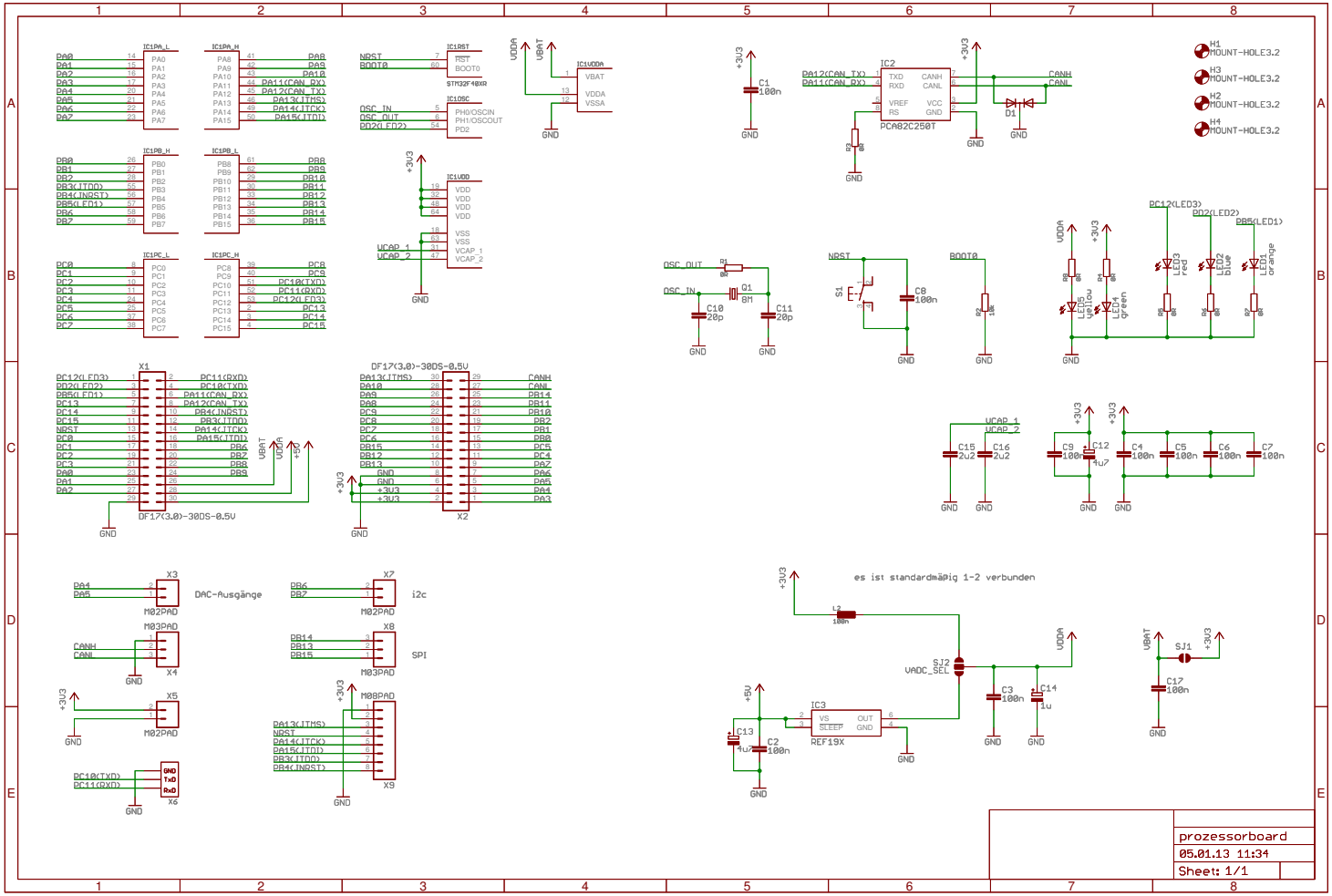
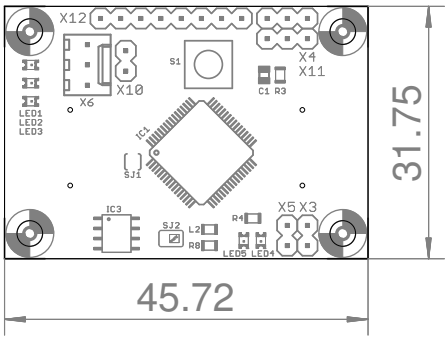
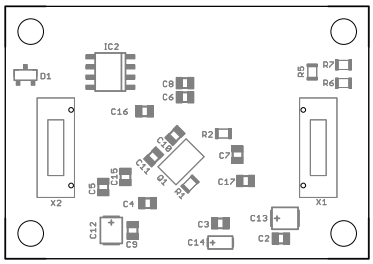


