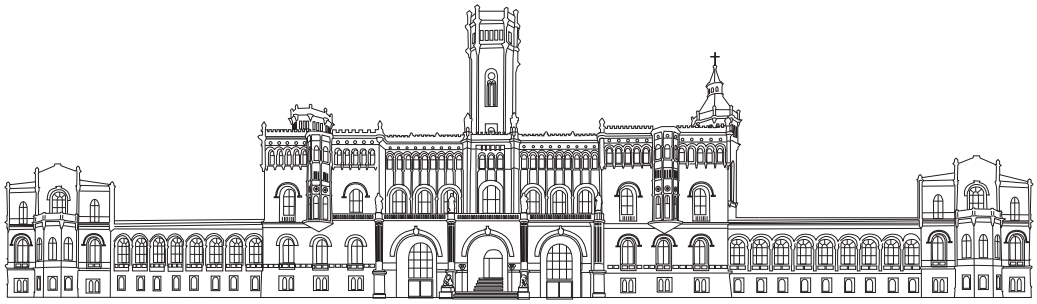

LEIBNIZ UNIVERSITÄT HANNOVER

INSTITUT FÜR ANTRIEBSSYSTEME
UND LEISTUNGSELEKTRONIK

FACHGEBIET LEISTUNGSELEKTRONIK
UND ANTRIEBSREGELUNG

Implementierung eines Frameworks zur Regelung eines PV-Modulwechselrichters unter Berücksichtigung begrenzter Rechnerressourcen

Masterarbeit

Luca Justin Zimmermann
Matrikelnummer: 10016470

Betreuer: M.Sc. Tobias Brinker
Erstprüfer: Prof. Dr. rer. nat. Kurt Schneider
Zweitprüfer: Jun.-Prof. Dr.-Ing. Jens Friebe

Eigenständigkeitserklärung

Luca Justin Zimmermann
Bunnenbergstraße 16 C
30165 Hannover

Matrikelnummer: 10016470
Studienrichtung: Informatik

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Hannover, den 1. Mai 2022

Masterarbeit

Betreuer: M. Sc. Tobias Brinker

Telefon: +49 (0) 511 / 762-2839

E-Mail: Tobias.Brinker@ial.uni-hannover.de

Institut für Antriebssysteme und Leistungselektronik

Fachgebiet Leistungselektronik
und Antriebsregelung
Jun.-Prof. Dr.-Ing. Jens Friebe

Implementierung eines Frameworks zur Regelung eines PV-Modulwechselrichters unter Berücksichtigung begrenzter Rechnerressourcen

Seit drei Jahrzehnten wird in Wissenschaft und Industrie die Idee von direkt ins Solarmodul integrierten Wechselrichtern verfolgt. Mit solchen AC-Solarmodulen können die Installation, modulare Flexibilität, Sicherheit und Effizienz von PV-Kleinanlagen gesteigert werden und durch die Skaleneffekte einer einheitlichen Elektronik werden enorme Kosten- und Qualitätsvorteile ermöglicht. Dies erfordert zunächst eine entsprechende anwendungsorientierte Forschung, welche das Ziel des Verbundforschungsprojektes **Voyager-PV** ist, in dessen Rahmen diese Masterarbeit ausgeschrieben wird.

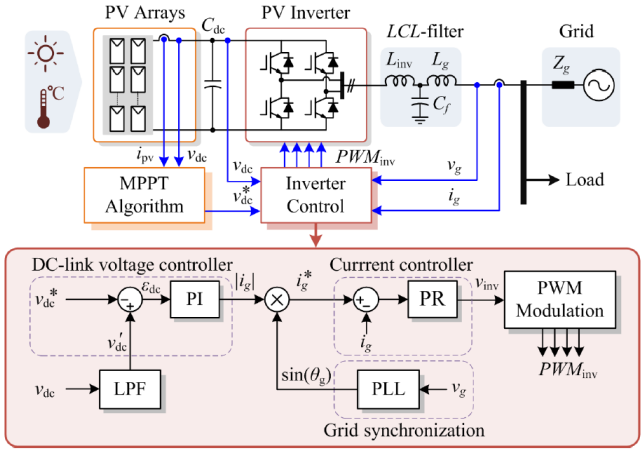

Abbildung 1: Systemkonfiguration und Regelungsstruktur eines einphasigen netzgekoppelten PV-Wechselrichters mit MPP-Tracking nach [1].

Inhalt der Masterarbeit ist die Entwicklung eines Frameworks zur Ausführung der Umrichterfunktionen auf einem Mikrocontroller der STM32 Familie unter Berücksichtigung dessen begrenzter Rechnerressourcen. Im Fokus der Arbeit steht daher neben der Entwicklung des Frameworks die systematische Optimierung hinsichtlich der Ausführungszeiten. Methodisch sollen dabei zunächst die Umrichterfunktionen nach aktuellem Standard implementiert werden und dann eine Bewertung der Ausführungsperformance (bspw. anhand der Zyklenzahl) vorgenommen werden. Anhand dessen sollen kritische Algorithmen identifiziert und zusätzliche Verfahren oder Vereinfachungen mit dem Ziel einer Effizienzsteigerung implementiert werden. Dabei sind Hardware Units (bspw. DMA, etc.) möglichst gewinnbringend zu verwenden. Die Funktion des Frameworks soll anhand von exemplarischen Projekten (siehe bspw. Abbildung 1) an bereits vorhandener Hardware im Labor validiert werden. Überdies ist eine Schnittstelle zur Live-Überwachung von Betriebszuständen (Variablen, Registern) an einem Oszilloskop mittels einer DMA basierten DAC Schnittstelle zu implementieren. Parallel dazu soll eine GUI zur Eingabe von Parametern (bspw. Reglerparametern, Skalierungsfaktoren, Sollwertvorgaben) implementiert werden.

Die Arbeit gliedert sich in folgende Abschnitte:

- Literaturrecherche und Einarbeitung in die Funktionsweise der Ansteuerung von Wechselrichtern und STM32 Mikrocontrollern
- Entwicklung des Frameworks mit den folgenden Funktionalitäten (PLL, Regler, MPPT)
- Methodische Untersuchung der Ausführungszeiten zur Identifikation von kritischen Algorithmen und Implementierung zusätzlicher Verfahren oder Vereinfachungen zur Effizienzsteigerung.
- Implementierung der Schnittstelle zur Live-Überwachung (Oszilloskop und GUI)
- Exemplarische Projekte (Auszug)
    - CCM Spannungsregelung für eine Split Full Bridge Topologie (Valley Switching)
    - Erstellung einer einphasigen dq-Regelung zur einphasigen Netzeinspeisung für verschiedene DC/AC Topologien

[1] A. Sangwongwanich, Y. Yang, D. Sera and F. Blaabjerg, "Interharmonics from grid-connected PV systems: Mechanism and mitigation," *2017 IEEE 3rd International Future Energy Electronics Conference and ECCE Asia (IFEEC 2017 - ECCE Asia)*, 2017, pp. 722-727.

10/2021

Inhaltsverzeichnis

Abbildungsverzeichnis **iii**

Tabellenverzeichnis **iv**

Quelltextverzeichnis **v**

Formelzeichenkonvention **vi**

Abkürzungsverzeichnis **vii**

1 Einleitung **1**

2 Grundlagen **3**
2.1 Wechselrichter . . . 3
2.2 Netzankopplung . . . 4
2.3 CORDIC . . . 4
2.4 Related Work . . . 5

3 Analyse **7**
3.1 Nutzergruppen . . . 7
3.2 Anforderungsanalyse . . . 7
3.2.1 Funktionale Anforderungen . . . 9
3.2.2 Nichtfunktionale Anforderungen . . . 10
3.2.3 Priorisierung und Arbeitspakete . . . 10

4 Hardwareentwicklung **12**
4.1 Komponentenauswahl . . . 12
4.1.1 Hauptprozessor . . . 12
4.1.2 Stromsensoren . . . 13
4.2 Sensorik . . . 14
4.3 Anschluss der ADCs . . . 16
4.4 Aufbau der Platine . . . 17

5 Implementierung **20**
5.1 Auswahl der Entwicklungsumgebung . . . 20
5.2 PLL . . . 21
5.2.1 QSG . . . 22
5.2.2 dq-Transformation . . . 26
5.2.3 SRF-PLL . . . 27
5.3 Stromregelung . . . 27
5.4 Generierung des PWM-Signals . . . 33

5.5 Verringerung der Ausführungszeit 35
5.5.1 CORDIC für Winkelfunktionen 35
5.5.2 Ersatz von Divisionen durch Multiplikationen, Nutzung der FPU, Compileroptionen 38
5.5.3 Vereinfachung des Bandpass-Filters der QSOGI-QSG 40
5.5.4 Einheitsvektor der SRF-PLL ohne Wurzelberechnung 40
5.5.5 Ergebnisse der vorgenommenen Änderungen 41
5.6 Ausgabe von Messwerten per DAC 41
5.7 Kommunikationsschnittstelle 42
5.8 Zeitlicher Ablauf des Programms 43
5.9 GUI 45
5.10 Fehlerbehandlung und Grenzwerte 47
5.11 Dokumentation 49

6 Auswertung **51**
6.1 Erfüllung der Anforderung 51
6.2 Ausblick 52

A Schaltplan und Platinenlayout **53**

Literaturverzeichnis **56**

Abbildungsverzeichnis

2.1 Taylorreihenentwicklung von $y = \sin(x)$ vom Grad 1, 3, 5 und 7 5

4.1 Aufbau des Spannungsteilers für die Stromsensorik 14
4.2 Aufbau des Spannungsteilers für die Spannungssensorik 15
4.3 Messung der Linearität zwischen anliegender und vom Microcontroller erfassten Spannung 16
4.4 Adapterplatine für Nucleo-64 Board mit Messsensorik 18
4.5 Anschluss der Adapterplatine an den Wechselrichter 19

5.1 Übersicht in der STM32CubeIDE 21
5.2 Verlauf von α und β der tDelay-PLL 23
5.3 Aufbau der tDelay-PLL ($f_{\mathrm{PLL}} = 140\,\mathrm{kHz}$) 23
5.4 Verlauf von α und β der DSC-PLL 24
5.5 Aufbau der DSC-PLL ($f_{\mathrm{PLL}} = 140\,\mathrm{kHz}$) 24
5.6 Aufbau der QSOGI-PLL 25
5.7 Rotation vom $\alpha\beta$- in das dq-Koordinatensystem 26
5.8 Zustände beim Aufschalten der Relais 29
5.9 Sprungantwort von 0 zu 500 mA RMS 30
5.10 Stromregelung mit 500 mA Wirk- und 200 mA Blindstrom 31
5.11 Stromregelung 300 mA mit AB-Modulation bei 65 V Wechselspannung und 150 V Zwischenkreisspannung 31
5.12 Spektrumsanalyse des Stroms 32
5.13 Strom Oberwellen bei 300 mA und 65 V ohne LCL-Filter, $f = 50\,\mathrm{Hz}$ 32
5.14 Aufbau des Wechselrichters 33
5.15 Halbbrückenansteuerung der unterschiedlichen Modulationsarten 34
5.16 Vergleich der Ausführungszeit mathematischer Operationen mit CORDIC und *math.h* 37
5.17 Bandpass der QSOGI-PLL 40
5.18 Zeitverlauf der einzelnen Programmteile 44
5.19 Grafische Benutzeroberfläche für Voyager-PV 46
5.20 Ansicht der von *Doxygen* generierten Dokumentation im Webbrowser . . . 50

A.1 Layout der entwickelten Adapterplatine 53
A.2 Schaltplan Seite 1 der entwickelten Adapterplatine 54
A.3 Schaltplan Seite 2 der entwickelten Adapterplatine 55

Tabellenverzeichnis

4.1 Belegung der ADC-Kanäle 16

5.1 Taktzyklen für unterschiedliche Float-Operationen auf dem STM32 36
5.2 Kommandos für die Kommunikationsschnittstelle 43
5.3 Aufteilung der GUI in verschiedene Bereiche 45

Quelltextverzeichnis

5.1 Berechnung ohne explizites Casting und mit Division 39
5.2 Optimierung mittels Casting und Multiplikation 39
5.3 Aufbereitung von Nachrichten für die grafische Benutzeroberfläche 47
5.4 Beispiel einer mit Doxygen annotierten Funktion 49

Formelzeichenkonvention

a	reelle Größe, Momentanwert
A	Effektivwert
$\hat{A}$	Amplitude
$\vec{a}$, $\vec{A}$	vektorielle Größe
$\boldsymbol{\vec{a}}$, $\boldsymbol{\vec{\alpha}}$	Raumzeiger, Raumvektor
$\lvert a \rvert$	Betrag einer reellen Größe

Abkürzungsverzeichnis

ADC	Analog/Digital Converter	12
AKW	Atomkraftwerk	1
API	Application Programming Interface	20
CORDIC	Computational Rotation Digital Computer	5
DAC	Digital/Analog Converter	9
DMA	Direct Memory Access	37
DSC	Delay Signal Cancellation	23
FFT	Fast Fourier Transformation	30
FIFO	First-in-First-out	22
FPGA	Field Programmable Gate Array	5
FPU	Floating Point Unit	38
GUI	grafische Benutzeroberfläche	8
HAL	High Abstraction Layer	20
HRTIM	High Resolution Timer	13
IAL	Institut für Antriebssysteme und Leistungselektronik	1
KISS	Keep It Simple Stupid	15
LED	Leuchtdiode	17
LL	Low Level	20
MPPT	Maximum Power Point Tracking	3
PI	Proportional Integral	6
PLL	Phase Locked Loop	4
PV	Photovoltaik	1
PVA	Photovoltaikanlage	1
PVW	Photovoltaikwechselrichter	1
PWM	Pulsweitenmodulation	13
QSOGI	Quadrature Second-Order Generalized Integrator	24
QSG	Quadrature Signal Generator	22
RMS	Effektivwert	13
RTOS	Echtzeitbetriebssystem	20
SRF	Synchronous Reference Frame	25
tDelay	Transport Delay	22
THD	Total Harmonic Distortion	4

1 Einleitung

Die aktuelle weltpolitische Situation zeigt, dass erneuerbare Energien nicht nur aus ökologischer Sicht sinnvoll sind. Europa importiert etwa 40% des Gasbedarfs aus Russland, Deutschland sogar 50% [1]. Im Rahmen des Ukrainekonflikts wurde die Zertifizierung der neuen Gaspipeline von Russland nach Deutschland ausgesetzt. Die deutsche Gasbranche befürchtet aufgrund dessen eine Versorgungslücke, die schwer beziehungsweise nur mit Mehrkosten geschlossen werden kann [2]. Der Gasexport von Russland nach Polen und Bulgarien ist inzwischen gestoppt worden. Letztere beziehen sogar 90% des Gasbedarfs aus Russland [3]. Eine Verdoppelung des jährlichen Ausbaus von Photovoltaik (PV) in der EU könnte den Gasverbrauch für die Stromerzeugung von 900 TWh um immerhin bis zu 30 TWh senken [4].

Trotz der sinkenden Preise für PV-Module [5] rentieren sich derzeit am Markt verfügbare Photovoltaikanlagen (PVA) meist nur für größere Flächen, da die zugehörigen Photovoltaikwechselrichter (PVW) und Gleichstromverkabelung unter den Modulen einen hohen Anteil der Investitionskosten ausmachen. Für kleinere Dach- oder Fassadenflächen sind die Systeme daher selten wirtschaftlich einsetzbar. Stattdessen werden größere Solarparks auf Freiflächen errichtet. Diese befinden sich jedoch meist in ländlichen Regionen, abseits der Verbrauchenden. Einerseits entstehen durch den Transport des Stroms von der Quelle zu den Abnehmenden Verluste, zudem müssen zum Anschluss gegebenenfalls neue Hochspannungsleitungen verlegt werden. Andererseits bieten Häuser nutzbare Fläche, die bereits bebaut ist, wohingegen die bisherige Freiflächen anstatt für Solarparks auch landwirtschaftlich genutzt werden könnte. Die Energie könnte damit in unmittelbarer Nähe genutzt werden.

Beim Überfall der Ukraine durch Russland am 24. Februar 2022 wurde unter anderem auch zentrale Infrastruktur, genauer Atomkraftwerke (AKWs), angegriffen. Neben den durch Beschuss entstandenen Bränden und beschädigten Hochspannungsleitungen am Kraftwerk Zaporizhzhia ist das AKW derzeit unter Kontrolle russischer Streitkräfte [6]. Durch die dezentrale Stromerzeugung kann die Abhängigkeit von der Infrastruktur verringert werden.

Im Rahmen des Verbundforschungsvorhabens Voyager-PV am Institut für Antriebssysteme und Leistungselektronik (IAL) werden daher die Möglichkeiten für integrierte Modulwechselrichter erforscht. Bei diesen ist der Wechselrichter entsprechend in das PV-Modul integriert. Durch eine einheitliche Elektronik und den Ersatz von teuren Gleichstromverbindungen unter den Modulen durch günstige Wechselstromanschlüsse sollen Vorteile bei Kosten und Installationsaufwand entstehen. Dank der Integration würden sich kleinere PVA dann ebenso rentieren und zum Ausbau der erneuerbaren Energien sowie zur Dezentralisierung der Strominfrastruktur beitragen.

Die wirtschaftlichen Aspekte sind jedoch nur peripher Bestandteil dieser Arbeit. Der Microcontroller stellt einen wesentlichen Teil der Kosten eines Modulwechselrichters dar. Neben

der Auswahl eines dementsprechend Kostengünstigen, aber für die Aufgabe ausreichend geeigneten *STM32*-Microcontrollers liegt das Hauptaugenmerk hier auf der effizienten Implementierung der für die Ansteuerung notwendigen Algorithmen.

In Kapitel 2 werden zunächst die Grundlagen zur eingesetzten Technologie erläutert und die Arbeit in den Kontext verwandter Arbeiten eingeordnet. Darauffolgend werden die Anforderungen für die erfolgreiche Umsetzung des Projekts analysiert, um anschließend zu bearbeitende Arbeitspakete zu erstellen. Durch den interdisziplinären Aspekt dieser Arbeit zwischen Informatik und Elektrotechnik ist im Rahmen dieser Arbeit eine Platine entwickelt worden. Die dafür notwendigen Schaltungen und die Auswahl des Microcontrollers, auf dem die Software implementiert wird, ist in Kapitel 4 beschrieben. In Kapitel 5 werden die mathematischen Konzepte der Netzsynchronisierung und Stromregelung erläutert, wie auch die eigentliche softwaretechnische Implementierung. Abschließend wird die entwickelte Lösung bewertet und das Erreichen der Anforderungen überprüft.

2 Grundlagen

PV-Module wandeln die Strahlungsenergie des Sonnenlichts in eine Gleichspannung. Aufgrund der unterschiedlichen Bestrahlungswinkel und -intensität, wetter- oder umgebungsbedingte Verschattung und Temperatur über den Tag variiert die erzeugte Spannung und mögliche Leistungsabgabe stark. Die im Rahmen dieser Arbeit behandelten PVA sind für den Netzanschluss vorgesehen. Aus der schwankenden Gleichspannung (DC) muss daher eine Wechselspannung (AC) für das Stromnetz mithilfe eines Wechselrichters generiert und zum Phasenwinkel synchronisiert werden. In den folgenden Abschnitten werden Konzepte unterschiedlicher Wechselrichter und die Netzankopplung kurz erläutert und die innerhalb des Verbundforschungsprojekts zu entwickelnde Lösung zu bereits existierenden Systemen verglichen.

2.1 Wechselrichter

Yaosuo Xue et al. vergleichen in [7] verschiedene Topologien für die Generierung von Wechselstrom aus Wind- und Solaranlagen mit kleinen Stromerzeugungssystemen. Dabei werden diese in Hinblick auf Kosten, Kompaktheit respektive Gewicht, Wirkungsgrad und Bandbreite der Eingangsspannung bewertet.

Wie eingangs bereits erwähnt, variiert die Klemmenspannung von PV-Modulen. Multi-Stage-Wechselrichter generieren zum Betrieb des eigentlichen Wechselrichters eine stabile Zwischenkreisspannung durch *[Buck-]Boost*-Schaltungen. Dabei wird aus der anliegenden, variierenden Gleichspannung die zur Erzeugung der Wechselspannung benötigte stabile Gleichspannung generiert (DC-DC-AC oder DC-AC-DC-AC). Die Zwischenstufe wird häufig auch für das *Maximum Power Point Tracking (MPPT)* genutzt, ein Algorithmus, um den maximalen Leistungspunkt des PV-Moduls zu ermitteln. Eine alternative Multi-Stage-Variante wandelt zunächst die Gleichspannung in eine Wechselspannung und passt diese in einer darauffolgenden Stufe auf das Netzspannungspotential an (DC-AC-AC). Diese bietet den Vorteil, dass auf große Kondensatoren zur Stabilisierung der DC-Zwischenkreisspannung verzichtet werden kann.

Den mehrstufigen Wechselrichtern stehen die Single-Stage-Varianten gegenüber. Diese generieren aus der Gleichspannung der PV-Module direkt die für den Netzanschluss benötigte Wechselspannung und benötigen durch den einfachen Aufbau aus nur einer Stufe weniger Komponenten. Gegenüber den Multi-Stage-Wechselrichtern sind diese somit Kostengünstiger. Durch den Aufbau ist allerdings die Bandbreite für das MPPT stark beschränkt. Der Wirkungsgrad ist durch den Verzicht auf die anderen Stufen hingegen sehr hoch.

Einige der Topologien erlauben auch den Blindleistungsbetrieb, während die anderen nur Wirkleistung erzeugen können. Regulatorisch müssen netzgekoppelte Wechselrichter sowohl Wirk- als auch Blindleistung bereitstellen [8]. Mit Voyager-PV soll einerseits durch

den mehrstufigen DC-DC-AC-Aufbau eine flexible, andererseits durch den Einsatz von neuen Technologien kompakte Lösung erforscht werden. Einheitliche Elektronik sollen die Kosten minimieren und die Integration in das PV-Modul die Effizienz steigern.

2.2 Netzankopplung

Die Erzeugung der Netzspannung ist mit entsprechender Hardware vergleichsweise einfach. Ausgehend von einer DC-DC-AC-Topologie muss der Prozessor für die Leistungselektronik ein sinusförmiges Signal in Abhängigkeit zur Zwischenkreisspannung modulieren. Auf die Ansteuerung wird in Kapitel 5.4 genauer eingegangen. Wie im nächsten Abschnitt erwähnt, ist dieser gesteuerte Betrieb auch mit einem sehr einfachen Microcontroller realisierbar.

Für die Einspeisung in das Stromnetz muss jedoch die erzeugte Wechselspannung in Phase mit der des Netzes sein. Dazu muss der Phasenwinkel mittels eines *Phase Locked Loop (PLL)* ermittelt und synchronisiert werden. Sind inverter- und netzseitige Wechselspannung phasen- und amplitudengleich, so findet kein Energieaustausch zwischen Wechselrichter und Netz statt. Der für einen Stromfluss notwendige Potentialunterschied wird nachfolgend von einem Stromregler gestellt. Der rechnerische Aufwand im geregelten Betrieb ist entsprechend höher, als im Gesteuerten, welcher jedoch nicht zur Netzankopplung geeignet ist.

Wie bereits im vorangehenden Abschnitt erwähnt, sind für die Netzankopplung Normen einzuhalten. Dazu zählt die zuvor genannte Bereitstellung von Blindleistung sowie eine geringe *Total Harmonic Distortion (THD)*, also Verunreinigung des Signals durch Oberwellen. Die entsprechenden Normen werden in Abschnitt 3.2 und 5.3 genauer erläutert.