Abbildung 6.3.: Schaltplan des Prozessorboards Version 2.0

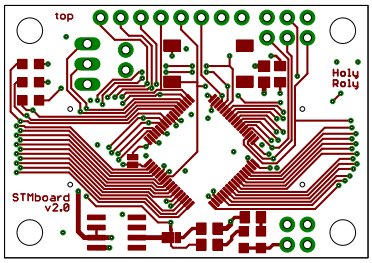


(a) top

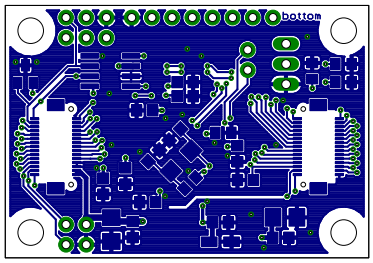


(b) bottom

Abbildung 6.4.: Bestückungsplan vom Prozessorboard 2.0



(a) top



(b) bottom

Abbildung 6.5.: PlatinenlLayout vom Prozessorboard 2.0

Die beiden inneren Lagen der Platine sind nicht abgebildet, sie bestehen entweder aus einer durchgängigen Masse oder $3,3V$ Fläche.

6.2. EVAL-Board: STM3240G

Zum Testen und Entwickeln der Software wurde das Evaluation-Board `STM3240G-EVAL` von STMicroelectronics verwendet. Die nachfolgende Auflistung gibt einen Überblick

über die am Evaluation-Board vorhandene Peripherie. Für die genannte Peripherie existiert im Eclipse-Projekt ein Softwaremodul. Die Softwaremodule wurden entweder selbst entwickelt oder Beispielcodes aus den diversen Beispielprojekten direkt von STMicroelectronics wurde adaptiert.

- STM32F407IGH6 Mikrocontroller
- LCD: 3,2“, 320x240 Pixel, Farb-LCD
- LED: 4 LED's (grün, orange, rot, blau)
- SRAM: 16Mbit SRAM (2Mb)
- CAN: CAN2.0A/B kompatibel
- RS232: standard 9pol D-Sub Stecker
- I^2C: IO-Expander (Joystick, LCD-touchscreen), MEMS, EEPROM
- Poti: Poti an einem ADC-Eingang
- 3 Taster: Wake-Up [3], Tamper [4], User
- MotorControl: 34pol IDC-Steckverbinder für das STM-MotorControlKit, siehe Abschnitt 6.3
- JTAG: 20pol IDC-Steckverbinder für standard JTAG Programmiergeräte
- Ethernet: 10/100Mbit Ethernet Schnittstelle, verbaute PHY:DP83848CVV

6.2.1. Peripherie

Auf dem Evaluation-Board ist jede vom Mikrocontroller unterstützte Peripherie verbaut. Dadurch kann man im Betrieb nicht jede Peripherie gleichzeitig nutzen, da oft mehrere unterschiedliche Peripherien die selben Ports verwenden. Am Evaluation-Board ist wird diese mögliche Doppelbelegung auf zwei unterschiedliche Arten gelöst: entweder durch die Abschaltung von einzelnen Peripherien über Jumper oder durch 0Ω-Widerstände in den Leitungen, die heraus gelötet werden müssen.

Ohne größeren Aufwand können: CAN, externer SRAM, RS232 oder das LCD genutzt werden. Die restliche Peripherie die noch auf dem Board verbaut ist, benutzt manche Ports doppelt. Eine genaue Auflistung der Ports und deren Verwendung ist in der Tabelle 6.2 aufgelistet.

6.2.2. Ports

Die Tabelle gibt einen Überblick über die Verwendung der Ports, für noch genauere Informationen, der Schaltplan ist in dem ”user manual“ UM1461 [5] zu finden. Für eine eigene Verwendung werden alle Ports zusätzlich auf eine Stiftleiste ausgeführt.

Die zweite Spalte gibt an welche Peripherien mit dem Port verbunden sind. Ist in der dritten Spalte ein Widerstand angegeben ist, muss dieser heraus gelötet werden, um diesen Port verwenden zu können. Ansonsten ist in der Tabelle beschrieben ob der Port ohne weiteres Verwendet werden kann, oder ob etwas beachtet werden muss.

[3] Taster kann verwendet werden um den μC aus dem Stand-By modus aufzuwecken
[4] real-time-clock alternate function input 1

Tabelle 6.2.: Belegung der einzelnen Port-Pins beim STM3240G-EVAL Board

Port	Std-Funktion	Beschreibung/Benutzng
PA0	Button - WakeUp	
PA1	ETH MII_RX_CLK/RMII_REF_CLK	JP6
PA2	ETH-MII/RMII	R44
PA3	ULPI_D0	siehe Notes 6.2.2
PA4	Audio out	R115
PA5	ULPI_CLK	R69
PA6	DCMI	siehe Notes 6.2.2
PA7	ETH-MII/RMII	RS2
PA8	MCO	verwendbar
PA9	USB OTG FS VBUS	verwendbar
PA10	USB OTG FS ID	R21
PA11	USB OTG FS DM	R17
PA12	USB OTG FS DP	R19
PA13	JTMS	JTAG Interface
PA14	JTCK	JTAG Interface
PA15	JTDI	JTAG Interface
PB0	ULPI_D1	siehe Notes 6.2.2
PB1	ULPI_D2	siehe Notes 6.2.2
PB2	BOOT1	
PB3	JTDO	JTAG Interface
PB4	JTRST	JTAG Interface
PB5	CAN2, ULPI_D7	siehe Notes 6.2.2
PB6	I2C	mehrere IC's angebunden
PB7	FSMC - NL	verfügbar IC (U7) nicht bestückt
PB8	Enc-Index (MC), ETH-MII	R3 (MC), RS5(ETH)
PB9	I2C	mehrere IC's angebunden
PB10	ULPI_D3	siehe Notes 6.2.2
PB11	ULPI_D4	siehe Notes 6.2.2
PB12	ULPI_D5	siehe Notes 6.2.2
PB13	CAN2, ULPI_D6	siehe Notes 6.2.2
PB14	ETH-MII/RMII	R41
PB15	FSMC INT	R53
PC0	ULPI_STP	siehe Notes 6.2.2
PC1	I_A (MC), ETH-MII	R37 (MC)
PC2	I_B (MC), ETH-MII	R76(MC), RS5(ETH)
PC3	I_C (MC), ETH-MII	R79(MC), RS5(ETH)
PC4	U_B (MC), ETH-MII/RMII	R13(MC), RS2(ETH)
PC5	U_{TEMP} (MC), ETH-MII/RMII	R5(MC), RS2(ETH)
PC6	SmartCard IO, I2S_MCK	R127(SC), verwendbar

Fortsetzung auf der nächsten Seite ...

Tabelle 6.2

Port	Std-Funktion	Beschreibung/Benutzng
PC7	LED4 (blau)	
PC8	MicroSD D0, Brake (MC)	pull-up bei SD Karte
PC9	MicroSD D1, I2S CLKin	pull-up bei SD Karte
PC10	RS232/IrDA TX, MicroSD D3	pull-up bei SD Karte
PC11	RS232/IrDA RX, MicroSD D3	pull-up bei SD Karte, JP22 1-2 -> rs232
PC12	MicroSD CLK	siehe Notes 6.2.2
PC13	Button - Tamper	
PC14	LSE - XTAL	
PC15	LSE - XTAL	
PD0	FSMC D2, CAN1	JP10(CAN)
PD1	FSMC D3, CAN1	JP3(CAN)
PD2	MicroSD CMD	pull-up bei SD Karte
PD3	FSMC CLK	verfügbar IC (U7) nicht bestückt
PD4	FSMC $\overline{OE}$	
PD5	FSMC $\overline{WE}$	
PD6	FSMC $\overline{WAIT}$	R54
PD7	FSMC $\overline{CS1}$	pull-up 3V3, R52
PD8	FSMC D13	
PD9	FSMC D14	
PD10	FSMC D15	
PD11	FSMC A16	
PD12	FSMC A17, Enc-A	
PD13	FSMC A18, Enc-B	
PD14	FSMC D0	
PD15	FSMC D1	
PE0	FSMC $\overline{BLE}$	
PE1	FSMC $\overline{BHE}$	
PE2	TRACE Clk	CN13
PE3	FSMC A19, TRACE D0	Auswahl durch JP2 2-3 FSMC
PE4	FSMC A20, TRACE D1	Auswahl durch JP1 2-3 FSMC
PE5	TRACE D2	CN13
PE6	TRACE D3	CN13
PE7	FSMC - D4	
PE8	FSMC - D5	
PE9	FSMC - D6	
PE10	FSMC - D7	
PE11	FSMC - D8	
PE12	FSMC - D9	
PE13	FSMC - D10	
PE14	FSMC - D11	

Fortsetzung auf der nächsten Seite ...