2.3 CORDIC

Für die zur Netzankopplung benötigte PLL ist die Berechnung von Winkelfunktionen essentiell. Diese lässt sich in Software auf verschiedene Weisen realisieren. Taylorreihen bieten eine Approximation einer Gleichung durch eine Potenzreihe. In Gleichung 2.1 ist die entsprechende Taylorreihe gezeigt. Die Genauigkeit der Annäherung wird über deren Grad n bestimmt. Mit steigendem Grad wächst der resultierende Term exponentiell, wodurch die Berechnungszeit im gleichen Maß steigt ($O(n^2)$). Wie in Abbildung 2.1 zu sehen, wird für den Bereich $[-\pi, \pi]$ erst mit dem 7. Grad eine entsprechend geringe Abweichung von der Sinusfunktion erreicht.

$$\sin(x) \quad = \sum_{k=0}^{n} (-1)^k \frac{x^{2k+1}}{(2k+1)!} \tag{2.1}$$

Rechnerisch weniger aufwändig ist die Realisierung mithilfe einer *Lookup Tabelle*. Dabei werden vorberechnete Werte für die Funktion in einer Tabelle abgelegt. Anhand des Winkels wird der nächstgelegene Eintrag gesucht und das Ergebnis ausgegeben. Die Genauigkeit hängt daher von der Intervallgröße ab. Je kleiner die Intervalle, desto mehr Einträge muss die Tabelle fassen, der Speicherbedarf erhöht sich bei einer Halbierung der Intervalle entsprechend exponentiell. Mittels Interpolation kann die Auflösung bedingt erhöht werden. Aufgrund der einfachen Suchfunktion in der Tabelle ist die Berechnungszeit statisch ($O(1)$),

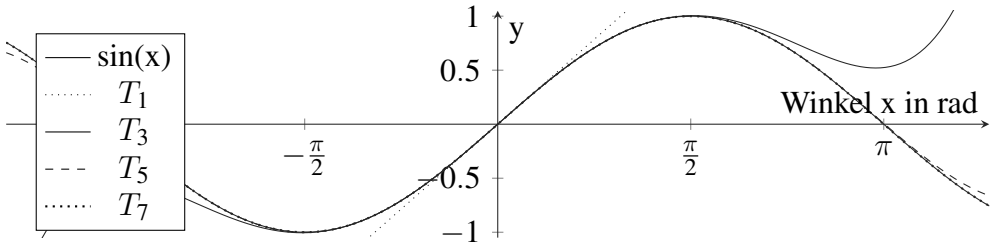

Abbildung 2.1: Taylorreihenentwicklung von $y = \sin(x)$ vom Grad 1, 3, 5 und 7

die Anforderungen an den Speicher steigen jedoch wie beschrieben. Dadurch, dass Kosinus und Sinus nur 90° phasenverschoben sind, die Kurve punktsymmetrisch und eine Halbwelle achsensymmetrisch ist, muss die Tabelle jedoch nur für eine Viertelperiode einer der Winkelfunktionen ausgelegt werden.

Der *Computational Rotation Digital Computer (CORDIC)* Algorithmus ist eine in Hardware sehr einfach mit Shift-Operationen und Additionen zu implementierende Lösung. Ähnlich wie die Genauigkeit der Taylorreihe vom Grad der Reihenentwicklung abhängig ist, ist der CORDIC abhängig von den durchlaufenen Zyklen. Jeder Zyklus führt die gleiche Berechnung aus, daher ist die Berechnungszeit im Gegensatz zur Taylorreihe nur linear davon abhängig ($O(n)$). Darüber hinaus kann der Algorithmus auch für die Berechnung von Exponentialfunktionen oder Quadratwurzeln umkonfiguriert werden. Auf die genaue Funktionsweise des CORDIC wird an dieser Stelle jedoch nicht weiter eingegangen.

2.4 Related Work

Im Rahmen dieser Arbeit wird das Softwareframework zur Ansteuerung des Wechselrichters und insbesondere der Netzsynchronisierung und Stromregelung entwickelt und für die gewählte Architektur optimiert. Die genutzten Algorithmen für die Synchronisierung und Regelung entsprechen dem Stand der Technik im Bereich netzeinspeisender Wechselrichter.

Lamo et al. stellen in [9] auf Basis eines *Field Programmable Gate Array (FPGA)* verschiedene PLLs hinsichtlich der Berechnungszeiten, Toleranz gegenüber Frequenz-, Spannungssprüngen und Verunreinigungen des Signals durch Oberschwingungen gegenüber. Auf FPGAs werden Hardwareblöcke entsprechend der Vorgaben der Entwickelnden konfiguriert. Dadurch ist im Gegensatz zu Microcontrollern, die instruktionsbasiert Anweisungen hintereinander ausführen, eine begrenzte Parallelisierung von Berechnung und Ansteuerung von Signalen möglich. Die genutzte Hardware unterstützt auch den vorig beschriebenen CORDIC Algorithmus nativ, welcher für die Berechnung der Winkelfunktionen auf Basis des Phasenwinkels genutzt wird. Die Taktfrequenzen eines FPGA sind in der Regel geringer als bei vergleichbaren Microcontrollern. Aufgrund der Möglichkeit, die Ausführung zu parallelisieren, können anwendungsspezifisch dennoch geringere Berechnungszeiten erreicht werden.

Bei Controllern beeinflusst die Größe des integrierten Flashspeichers, in dem die Instruktionen und Nutzdaten gespeichert sind, vergleichsweise wenig den Preis des Bausteins. Die Anzahl der zur Verfügung stehenden konfigurierbaren Gatter von FPGAs wirken sich hin-

gegen superlinear auf den Preis aus. Ein für die in dem Paper genannte Anzahl an Hardwareeinheiten ausreichendes FPGA kostet ohne den benötigten Konfigurationsspeicher das vierfache des für diese Arbeit genutzten Microcontrollers [10]. Die Implementierung umfasst jedoch nur die für die Regelung notwendigen Algorithmen und keine Ansteuerung von Hardware oder eine Steuerungsmöglichkeit über eine Schnittstelle. Für die Erweiterung des Programms sind entsprechend weitere Logikzellen notwendig, wodurch ein größeres FPGA benötigt wird, das mit höheren Kosten verbunden ist.

Die Implementierung einer Wechselrichteransteuerung wird in [11] auf einem preisgünstigen 8 Bit Microcontroller umgesetzt. Die Lösung umfasst jedoch nur den gesteuerten Betrieb, also, wie im vorigen Abschnitt beschrieben, die Erzeugung einer Wechselspannung, deren Phasenwinkel nicht mit dem Stromnetz synchronisiert ist. Für den Anschluss an das Stromnetz ist jedoch eine Regelung notwendig, für die die rechnerischen Kapazitäten und zur Verfügung stehenden Eingänge des gewählten Microcontrollers nicht ausreichend sind. Darüberhinaus beträgt die THD des Stroms von teilweise über 9%, welche, wie später in Abschnitt 3.2 und 5.3 erklärt, nicht zum Einhalten der Norm genügt. Der Ansatz ist daher nur für Insellösungen, also eine vom Stromnetz autarke Versorgung, einsetzbar.

Mit einem Microcontroller der auch für diese Arbeit genutzten *STM32*-Serie wird in [12] das Verhalten der PLL bei Spannungssprüngen untersucht. Dafür wird eine *acSOGI* PLL zusammen mit der *Park-Transformation* und einem *Proportional Integral (PI)*-Regler genutzt. Das Konzept wird in der vorliegenden Arbeit in Abschnitt 5.2 weiter erläutert. Die Untersuchung zeigt, dass der Microcontroller mit der Implementierten PLL auch bei Spannungs- oder Phasensprüngen die anliegende Netzspannung tracken kann. Eine Ansteuerung eines Wechselrichters in Kombination mit einer Regelung auf Basis der PLL erfolgt jedoch nicht. Ob die für einen Wechselrichter notwendige Software im Gesamten auf dem Controller implementiert werden kann wird daher nicht bewertet.

3 Analyse

Die Aufgabenstellung enthält grundlegende Anforderungen an das Software-Framework. In diesem Kapitel werden die verschiedenen Anforderungen an das System genauer analysiert, damit das Framework mit der bereits existierenden Wechselrichterplatine kompatibel ist und die gewünschten Ergebnisse erzielt. Dazu wird im Folgenden zunächst einerseits ermittelt, für welche Nutzenden das System ausgelegt werden soll, und andererseits, welche Kenntnisse bei diesen vorausgesetzt werden können. In der anschließenden Anforderungsanalyse werden die Anforderungen an das Framework selbst konkretisiert und nachfolgend priorisiert und in Arbeitspakete zusammengefasst.

3.1 Nutzergruppen

Die entwickelte Lösung soll langfristig in kommerziellen Photovoltaikmodulen zum Einsatz kommen. Für die Anlagenbetreibende sind keine Steuerungsmöglichkeiten vorgesehen, sondern diese den netzbetreibenden Unternehmen über entsprechende Kommunikationsdienste vorbehalten. Im Verbund von mehreren Solaranlagen soll so eine netzdienliche Stromeinspeisung ermöglicht werden. Dies umfasst die gesteuerte Einspeisung nicht nur von Wirkstrom sondern auch Blindstrom. Für die Anlagenbetreibenden sollen Messwerte und Statistiken in Echtzeit zur Verfügung gestellt werden.

Bis zur Entwicklung eines serienreifen Produkts müssen jedoch noch Messungen und Anpassungen an der Schaltung vorgenommen werden. Somit ist die Software sowohl für Forschungszwecke, als auch modular genug auszulegen, sodass nur wenige Anpassungen für den Erprobungsbetrieb des Modulwechselrichters im Rahmen des Verbundforschungsvorhabens *Voyager-PV* notwendig sind. Im Rahmen der weiteren Untersuchungen werden hauptsächlich Projektbearbeitende mit lediglich grundlegenden Informatikkenntnissen das System benutzen.

3.2 Anforderungsanalyse

Die Projektbearbeitenden nutzen vornehmlich *MathWorks MATLAB* respektive *Simulink*, um Schaltungen zu simulieren und aus den Modellen Quellcode zu generieren. Für die einfache Weiterentwicklung und Anpassung ist daher eine hohe Kongruenz zwischen dem für den Wechselrichter vorliegenden *Simulink*-Modell und dem Quelltext für den Microcontroller sinnvoll (NFA4). Ebenso existieren Prozesse zur Programmierung von *STM32* Microcontrollern mithilfe von *Simulink*. Im weiteren Verlauf wird das Projekt auch auf Basis des damit generierten Codes erweitert. *Simulink* bietet den C respektive C++ Export für *STM32* Microcontroller an, sodass auch die Software in einer der Sprachen geschrieben werden soll (NFA1). Zur weiteren Nutzung dieser Toolchain und aus den Anforderungen des IAL ergibt

sich damit auch der Einsatz eines *STM32* Microcontrollers (NFA3).

Diese Arbeit behandelt hauptsächlich die Funktionsweise, Validierung, mathematischen Prinzipien und Methodiken für die Entwicklung. Damit nach Abschluss der Arbeit andere Personen ohne Rückfragen den Quelltext anpassen können, bedarf es einer ausreichenden Dokumentation dessen. Die Mindestanforderung für die Dokumentation sind ausführliche Kommentare im Quelltext (NFA5).

Nach IEC/EN 61727 sind Schwingungen im Netzstrom bis zur 40. Oberwelle – bei einer nominalen Netzfrequenz von 50 Hz entspricht das 2 kHz – auf unter 5% zu begrenzen [13]. Innerhalb der PLL wird der für die Strom-Sollwerte benötigte Winkel berechnet. Bei einer Ausführungsfrequenz von unter 2 kHz ist eine Treppenbildung die Folge, welche wiederum Oberschwingungen im Netz generiert. Die vom Wechselrichter zu generierende Spannung wird durch die Stromregelung berechnet, die entsprechend ebenso mindestens so schnell wie die PLL ausgeführt werden muss. Dies impliziert für den Wechselrichter eine Stellung des Stroms und der Spannung mit mindestens 2 kHz (FA2, FA3). Der Wechselrichter ist nach Vorgabe mit einem 140 kHz PWM-Signal anzusteuern. Eine entsprechend mit gleicher Frequenz ausgeführte PLL und Stromregelung ist wünschenswert. Neben den beiden Berechnungen ist auch noch genug Rechenzeit für eine Kommunikationsschnittstelle und MPPT bereitzustellen. Das MPPT soll mit mindestens 1 Hz ausführbar sein (FA10).

Das System muss nach der Norm VDE-AR-N 4105 sowohl Wirk- als auch Blindleistung zur Verfügung stellen [8]. Das Verhältnis wird durch die netzbetreibenden Unternehmen mitgeteilt. Eine feste Vorgabe im Programm ist daher nicht möglich. Es wird somit eine Schnittstelle benötigt, die Werte dynamisch zu konfigurieren. Ebenso sollen Messwerte des Systems wie eingangs erwähnt für die Anlagenbetreibenden abrufbar sein. Für den vernetzten Betrieb soll im späteren Verlauf zur Kommunikation ein *WirePas*-Modul mit SPI-Anbindung genutzt werden. Die Einbindung des Moduls erfolgt erst im Anschluss an diese Arbeit, daher muss für den Erprobungsbetrieb im Labor eine alternative Kommunikationsschnittstelle eingerichtet werden, deren konkrete hardwareseitige Anbindung jedoch nicht seitens des IAL spezifiziert wird. Die vom Microcontroller erfassten Messwerte sind in Echtzeit auf einen Computer zu übertragen und dort anzuzeigen (FA6), wie auch Soll- und Regelwerte auf Basis der Eingabe der Benutzenden von diesem wieder zurück an den Controller zu übermitteln (FA5). Um eine Fehlbedienung zu verhindern, ist die Schnittstelle so zu beschränken, dass nur plausible Werte akzeptiert werden (FA8).

Zur Ansteuerung der Schnittstelle wird darüber hinaus eine grafische Benutzeroberfläche (GUI) gewünscht (FA7), die unabhängig von dem genutzten Betriebssystem die Anzeige und Eingabe ermöglicht (FA9). Eine GUI vereinfacht für die Bedienenden die Parametrisierung der Steuerung, da keine Befehle manuell eingegeben werden müssen, sondern mittels Schaltfläche ausgelöst werden. Aufgrund des Personenkreises an Projektbearbeitenden mit mitunter wenig Informatikkenntnissen ist die Oberfläche einfach zu gestalten (NFA2).

Die Datenrate der Kommunikationsschnittstelle reicht zur Übertragung von sich langsam ändernden Werten aus. Für die Überprüfung der korrekten Funktionsfähigkeit ist diese aber nicht ausreichend, unter anderem, weil die Werte durch den Microcontroller selbst generiert werden, aber nicht mit den äußeren Einflüssen des Systems (Spannungs-/Strommessung)

direkt verglichen werden können. Die *STM32* bieten *Digital/Analog Converter (DACs)*, womit interne Werte zu einem analogen Ausgangssignal gewandelt werden. Das ausgegebene Signal des Microcontrollers kann zusammen mit der Spannung und dem Strom des Wechselrichters mit einem Oszilloskop gemessen und verglichen werden. Besonders für die Entwicklung und Verifizierung ist daher die Ausgabe über einen oder mehrere DACs notwendig (FA12).

Bei der Erfassung der Messwerte mit dem Microcontroller ist hingegen wichtig, dass das Sampling von Signalen, die miteinander verrechnet werden müssen, zum gleichen Zeitpunkt erfolgt. Wenn beispielsweise die Referenzspannung für einen Sensor mit Versatz zu dem eigentlichen Sensorausgang gemessen wird, so besteht keine direkte Korrelation mehr zwischen den Werten. Für eine hohe Korrelation sind daher zusammengehörige Werte gleichzeitig im Rahmen der Möglichkeiten des Microcontrollers zu erfassen (FA13). Die Platine des Wechselrichters umfasst auch die bisher genutzte Stromsensorik. In der vorigen Arbeit haben sich der Aufbau und die dabei genutzten Sensoren jedoch für die Anwendung als unzureichend herausgestellt [14]. Für den Einsatz wird daher eine zusätzliche Platine für die anwendungsspezifische Messtechnik benötigt (FA1).

Trotz der Prüfung der Soll- und Regelparameter über die Schnittstelle kann unter anderem eine falsche Messung des Stroms etwa aufgrund falsch angeschlossener Kabel auftreten. Ebenso sind Schwankungen des angeschlossenen Stromnetzes außerhalb des definierten Arbeitsbereichs nicht ausgeschlossen. Damit eine Fehlfunktion durch Überschreitung von Grenzwerten oder fehlerhafte Konfiguration nicht zur Zerstörung der Elektronik führt, ist eine Sicherheitsfunktion zu implementieren, die das Stromnetz von dem Wechselrichter trennt und diesen in einen sicheren Zustand überführt (FA11). Die Fehlerbeschreibung ist den Benutzenden anzuzeigen (FA6).

Im Rahmen des Verbundforschungsvorhabens Voyager-PV sind unterschiedliche Modulationsarten hinsichtlich der Verlustleistung und elektromagnetischer Verträglichkeit zu untersuchen. Die Software soll diese entsprechend unterstützen und durch das Ändern eines Parameters im Quelltext die jeweilige Modulation ausführen (FA4).

Aus der Aufgabenstellung und weiteren Gesprächen ergeben sich somit die im folgenden aufgelisteten Anforderungen.

3.2.1 Funktionale Anforderungen

FA1 Erstellung einer Platine mit anwendungsspezifischer Messsensorik

FA2 Berechnung der PLL mit mindestens 2 kHz

FA3 Ausführung der Stromregelung mit mindestens der Frequenz der PLL

FA4 Änderung der Modulationsart (A, B, AB, BA) im Programm

FA5 Einstellen von Sollwerten und Reglerparametern per Kommunikationsschnittstelle

FA6 Ausgabe möglicher Fehler und Messwerte über die Schnittstelle

FA7 Bedienung der Schnittstelle über eine GUI

FA8 Schnittstelle akzeptiert nur plausible Werte

FA9 Ausführbarkeit der GUI auf allen gängigen Betriebssystemen

FA10 MPP-Tracking mit 1 Hz ausführbar

FA11 Selbstständige Trennung vom Stromnetz im Fehlerfall

FA12 Ausgabe von ausgewählten Variablen per DAC

FA13 Gleichzeitiges Sampling von zusammengehörigen Analogwerten

3.2.2 Nichtfunktionale Anforderungen

NFA1 Implementierung auf dem Microcontroller in Standard-C/C++

NFA2 Zur Bedienung der GUI werden keine Informatik-Kenntnisse vorausgesetzt

NFA3 Nutzung eines STM32 Microcontrollers

NFA4 Starke Kongruenz von Quellcode und Simulink-Modell

NFA5 Dokumentation des Projekts durch Kommentare im Quellcode und diese Arbeit

3.2.3 Priorisierung und Arbeitspakete

Die vorherig aufgelisteten Anforderungen sind teilweise stark voneinander abhängig. Nachfolgend werden daher zusammengehörige Arbeitspakete gebildet und unabhängig oder mit geringer Priorität realisierbare Aufgaben separat aufgelistet.

Wahl der Plattform

Bevor mit der Implementierung begonnen werden kann, sind die kritischen Berechnungen der Regelung zu identifizieren. Auf Basis dieser kann eine Bewertung erfolgen, welcher Microcontroller geeignet ist. Dazu passend wird eine Entwicklungsumgebung ausgesucht, die den Controller und gegebenenfalls dessen Besonderheiten unterstützt.

Entwicklung einer Adapterplatine

Nach der Festlegung auf einen Microcontroller ist die benötigte Hardware zu bestellen. Die Zeit bis zur Lieferung wird für die Entwicklung der Adapterplatine genutzt.

Test des Microcontrollers, Validierung der Platine

Damit die zu implementierende PLL und Stromregelung überprüfbar sind, müssen grundlegende Funktionalitäten, wie die korrekte Ausgabe von Werten über die DACs zuvor sichergestellt werden. Die DACs werden ebenso wie das Sampling der Analogeingänge für die Validierung und Charakterisierung der Sensorik auf der zu entwickelnden Platine benötigt. Für Testzwecke ist der Wechselrichter gesteuert zu betreiben.

Implementierung der Regelung

Im Anschluss erfolgt die Implementierung der Funktionen, um den Wechselrichter auch geregelt betreiben zu können. Damit einhergehend sind Maßnahmen zur Vereinfachung der Algorithmen und Optimierung auf die Prozessorarchitektur.

Kommunikationsschnittstelle

Die Kommunikationsschnittstelle kann unabhängig von den anderen Arbeitspaketen implementiert werden. Allerdings wird diese erst benötigt, nachdem die Regelung funktionsfähig ist. Daher wird eine Realisierung im Anschluss an die Fertigstellung der Regelung angestrebt.

Nebenläufige Aufgaben

Die Robustheit des Systems, also die ordnungsgemäße Funktionalität des Systems auch bei fehlerhaften Eingaben oder Messwerten, ist dauerhaft sicherzustellen. Eine Zerstörung der Hardware durch eine Fehlfunktion führt einerseits zu unvorhergesehenen Kosten und andererseits zu Verzögerungen des Projekts. Die Dokumentation ist während der Bearbeitung nicht mit hoher Priorität durchzuführen, muss allerdings bis zum Abschluss vorliegen und vereinfacht die Nutzung, Wartbarkeit und Pflege der eigenen Software.

Arbeitspunkte mit geringer Priorität

Der Wechselrichter wird hauptsächlich in einer Modulationsart betrieben und die Funktionsfähigkeit lässt sich damit verifizieren. Ebenso lassen sich die verschiedenen Modulationsarten im bereits gesteuerten Betrieb testen und benötigen daher nicht die Implementierung von PLL und Stromregelung. Die Anpassungen für die Modulationsarten können daher unabhängig vom Projektfortschritt realisiert werden.

Eine grafische Benutzeroberfläche vereinfacht die Bedienung der Kommunikationsschnittstelle, diese ist jedoch abhängig vom Aufbau auch mit bereits existierenden Programmen nutzbar. Dieser Aufgabe ist eine entsprechend geringe Priorität zugeordnet.

4 Hardwareentwicklung

In der vorherigen Arbeit zu dem Projekt wurde ein *STM32F303 Nucleo-64* Microcontroller-Board genutzt. Aufgrund von inkompatiblen Pinbelegungen wurde die Verbindung zur Wechselrichterplatine über Kabel anstelle des Stecksockels hergestellt. Dieser Aufbau ist aufgrund der elektromagnetischen Störung, die in die Signalleitungen eingekoppelt werden, bei höheren Spannungen nicht praktisch nutzbar [14]. Zudem ist die Strom- und Spannungssensorik weit vom Microcontroller entfernt, wodurch bereits auf der Wechselrichterplatine störende Beeinflussungen auftreten können. Daher soll ein separater Adapter für den Microcontroller entwickelt werden, der einerseits die richtige Zuordnung der Pins sicherstellt und andererseits die Sensorik nah genug an die Eingänge verlegt, sodass der Einfluss der Leistungselektronik auf die Messwerte verringert wird.

4.1 Komponentenauswahl

In der Arbeit von Teichert wird ebenso erwähnt, dass die Leistungsfähigkeit des Systems aufgrund der Möglichkeiten des verwendeten Microcontrollers und Messbereichs des Stromsensors begrenzt wird [14]. Innerhalb der *STM32*-Familie soll daher ein für die Aufgabe besser geeigneter Controller gewählt werden, der jedoch etwa im gleichen Preisbereich liegt. Ebenso wird folgend die Auswahl des Stromsensors auf Basis der Verfügbarkeit noch einmal evaluiert.

4.1.1 Hauptprozessor

STMicroelectronic bietet auf der Produktseite zur *STM32*-Familie eine Übersicht über die einzelnen Prozessoren und Vorschläge für unterschiedliche Anwendungsszenarien. Die bisher genutzte *STM32F3*-Serie listet der Hersteller unter Mixed-Signal-optimierten, also sowohl für die Verarbeitung von digitalen, als auch analogen Signalen ausgelegten Mainstreamprozessoren. Daneben werden die Prozessoren noch in *High-Performance*, vorrangig für die Ansteuerung von Displays, *Ultra-Low-Power* für Batterieanwendungen und *Wireless* unterteilt. Als Auswahl für mögliche Anwendungsszenarien wird unter anderem auch *STM32 Digital Power Ecosystem* aufgeführt. Auf der Unterseite beschreibt der Hersteller dieses weiter mit den Bereichen Beleuchtung, Schweißen und Wechselrichtern für Solaranlagen [15], also dem Anwendungsfall dieses Projekts.

Neben dem bisher genutzten *STM32F3* steht dort *STM32G4* und *STM32H7* zur Auswahl. Die *F3*-Serie ist auf eine maximale Taktfrequenz von 72 MHz beschränkt, während die Kosten der *H7*-Serie doppelt so hoch sind, dafür jedoch auch mit bis zu 550 MHz Taktfrequenz und bis zu 3 16 Bit statt mit nur 12 Bit Analog/Digital Converter (ADCs) ausgestattet ist. Die *G4*-Prozessoren bieten bei etwa gleichem Kostenaufwand wie bei dem vorherig genutzten Prozessor für diese Anwendung vorteilhafte Funktionen und eine Frequenz von bis zu 170 MHz. Alle Microcontroller der *G4*-Baureihe (*4x1*, *4x3* und *4x4*) bieten einen

CORDIC, welcher unter anderem Winkelfunktionen effizient und numerisch stabil in Hardware berechnet (vgl. Abschnitt 2.3). In der vorhergehenden Arbeit hat sich die Berechnung von Sinus und Kosinus als ein rechenintensiver Teil herausgestellt [14], welcher mit dem CORDIC vereinfacht werden kann. Ab dem *G4x4* stehen dazu noch *High Resolution Timer (HRTIM)* zur Verfügung, welche dem Namen entsprechend eine hohe Auflösung für Pulsweitenmodulation (PWM) erlauben. Die beiden Modelle *G474* und *G484* unterscheiden sich dadurch, dass der G484 noch Kryptografie-Funktionen unterstützt, die aber für den Wechselrichter nicht notwendig sind. Innerhalb der *STM32G474* wird nur noch über die Bauform und Speicher unterschieden. Der *STM32G474RE* wird von *STMicroelectronic* als Adapterplatine in dem bereits genutzten *Nucleo-64*-Format angeboten. Auf der Platine sind bereits alle für den Betrieb benötigten Bauteile, wie Programmieradapter, Spannungsregler, Quartz und weiteres vorhanden, sodass der Entwicklungsaufwand für den eigenen Adapter verringert wird. Ebenso lässt sich der *Nucleo* zwischen verschiedenen Revisionen einfach umstecken.

Aufgrund des etwa gleichen Stückpreises im Vergleich mit dem vorherig eingesetzten Prozessor in Kombination mit für die Anwendung in Hardware realisierten Algorithmen und einer hohen Auflösung für die Ansteuerung des Wechselrichters, wird der *STM32G474RE* gewählt.

4.1.2 Stromsensoren

Auf der Wechselrichter-Platine ist ein *ACS724LLCTR-20AB-T* Stromsensor verbaut. Der Sensor wandelt Ströme von -20 bis 20 A zu einer Spannung im Bereich von 0 bis 5 V [16]. Insgesamt ist der Wechselrichter nur auf eine Leistung von 600 W und damit weniger als 2,6 A Effektivwert (RMS) respektive 3,7 A Spitzenstrom ausgelegt. Von dem verfügbaren Bereich wird somit nur knapp 20% ausgenutzt. Die ADCs können das Signal nur mit einer begrenzten Anzahl an Schritten auflösen, daher ist die effektive Auflösung für den benötigten Bereich mit $128\,{}^{\mathrm{mA}}/_{\mathrm{LSB}}$ sehr grob. Zudem arbeitet der Controller im Spannungsbereich 0 bis 3,3 V, die Spannung muss entsprechend durch einen Teiler angepasst werden, wodurch die Auflösung sich je nach Teilerverhältnis weiter verschlechtert.

Der Hersteller *Allegro Microsystems* bietet mit dem *ACS725LLCTR-05AB-T* den Stromsensor ebenso in einer Variante für 3,3 V Versorgungsspannung und einen Bereich von ± 5 A an [17]. Aufgrund der anhaltenden Chip-Krise [18] ist der Sensor allerdings nicht verfügbar. Der für 5 V ausgelegte *ACS724LLCTR-05AB-T* ist ebenso nicht als einzelner Chip verfügbar, jedoch wird dieser auf Adaptern angeboten, sodass der Sensor von diesen entlötet und auf die Platine gelötet wird. Die Ausgabe der Sensoren ist abhängig von der Versorgungsspannung, diese muss, wie in der Arbeit von Teichert [14] vorgeschlagen, mit erfasst werden. Die den Sensoren und Versorgungsspannung nachgelagerten Spannungsteiler müssen so ausgelegt werden, dass auch Schwankungen der Versorgung über 5 V hinaus mit erfasst werden können, die Spannung aber nicht zu stark und damit zu Lasten der Auflösung reduziert wird.

Auch wenn der *ACS724LLCTR-05AB-T* für die Applikation nicht der bevorzugte Stromsensor ist, muss dieser, aufgrund der Lieferschwierigkeiten besserer Sensoren, eingesetzt werden. Für eine weitere Entwicklung wird bei Verfügbarkeit der Einsatz des *ACS725LLCTR-*

05AB-T empfohlen. Dadurch kann sowohl die Erfassung der Versorgungsspannung entfallen, wie auch der Spannungsteiler, der einerseits den Erfassungsbereich beschränkt und andererseits durch die Toleranzen der Widerstände die Genauigkeit reduziert.

4.2 Sensorik

Wie bereits im vorherigen Abschnitt beschrieben ist für die Stromsensorik noch ein Spannungsteiler, wie in Abbildung 4.1 gezeigt, notwendig, um einerseits den Messbereich von 5 V auf die für den Microcontroller geeigneten 3,3 V anzupassen und andererseits die 5 V Versorgungsspannung selbst mit einem ADC zu erfassen. Um $U_1 = 5\,\mathrm{V}$ auf $U_2 = 3{,}3\,\mathrm{V}$ zu übersetzten, wird nach Gleichung 4.1 respektive 4.2 bei einem Widerstand gegen Masse von $R_2 = 10\,\mathrm{k\Omega}$ ein Widerstand von 5,15 kΩ benötigt. Der nächstgrößere Widerstand in der E96 Reihe ist 5,23 kΩ und erlaubt Spannungen von bis zu 5,026 V. Die Spannung kann teilweise auch mehr als 26 mV schwanken, sodass der entsprechend nächste Widerstand von 5,62 kΩ nach Gleichung 4.3 gewählt wird und eine Schwankung bis 155 mV erlaubt.

$$R_1 = R_2 \cdot \left(\frac{U_1}{U_2} - 1\right) \tag{4.1}$$

$$5{,}15\,\mathrm{k\Omega} \approx 10\,\mathrm{k\Omega} \cdot \left(\frac{5\,\mathrm{V}}{3{,}3\,\mathrm{V}} - 1\right) \tag{4.2}$$

$$5{,}15\,\mathrm{V} \approx \frac{3{,}3\,\mathrm{V} \cdot (10\,\mathrm{k\Omega} + 5{,}62\,\mathrm{k\Omega})}{10\,\mathrm{k\Omega}} \tag{4.3}$$

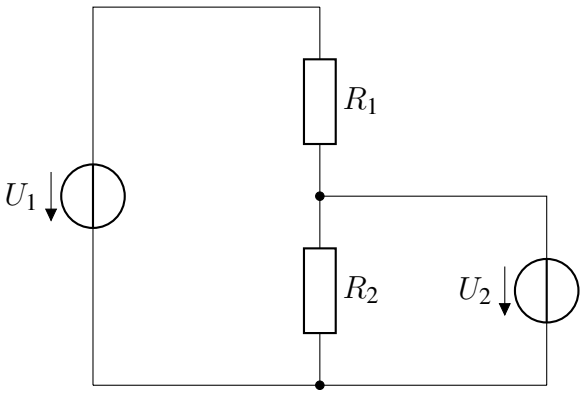

Abbildung 4.1: Aufbau des Spannungsteilers für die Stromsensorik

Die zu messende Wechselspannung beträgt nominal $U_{in} = 230\,\mathrm{V}$. Bei einer erlaubten Schwankung von $\pm 10\%$ ergibt sich somit $\hat{U}_{\mathrm{max}} = \pm 358\,\mathrm{V}$. Der Microcontroller kann wie erwähnt jedoch nur Spannungen im Bereich 0 bis 3,3 V erfassen. Somit bedarf es einer Anpassung des Spannungspegels.

Für die Messung der Wechselspannung wurden zunächst Aufbauten mit Operationsverstärkern evaluiert. Die getestete Schaltung weist eine hohe Komplexität auf und ist durch den Einsatz des Operationsverstärkers vergleichsweise teuer. Zudem erfasst die Schaltung nur

positive Spannungen, sofern eine gegenüber der Masse des Microcontrollers floatendes Potential gemessen wird. Bei wechselrichterseitiger Verwendung ist dies unproblematisch sofern die negativen Zwischenkreisspannung auf dem Potential der Masse liegt, da dieser aus Sicht des Microcontrollers nur positive Spannungen generiert. Am Stromnetz mit Phase und Neutralleiter ist letzterer jedoch bei einem Spannungspotential von Null, während auf der Phase die Spannung alterniert. Der Microcontroller kann nur eine der beiden Halbwelle messen, die andere, mit Spannungen von $< 0\,\mathrm{V}$, wird durch die Schaltung beschnitten, sodass keine Messung möglich ist.

Das Prinzip *Keep It Simple Stupid (KISS)* – also ein System so einfach wie möglich und dadurch auch verständlich zu gestalten – aus der Informatik lässt sich auch in der Elektrotechnik anwenden. Einerseits ist durch die Komplexität der Schaltung die Funktionsweise nicht leicht verständlich. Anderseits zeigt die Evaluation auch, dass das gewünschte Ergebnis nicht erreicht wird. Die alternative Schaltung sieht daher starke Vereinfachungen vor und besteht im Ersatzschaltbild in Abbildung 4.2 aus je drei Widerständen je Anschlusspunkt der Wechselspannung. Durch die identischen Widerstände R_1 und R_2 respektive R_5 und R_6 wird die Spannung am Microcontroller-Eingang auf dessen mittlere Versorgungsspannung gebracht. Ohne wechselspannungsseitiges Potential beträgt U_L und U_N somit $\mathrm{VCC}/2$. Die Widerstände R_3 beziehungsweise R_4 sind so gewählt, dass die anliegende Wechselspannung U_L respektive U_N so beeinflusst, dass ein Potential von 358 V in 3,3 V respektive $-358\,\mathrm{V}$ in 0 V resultieren. Im Microcontroller wird die Differenz aus U_L und U_N gebildet, wodurch einerseits der auf $\mathrm{VCC}/2$ verschobene Nullpunkt kompensiert wird und andererseits sowohl der wechselrichter- als auch netzseitige Einsatz möglich ist.

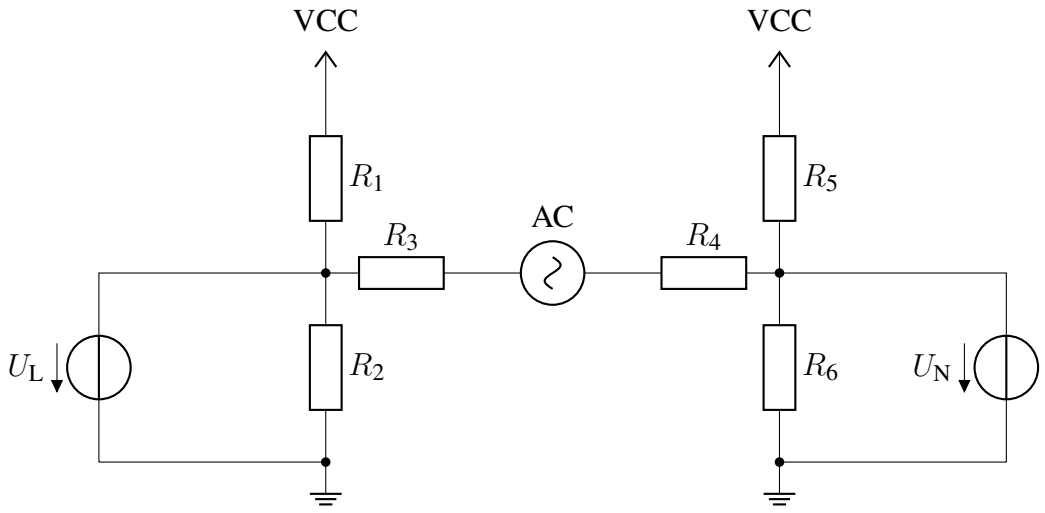

Abbildung 4.2: Aufbau des Spannungsteilers für die Spannungssensorik

Die Schaltung erreicht in der Charakterisierung eine ausreichende Linearität. In Abbildung 4.3 ist im Bereich von -20 bis 10 V, was weniger als 5% des Messbereichs entspricht, eine leichte Abweichung zu erkennen. Für die Erfassung der Zwischenkreisspannung U_DC wird die gleiche Schaltung vorgesehen, die bis auf den Widerstand gegen VCC auch bereits auf der Wechselrichterplatine vorhanden ist.

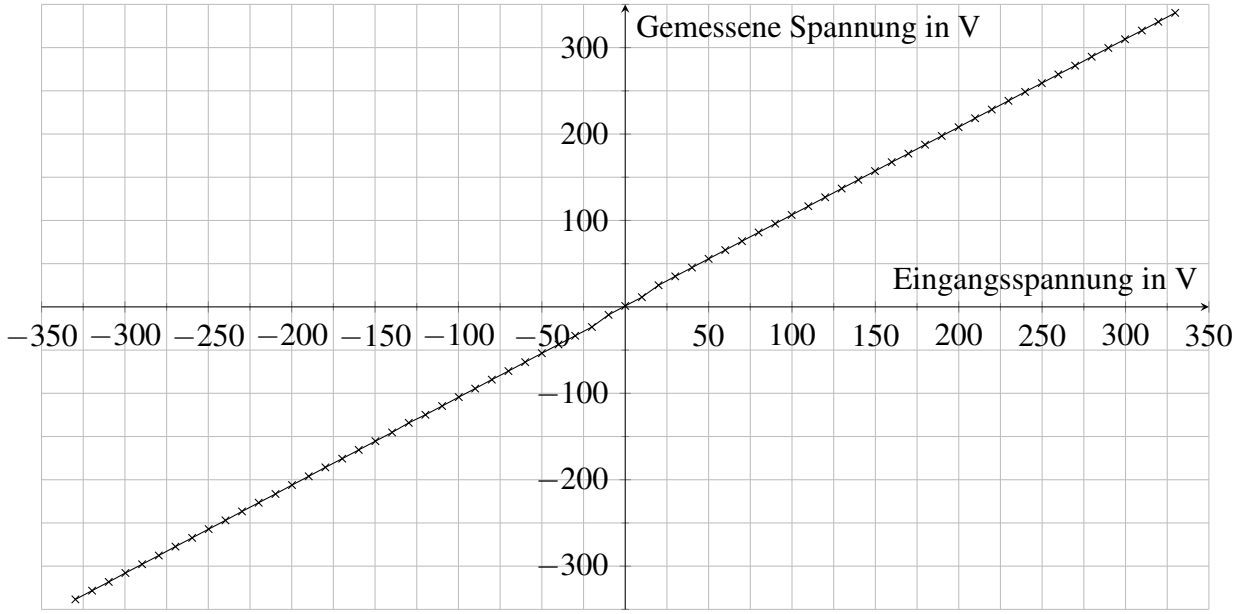

Abbildung 4.3: Messung der Linearität zwischen anliegender und vom Microcontroller erfassten Spannung

4.3 Anschluss der ADCs

Die vom Stromsensor als Messwert ausgegebene Spannung ist, wie in Abschnitt 4.1.2 erwähnt, abhängig von der Versorgungsspannung. Wie in Kapitel 3.2 erläutert, ist es für eine hohe Korrelation der erfassten Werte nötig, die am ADC anliegenden Spannungen der zusammengehörigen Kanäle gleichzeitig zu sampeln.

Der Microcontroller bietet 5 ADCs mit jeweils mehreren Kanälen, die über Multiplexer angebunden sind. Daraus folgert, dass maximal fünf Werte gleichzeitig erfasst werden können. Damit FA13 erfüllt wird, wird die Sensorik so verbunden, dass die Wechselspannung an vier (je zwei Messungen netz- und inverterseitig) unterschiedlichen ADCs anliegt. Für die Strommessung werden drei ADCs benötigt, der vierte ADC wird für die Messung der Zwischenkreisspannung genutzt. Der interne Temperatursensor des Prozessors ist auf ADC5 verfügbar, wird aber für den Betrieb im Labor nicht ausgewertet. Die genaue Belegung ist Tabelle 4.1 zu entnehmen.

Messung	**ADC**	**Kanal**
Spannung, netzseitig, L	4	5
Spannung, netzseitig, N	2	12
Spannung, wechselrichterseitig, L	3	1
Spannung, wechselrichterseitig, N	1	5
Strom, L	4	3
Strom, N	1	14
Referenzspannung Stromsensor	3	5
Zwischenkreisspannung	2	5

Tabelle 4.1: Belegung der ADC-Kanäle

4.4 Aufbau der Platine

Der Microcontroller wird im *Nucleo-64* Format bereits auf einer Trägerplatine mit Stiftleisten geliefert. Dementsprechend sind zum Anschluss Buchsenleisten auf dem Adapter vorgesehen. Die niederspannungsführende Elektronik ist am Rand links und rechts mit ausreichend Abstand angeordnet. Unten in Abbildung 4.4 befinden sich 4 mm-Sicherheitsbuchsen für den Netzanschluss, im oberen Bereich sind für eine vielseitige Konfiguration *Würth Elektronik Redcube*-Schraubkontakte eingesetzt. Durch die Relais wird der Wechselrichter von dem Netz zweipolig getrennt und kann bei erfolgter Synchronisierung verbunden werden. Die Stromsensoren sind in den oberen Ecken platziert und können wie in Abbildung 4.5 für die Messung des Netzstroms konfiguriert werden. Ebenso ist durch die Schraubkontakte die alternative Konfiguration zur Messung des Drosselstroms möglich.

Im unteren Bereich befinden sich Leuchtdioden (LEDs), welche den Zustand des Systems anzeigen. So sind zum einen die Enable-Signale für die Halbbrücken des Wechselrichters wie auch die Relais-Signale durch einzelne LEDs visualisiert. Zum anderen sind noch zwei weitere LEDs verbaut, welche Fehler (rot) respektive eine erfolgreiche Synchronisation (grün) signalisieren. Für die Messung mittels Oszilloskop sind zudem vier Testpunkte vorhanden.

Auf die Details des Platinenlayouts wird in dieser Arbeit aufgrund der thematischen Ausrichtung nicht weiter eingegangen. Die Anforderung FA1 ist jedoch erfüllt und wurde validiert.

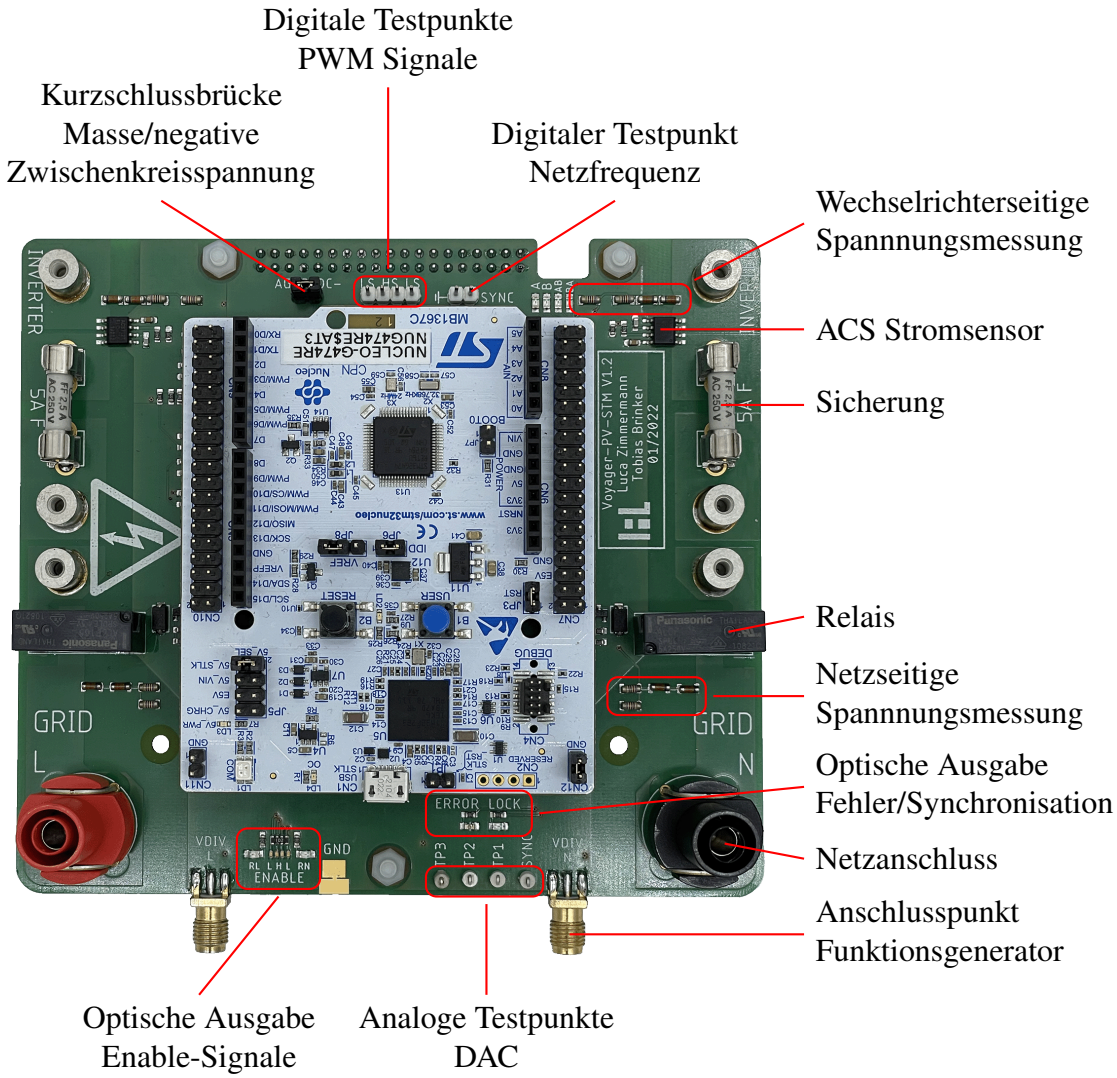

Abbildung 4.4: Adapterplatine für Nucleo-64 Board mit Messsensorik
Die hier noch aufgesteckte Kurzschlussbrücke an JP5 (*5V_SEL*) muss für den Betrieb entfernt werden.