Tabelle 6.2

Port	Std-Funktion	Beschreibung/Benutzng
PE15	FSMC - D12	
PF0	FSMC - A0	
PF1	FSMC - A1	
PF2	FSMC - A2	
PF3	FSMC - A3	
PF4	FSMC - A4	
PF5	FSMC - A5	
PF6	SmartCard OFF	R126
PF7	SmartCard Reset	
PF8	LCD CS	n.c. frei
PF9	Potentiometer	
PF10	Audio In	R196
PF11	USB OTG FS overcurrent	R15
PF12	FSMC - A6	
PF13	FSMC - A7	
PF14	FSMC - A8	
PF15	FSMC - A9	
PG0	FSMC - A10	
PG1	FSMC - A11	
PG2	FSMC - A12	
PG3	FSMC - A13	
PG4	FSMC - A14	
PG5	FSMC - A15	
PG6	LED1 (grün)	
PG7	SmartCard Clk	verwendbar
PG8	LED2 (orange)	
PG9	FSMC $\overline{CS2}$	R47
PG10	FSMC $\overline{CS3}$	
PG11	ETH-MII/RMII	RS6
PG12	SmartCard CMCVCC, LCD VSYNC	R128 (SmartCard), n.c. (LCD)
PG13	ETH-MII/RMII	RS6
PG14	ETH-MII/RMII	RS6
PG15	Button - User	
PH0	HSE - XTAL	
PH1	HSE - XTAL	
PH2	ETH-MII	RS3
PH3	ETH-MII	RS3
PH4	ULPI_NXT	R61
PH5	USB OTG FS switch	R18
PH6	ETH-MII	RS5
PH7	ETH-MII	RS3

Fortsetzung auf der nächsten Seite ...

Tabelle 6.2

Port	Std-Funktion	Beschreibung/Benutzng
PH8	DCMI, NTC (MC)	siehe Notes 6.2.2)
PH9	DCMI	siehe Notes 6.2.2
PH10	DCMI, PFC2 (MC)	R59 (MC)
PH11	DCMI, PFC1 (MC)	R57 (MC)
PH12	DCMI, PFC (MC)	
PH13	TIM8_CH1N (MC), MicroSD detect	R2 (MicroSD)
PH14	TIM8_CH2N (MC), DCMI	
PH15	TIM8_CH3N (MC), SmartCard 3/5V	
PI0	I2S WS	verwendbar
PI1	I2S SCK	verwendbar
PI2	IO Exp. Interrupt	R136
PI3	I2S SD	verwendbar
PI4	TIM8 BKIN (MC), DCMI	R14(MC), siehe Notes 6.2.2
PI5	TIM8 CH1 (MC), DCMI	siehe Notes 6.2.2
PI6	TIM8 CH2 (MC), DCMI	siehe Notes 6.2.2
PI7	TIM8 CH3 (MC), DCMI	siehe Notes 6.2.2
PI8	LCD HSYNC	n.c. frei
PI9	LED3 (rot)	
PI10	ETH-MII	RS3
PI11	ULPI_DIR	R62, siehe Notes 6.2.2

Notes Hier sind noch zusätzliche Informationen über die Peripherie.