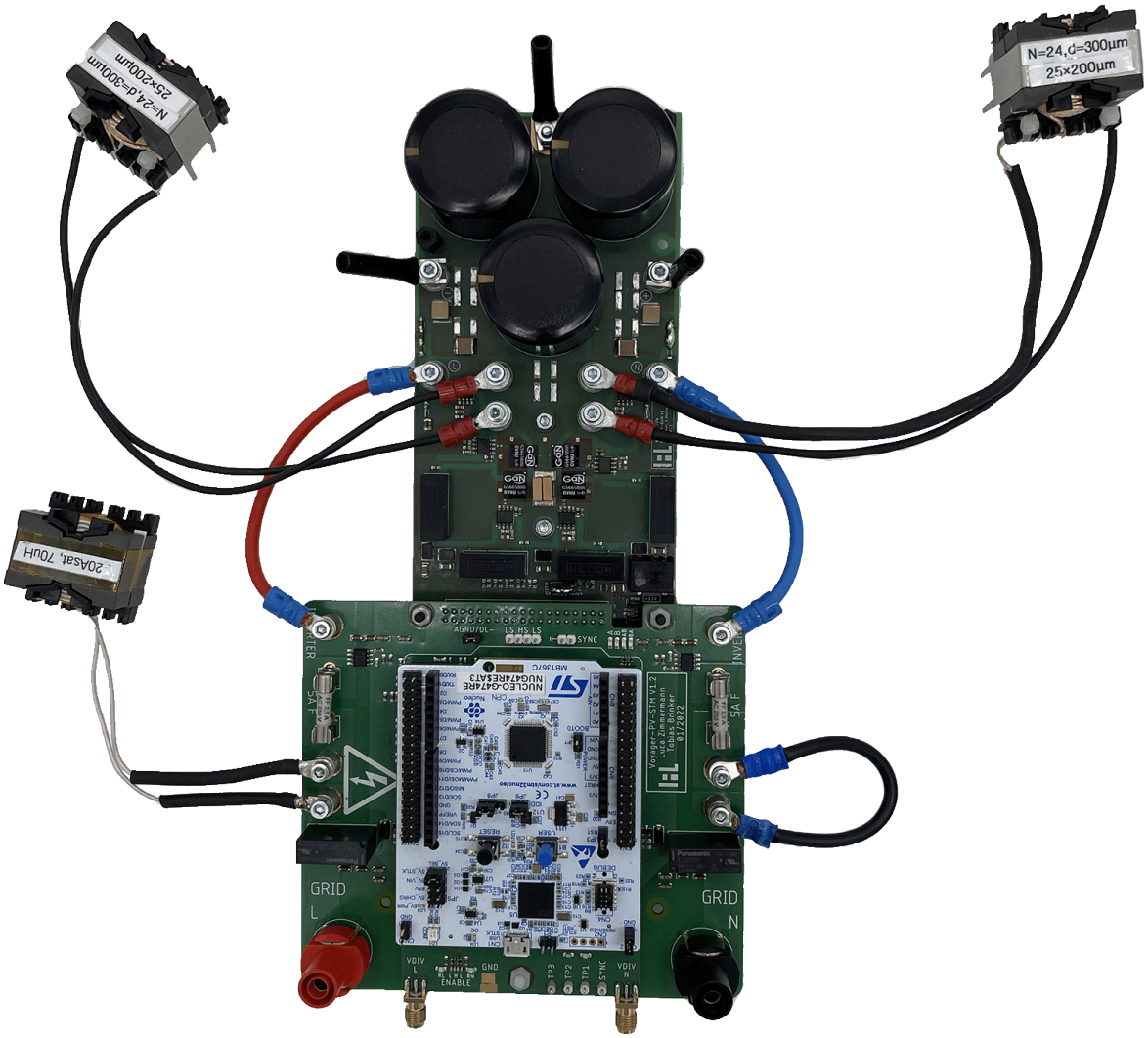

Abbildung 4.5: Anschluss der Adapterplatine an den Wechselrichter, hier in der Konfiguration der Stromsensoren für die Messung des Netzstroms

5 Implementierung

Ziel dieses Kapitel ist, sowohl die mathematischen Konzepte für die PLL und Stromregelung darzustellen, als auch deren effiziente Implementierung auf dem Microcontroller. Hierfür wird einerseits evaluiert, welche Werkzeuge für die Aufgabe geeignet sind, andererseits die Toolchains und mathematische Vereinfachungen wirksam zur Beschleunigung der Berechnungen eingesetzt. Daran schließt sich die Ansteuerung der Hardware und Kommunikation mit einem verbundenen Computer an. Abschließend wird auf die Grenzwerte des Systems eingegangen, welche eine Sicherheitsabschaltung ausführen sollen, und die Erstellung der Dokumentation erläutert.

5.1 Auswahl der Entwicklungsumgebung

STMicroelectronics listet auf der Produktseite unter den Entwicklungstools lediglich *WinIDEA Open* auf, eine Entwicklungsumgebung, die nur unter Windows lauffähig ist. Auf der Produktverpackung des *Nucleo*-Boards wird hingegen *Mbed* beworben. *ST* bietet jedoch auch die auf Eclipse aufsetzende *STM32CubeIDE* an, die nicht an ein Betriebssystem gebunden ist. Aufgrund der Beschränkung von *WinIDEA Open* auf Windows-Systeme wird im folgenden kurz die Entscheidung zwischen *Mbed* und der *STM32CubeIDE* erläutert.

Mbed ist die webbasierte Entwicklungsumgebung von *Arm*, entsprechend passend zu der *Arm Cortex-M4* Prozessorbasis. Alternativ dazu lässt sich auch *Mbed Studio* als Desktopumgebung nutzen. Wahlweise lässt sich das *Echtzeitbetriebssystem (RTOS)* oder *Bare-Metal* als Basis für den Microcontroller nutzen. Das RTOS bietet dabei die Vorteile der parallelen Programmierung mit Multithreading und einer weiterhin deterministischen Ausführung. Allerdings bedingt das Betriebssystem durch die zusätzliche Abstraktionsschicht auch einen leicht erhöhten Overhead und reduzierte Kontrolle über die Hardware selbst. Bei der Wahl der *Bare-Metal* Basis bildet weiterhin die *Mbed-Application Programming Interface (API)* das Grundkonstrukt und ist daher immer noch leicht abstrahiert.

Die *STM32CubeIDE* bietet zur Konfiguration der Hardware eine grafische Oberfläche, wie in Abb. 5.1 gezeigt. Der ausgewählte Prozessor wird in seiner Bauform angezeigt, so können bei der Nutzung des Prozessors ohne Trägerplatine die Pins entsprechend für kurze Wege zu den Komponenten konfiguriert werden. Für die Testaufbauten wird ein *Nucleo-64* Board genutzt, also eine zusätzliche Platine, auf der die Pins abweichend angeordnet sind, sodass die Darstellung zum Entwicklungszeitpunkt zweitrangig ist. Bei der Konfiguration lassen sich jedoch Konflikte zwischen Belegungen schnell erkennen, da eingeschränkte Verfügbarkeit in gelb und sich widersprechende Einstellungen in pink hervorgehoben werden. Als API besteht die Auswahl zwischen *High Abstraction Layer (HAL)* für einfache Benutzung und *Low Level (LL)* für verkürzte Zugriffszeiten respektive weniger Overhead. Die HAL-API bietet automatisch eine Überprüfung der korrekten Ausführung an und gibt im Erfolgsfall ein OK zurück, während bei der LL-API in der Regel nur Register manipuliert werden, ohne

Fehler-Flags zu kontrollieren. Diese Prüfung muss durch die entwickelnde Person manuell implementiert werden.

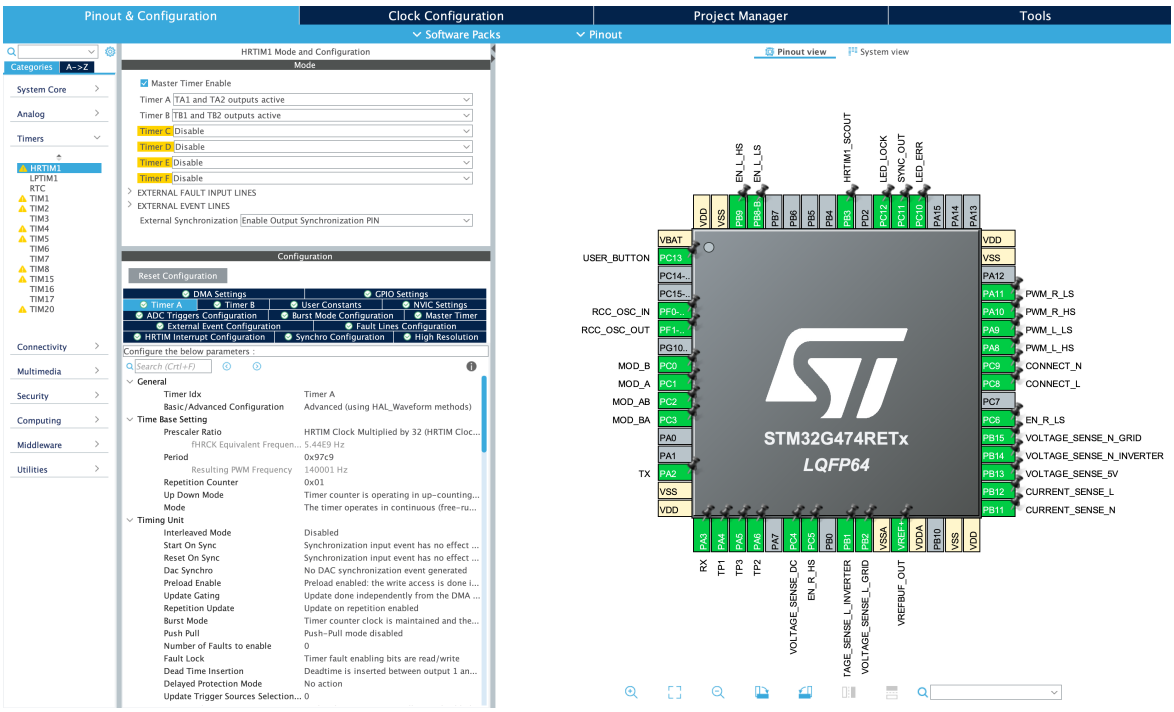

Abbildung 5.1: Übersicht in der STM32CubeIDE

Beide Umgebungen bieten somit den Entwickelnden die Möglichkeit zwischen zwei Abstraktionsstufen, wobei *Mbed* grundsätzlich stärker abstrahiert als die *CubeIDE*. Da sich bei der *CubeIDE* Parameter, wie beispielsweise die Timer-Periode, in der grafischen Oberfläche ändern lassen und die resultierende Frequenz direkt angezeigt wird, ist die Benutzung für die Projektbearbeitenden mit wenig Informatikkenntnissen anschaulicher. Zudem ermöglicht die *CubeIDE* aufgrund der Ausrichtung rein auf die *STM32* Prozessoren und nicht die gesamte *Arm*-Architektur deutlich feinere Einstellungsmöglichkeiten, um das volle Potential der *STM*-Prozessoren zu nutzen. Hardwareeinheiten wie der CORDIC sind über *Mbed* nicht direkt ansprechbar. Da dies für die Entwicklung essentiell ist, wird die *STM32CubeIDE* als Entwicklungsumgebung ausgewählt. Als Programmiersprache wird aufgrund der Konzentration auf Algorithmen und damit eher funktionale/prozedurale Programmierung C gegenüber C++ bevorzugt.

5.2 PLL

Die PLL ermittelt aus einer abgetasteten Wechselspannung den Phasenwinkel und daraus den Sinus und Kosinus. Letztere werden im Folgenden für die Stromregelung benötigt, um den jeweils aktuellen Sollwert für Wirk- bzw. Blindstrom zu berechnen. Um den entsprechenden Normen zu entsprechen, wie in FA2 gefordert, muss die Ausführungsfrequenz der PLL mindestens 2 kHz betragen. Andernfalls würde der resultierende Sinus eine Treppenbildung aufweisen, welche zu ungewollten harmonischen Schwingungen im Stromnetz führt.

Die PLL und Stromregelung liegt in einem Blockschaltbild in *MATLAB* respektive *Simulink* vor. *Simulink* bietet die Option der Synthese an, also automatische Umwandlung der Blöcke in Programmcode. Um die Codequalität zu steigern, werden jedoch die Algorithmen von Grund auf in C implementiert und das Modell nur als Referenz und Testumgebung herangezogen. Umgekehrt lassen sich auch eigene C-Programmabschnitte in *Simulink*-Blöcke implementieren und die korrekte Funktionsweise der Berechnungen überprüfen.

Die Berechnungen in den PLLs sind abhängig von der Ausführungsfrequenz f_{PLL} der PLL, respektive dem Kehrwert, der Periodendauer T_{PLL}.

5.2.1 QSG

Der Quadrature Signal Generator (QSG) erzeugt aus einem Eingangssignal (Netzspannung) mit dem Phasenwinkel θ die Signale $\alpha = \sin(\theta)$ und $\beta = \cos(\theta)$. Dementsprechend ist α identisch mit dem Eingangssignal und β um 90° verschoben. Das β Signal wird im folgenden in der dq-Transformation benötigt, die eigentlich für dreiphasige Systeme ausgelegt ist.

Für die QSG-PLL wurden in einer vorhergehenden Arbeit vier verschiedene Implementierungen bezüglich ihrer Genauigkeit, Geschwindigkeit und Effizienz untersucht [14]. In dieser Arbeit wird daher auf Basis der Ergebnisse, einer Bewertung der Komplexität und der damit einhergehenden Berechnungszeit nur eine Auswahl getroffen und die Varianten nicht erneut einzeln getestet.

tDelay-PLL

Die *Transport Delay (tDelay)* Implementierung nutzt einen *First-in-First-out (FIFO)*-Speicher der Größe $f_{\text{PLL}}/4 \cdot f_{\text{Netz, nominal}}$, um ein um 90° phasenverschobenes Signal β zu generieren, für α wird das Eingangssignal ohne weitere Verarbeitung genutzt. Aufgrund der starren Verzögerung um eine Viertelperiode der nominalen Netzfrequenz, wie in Abbildung 5.2 zu sehen, ergibt sich bei Frequenzschwankungen auch eine unerwünschte Abweichungen im β Signal. Für die nominale Netzfrequenz von 50 Hz ergeben sich somit die Gleichungen 5.1 und 5.2.

$$\alpha_{\text{tDelay}} = v_{\text{in},\, t} \tag{5.1}$$

$$\beta_{\text{tDelay}} = v_{\text{in},\, t - \frac{T_{\text{Netz, nominal}}}{4}} \tag{5.2}$$

Abhängig von der definierten Frequenz der PLL beträgt die Anzahl der Einträge bei einer Netzfrequenz von $f_{\text{Netz}} = 50\,\text{Hz}$ ein Zweihundertstel der Frequenz f_{PLL}. Aufgrund der Vorgabe aus FA2 und damit eine Mindestfrequenz von 2 kHz, muss der Speicher jedoch mindestens 10 Einträge umfassen. Für `floats` mit einer Breite von 32 Bit würde dies 40 Byte bedeuten.

In Abbildung 5.3 ist das *Simulink*-Blockschaltbild zu sehen, basierend auf einer PLL-Frequenz von 140 kHz. Damit ergibt sich für den Speicher eine Länge von 700 Einträgen, entsprechend 2800 Byte.

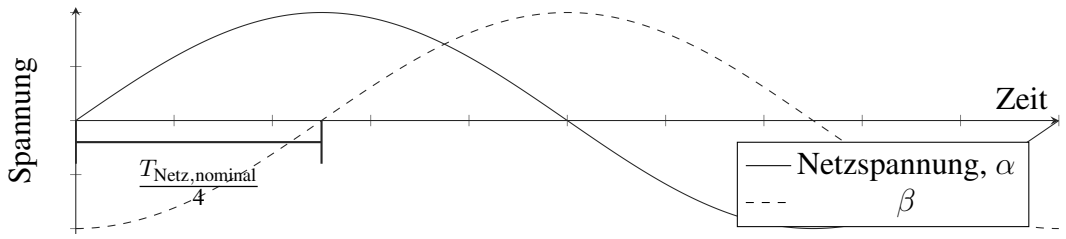

Abbildung 5.2: Verlauf von α und β der tDelay-PLL

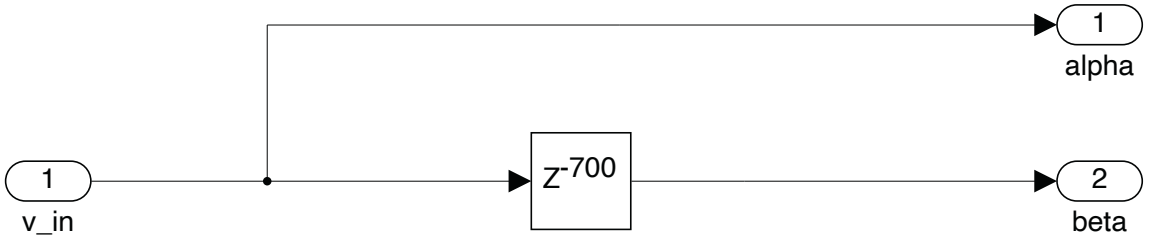

Abbildung 5.3: Aufbau der tDelay-PLL ($f_{\text{PLL}} = 140\,\text{kHz}$)

DSC-PLL

Die Delay Signal Cancellation (DSC) Variante ist ähnlich zur tDelay-PLL, nutzt jedoch zwei serielle Speicher. α_{DSC} entspricht dabei β_{tDelay}, wird also durch die erste Verzögerung um 90° phasenverschoben. Für die zweite Komponente wird das Signal α_{DSC} um eine weitere Viertelperiode mittels eines Speichers auf insgesamt eine halbe Periode (bezogen auf das Eingangssignal) verzögert, wie in Abbildung 5.4 gezeigt. β_{DSC} kann dann durch den Mittelwert der invertierten Netzspannung und dem 180° phasenverschobenen Signal berechnet werden.

$$\alpha_{\text{DSC}} = v_{\text{in},\, t-\frac{T_{\text{Netz, nominal}}}{4}} \tag{5.3}$$

$$\beta_{\text{DSC}} = \frac{v_{\text{in},\, t-\frac{T_{\text{Netz, nominal}}}{2}} - v_{\text{in}}}{2} \tag{5.4}$$

Die DSC-PLL ist trotz der festen Auslegung der Speichergröße respektive der Verzögerungszeit durch die Mittelwertbildung für β für die Frequenzermittelung mit Abweichungen von der nominalen Netzfrequenz besser geeignet, als die tDelay-PLL. Insgesamt verschiebt sich das Signal jedoch um eine Viertelperiode. Auf Veränderungen im Eingangssignal kann entsprechend erst verzögert reagiert werden. Bei der Implementierung als Ringspeicher wären insgesamt vier Additionen respektive Subtraktionen und die Division durch zwei notwenig. Wie in Abschnitt 5.5.2 festgestellt, benötigen Divisionen jedoch 14 statt einem Takt, eine Multiplikation mit $0,5$ würde dies auf einen Takt reduzieren. In Summe werden ohne Lade- und Speicheroperationen somit fünf Takte benötigt. Der Speicherbedarf ist doppelt so groß, wie bei der tDelay-PLL.

Wie auch für die DSC-PLL basiert das *Simulink*-Blockschaltbild in Abbildung 5.5 auf einer PLL-Frequenz von 140 kHz. Insgesamt werden bei dieser Frequenz 5200 Byte für die zwei

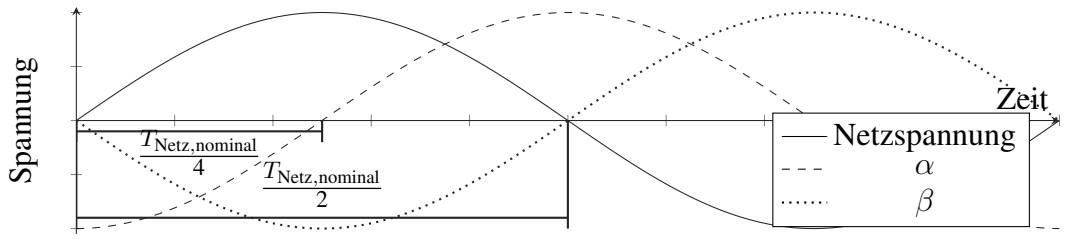

Abbildung 5.4: Verlauf von α und β der DSC-PLL

Verzögerungsglieder belegt.

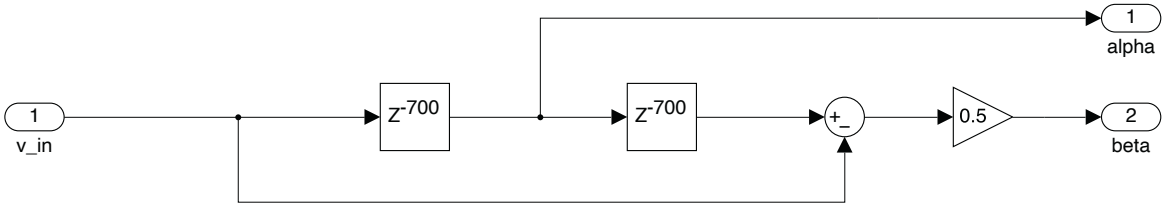

Abbildung 5.5: Aufbau der DSC-PLL ($f_{\text{PLL}} = 140\,\text{kHz}$)

TwoSample-PLL

Darüber hinaus wird auch noch die TwoSample-PLL untersucht, die bei einer Verunreinigung des Eingangssignals mit harmonischen Schwingungen nicht mehr zur Frequenzermittlung genutzt werden kann und daher hier nicht weiter betrachtet wird. Die Ergebnisse decken sich hierbei auch mit denen von Lamo, et al. [9].

QSOGI-PLL

Aufgrund der sehr guten Bestimmung der Frequenz, als auch dem schnellen Einrasten wird die *Quadrature Second-Order Generalized Integrator (QSOGI)*-PLL in der vorhergehenden Arbeit als Referenzimplementierung betrachtet [14]. Auch in dem bereits erwähnten Review [9] wird dieser PLL – neben der bereits ausgeschlossenen TwoSample-PLL – eine kurze Antwortzeit auf Frequenz- und Phasensprünge bestätigt. Der dort erwähnte Rechen- und Hardwareaufwand bezieht sich jedoch auf ein FPGA, bei dem Berechnungen stark parallelisiert werden können, und muss daher für den Microcontroller neu bewertet werden. Die Terme zur Ermittlung von α und β sind im Vergleich zu den speicherbasierten PLLs deutlich komplexer.