- ULPI: USB OTG IC, Datenleitungen können verwendet werden wenn JP31 nicht vorhanden ist
- DCMI: Kamera Stecker, ist keine Kamera angesteckt können die Pins frei verwendet werden
- MCO: master clock output, wird von einiger Peripherie als Clock verwendet, Audio (R211), Ethernet (R208). Der Widerstand in Klammer ist zum Wegschalten.
- I2C: am I2C Bus (PB6, 9) hängt der IO-Expander, Audio, EEPROM, MEMS
- MC: beim Motor-Control Verbinder sind auf der Power-Stage nicht alle Pins verwendet: PFC, PFC1, PFC2, Encoder-Index können verwendet werden.
- MicroSD: die Pins können verwendet werden, wenn keine Karte eingesteckt ist. Wo angegeben, ist ein pull-up Widerstand vorhanden.
- Beim CAN-Bus kann man sich aussuchen (mittels Jumper) ob CAN1 oder CAN2 verwendet wird, Es wird Standardmäßig CAN2 verwendet, CAN1 wird nicht benutzt, weil dieser die Datenleitungen vom SRAM Chip verwendet.
- CAN2 kollidiert mit einem USB-OTG Chip, um diesen zu deaktivieren muss man JP31 entfernen.
- Die RS232-Schnittstelle teilt sich die Datenleitung mit der verbauten MicroSD-

Karte, da die MicroSD-Karte nie zur Verwendung gedacht ist, kann die Peripherie immer verwendet werden. Mit JP22 kann eingestellt werden welche Übertragungsart verwendet werden soll, entweder Standard RS232 oder IrDA.

6.3. Power Stage: MB459-B01

Dieses Evaluation-Board [4] ist geeignet für die Ansteuerung von dreiphasigen Synchron- oder Asynchronmaschinen. Die maximale Leistung beträgt ca. 100W. Die Verbindung mit dem STM3240G Developer-Board ist über ein 34 poliges Flachbandkabel realisiert. Der Stecker und die Belegung ist ein STMicroelectronics interner Standard. Es wird der mitgelieferte Motor verwendet. Die Hardwaredaten des Motors sind in Tabelle 6.3 aufgelistet. Als Versorgungsspannung für das Board wird 24V verwendet (Nennspannung des Motors).

Der Jumper W15 verbindet die 5V der Power-Stage mit dem Evaluation-Board, dadurch kann das verwendete Evaluation-Board von der Power-Stage versorgt werden. Bei dem verwendeten Evaluation-Board STM3240G, sollte diese Art der Versorgung nicht zu lange verwendet werden. Das Board benötigt ca. $268mA$, dieser Stromverbrauch lässt den linearen Spannungsregler auf der Power-Stage überhitzen.

Konfiguration der Platine für einen permanent erregen Synchronmotor:

- `W1` - Position "<35V only"
- `W4` - vorhanden
- `W5` - nicht vorhanden
- `W6` - vorhanden
- `W7` - vorhanden und in der selben Position, wie am Bestückungsdruck aufgezeichnet
- `W8` - vorhanden
- `W9` - vorhanden
- `W10` - vorhanden
- `W11` - vorhanden
- `W12` - nicht vorhanden
- `W13` - nicht vorhanden
- `W14` - nicht vorhanden
- `W15` - nicht vorhanden
- `W16` - vorhanden und entgegengesetzt zur aufgezeichneten Position eingesetzt (Bestückungsdruck)
- `W17` - vorhanden
- `W18` - vorhanden
- `W19` - vorhanden

Motor Shinano PMSM mit eingebauten Hall-Sensoren und einen inkremental Encoder

Anschlüsse des Motors (Windungsanschluss, Encoder):

Nennleistung	P_N	$80\,W$
Nennspannung	U_N	$24\,V$
Nennstrom	I_N	$4.6\,A$
Nenndrehzahl	N_N	$3000\,rpm$
Nennmoment	M_N	$0.25\,Nm$
Polpaarzahl	Z_P	2
Phasenwiderstand	R	$0.6\,\Omega\ \pm 10\%$
Phaseninduktivität	L	$1.4\,mH\ \pm 30\%$
Rotorträgheitsmoment	J	$11 \cdot 10^{-6}\,kg\,m^2$
Drehmomentkonstante	k_m	$59\,mNm/A$
Back EMF Konstante	k_e	$0.059Vs/rad$

Tabelle 6.3.: Parameter des Shinano Motors

Pin	Funktion	Farbe
1	Spule W1	schwarz
2	Spule W2	weiß
3	Spule W3	rot

Tabelle 6.4.: Windungsanschluss des Shinano Motors

Pin	Funktion	Farbe
1	5V	rot
2	GND	schwarz
3	Channel A	blau
4	Channel B	gelb

Tabelle 6.5.: Encoder Anschluss des Shinano Motors

Strommessung Auf dem Developer ist eine Strommessung über drei Shunt-Widerstände realisiert. Diese messen den Strom, durch den low-side MOSFET, mittels eines Shunts. Die nötige Verstärkerschaltung ist in Abbildung 6.6 zu sehen. Sie verstärkt die Spannung, welche am Shunt abfällt, um den Messbereich vom ADC auszunutzen. Mit den Widerständen am Eingang des Verstärkers wird ein Offset erzeugt, dieser wird benötigt um über den Shunt auch negative Ströme messen zu können.

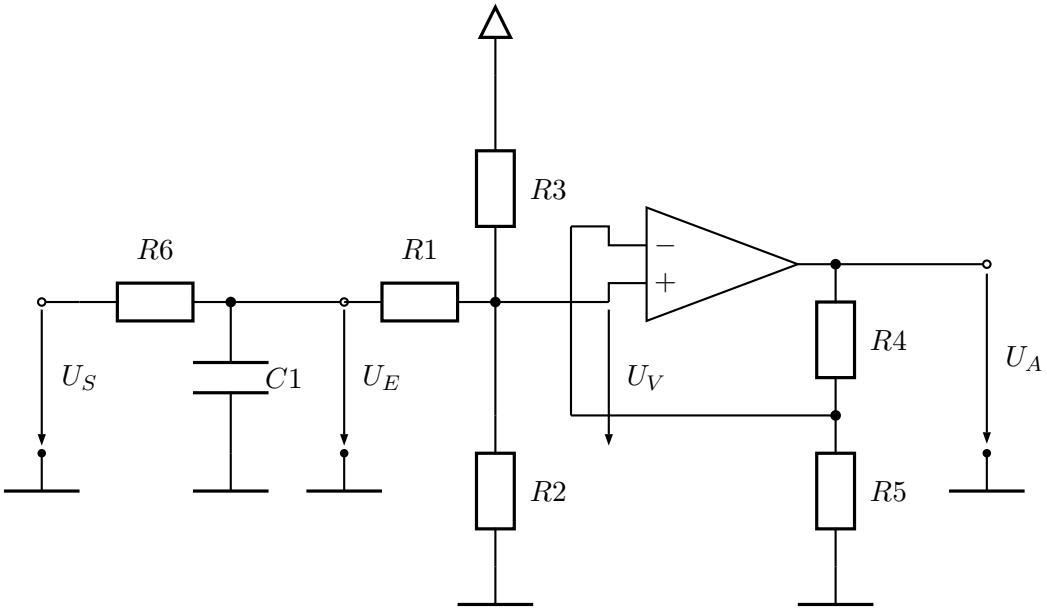


Abbildung 6.6.: Verstärkerschaltung für einen Shunt

Widerstandswerte:

- $R_S = 0,22\Omega$
- $R_1 = 560\Omega$
- $R_2 = 560\Omega$
- $R_3 = 4,7k\Omega$
- $R_4 = 5,4k\Omega$
- $R_5 = 1k\Omega$
- $R_6 = 100\Omega$
- $C_1 = 10nF$

Für die Berechnung der Verstärkung der OPV-Schaltung wird das Tiefpassfilter ver-

nachlässigt. Verstärkung des nicht invertierenden Verstärkers:

$$G = 1 + \frac{R_4}{R_3}$$
$$G = 6,4$$
$$U_A = G \cdot U_V$$

Die Offset-Anpassung wird mit dem Superpositionsprinzip berechnet:
Fall I: $U_E = 0$

$$R_g = R_3 + \frac{1}{\frac{1}{R_2} - \frac{1}{R_3}}, I_3 = \frac{U_B}{R_g}, U_3 = I_3 R_3$$
$$U_V = U_B - U_3$$

Fall II: $U_B = 0$

$$R_g = R_1 + \frac{1}{\frac{1}{R_2} - \frac{1}{R_3}}, I_1 = \frac{U_e}{R_g}, U_1 = I_1 R_1$$
$$U_V = U_E - U_1$$

Beide Fälle zusammenfassen: I + II:

$$U_V = U_B + U_E - U_1 - U_3$$
$$U_V = U_B + U_E - \frac{R_1 U_E}{R_1 + \frac{1}{\frac{1}{R_2} - \frac{1}{R_3}}} - \frac{R_3 U_B}{R_3 + \frac{1}{\frac{1}{R_1} - \frac{1}{R_2}}}$$
$$U_V = \frac{U_E \frac{1}{R_1} + U_B \frac{1}{R_3}}{\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3}}$$
$$U_V = \frac{U_E}{k\ R_1} + \frac{U_B}{k\ R_3}, k = \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3}$$