In dem *Simulink*-Modell in Abbildung 5.6 werden Verzögerungsglieder ($1/z$) genutzt. Für die Überführung in für den Microcontroller geeigneten Code werden stattdessen drei Hilfszustände genutzt. Zustand $state_1$ in Gleichung 5.5 ist dabei direkt von der Eingangsspannung und weiteren Variablen abhängig. Insgesamt sind für diesen Term drei Multiplikationen und zwei Additionen notwendig, die im optimalen Fall jeweils einen Taktzyklus benötigen. Darüber hinaus wird α mit einer Subtraktion (5.6) und β mit einer weiteren Multiplikation (Gl. 5.7) mit weiterhin je einem Takt berechnet. $state_3$ und $state_2$ in Gleichung 5.8 und 5.9 sind lediglich Zuweisungen des vorigen Zustands. Der Speicherbedarf ist im Gegensatz zur tDelay- und DSC-PLL auch bei verändertem PLL-Takt konstant. Wie bei der tDelay-PLL ist

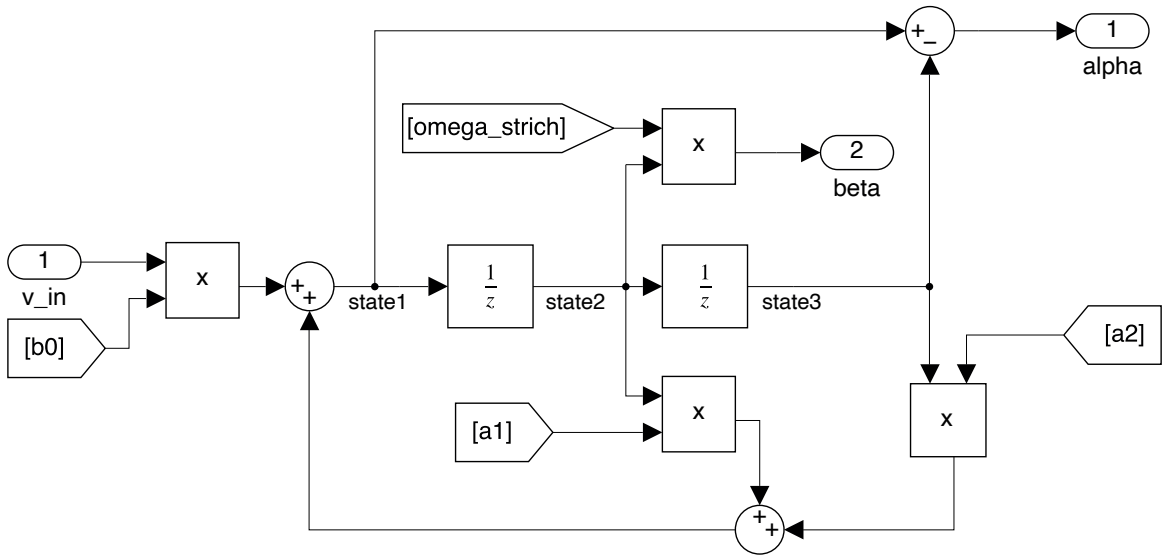

Abbildung 5.6: Aufbau der QSOGI-PLL

α deckungsgleich mit der Eingangsspannung, jedoch ist der Versatz von β nicht mehr fest von der Speichergröße, sondern in Abhängig von der Netzfrequenz (vgl. Abb. 5.2).

$$
\begin{aligned}
state_1 &= b_0 \cdot v_{\text{in}} + a_1 \cdot state_2 + a_2 \cdot state_3 && (5.5)\\
\alpha &= state_1 - state_3 && (5.6)\\
\beta &= state_2 \cdot \omega' && (5.7)\\
state_3 &= state_2 && (5.8)\\
state_2 &= state_1 && (5.9)
\end{aligned}
$$

Die für $state_1$ benötigten Variablen ergeben sich aus Berechnungen in Abhängigkeit von der Kreisfrequenz ω, welches $f_{\text{Netz}} \cdot 2 \cdot \pi$ entspricht. Die Rückkopplung der im weiteren Verlauf von der Synchronous Reference Frame (SRF) PLL ermittelten Netzfrequenz über ω fungiert als Bandpassfilter. Für die Terme a_1 (Gl. 5.13) bis b_0 (Gl. 5.15) können sich wiederholende Berechnungen in die Gleichungen 5.10 bis 5.12 ausgelagert werden, um so einerseits die Berechnungszeit, aber auch die Übersicht zu verbessern.

$$
\begin{aligned}
x &= \omega \cdot 2 \cdot \sqrt{2} \cdot T_{\text{PLL}} && (5.10)\\
y &= (\omega \cdot T_{\text{PLL}})^2 && (5.11)\\
z &= x + y + 4 && (5.12)\\
a_1 &= \frac{2 \cdot (4 - y)}{z} && (5.13)\\
a_2 &= \frac{x - y - 4}{z} && (5.14)\\
b_0 &= \frac{x}{z} && (5.15)
\end{aligned}
$$

Betrachtet man jedoch die Optimierungsansätze in Abschnitt 5.5.3, so kann die Berechnung der voherigen Gleichungen auf eine einmalige Ausführung mit der nominalen Netzfrequenz

beschränkt werden. Der Rechenaufwand liegt somit, abgesehen von den Lade- und Speicherzyklen, bei 4 Multiplikationen, zwei Additionen und einer Subtraktion, insgesamt sieben Operationen, die sich auf jeweils einen Takt beschränken. Aufgrund der guten Ergebnisse und der im optimierten Zustand dennoch geringen Ausführungszeit wird diese PLL auf dem Microcontroller implementiert. Sollte die Rückkopplung entgegen der Empfehlungen in Abschnitt 5.5.3 dennoch implementiert werden, so empfiehlt sich nach Abschnitt 5.5.2 die Anpassung der Gleichung 5.12 zu $z = {}^{1}/_{x+y+4}$, sodass die Division nur einmalig durchgeführt wird. Für die Gleichungen 5.13 bis 5.15 ist die Division durch eine Multiplikation zu ersetzen.

5.2.2 dq-Transformation

Die dq-Transformation vereinfacht die Synchronisierung zur Netzfrequenz. Das genutzte Koordinatensystem erlaubt aufgrund des statischen Vektors den Einsatz eines PI-Reglers, welcher nur wenige Operationen benötigt.

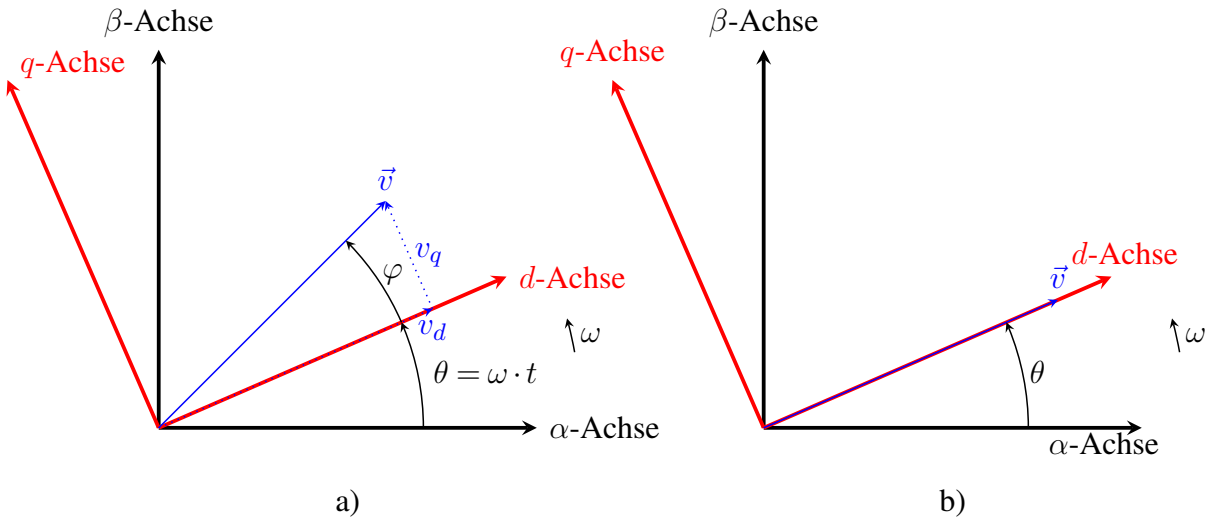

Abbildung 5.7: Rotation vom $\alpha\beta$- in das dq-Koordinatensystem

$\vec{v}$ entspricht der gemessenen Netzspannung, v_d und v_q deren Komponenten in dq-Koordinaten. φ ist der Winkel, der auszuregeln ist, so wie in b) gezeigt.

Der Vektor $\vec{v}$ rotiert im $\alpha\beta$-Koordinatensystem mit der Kreisfrequenz ω. Die Länge des Vektors entspricht der Wechselspannung mit einem Effektivwert von 230 V, was einer Spitzenspannung $230\,\mathrm{V} \cdot \sqrt{2} \approx 325\,\mathrm{V}$ entspricht. Die Kreisfrequenz entspricht in der avisierten Applikation der $2 \cdot \pi f_{\mathrm{Netz}}$.

Bei der dq-Transformation wird das statische Koordinatensystem mit rotierendem $\alpha\beta$-Vektor in ein rotierendes Koordinatensystem mit statischem dq-Vektor überführt. Der Winkel φ beschreibt dabei die Phasenverschiebung zwischen Eingangssignal in $\alpha\beta$-Koordinaten und der dq-Transformierten. Mit der SRF PLL soll diese Abweichung nachfolgend, wie in Abbildung 5.7 b), ausgeregelt werden. Die Berechnungen der d und q Komponente in Gleichung 5.17 und 5.18 benötigen Multiplikationen mit dem Sinus und Cosinus von θ. Der aktuelle

Winkel θ wird erst nachfolgend durch die SRF-PLL ermittelt und daher aus dem vorangegangen Zyklus genutzt.

$$\theta = \omega \cdot t \tag{5.16}$$
$$v_{\mathrm{d}} = v_\alpha \cdot \cos(\theta_{t-1}) + v_\beta \cdot \sin(\theta_{t-1}) \tag{5.17}$$
$$v_{\mathrm{q}} = -v_\alpha \cdot \sin(\theta_{t-1}) + v_\beta \cdot \cos(\theta_{t-1}) \tag{5.18}$$

5.2.3 SRF-PLL

Die SRF-PLL ist zuständig für die Synchronisierung zwischen dem Vektor $\vec{v}$ in $\alpha\beta$-Koordinaten und dem dq-Koordinatensystem (vgl. Abschnitt 5.2.2). Da der Vektor in dq-Koordinaten nahezu statisch ist, lassen sich über dessen Komponenten Abweichungen mittels eines PI-Reglers mathematisch einfach kompensieren. Angestrebt wird $\cos(\varphi) = 1$, also Winkel von $\varphi = 0°$ und damit eine q-Komponente von ebenfalls 0.

Für den PI-Regler lässt sich damit die negative q-Komponente als Fehlerwert übergeben, beziehungsweise 0 als Sollgröße und q als Ist-Wert. Durch die Netzimpedanz kann es beim Zuschalten von größeren Lasten zu Spannungseinbrüchen kommen. Besonders in Momenten eines Kurzschlusses im Netz kann die Spannung drastisch einbrechen. Da die Regelung für die Frequenz weiterhin stabil arbeitet, muss q entsprechend wie in Gleichungen 5.19 aufgezeigt normiert werden.

$$error_{\mathrm{SRF}} = -\frac{q}{|\vec{v}|} \tag{5.19}$$

Die Steuergröße u_t des Reglers wird zunächst auf einen Wertebereich von $2 \cdot \pi \cdot f_{\text{Netz, nominal}} \cdot 10\%$ begrenzt, sodass der Regler bei 50 Hz Netzfrequenz nur zwischen 45 Hz und 55 Hz schwingen kann. ω lässt sich nach Gleichung 5.20 aus u_t errechnen und sich daraus auch wieder die Netzfrequenz ableiten (vgl. Gl. 5.21). Die Gleichung $\theta = \omega \cdot t$ kann wie in 5.22 zu einer inkrementellen Berechnung umgeformt werden.

$$\omega_t = u_{\mathrm{SRF},t} + 2 \cdot \pi \cdot F_{\mathrm{Netz,nominal}} \tag{5.20}$$
$$f_{\mathrm{Netz},t} = \frac{\omega_t}{2 \cdot \pi} \tag{5.21}$$
$$\theta_t = \theta_{t-1} + \omega_t \cdot T_{\mathrm{PLL}} \tag{5.22}$$

5.3 Stromregelung

Der Strom wird für den Laborbetrieb durch die bedienende Person vorgegeben. Dabei kann sowohl der Anteil von Wirk- als auch Blindstrom beliebig gestellt werden.

Für die Strommessung sind zwei Stromsensoren wechselrichterseitig verbaut und von dem Stromnetz durch Relais getrennt. Der gemessene Strom I_{ist} wird mit der Sollwertvorgabe verglichen und der sich dabei ergebende Fehlerwert aus Gleichung 5.23 mithilfe eines PI-Regler gegen Null geregelt.

$$error_{\mathrm{I},t} = I_{\mathrm{soll},t} - I_{\mathrm{ist},t} \tag{5.23}$$

Sowohl für den Fall, dass die Halbbrücken des Umrichters nicht eingeschaltet, als auch wenn der Umrichter nicht mit dem Netz über die Relais verbunden ist, kann kein Strom gemessen werden. Der PI-Regler hat in diesem Moment keinen Einfluss auf das Stellglied, da dieses vom Stromnetz entkoppelt ist und kein Strom fließen kann. So lange entweder die Halbbrücken oder Relais nicht aktiviert sind, kann keine Reglerfreigabe erteilt und es muss somit eine Regelgröße von Null übergeben werden. In dieser Zeit wird lediglich versucht, die Spannung des Umrichterausgangs an die Netzspannung anzupassen (siehe Abschnitt 5.4). Bei der Nutzung der gemessenen Eingangsspannung werden Störungen und Messfehler übernommen. Um dies zu verhindern, wird die Spannung durch einen QSG (vgl. Abschnitt 5.2.1) gefiltert.

Sobald Halbbrücken und Relais aktiviert wurden, kann der Regler aktiviert und eine Stellgröße übergeben werden. Da Relais durch die elektromechanische Kopplung nicht wie Halbleiter nahezu unmittelbar mit dem Steuerimpuls auch die Schaltung ausführen, muss die Schaltverzögerung entsprechend im Programm kompensiert werden. Bei Messungen mit den verwendeten Panasonic ALDP105W wurden bei Raumtemperatur experimentell 2,8 ms zwischen Ansteuerung und Schließen der Kontakte ermittelt. Wenn Spannung an den Spulen der Relais anliegt, werden mittels eines Zählers ausreichend PLL-Takte gewartet, bevor die Regelung tatsächlich aktiviert wird, um sicherzustellen, dass die Relaiskontakte mechanisch geschlossen sind. Um eine ungewünschte Anregung des Systems beim Aufschalten des Umrichters zu vermeiden, wird die Regelgröße mit einem über die Dauer von einer halben Periodendauer $T_{\mathrm{Netz,nominal}}/2 = 10\,\mathrm{ms}$ linear von 0 auf 100% steigernden Faktor multipliziert (Anfahrrampe), in Abbildung 5.8 mit der gepunkteten Linie zu sehen.

Während der Tests wurde festgestellt, dass bei Aufschaltungen außerhalb des Nulldurchgangs auch bei einem eingestellten Nullstrom die Regelung stark schwingt und Überschwingungen produziert. In Aufschaltmomenten abseits des Nulldurchgangs ist die Spannungdifferenz groß, sodass die Spannungszeitfläche über die Netzimpedanz und Drosseln des Wechselrichters zu einem Strompeak führt. Um dies zu verhindern wurde die Aufschaltung so verändert, dass das Relais erst im Nulldurchgang geschlossen ist. Innerhalb des gestrichelten Bereichs in Abbildung 5.8 erfolgt die Anforderung zur Verbindung von Wechselrichter und Stromnetz. Daraufhin wird nach dem nächsten Nulldurchgang (Ende der gestrichelten Fläche) eine halbe Periodenlänge (10 ms) abzüglich der Schaltverzögerung des Relais (2,8 ms) gewartet, bevor die Steuerspannung angelegt wird. Die tatsächliche Schließung des Kontakts erfolg somit ab der Anforderung im übernächsten Nulldurchgang.

Die Erkennung von Nulldurchgängen und tatsächlicher Ansteuerung der Relais erfolgt durch die PWM-Steuerung, welche im nachfolgenden Kapitel erläutert wird.
Der effektive Sollstrom ergibt sich nach der Gleichung 5.24 aus dem Wirkstromanteil I_{p} und Blindstromanteil I_{q}. Wirkstrom ist der Anteil, der mit dem Phasenwinkel der Spannung übereinstimmt. Blindstrom hingegen ist, je nach dem, ob eine induktive oder kapazitive Last vorliegt, um 90° respektive $\pi/2$ vor- oder nacheilend.

Durch die Funktionsweise der PLL ist θ um $\pi/2$ nacheilend. Im *Simulink* Modell wird da-

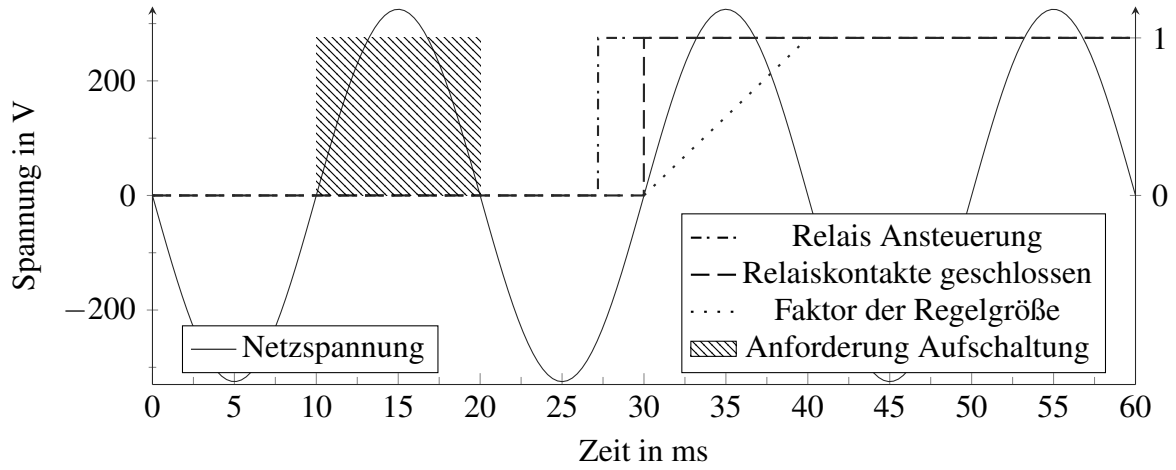

Abbildung 5.8: Zustände beim Aufschalten der Relais

für zum ausgegebenen Winkel θ $\pi/2$ addiert, sodass die Verschiebung kompensiert wird. Für den Microcontroller bedeutet das aber eine zweite Berechnung der Winkelfunktionen, die sich nachteilig auf die Anzahl der benötigten Taktzyklen auswirkt. Um eine Verschiebung von $-\pi/2$ zu erreichen, wird wie in der Gleichung 5.25 und 5.26 die Umformung $\sin(x + \pi/2) = \cos(x)$ respektive $\cos(x + \pi/2) = -\sin(x)$ genutzt, ohne dass eine erneute Berechnung von Sinus oder Cosinus erforderlich ist. Bei der Wahl der DSC-PLL muss entsprechend um π verschoben werden, was lediglich eine der Änderung der Vorzeichen zur Folge hat.

$$I_{\text{soll},t} = I_{\text{p,soll}} \cdot \sqrt{2} \cdot \sin(\theta_t) + I_{\text{q,soll}} \cdot \sqrt{2} \cdot \cos(\theta_t) \tag{5.24}$$

$$I_{\text{soll},t,\text{PLL}} = I_{\text{p,soll}} \cdot \sqrt{2} \cdot \sin(\theta_t + \frac{\pi}{2}) + I_{\text{q,soll}} \cdot \sqrt{2} \cdot \cos(\theta_t + \frac{\pi}{2}) \tag{5.25}$$

$$I_{\text{soll},t,\text{PLL}} = I_{\text{p,soll}} \cdot \sqrt{2} \cdot \cos(\theta_t) + I_{\text{q,soll}} \cdot \sqrt{2} \cdot (-\sin(\theta_t)) \tag{5.26}$$

Die Sollwerte werden über die Schnittstelle als RMS-Wert übergeben und als Variable gespeichert. Die jedoch notwendige Amplitudenhöhe ist bei einfrequent sinusförmigen Signalen um Faktor $\sqrt{2}$ größer, als der RMS-Wert. Damit nicht bei jeder Ausführung der Stromregelung erneut $I_{\text{p,RMS}} \cdot \sqrt{2}$ respektive $I_{\text{q,RMS}} \cdot \sqrt{2}$ berechnet wird, wird beim Setzen des Sollwerts die Berechnung mit ausgeführt und in einer weiteren Variable gespeichert.

Damit ein Strom in das Netz eingespeist wird, muss der Wechselrichter eine etwas höhere Spannung als die im Netz anliegende generieren. Die Spannungsvorgabe ergibt sich aus der Eingangsspannung v_{in} und der Steuergröße u des PI-Reglers. Für die Stellgröße y_t des PWM-Signals wird ein prozentualer Wert benötigt, der in Abhängigkeit von der Gleichspannungsversorgung V_{DC} berechnet wird (Gleichung 5.27).

$$y_{\text{I},t} = \frac{v_{\text{in},t} + u_{\text{I},t}}{v_{\text{DC},t}} \tag{5.27}$$

Für Testaufbauten im Labor wurden geringere Spannungen eines einstellbaren Trenntransformators genutzt. In Abbildung 5.9 ist zu sehen, wie zum Zeitpunkt der Sollwertänderung von 0 zu 500 mA RMS die Spannung entsprechend von 60 auf 70 V ansteigt, um den Strom einzustellen.

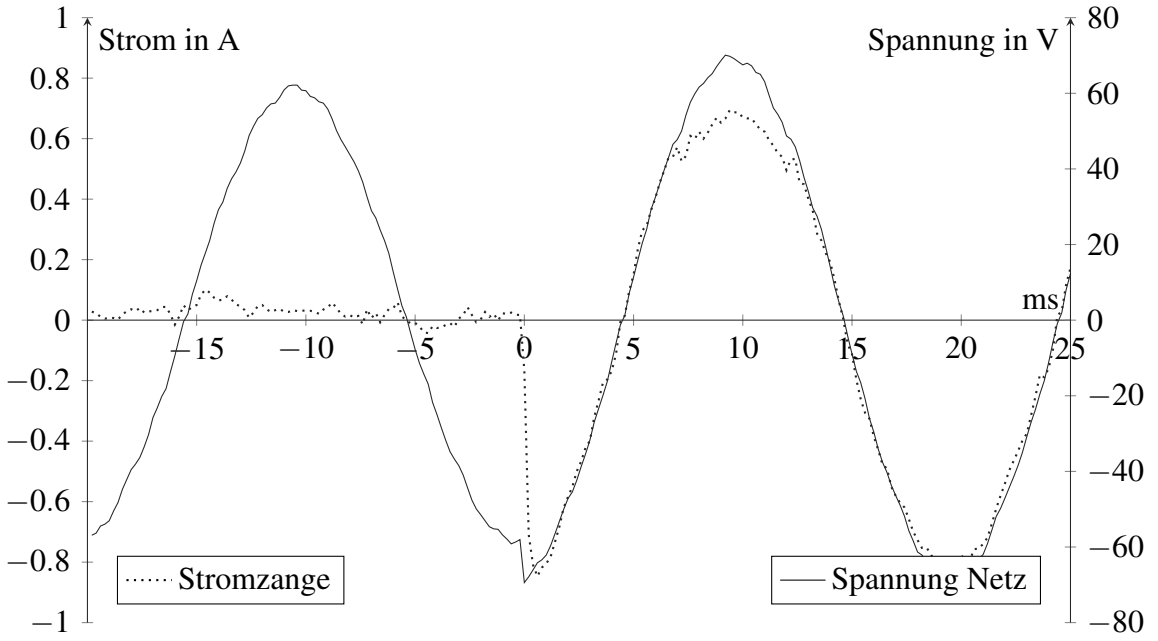

Abbildung 5.9: Sprungantwort von 0 zu 500 mA RMS

Gemäß VDE-AR-N 4105 müssen Erzeugungsanlagen ab 3,68 kVA Scheinleistung auch Blindleistung bereitstellen [8]. Es wird angenommen, dass zukünftig auch Erzeugungsanlagen im avisierten Leistungsbereich blindleistungsfähig sein müssen. In Abbildung 5.10 wird diese Fähigkeit mittels eines eingestellten Wirkstroms von 500 mA und einem Blindstrom von 200 mA validiert. Durch Änderung des Vorzeichens bei der Sollwertvorgabe für den Blindstrom kann dieser auch nacheilend gestellt werden.