Die beiden Systeme: Widerstandsnetzwerk und Verstärker zusammenfassen

$$U_A = 6,4 \frac{U_B}{R_3\ k} + 6,4 \frac{U_E}{R_1\ k}$$

Mit den oben angegebenen Widerstandswerten und einer Versorgungsspannung von $U_B = 5V$ ergibt sich folgender Eingangsspannungsbereich:
$U_{E,min} = -0,6V$, $U_{E,max} = 0,5V$.
Mit einem Widerstandswert von $R_S = 0,22\Omega$ bedeutet das einen Strommessbereich von ungefähr $I_{mess} = \pm 2,3A$. Eine vereinfachte Formel für das Mikrocontrollerprogramm: $I[mA] = 1,2 * (ADCvalue - ADCoffset)$.

6.4. STM-Motor-Board

Für die Entwicklung des Reglers verwendete Hardware. Sie besteht aus dem Evaluation-Board STM32F40G (siehe 6.2) und dem Leistungsteil von STMicroelectronics MB459-B01 (siehe 6.3). Als Motor wird der beiliegende Motor von Shinano verwendet, die Daten sind in der Tabelle 6.3 aufgelistet.

Um diese Kombination verwenden zu können, müssen am Evaluation-Board ein paar kleine Umbauarbeiten vorgenommen werden:

- die MicroSD-Karte muss entfernt werden
- das Kamera-Modul muss ebenfalls entfernt werden
- das Netzwerk kann nicht verwendet werden, dazu müssen folgende 0R-Widerstandsnetzwerke entfernen werden: RS2,3,5,6.
- für die Strommessung müssen folgende Lötbrücken geschlossen werden: SB11 - MC current A, SB12 - MC current B
- beim Inkremental-Encoder müssten auch zwei Lötbrücken geschlossen werden: SB14, SB15. Die dadurch in Anspruch genommenen Pin's werden auch noch vom FSMC [5] verwendet. Der FSMC wird benötigt um den SRAM und das LCD zu verwenden. Um diese beiden Peripherien trotzdem nutzen zu können, wurde der Encoder mit einem anderen Timer verbunden: Timer3 (PA6-TIM3_CH1, PA7-TIM3_CH2).
- folgende Widerstände wurden entfernt um die beiden DAC-Ausgänge verwenden zu können: R115, R96

6.5. TERA Motorsteuerung

Diese Steuerung wurde im Team von mehreren Mitgliedern entwickelt, eingesetzt wird sie im UCC. Ihr Aufbau ist sehr modular gestaltet. Die Steuerung besteht aus einem Trägerboard, auf dem die einzelnen selbst entwickelten Module aufgesteckt werden. Sie wird in folgende Module unterteilt:

- Prozessorboard: Mikrocontroller mit nötiger Peripherie
- DC/DC-Wandler: stellt von der Batteriespannung die nötigen Spannungen für die einzelnen Komponenten zur Verfügung
- Halbbrücken: drei idente Module verbaut, das Modul besteht aus zwei MOSFETs, Stützkondensator und MOSFET Treiber
- Strommessung: ebenfalls drei idente Module verbaut, das Modul besteht nur aus dem Stromsensor, es wird ein integrierter HALL-Stromsensor von Allegro (ACS712) verwendet. Dieser Sensor gibt eine Spannung von $U_A = 0 - 5V$ aus. Diese Spannung bildet den gesamten Messbereich von $\pm 20A$ ab. Für die Anpassung an den ADC des Mikrocontrollers wird ein einfacher passiver Spannungsteiler verwendet.

[5] flexible static memory controller

A. Dateiübersicht

Übersicht der einzelnen Dateien im Projekt. Zu jeder Datei ist eine kurze Beschreibung zu ihren beinhaltenden Funktionen angegeben. Ist bei einer Header-Datei nichts angegeben, enthält sie nur die Funktionsdeklarationen der entsprechenden C-Datei.