Abbildung 5.11 zeigt die auf $I = 300\,\text{mA}$ eingestellte Stromregelung bei einer reduzierten Netzspannung von 65 V und einer Zwischenkreisspannung von 150 V. Die mit einem Leistungsmessgerät ermittelte THD THD_U liegt bei etwa 1%. Nach EN 50160 beträgt das Maximum 8%, welches entsprechend eingehalten wird [19][20]. Abbildung 5.12 zeigt die Fast Fourier Transformation (FFT) des Netzstroms, also das Frequenzspektrum des Ausschnitts aus Abb. 5.11. Um 140 kHz ist hier ein Peak zu sehen, welcher der Schaltfrequenz der Leistungstransistoren entspricht. Mit dem Leistungsmessgerät wurden entsprechend die Oberwellen des Stroms aufgezeichnet (vgl. Abb. 5.13). Die THD THD_I über die ersten 40 Oberwellen beträgt 8,1%. Das Maximum nach IEC/EN 61727 wird mit 5% definiert [19][13]. Der Laborversuch wurde an einem Trenntransformator sowohl ohne Filter, sowie bei einem Bruchteil der Leistung des Wechselrichters (20 W) durchgeführt. Die Einhaltung der Grenzwerte muss daher in weiteren Versuchen bei direktem Netzanschluss erneut geprüft werden, ist aber nicht Bestandteil dieser Arbeit. Die grundsätzliche Funktionsfähigkeit Stromregelung zum Einspeisen wurde jedoch validiert.

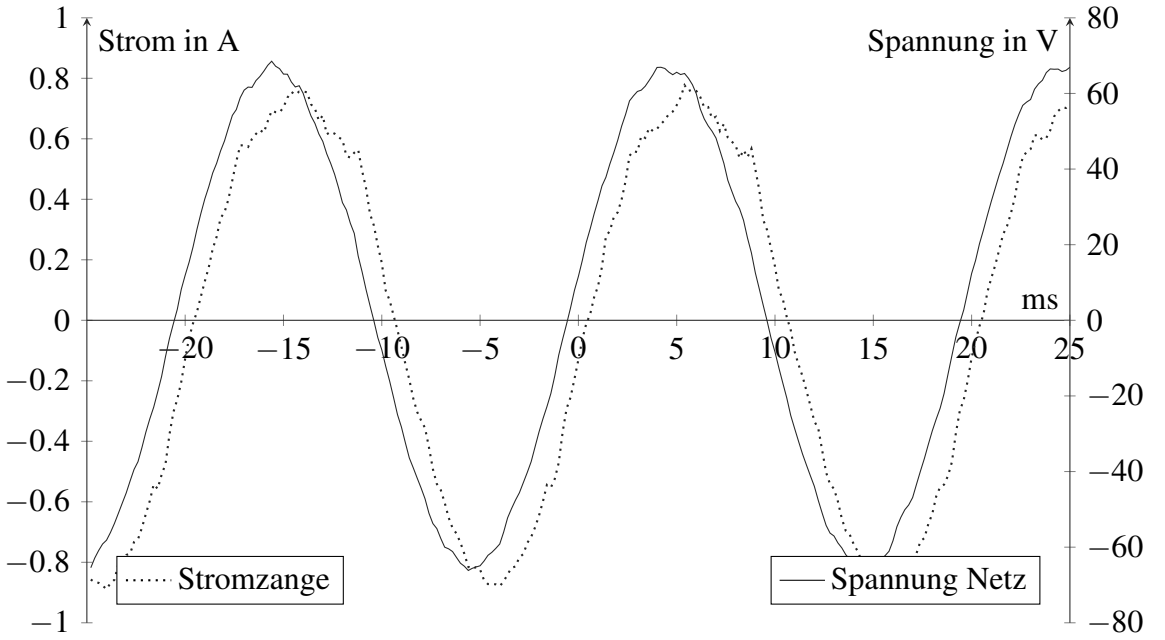

Abbildung 5.10: Stromregelung mit 500 mA Wirk- und 200 mA Blindstrom

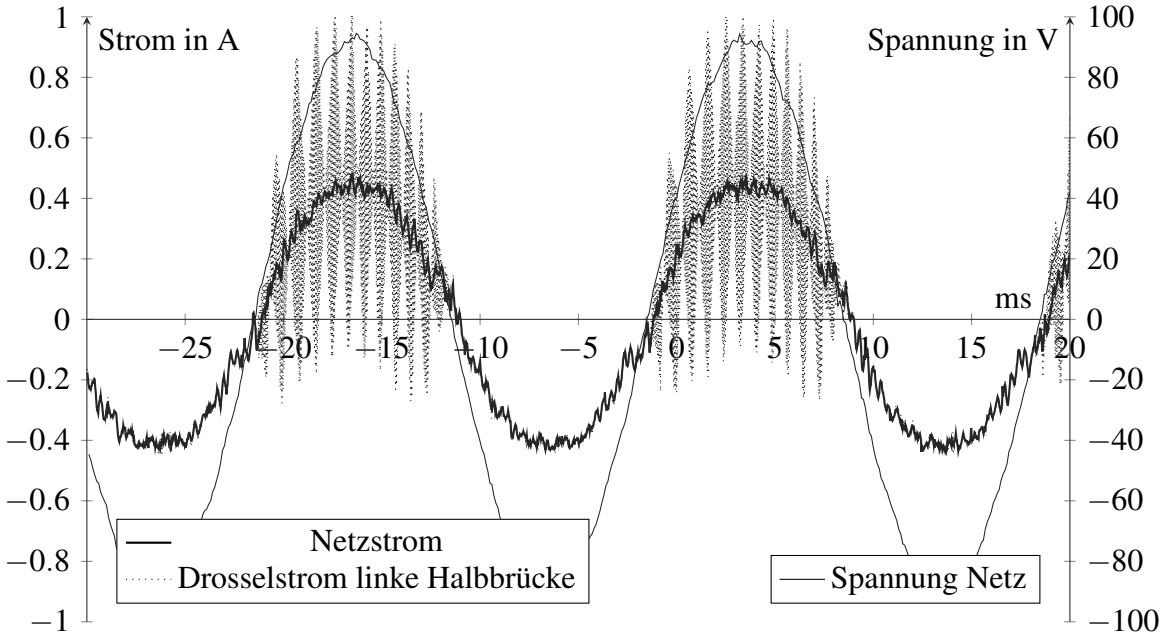

Abbildung 5.11: Stromregelung 300 mA mit AB-Modulation bei 65 V Wechselspannung und 150 V Zwischenkreisspannung

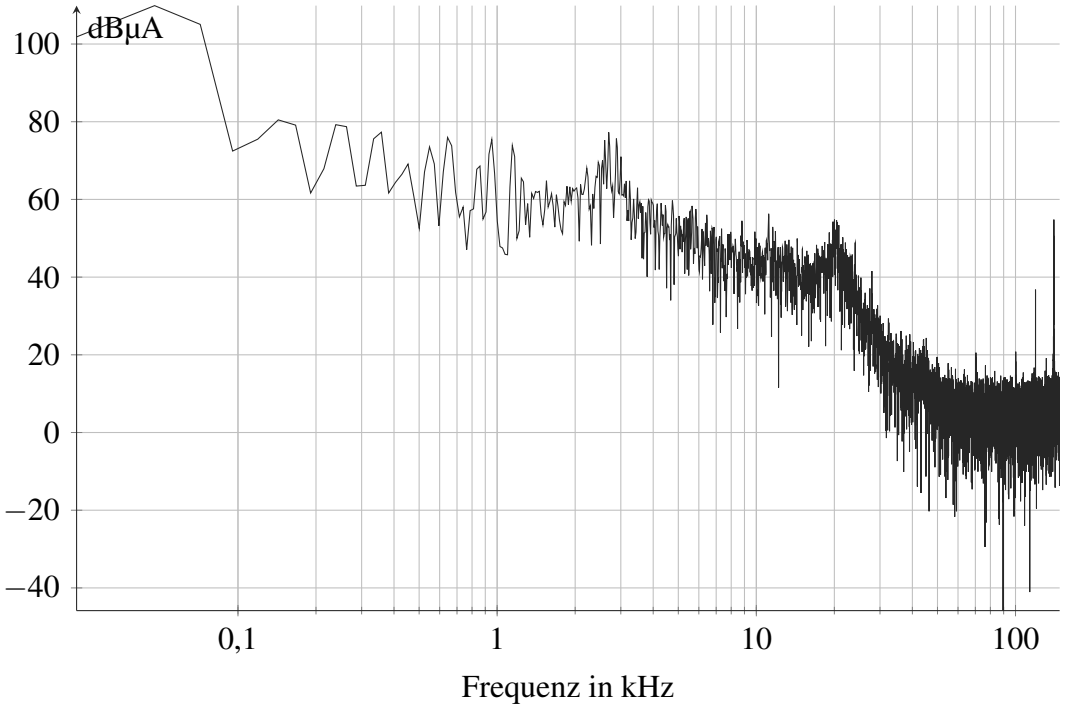

Abbildung 5.12: Spektrumsanalyse des Stroms aus Abb. 5.11

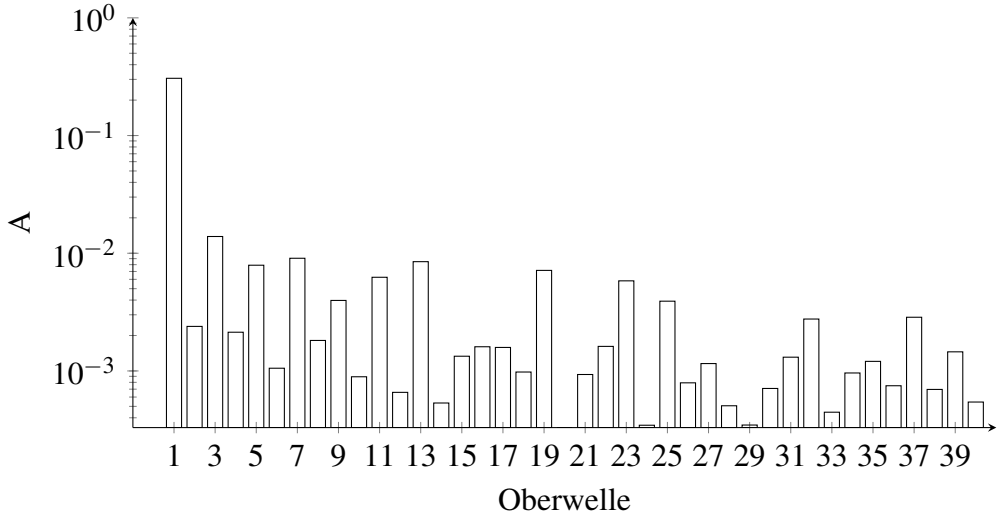

Abbildung 5.13: Strom Oberwellen bei 300 mA und 65 V ohne LCL-Filter, $f = 50\,\text{Hz}$

5.4 Generierung des PWM-Signals

Die Ausgabe der Wechselspannung erfolgt mittels PWM. Dabei wird ein Ausgang mit einem Signal konstanter Frequenz, aber veränderlichem Tastgrad (Duty-Cycle) angesteuert. Der Tastgrad bestimmt, welchen Anteil der Periodenlänge der Ausgang an- bzw. ausgeschaltet ist. Ein mit 50% Duty-Cycle angesteuerter Treiber an einer 100 V Spannungsversorgung erzeugt einen Spannungsmittelwert von 50 V an dessen Ausgang. Dieses Prinzip wird im Wechselrichter genutzt, um aus einer konstanten Gleichspannung die sinusförmigen Halbwellen für die Wechselspannung zu generieren.

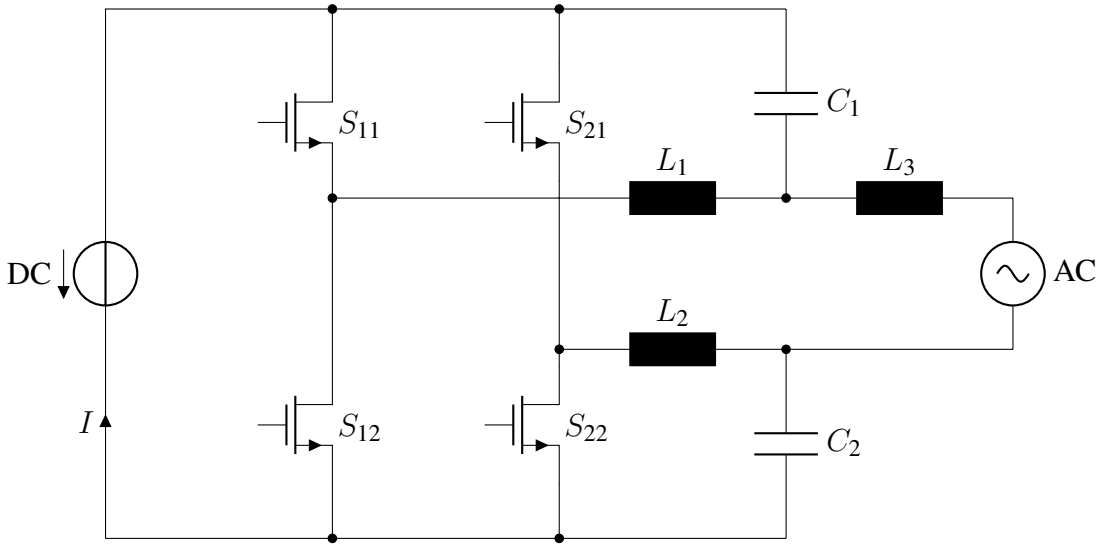

Abbildung 5.14: Aufbau des Wechselrichters

Der Wechselrichter besteht, wie in Abbildung 5.14 gezeigt, aus zwei sogenannten Halbbrücken, die jeweils einen Treiber für die positive (S_{11}, S_{21}) und negative Gleichspannungsseite (S_{12}, S_{22}) aufweisen. Um keinen Kurzschluss zu verursachen, dürfen die beiden Schalter einer Halbbrücke nicht zum gleichen Zeitpunkt aktiviert werden. Zwar weisen die Leistungstransistoren nicht wie die in 5.3 beschriebenen Relais Verzögerungen im Millisekunden-Bereich durch elektromechanische Kopplung auf, unterliegen aber ebenso Verzögerungen und Steig- und Fallzeiten von wenigen Nanosekunden. Das bedeutet, dass nach dem Anlegen der Steuerspannung der Halbleiter einen Moment benötigt, bis die volle Leitfähigkeit erreicht wird respektive beim Ausschaltern wieder vollständig sperrt. Dementsprechend ist eine Ansteuerung des komplementär geschalteten Treibers mit einem invertierten Signal nicht möglich. Stattdessen muss zwischen der Deaktivierung des einen und Aktivierung des anderen Schaltglieds eine sogenannte Totzeit vorgesehen werden. Für die verwendeten Leistungstransistoren wurde diese auf 100 ns festgelegt.

In der *CubeIDE* lässt sich in der grafischen Oberfläche sowohl die Frequenz, als auch die Totzeit direkt konfigurieren. Die *STM32G4x4* Microcontroller bieten einen HRTIM, welcher dem Namen entsprechend eine deutlich höhere Auflösung als die alternativ verfügbaren Timer bietet. Der Prozessorgrundtakt von 170 MHz wird dabei um Faktor 32 auf eine Frequenz von 5,44 GHz gesteigert. Um die angestrebte PWM-Frequenz von $f_{\text{PWM}} = 140\,\text{kHz}$ zu erreichen, muss eine Periodenlänge von $^{5{,}44\,\text{GHz}}/_{140\,\text{kHz}} \approx 38857$ Takten eingestellt werden. Die

Totzeit von 100 ns ergibt sich mit dem Faktor 8 als $170\,\text{kHz} \cdot 8 \cdot 100\,\text{ns} \approx 136$.

Von der Stromregelung wird der PWM-Steuerung der Tastgrad D als Wert zwischen -1.0 und 1.0 übergeben. Für die nachfolgende Ansteuerung der Treiber wird das Vorzeichen als `phase_dir` gespeichert (1 = positiv, 0 = negativ) und der Absolutwert gebildet. Der Timer benötigt für den Umschaltzeitpunkt einen Vergleichswert, definiert in Anzahl an Takten nach Periodenbeginn. Durch Multiplikation des Tastgrades mit der Periodenlänge lässt sich dieser Wert ermitteln $n_{\text{CMP}} = D \cdot 38857$.

Insgesamt sind die vier verschiedenen Modulationsarten aus Abbildung 5.15 zur Generierung der Wechselspannung implementiert. Bei der A-Modulation wird zunächst auf der linken Halbbrücke (S_{11}, S_{12}) eine positive Halbwelle generiert, während die rechte Halbbrücke (S_{21}, S_{22}) auf die negative Gleichspannung schaltet. Für eine negative Spannung wird nachfolgend die rechte Halbbrücke auf das positive Zwischenkreispotential geschaltet und über die linke die positive Gleichspannung abzüglich der Halbwelle moduliert. Dadurch ergibt sich eine Umschaltung der Schalter S_{21} und S_{22} mit der Netzfrequenz, während S_{11} und S_{12} mit 140 kHz Pulsweitenmodulation betrieben werden. Abbildung 5.14 zeigt den Aufbau für AB- und BA-Modulation, für die A-Modulation ist L_2 zu entfernen. Die B-Modulation invertiert die Signale und Halbbrücken, sodass die linke Halbbrücke mit Netzfrequenz schaltet. Entsprechend muss für diese Variante L_1 entfernt werden.

Bei der in dieser Arbeit vornehmlich genutzten AB-Modulation wird zuerst auf der linken Halbbrücke eine Halbwelle moduliert und dann auf der rechten Halbbrücke. Die jeweils andere Brücke wird auf die negative Gleichspannung geschaltet. Für die BA-Modulation wird die Reihenfolge der Halbwellenansteuerung getauscht und statt der negativen wird die positive Gleichspannung als Referenz genutzt, sodass sich für die Halbwellen ein Tastgrad von $1 - D$ ergibt. Im Programmcode lässt sich über eine Definition die Modulationsart ändern, sodass FA4 erfüllt ist.

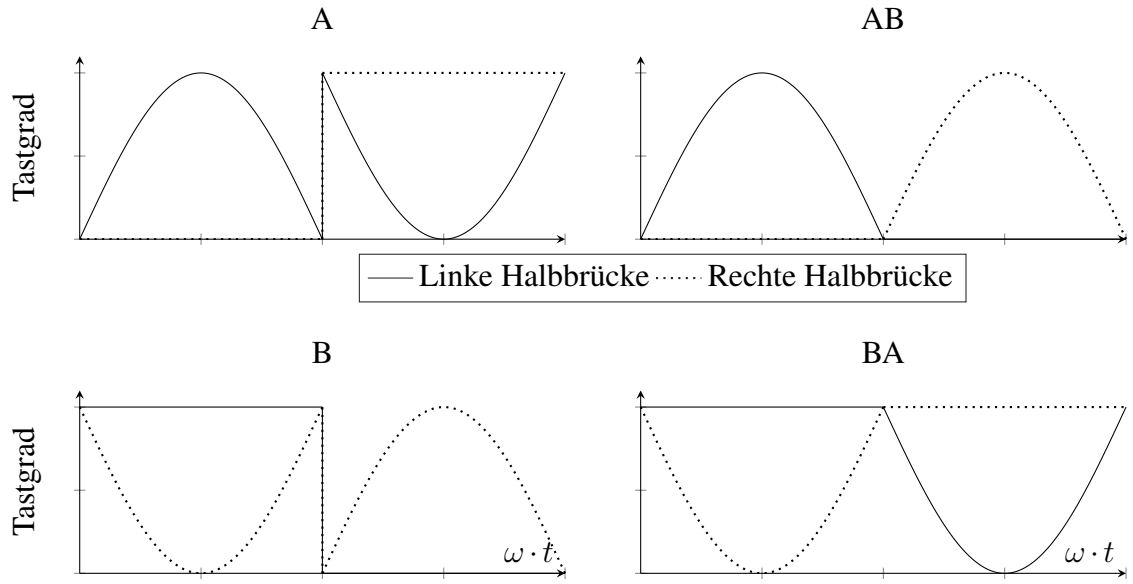

Abbildung 5.15: Halbbrückenansteuerung der unterschiedlichen Modulationsarten

Bei Tests zeigten sich sowohl bei sehr niedrigen als auch hohen Tastgraden Artefakte, trotz

Anforderung von nahezu 0% eine oder mehrere Perioden eine Ansteuerung mit 100%. Die Ursache lässt sich darauf zurückführen, dass bei einer Vorgabe kleiner als `0x60` (96) der Timer nicht mehr korrekt den Ausgang stellen kann. Dokumentiert ist dieses Verhalten in Datenblättern des Herstellers nicht, jedoch lässt sich in der grafischen Oberfläche der *CubeIDE* kein Vergleichswert kleiner als `0x60` einstellen. Als Gegenmaßnahme werden daher alle Werte kleiner als 96 auf 0 gesetzt. Der minimal einstellbare Tastgrad ergibt sich damit als $^{96}/_{38857} = 2,5‰$.

Wie in 5.3 beschrieben, müssen Nulldurchgänge der Wechselspannung erkannt werden. Dadurch, dass der Vergleichswert in Nulldurchgängen durch vorherig genannte Maßnahme exakt 0 beträgt, lässt sich dies als Indikator nutzen. Basierend darauf können die beschriebenen Zähler gestartet werden.

5.5 Verringerung der Ausführungszeit

Für das schnelle Einrasten der PLL und eine stufenarme Stromregelung ist auch die Ausführungszeit und damit die -frequenz von entscheidender Bedeutung. Besonders komplexe Berechnungen, wie etwa Winkel- oder Wurzelfunktionen, benötigen eine signifikante Anzahl an Taktzyklen. Der vorliegende Microcontroller mit einem *Cortex-M4* Prozessor benötigt 416 respektive 405 Takte für `sinf()` und `cosf()`, `sqrtf()` wird mit 58 Takten angegeben [21].

Bei einer angestrebten PLL-Frequenz gleich der PWM-Frequenz, also 140 kHz, und einer 170 MHz Taktfrequenz des Controllers ergeben sich rund 1214 Takte für alle Berechnungen der PLL. Eine einfache Ausführung der Winkelfunktionen würde nach Tabelle 5.1 entsprechend $^{405}/_{1214} \approx 34\%$ der insgesamt zur Verfügung stehenden Zyklen belegen. Nach FA10 muss neben der PLL und Stromregelung auch noch das MPPT mit 1 Hz ausführbar sein.

5.5.1 CORDIC für Winkelfunktionen

Die *STM32-G4*-Serie umfasst auch eine CORDIC-Einheit, weshalb, wie in Abschnitt 4.1.1 beschrieben, diese Prozessorserie ausgewählt wurde. Der Vorteil der CORDIC liegt einerseits in der geringeren Berechnungszeit, als auch darin, dass die Berechnungen nebenläufig in der Hardware-Einheit ablaufen können.

Um die Berechnungen in Hardware zu vereinfachen, nutzt die CORDIC das Q31-Festkommaformat. Für Sinus und Cosinus ergibt sich damit bei der Eingabe der Wertebereich $[-1,1)$ anstatt $[-\pi,\pi)$, wobei die *math.h* Funktion beliebige Werte verarbeiten kann. Übergebene Variablen müssen folglich durch π dividiert werden und sichergestellt werden, dass der Wertebereich eingehalten wird. Da das Ergebnis der Winkelfunktionen nur $[-1,1]$ annehmen können, ergibt sich bei der Ausgabe bis auf die Abweichung bei $+1$ um 1 LSB keine Einschränkung.

Die reine Berechnungszeit der CORDIC wird mit 29 Taktzyklen angegeben [21]. Innerhalb dieser Zeit werden für einen übergebenen Winkel sowohl Sinus, als auch Cosinus ermittelt. Jedoch muss – sofern nicht vollständig im Festkommaformat Berechnungen vorgenommen werden – die Fließkommazahl von `float` in das Q31-Format und wieder zu-

Operation	**Code**	**Taktzyklen**
Sinus & Kosinus	C: `sinf()` & `cosf()`	416
Sinus & Kosinus	C: `arm_sin_cos_f32()`	405
Addition	ASM: `VADD.F32`	1*
Subtraktion	ASM: `VSUB.F32`	1*
Multiplikation	ASM: `VMUL.F32`	1*
Division	ASM: `VDIV.F32`	14*
Multiplikation & Addition (MAC)	ASM: `VMLA.F32`	3*
Negieren	ASM: `VNEG.F32`	1*
Negieren & Multiplizieren	ASM: `VNMUL.F32`	1*
Quadratwurzel	C: `sqrtf()`	58
Quadratwurzel	ASM: `VSQRT.F32`	14*
Typwandlung	ASM: `VCVT.F32.X` / `VCVT.X.F32`	1*
Laden	ASM: `VLDR.32`	$1+N$
Speichern	ASM: `VSTR.32`	$1+N$

* Je ein Taktzyklus länger, wenn das Ergebnis im nächsten Schritt genutzt wird

Tabelle 5.1: Taktzyklen für unterschiedliche Float-Operationen auf dem *STM32* [22] [21]

rück gewandelt werden. Mit einer einfachen Multiplikation (`VMUL.F32` 1 Takt) respektive Division (`VDIV.F32` 14 Takte) durch 2^{31} und Typecasting (`VCVT.F32` 1 Takt) ist die Umwandlung zwischen den beiden Formaten nicht zeitintensiv. Für die Umwandlung von `float` zu Q31 werden damit im Idealfall 3 $((1+1)+1)$ Takte und die Rückwandlung 16 $((14+1)+1)$ Takte benötigt. Der zusätzliche Takt bei Multiplikationen, Divisionen, Additionen und Subtraktionen ergibt sich aus dem Umstand, dass die Hardware auf Pipelining optimiert ist und sobald das Ergebnis einer Berechnung direkt im nächsten Schritt verarbeitet werden soll, entsprechend einen Taktzyklus auf das Vorliegen eben dieses gewartet werden muss. Die Ladezyklen von Variablen oder Konstanten aus dem Speicher in die Register wird hier nicht betrachtet. Durch Compileroptimierungen (vgl. Abschnitt 5.5.2) wird die Berechnung jedoch als Umwandlung zwischen Gleit- und Festkommaformat identifiziert und durch die entsprechend einen respektive zwei Takte benötigende Funktion `VCVT.S32.F32 S15,S15,#31` beziehungsweise `VCVT.F32.S32 S0,S0,#31` ersetzt (vgl. Tabelle 5.1). Gegenüber den 416 Takten der Softwareimplementierung der Winkelfunktionen ist hier ein signifikanter Vorteil ersichtlich. Da der Microcontroller eine Floatingpoint-Einheit besitzt, die Berechnungen überwiegend gleich schnell, wie Integeroperationen durchführt, wird auf den Einsatz von Festkommazahlen abseits der CORDIC verzichtet. Multiplikationen oder Divisionen in dem Format sind durch notwenige Bitshifts teilweise sogar zeitintensiver, für die Berechnung der Quadratwurzel wäre die Rechenzeit sogar mehr als fünffach so hoch, wie in Abb. 5.16 zu sehen.

Ein weiterer Vorteil der CORDIC ist darüber hinaus, dass die Hardwareeinheit die Berechnungen parallel zum sonstigen Programmcode ausführen kann. Die tatsächlich für den Controller verbrauchten Takte beschränken sich somit auf die Umwandlung zwischen den Datentypen und dem Schreiben und Lesen der CORDIC-Register. Während der eigentlichen 29 Takte kann der Prozessor andere Berechnungen erledigen und das Ergebnis zu einem späteren Zeitpunkt aus den Registern lesen.

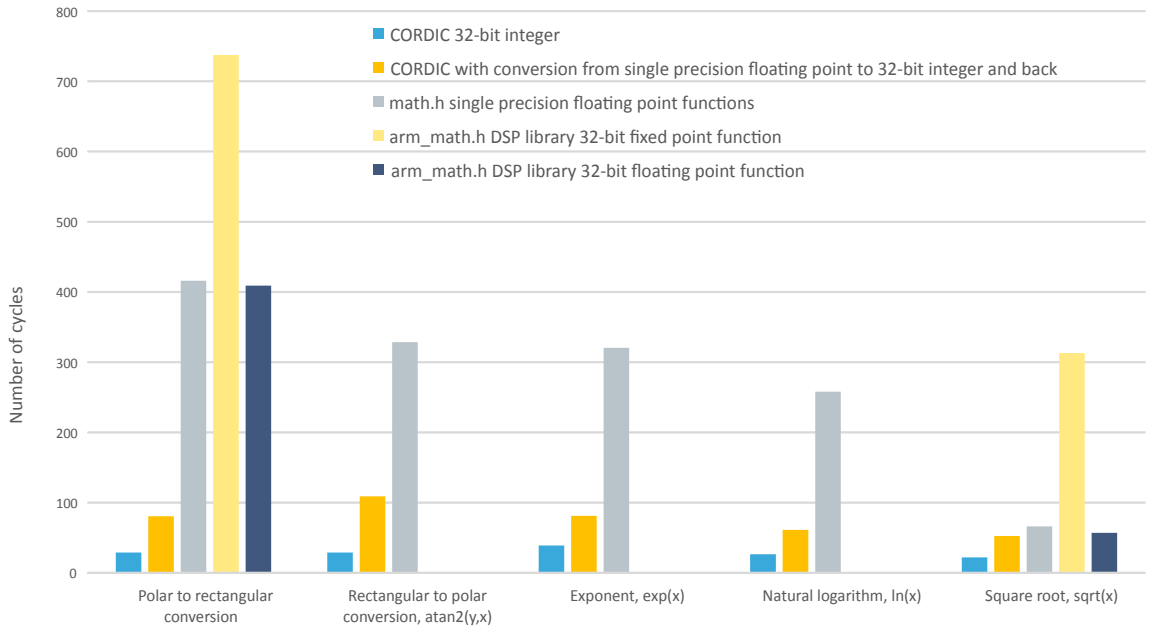

Abbildung 5.16: Vergleich der Ausführungszeit mathematischer Operationen mit CORDIC und *math.h* [21]

Konkreten Einsatz findet die CORDIC für die Implementierung der PLL in der dq-Transformation. Da der für die Berechnung von v_d und v_q benötigte Winkel θ je einen PLL-Takt zuvor ermittelt wird, lassen sich auch direkt anschließend die Register mit dem Winkel beschreiben. Zwischen dem Übergeben von θ an die CORDIC und der Nutzung von $\sin(\theta)$ und $\cos(\theta)$ in der dq-Transformation muss noch die QSG-PLL ausgeführt werden. Die für die Berechnung notwendigen 29 Takte werden somit überschritten, wodurch sichergestellt wird, dass das Ergebnis bei der Ausführung der Transformation vorliegt. Durch den Einsatz der CORDIC können somit etwa 370 Takte eingespart werden.

Die CORDIC lässt sich darüber hinaus auch für Direct Memory Access (DMA) konfigurieren. Anstatt einzelne Werte nacheinander in die Register zu schreiben und wieder zu lesen, kann hierbei ein Speicherbereich mit Variablen an die CORDIC übergeben werden. Im Gegensatz zu anderen Hardwareeinheiten des Controllers, die bei einer Änderung der Variablen automatisch getriggert werden, muss die Berechnung wieder durch Software gestartet werden. Für die PLL Implementierung wird je Zyklus nur ein Winkel ermittelt, der durch die CORDIC verarbeitet werden muss. Dadurch ergibt sich beim Einsatz der DMA-Konfiguration ein Overhead gegenüber dem einfachen Schreiben und Lesen in die Register.

Bei der Quadratwurzel sind mit der Softwareimplementierung `sqrtf()` 58 Takte notwendig, bei der CORDIC hingegen 23 Takte. Berücksichtigt man jedoch die notwendige Zeit inklusive der Umwandlung zwischen den Datentypen, so werden 53 Takte (davon die 23 der CORDIC parallelisiert) angegeben [21]. Im Berechnungsablauf ergeben sich Abhängigkeiten, sodass das Ergebnis der Quadratwurzel direkt in der nächsten Operation genutzt wird. Innerhalb der 23 Takte der CORDIC sind für die kritische Ausführung der PLL somit keine anderen sinnvollen Berechnungen möglich. Für die PLL ergäben sich somit lediglich eine Reduktion um 5 Takte. Selbst unter Betrachtung des Vorteils von 58 gegenüber

$53 - 23 = 30$ Takten, falls in der Zeit andere Aufgaben ausgeführt werden, muss bedacht werden, dass die CORDIC zwischen den Modi Winkelfunktion und Quadratwurzel umkonfiguriert werden muss. Da lediglich eine Quadratwurzel in der PLL zu berechnen ist, lohnt sich der Einsatz hierfür nicht. Wie im nachfolgenden Kapitel erklärt, kann die Implementierung von der *math.h*-Bibliothek durch den Compiler auf die *Cortex-M4* Assembler-Operation `VSQRT.F32` optimiert werden, die nach Tabelle 5.1 nur 14 Takte und damit weniger als mit der CORDIC benötigt. Zudem kann diese Operation nach Vereinfachungen in 5.5.4 gänzlich wegfallen.

5.5.2 Ersatz von Divisionen durch Multiplikationen, Nutzung der FPU, Compileroptionen

Wie im vorherigen Abschnitt beschrieben, benötigen Divisionen des Microcontrollers mit *Cortex-M4* 14 Takte, während Multiplikationen nur einen Takt benötigen. Da viele Berechnungen als Divisor eine Konstante haben, lässt sich die Berechnung von x/c zu $x \cdot c'$ umformen mit $c' = 1/c$. In der *math.h* sind bereits Konstanten, wie etwa `M_1_PI` für $1/\pi$ definiert. Die Verwendung der Bibliothek ist auf dem Controller jedoch nicht ohne weiteres taktoptimal nutzbar.

math.h definiert die Konstanten `M_PI` mittels `#define`, also als reine Zahlenfolge ohne Datentyp. Sofern nicht explizit eine Gleitkommazahl mit einem zusätzlichen `f` (beispielsweise `1.0f`) ausgezeichnet wird, so wird diese nicht als `float` sondern `double` interpretiert. Einerseits bietet der Microcontroller nur eine Floating Point Unit (FPU) für Gleitkommazahlen mit einfacher Genauigkeit, also `float`, womit alle Berechnungen mit Gleitkommazahlen doppelter Genauigkeit nicht vollständig oder gar nicht in Hardware, sondern in Software realisiert werden müssen. Andererseits sind Operationen von unterschiedlichen Datenformaten nur mit voriger Umwandlung, also zusätzliche Instruktionen für den Controller, möglich. Um diese Problematik zu umgehen, ist eine explizite Angabe des Datentyps notwendig, etwa `theta.angle * (float)M_1_PI - 1.0f`.

Der Vergleich von
`float_to_fixed(theta.angle / M_PI - 1.0f);`
in Quellcode 5.1 und
`float_to_fixed(theta.angle * (float)M_1_PI - 1.0f);`
in Quellcode 5.2 zeigt das immense Optimierungspotential. Die Multiplikation $\theta \cdot 1/\pi$ mit einfacher Genauigkeit und anschließender Subtraktion von 1 benötigt `vmul.f32` und `vsub.f32` mit jeweils einem Taktzyklus. Bei der Berechnung mit $\theta \div \pi$ und doppelter Genauigkeit werden die Softwarefunktionen `__extendsfdf2` (`float` zu `double`), `__divdf3` (Division von `double` durch `double`), `__subdf3` (Subtraktion) und `__truncdfsf3` (`double` zu `float`) aufgerufen. Die generelle Interpretation von Gleitkommazahlen mit einfacher Präzision kann man jedoch auch an den Compiler mittels `-fsingle-precision-constant` auslagern, wodurch der Code aufgeräumter ist und eine versehentliche Operation mit doppelter Präzision nahezu ausgeschlossen wird. Eine andere, auch zusätzliche Option ist `-Wdouble-promotion`, wodurch Warnungen ausgegeben, aber keine automatischen Optimierungen durchgeführt werden. Dadurch lassen sich Operationen, die dennoch `double` nutzen, identifizieren und bei Bedarf auf `float` anpassen.

```
1  it      ge
2  vsubge.f32      s3, s3, s15
3  vstr    s3, [r3]
4  vmoveq  r0, s3
5  bleq    0x8000598 <__extendsfdf2>
6  add     r3, pc, #156    ; (adr r3, 0x8002b38 <pll_srf+416>)
7  ldrd    r2, r3, [r3]
8  bl      0x800089c <__divdf3>
9  ldr     r3, [pc, #224]  ; (0x8002b84 <pll_srf+492>)
10 movs    r2, #0
11 bl      0x80002d8 <__subdf3>
12 bl      0x8000bf8 <__truncdfsf2>
13 vpop    {d8-d9}
14 vmov    s15, r0
15 ldr     r3, [pc, #208]  ; (0x8002b88 <pll_srf+496>)
16 vcvt.s32.f32    s15, s15, #31
```

Quelltext 5.1: Berechnung ohne explizites Casting und mit Division

```
1  it      ge
2  vsubge.f32      s3, s3, s15
3  vpop    {d8-d9}
4  vmul.f32        s15, s3, s14
5  vmov.f32        s14, #112       ; 0x3f800000  1.0
6  vsub.f32        s15, s15, s14
7  vstr    s3, [r3]
8  vcvt.s32.f32    s15, s15, #31
```

Quelltext 5.2: Optimierung mittels Casting und Multiplikation

In der vorangegangen Arbeit zur Implementierung auf einem *STM32* wurden keinerlei Optimierungen durch den Compiler erzwungen. Die Generierung der Binärdateien wurde immer mit `-O0` ausgeführt, sodass der Code sozusagen wortwörtlich in Instruktionen übersetzt wird. Grundsätzlich ist dies die Standardeinstellung und führt beim Debugging zu deterministischen Ergebnissen, da jede Variable für sich untersucht werden kann. Während `-O2` den Code nahezu, bzw. `-Os` gänzlich ohne Speichernachteil auf Geschwindigkeit optimiert, kann die Stufe `-O3` einen größeren Speicherbedarf generieren, die Ausführungszeit wird jedoch weiter gesenkt. Da das Projekt auf dem Microcontroller weniger als 10% des verfügbaren Speichers belegt und eine schnelle Ausführungszeit eines der Hauptziele ist, kann selbst mit `-O3` noch auf die kostengünstigere Variante des Chips mit weniger Speicher gewechselt werden. Wie bereits im vorigen Abschnitt erwähnt, wird auch die Quadratwurzelfunktion durch die Einstellung auf die Funktion des *Cortex-M4* übersetzt, sodass der Bedarf von 58 auf 14 Takte sinkt.

Darüber hinaus steht die Optimierung `-Ofast` zur Auswahl, diese kann die Ausführungszeit zwar verringern, ist jedoch nicht mehr Standardkonform und kann zu unerwartetem

Verhalten führen. Für den am Stromnetz angeschlossenen Umrichter ist die korrekte Ausführung jedoch sicherheitsrelevant. Die Geschwindigkeit wird daher nicht auf Kosten der Sicherheit optimiert, sondern mit `-O3` kompiliert.

Die Optimierungen durch den Compiler sind nicht nur für die Ausführungszeit vorteilhaft. Dadurch, dass etwa `M_TWOPI * F_MIN * F_GRID` ausgeschrieben im Quellcode erhalten bleibt und nicht mit einer vorberechneten Hilfskonstanten ausgetauscht wird, ist ein direkter Vergleich mit dem Blockschaltbild möglich. Für nachfolgende Arbeiten und Personen, die eher mit *Simulink* vertraut sind, erhöht sich dadurch die Lesbarkeit und NFA4 wird erfüllt.

5.5.3 Vereinfachung des Bandpass-Filters der QSOGI-QSG

Die QSG der QSOGI-PLL ist mit einem Bandpass-Filter für ω ausgestattet. Der Filter ist entsprechend auf die nominale Netzfrequenz von 50 Hz eingestellt, bietet jedoch eine Bandbreite von ebenfalls 50 Hz, also deckt insgesamt einen Frequenzbereich von 25 bis 75 Hz ab. Da für das Photovoltaikmodul jedoch nur der Arbeitsbereich zwischen 47,5 Hz und 51,5 Hz vorgesehen ist, kann die Filterfunktion vernachlässigt werden und ω entsprechend für die nominale Netzfrequenz eingestellt werden. Von ω abhängige Variablen a_1, a_2, b_0 und ω' können infolgedessen als Konstanten definiert werden und somit 30 Takte je PLL-Durchlauf eingespart werden. Das im Blockschaltbild 5.17 von der SRF-PLL zurückgeführte ω wird durch $2 \cdot \pi \cdot 50\,{}^{rad}/_{\mathrm{s}}$ ersetzt.

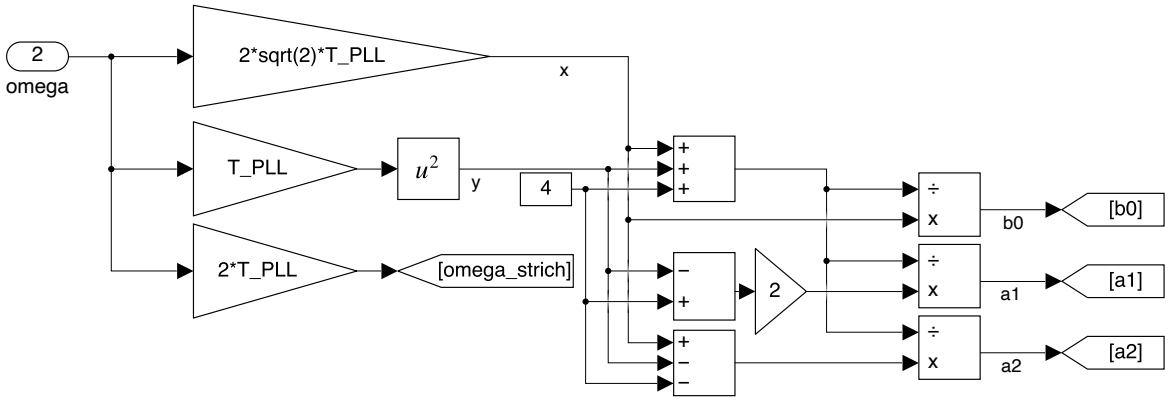

Abbildung 5.17: Bandpass der QSOGI-PLL

5.5.4 Einheitsvektor der SRF-PLL ohne Wurzelberechnung

Wie in Abschnitt 5.2.3 erläutert, wird für die Ausregelung der q-Komponente deren Einheitsvektor genutzt. Da die Winkelschritte von ω bei entsprechend hoher Ausführungsfrequenz der PLL nahe 0° liegen, lässt sich vereinfachen, dass d der Vektorlänge $|\vec{v}|$ entspricht. Im *Simulink*-Modell ist daher die Vereinfachung von $error_{\mathrm{SRF}} = {v_{\mathrm{q}}}/{\sqrt{v_{\mathrm{q}}^2+v_{\mathrm{d}}^2}}$ zu $error_{\mathrm{SRF}} = {v_{\mathrm{q}}}/{v_{\mathrm{d}}}$ vorgeschlagen. Die Einsparungen sind in diesem Fall zwei Multiplikationen, eine Addition und Quadratwurzel, die insgesamt zu 20 Takten weniger Prozessorlast führt.

Bei ersten Tests zeigte sich, dass die PLL mit dieser Vereinfachung nicht immer korrekt einrastet. Anstatt der Phasenverschiebung von θ zum Phasenwinkel der Netzspannung mit ${\pi}/{2}$

nacheilend rastete die PLL auch mit $\pi/2$ voreilend ein. In den Simulationen zeigte sich dieses Verhalten jedoch nur mit dem in *Simulink* implementierten Codeblock des C-Programms, nicht jedoch mit dem Blockschaltbild.

Variablen von Codeblöcken in *Simulink* werden nur beim Kompilieren initialisiert. Stoppt man die Simulation und startet einen weiteren Durchlauf, so behalten die Variablen die Werte des letzten Durchlaufs, anstatt einen Reset-Zustand einzunehmen. Die *Simulink*-Variablen werden durch eine Initialisierungsfunktion bei jeder Simulation neu gesetzt. Zudem startet der Spannungsverlauf in dem Modell immer mit einer positiven Halbwelle, also einem Phasenwinkel von $\theta = 0°$.

Zur Analyse der Fehlfunktion wurde die Spannung invertiert, sodass eine negative Halbwelle respektive ein Phasenwinkel $\theta = 180°$ zu Beginn anlag. Durch diese Modifikation wurde die gleiche Fehlfunktion, die zufällig mit dem Codeblock auftrat, provoziert. Während die Quadratwurzel im nicht vereinfachten Term immer ein positives Ergebnis liefert und das Vorzeichen dementsprechend nur von v_q abhängt, ergibt sich im zweiten Term eine Wechselwirkung der Vorzeichen von v_d und v_q. Da die Länge des Vektors nie negativ werden kann, ist eine einfache Umformung zu $error_\mathrm{SRF} = {v_\mathrm{q}}/{|v_\mathrm{d}|}$ notwenig und ergibt eine zuverlässige Einrastung mit den gewünschten $\pi/2$ nacheilend.

5.5.5 Ergebnisse der vorgenommenen Änderungen

Trotz der diversen Vereinfachungen von Algorithmen, Code-Aufbau und Compilereinstellungen werden volle 140 kHz PLL-Frequenz nicht erreicht. Insgesamt konnte die Ausführungszeit jedoch von knapp 85 000 μs (12 kHz) auf etwa 7500 μs, also ein Zehntel, und damit nur wenig mehr als $T_\mathrm{PWM} = 7142\,\mu$s reduziert werden. Dies umfasst bereits die in 5.10 beschriebene Behandlung von Fehlern und Grenzwertüberschreitungen. Dabei muss auch beachtet werden, dass neben der PLL und Stromregelung noch die Kommunikationsschnittstelle und andere Dienste Prozessorzeit benötigen. Daher wurde festgelegt, dass die PLL mit halber Frequenz der PWM, also 70 kHz ausgeführt wird, was eine Prozessorauslastung von ${7500\,\mu\mathrm{s}}/{14\,285\,\mu\mathrm{s}} = 53\%$ bedeutet. Da FA2 jedoch nur 2 kHz als Minimum definiert, ist die Anforderung dennoch erfüllt.