- advanced_timer.c: Funktionen für den Advanced-Timer (Peripherie im Microcontroller), dieser Timer steuert die Halbbrücken an
- advanced_timer.h:
- config.h: Konfiguration für das Projekt, siehe 5.3.1
- controller.c: stellt einen PI-Regler bereit, die Control-Loop für die beiden Strom- und den Drehzahlregler, weiters sind die Regelschleifen für die Steuerungen über CAN oder Benutzereingaben implementiert
- controller.h:
- drive.c: Hauptprogramm, Initialisierungen der einzelnen Module, beinhaltet die main-loop (bearbeitet das User-Interface)
- drive_config_values.h: Präprozessor-Definitionen der möglichen Einstellungen
- drive_config.h: Konfiguration des Programms
- drive_it.c: ADC Interrupt, triggert den Stromregler
- drive_it.h:
- eeprom_config.c: Funktionen für die Speicherung von Einstellungen im simulierten EEPROM
- eeprom_config.h:
- feedback.c: allgemeine Funktionen für die Auswertung eines Winkelsensors (spezielle Implementierung in den einzelnen Dateien), Drehzahlmessung
- feedback.h:
- feedback_encoder.c: Funktionen für einen beliebigen inkrementalen Encoder, der Sensor muss zwei um 90° verschobene Signale generieren
- feedback_encoder.h:
- feedback_rotary.c: Funktionen für den absoluten Encoder von AMS: AS5145H
- feedback_rotary.h:
- feedback_simul.c: verwendet einen Timer um eine Bewegung des Rotors zu simulieren
- feedback_simul.h:
- globals.c: beinhaltet globale Variablen (Zustandsvariablen nötig für Regler und Programmablauf)

- globals.h:
- memory.c: Funktionen für das Mitloggen von mehreren Variablen in den externen SRAM
- memory.h:
- motor_control_layer.c: Funktionen zum Steuern des Motor's (Ein-/Ausschalten) und Initialisierung der benötigten Peripherie
- motor_control_layer.h:
- svpwm.c: Funktionen für die Raumzeigermodulation (Berechnung der Duty-Cycles) und Strommessung (Initialisierung der ADCs)
- svpwm.h:
- svpwm_ics.c: Funktionen für die Strommessung bei direkter Messung der Phasenströme
- svpwm_3shunt.c: Funktionen für die Strommessung bei Messung des Stromes durch den low-side MOSFET
- types.h: Definition eigener Datentypen
- timebase.c: Funktionen um eine Zeitbasis für wiederkehrende Aufgaben (z.B. periodische CAN-Nachrichten)
- timebase.h:
- transformation.c: Sinus und Kosinus für die eigene Wertedarstellung bzw. Winkeldarstellung, Transformation zwischen Rotor- und Statorkoordinatensystem, Raumzeiger
- transformation.h:
- trigger.c: Funktionen zur Implementierung verschiedener Trigger: bestimmte Zeit, Tastendruck durch Benutzer, vordefinierter Signalwert
- trigger.h:
- ui.c: User Interface über die serielle Schnittstelle, Anzeigen eines Menüs und aktueller Daten, Auswertung der Benutzereingaben
- ui.h:
- ui_print_fcn.c: Funktionen zum Senden der einzelner Menüs für das User-Interface
- ui_print_fcn.h:
- ui_control_fcn.c: Funktionen die auf Tastendruck (über das User-Interface) vorher definierte Tätigkeiten ausführen
- ui_control_fcn.h:

Literaturverzeichnis

[1] *application note AN3969.* : *application note AN3969* 43

[2] *Datenblatt STM32F405xx.* : *Datenblatt STM32F405xx* 18

[3] *reference manual RM0090.* : *reference manual RM0090* 18

[4] *user manual UM0379.* : *user manual UM0379* 71

[5] *user manual UM1461.* : *user manual UM1461* 66

[6] Daniel, Dumitru ; Mario, Liviu ; Giuclea, Raducu ; Sarca, Aurelian: A novel method for PM synchronous machine rotor position detection. 2007. – Forschungsbericht 36

[7] Gausch ; Hofer ; Schlacher: *Digitale Regelkreise.* 1991 11

[8] Schröder, Dierk: *Leistungselektronische Schaltungen.* Springer-Verlag, 2008. – ISBN 978–3–540–69300–0 12

[9] Schröder, Dierk: *Elektische Antriebe - Grundlagen.* Springer-Verlag, 2009. – ISBN 978–3–642–02989–9 1, 2

[10] Schröder, Dierk: *Elektische Antriebe - Regelung von Antriebssystemen.* Springer-Verlag, 2009. – ISBN 978–3–540–89612–8 12

[11] Shepherd, William ; Zhang, Li: *Power converter circuits.* 2004 29