Die hohe, 35-fache der geforderten Ausführungsfrequenz ist jedoch wichtig, um die Verluste des Wechselrichters gering zu halten. Weiterhin steht knapp die Hälfte der Prozessorzeit für andere Aufgaben zur Verfügung und erlaubt die Implementierung des weniger komplexen MPPT, damit ist auch FA10 erfüllt.

5.6 Ausgabe von Messwerten per DAC

In der Anforderungsanalyse wurde bereits die Notwendigkeit der Ausgabe von Variablen und Messwerten über die DACs erläutert (vgl. Abschnitt 3.2). Der Microcontroller stellt dafür drei Kanäle bereit, auf denen unterschiedliche Analogsignale ausgegeben werden können.

Die Kombinationen sind durch die Anzahl an Variablen, deren für die Messung relevanter Wertebereich und mit den drei Kanälen so zahlreich, dass eine Konfiguration über die

Schnittstelle als nicht sinnvoll bewertet wird. Zudem wird der DAC nur in der Entwicklungsphase genutzt, sodass die Anpassung, welcher Wert auf welchem Kanal ausgegeben wird, im Programm selbst erfolgt.

Die DACs bieten eine Auflösung von 12 Bit, also 4096 Einzelwerten. Für Variablen, die nur positive Werte annehmen, reicht die Skalierung auf den Bereich [0, 4095]. So kann beispielsweise θ aus der PLL nach der Normierung auf [0, 2) für die Ausgabe durch eine Multiplikation mit 2047 auf den Wertebereich des DAC erweitert werden. Für Variablen, die (symmetrisch) alternierende Signale, wie die Netzspannung, darstellen, muss zunächst der Nullpunkt auf den Mittelwert des DACs verschoben werden. Im Falle der Netzspannung beträgt der Maximalwert der Variable ± 358. Nach $4095/358 \cdot 2$ ergibt sich ein Skalierungsfaktor von etwa $5,72$. Der zur Ausgabe auf DAC 1 Kanal 1 respektive Testpunkt 1 benötigte Befehl lautet entsprechend

```
HAL_DAC_SetValue(TP1, DAC_ALIGN_12B_R,
2047 + get_voltage_grid() * 5.7f);
```

Im Quelltext *voyager_pll.c* sind in Kommentaren bereits einige Variablen beispielhaft implementiert, welche von Entwickelnden entweder direkt durch Entkommentieren oder als Vorlage für weitere Variablen genutzt werden. Die Anforderung FA12 wird somit erfüllt.

5.7 Kommunikationsschnittstelle

Während der Forschungsphase wird die auf dem Microcontroller laufende Software noch weiter angepasst, um etwa die Ergebnisse aus veränderten Regelparametern im Code zu persistieren. Dazu wird die am Nucleo-Board vorhandene USB-Schnittstelle genutzt, die neben dem Übertragen des Programms auch eine serielle Kommunikationsschnittstelle bietet. Somit muss für den Anschluss an einem Computer nur das bereits zum Programmieren verwendete Kabel angeschlossen werden und keine separate Hardware genutzt werden. Im fertigen Produkt soll die Kommunikation über ein Funkmodul realisiert werden. Daher ist die Schnittstelle entsprechend modular aufgebaut und kann später einfach durch eine andere (Teil-)Implementierung ersetzt werden.

Für die Übertragungsgeschwindigkeit wurden 115 200 baud mit 8N1 gewählt, also 8 Datenbits, je 1 Start- und Stoppbit, was in $115\,200\,\text{baud}/10\,\text{bit} = 11\,520\,\text{char}/\text{s}$ resultiert. Wie auch andere Dienste lässt sich die serielle Schnittstelle für DMA konfigurieren, dafür muss jedoch im Vorhinein die zu empfangende Datenmenge bekannt sein. Da die implementierten Kommandos unterschiedlich lang sind und sich durch Übertragungsfehler die Leseposition (erwartet werden beispielsweise 10 Zeichen, übertragen nur 9 und die Signailisierung, dass erfolgreich gelesen wurde, erfolgt beim ersten Zeichen der neuen Nachricht) verändern kann, ist diese Variante hier nicht sinnvoll. Alternativ lässt sich auch blockierend lesen, also es wird auf eine Nachricht gewartet. Dies würde allerdings den Ablauf von nicht kritischen Aufgaben entsprechend pausieren. Da nur unregelmäßig Befehle übertragen werden, würde der Hauptprozess im schlimmsten Fall komplett unterbrochen werden, sollten keine neuen Werte empfangen werden. Entsprechend ist auch diese Option für den Anwendungsfall nicht praktikabel. Als dritte Möglichkeit kann der Microcontroller den Empfang eines neuen Zeichens per Interrupt signalisieren. Ein Interrupt, basierend auf der Baudrate, tritt nur mit 11,52 kHz und damit geringerer Frequenz als die Ausführung der PLL auf. Somit kann die

Nachricht	Funktion	Beispiel	Ergebnis
`Cxxxx`	Fehler zurücksetzen	`C000A`	Setzt Fehler `0x0008` und `0x0004` zurück
`Eb`	Halbbrücken ein-/ausschalten	`E1`	Halbbrücken aktivieren
`Rb`	Relais schließen/öffnen	`R1`	Relais schließen
`Pnnn`	P-Anteil des Stromreglers mit $n \cdot 0,1$ multiplizieren	`P015`	$PI_\mathrm{P} \cdot 1,5$
`Knnnnn`	I-Anteil des Stromreglers mit $n \cdot 0,1$ multiplizieren	`K00002`	$PI_\mathrm{P} \cdot 0,2$
`Ipp;qq`	RMS Wirk- und Blindstrom-vorgabe setzen	`I20;05`	$I_\mathrm{p} = 2,0$ und $I_\mathrm{q} = 0,5$

Tabelle 5.2: Kommandos für die Kommunikationsschnittstelle

PLL die Bearbeitung der Nachrichten auch unterbrechen, ohne dass Zeichen aufgrund eines verpassten Interrupts nicht gelesen werden.

Die implementierten Kommandos sind in Tabelle 5.2 aufgeführt. Als Signal, dass die Nachricht zu Ende ist, muss das Newline-Symbol `\n` am Ende eingefügt werden. Da das Lesen von Gleitkommazahlen aus Zeichenketten aufwendig und nur als nicht in Hardware unterstütztem `double`-Format möglich ist, werden diese Werte als Ganzzahlen übertragen und vom Controller mit einem Faktor in eine rationale Zahl im `float`-Format überführt. Messwerte und Fehlerstatus (FA6) vom Controller werden als `Name=Wert` durch Semikolon getrennt und mit einem Newline-Zeichen terminiert übertragen.
Tabelle 5.2 listet die implementierten Kommandos auf. Mit den Befehlen `I...`, `P...` und `K...` sind, wie in FA5 gefordert, Sollwertvorgaben und Regelparameter einstellbar.

Da der Microcontroller von der Versorgungsspannung des Wechselrichters und der Debugger, welcher für Anbindung serielle Schnittstelle über USB zuständig ist, über den angeschlossenen Computer versorgt wird, lassen sich die beiden Einheiten unabhängig voneinander einschalten. Allerdings hat das Abschalten der Versorgungsspannung für den Microcontroller bei gleichzeitigem aufrechterhalten der Spannung am Debugger zur Folge, dass keine Nachrichten mehr vom Controller über USB versendet werden. Die Empfangsrichtung, also Kommandos an den Controller senden, funktioniert weiterhin. Um die Kommunikation in beide Richtungen wiederherzustellen, muss der Debugger stromlos geschaltet werden, beispielsweise durch Abziehen des USB-Kabels, während der Controller aktiv ist.

5.8 Zeitlicher Ablauf des Programms

Für die Messwerterfassung ist der genaue Zeitpunkt innerhalb des PWM-Pulses relevant. Wenn bei gleichbleibendem Tastgrad zu unterschiedlichen Zeitpunkten abgetastet wird, variiert infolgedessen der ermittelte Wert. Ebenso ist für die PLL und Stromregelung aufgrund der taktfrequenzabhängigen Parameter die Ausführung mit entsprechender Frequenz notwendig.

Die `main()`-Funktion des Programms bearbeitet zeitunkritische Teile, wie etwa die Ausga-

be über die Kommunikationsschnittstelle. Die PWM-Frequenz von 140 kHz wird durch den HRTIM1 basierend auf dem Vergleichswert CMP1 generiert. Für eine andere im Rahmen des Projektes implementierte, aber in dieser Arbeit nicht weiter behandelten Modulationsart wird zusätzlich noch CMP2 genutzt, um die beiden Ausgänge des Timers nicht nur komplementär, sondern auch unabhängig voneinander zu nutzen. Der Timer kann neben der Signalausgabe zusätzlich als Interruptquelle eingerichtet werden, sodass die PLL und Regelung mit der Steuerung synchronisiert werden kann. Wie in Abschnitt 5.5.5 erläutert, ist die maximale Ausführungsfrequenz bei etwa 133 kHz bei voller Prozessorauslastung erreicht. Mittels des Repetition-Counters wird nur bei jeder zweiten PWM-Periode ein Interrupt ausgelöst, sodass die Ausführung mit 70 kHz erfolgt.

Auch die Analog/Digital Converter sind über den Timer ansteuerbar. Damit ein Mittelwert erfasst wird, muss die Abtastung entsprechend mittensynchron zur PWM-Ansteuerung erfolgen. Der Einschaltzeitpunkt des PWM-Signals liegt zum Anfang der Periode, während der Ausschaltzeitpunkt durch den Vergleichswert CMP1 definiert wird. Für die mittige Abtastung wird ein weiterer Vergleichswert CMP3 genutzt, welcher dem halben Wert von CMP1 entspricht.

Im vorigen Abschnitt stellt sich für die Implementierung der seriellen Kommunikation nur die Variante mit Interrupts für dieses Projekt als geeignet heraus. Damit die kritischen Prozesse der PLL und Stromregelung jedoch nicht durch die Kommunikationsschnittstelle unterbrochen werden, wird letzterem eine niedrigere Priorität zugewiesen. Bei einem eingehenden Zeichen setzt der Microcontroller ein entsprechendes Flag, wartet aber, wie in Abbildung 5.18 zu sehen, auf den Abschluss der Interrupts mit höherer Priorität. Der gestrichelte Bereich zeigt den durch CMP1 bedingten Duty-Cycle.

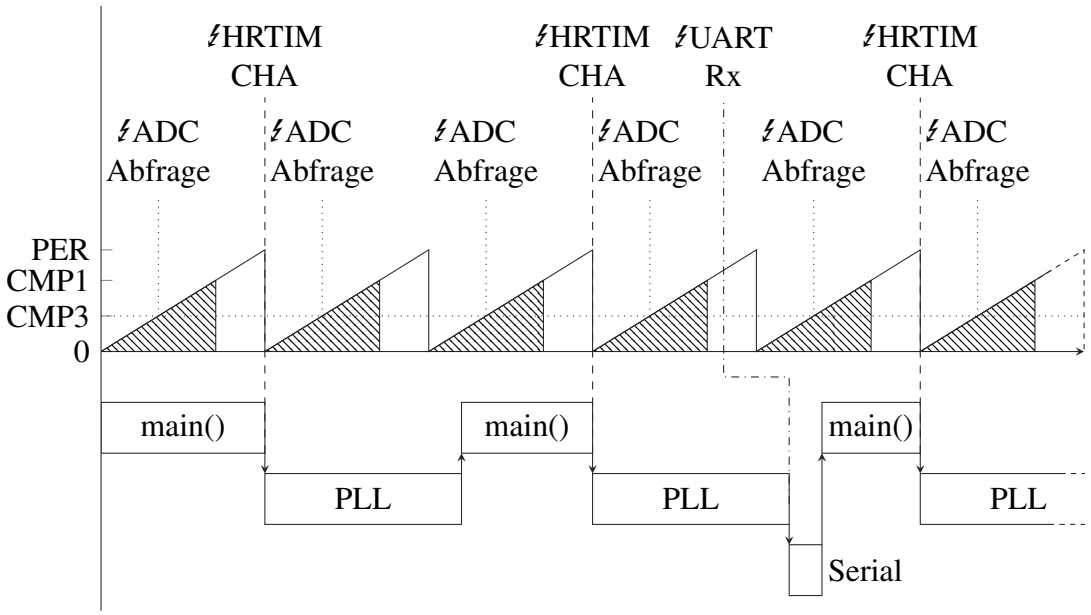

Abbildung 5.18: Zeitverlauf der einzelnen Programmteile

connect_window	
stats_window	error_window
current_window	
control_window	

Tabelle 5.3: Aufteilung der GUI in verschiedene Bereiche

5.9 GUI

Für eine unkomplizierte Handhabung der Steuerung über die Kommunikationsschnittstelle wird eine GUI implementiert. Häufig eingesetzte Werkzeuge sind dafür *Java* mit *Swift*, C/C++ mit *Qt* und *Python* ebenfalls mit *Qt* oder *tkinter*.

Die Nutzung von *Qt* in C bringt den Vorteil mit, dass innerhalb des Systems nur eine Programmiersprache genutzt wird. Nachteilig ist dabei jedoch, dass der Programmcode für unterschiedliche Betriebssysteme in ein eigenes ausführbares Programm kompiliert werden muss. Ebenso existieren für die verschiedenen Betriebssysteme unterschiedliche APIs für die Ansteuerung der seriellen Schnittstelle. Der Programmcode muss also mehrere Implementierungen unterstützen.

Unter *Java* werden die Unterschiede zwischen den Betriebssystemen zu großen Teilen durch die Bibliotheken und die Laufzeitumgebung vor den Nutzenden versteckt. Für die Einbindung der seriellen Schnittstelle werden verschiedene Bibliotheken angeboten. Die Komplexität des für die vergleichsweise kleine GUI zu generierenden Quelltextes und der Toolchain (Quelltext muss ebenfalls kompiliert werden) wird jedoch deutlich höher als bei *Python* bewertet.

Python ist eine Interpretersprache, die ebenfalls größtenteils betriebssystemunabhängig ausgeführt werden kann. Wie auch in C/C++ kann die Oberfläche mit *Qt* gestaltet werden. Eine technisch einfachere und kompakter zu Implementierende Alternative bietet *tkinter*. Mit *pyserial* lässt sich die serielle Schnittstelle ansteuern. Die Realisierung mittels *Python3* erfolgt aufgrund der Simplizität des Quellcodes, der damit einhergehenden Wart- und Erweiterbarkeit für nachfolgende Entwickelnde, und der bereits in der Vergangenheit zuverlässig eingesetzten Bibliotheken.

Die Elemente innerhalb des Fensters sind mittels des Grid-Managers von *tkinter* ausgerichtet und in mehrere Untergruppen aufgeteilt. Der Grid-Manager erstellt eine tabellarische Struktur mit Zeilen und Spalten, die für die jeweiligen Gruppen genutzt wird. Zusammengehörige Elemente, wie etwa die aktuellen Messwerte oder die Fehler-Übersicht sind noch einmal in eigene Grids unterteilt, die dann dem Hauptfenster zugeordnet werden. Damit bleibt bei einer Erweiterung der GUI die Anordnung ausreichend modular, sodass Gruppen neue Positionen zugeordnet werden können, aber die Elemente innerhalb der Gruppe entsprechend zueinander ausgerichtet bleiben. Tabelle 5.3 zeigt das aktuelle Layout der Gruppen, in Abbildung 5.19 ist ein Bildschirmfoto der tatsächlichen GUI zu sehen.
Für die Anzeige aktueller Werte müssen eingehende Nachrichten der seriellen Schnittstelle bearbeitet werden. Hierfür wird vor dem Ausführen des *tkinter*-Loops ein Thread gestartet, welcher bei verbundener Schnittstelle Nachrichten zeilenweise einliest. Inhalte beginnend

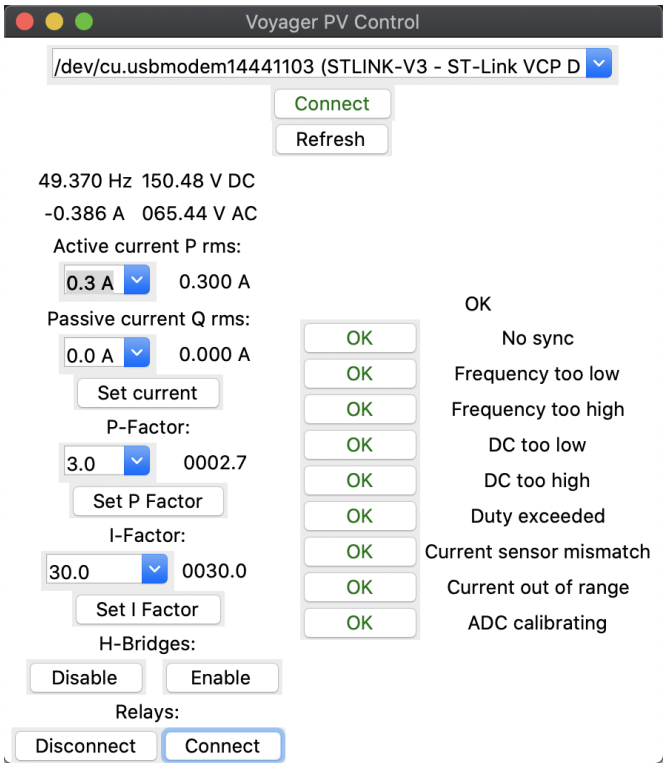

Abbildung 5.19: Grafische Benutzeroberfläche für Voyager-PV

mit `E:` sind Fehlermeldungen oder eine OK-Mitteilung und werden über der Fehlerübersicht angezeigt. Mit `S:` beginnende Nachrichten enthalten Statusinformationen entsprechend des Formats aus Abschnitt 5.7. Diese werden an den Separatorenzeichen (Semikolon) aufgetrennt und in einem assoziativen Array (`dict`) abgelegt. Anhand der Keys werden die Werte dann aufbereitet und für die entsprechenden Anzeigenelemente übernommen. Die Aufbereitung der Fehlerzustände ist exemplarisch in Quellcode 5.3 gezeigt.

```
def parse_message(self, line):
  if line.startswith(b'E:'):
    self.label_errormessage.config(text=line[3:-1])
  elif line.startswith(b'S:'):
    parts = line[3:].split(b';')
    vals = {}
    for part in parts:
      v = part.split(b'=')
      vals[v[0]] = v[1]
    try:
      err = int(vals[b'ERR'], 16)
      for bit in self.errors:
        if err & bit:
          self.error_buttons[bit].config(text="ERR", style='red.
              TButton')
        else:
          self.error_buttons[bit].config(text="OK", style='green.
              TButton')
    except IndexError:
      pass # No error value sent in this message
```

Quelltext 5.3: Aufbereitung von Nachrichten für die grafische Benutzeroberfläche

5.10 Fehlerbehandlung und Grenzwerte

Während bei Datenbanken oder anderer Software eine Fehlkonfiguration durch ein Backup häufig wieder rückgängig gemacht werden kann, hätte eine falsche Einstellung am Wechselrichter unter Umständen eine Zerstörung der Hardware zur Folge. Dementsprechend ist die Robustheit, also die Unterbindung von undefinierten Zuständen und Eingaben, eine wichtige Eigenschaft des Programms, wie in FA11 und FA8 gefordert. In diesem Abschnitt wird beschrieben, welche Maßnahmen unternommen wurden, um entsprechende Fälle abzufangen und einen sicheren Zustand herzustellen.

Die GUI ermöglicht zwar nur begrenzte Stellmöglichkeiten, die genutzte Schnittstelle kann jedoch auch mit anderen Programmen, etwa einem seriellen Terminal, angesprochen werden. Ebenso können durch Störung der Übertragung Zeichen nicht oder falsch übertragen werden. Daher ist die Beschränkung seitens der GUI nicht ausreichend. Kommandos haben, wie in 5.7 beschrieben, ein festes Format. Sollte ein Zeichen fehlen, so wird die Anfrage verworfen und eine Fehlermeldung ausgegeben. Das Stellen eines negativen Wirkstroms hätte zur Folge, dass der Wechselrichter anstatt als Quelle als Stromsenke arbeitet, wofür die Hardware nicht ausgelegt ist. Aufgrund dessen ist es nur möglich einen positiven Wirkstrom zu stellen, der in Summe mit dem Blindstrom aber unterhalb der maximalen Leistungsfähigkeit des Systems liegen muss. Auch beim Schließen der Relais, ohne dass der Wechselrichter eine Wechselspannung erzeugt, also die Halbbrücken deaktiviert sind, arbeiten die Treiber in die andere Richtung als Gleichrichter und entsprechend auch als Senke. Das Einschalten der Relais wird daher so lange verhindert, bis die Halbbrücken aktiviert wurden, also eine Wechselspannung wechselrichterseitig erzeugt wird. Umgekehrt hat auch das Abschalten der Brücken das Öffnen der Relais zur Folge.

Alle von der Software als kritischen Zustand bewerteten Situationen öffnen nach FA11 mindestens die Relais, um den Wechselrichter vom Netz zu trennen. Je nach Notwendigkeit werden zusätzlich auch die Halbbrücken deaktiviert, etwa bei zu hoher Gleichspannungsversorgung oder Strommessungen außerhalb des definierten Bereichs. Nur wenn kein Fehler mehr erkannt wurde, lassen sich die Relais wieder schließen. Für den derzeitigen Einsatz in der Forschung müssen alle Fehler manuell über die Schnittstelle oder einen Reset des Systems zurückgesetzt werden. Eine Ausnahme davon ist der Verlust der Netzsynchronisation.

Wie in 5.2.3 beschrieben ist der q-Anteil der Indikator für die Abweichung des ermittelten und tatsächlichen Phasenwinkels. Bei überschreiten eines eingestellten Schwellwerts wird sowohl über die LEDs als auch mit dem Flag `ERROR_NO_SYNC` signalisiert, dass keine Synchronisierung vorliegt. Der Zustand setzt sich jedoch automatisch zurück, sobald der Schwellwert für mindestens eine Periodenlänge der Netzfrequenz unterschritten ist. Somit wird sichergestellt, dass der q-Anteil nicht nur zufällig unter dem Schwellwert ist.

Ein weiteres Flag, welches kein Fehler ist und sich selbst zurücksetzt, ist die Kalibrierung der Analog/Digital Converter. In regelmäßigen Abständen und starken Temperaturschwankungen, die gerade bei Solarmodulen mit direkter Bestrahlung und Erwärmung der Elektronik auftreten, sollten die Wandler eingemessen werden. Diese Funktion wird vom Controller derzeit nur beim Starten des Systems ausgeführt und muss im späteren Betrieb basierend auf der Prozessortemperatur erneut angestoßen werden. Dafür müssen jedoch die Messungen von analogen Eingängen unterbrochen werden. Folglich stehen in diesem Moment keine Werte für die PLL und Stromregelung zur Verfügung, daher wird der Zustand wie ein Fehler behandelt und trennt den Wechselrichter vom Netz.

Die nicht selbst zurücksetzenden Zustände sind, wie unter anderem bereits erwähnt, Ströme außerhalb des vorgesehenen, respektive von den Sensoren spezifizierten Bereichen und das Überschreiten einer gesetzten maximalen Gleichspannung. Im Laborbetrieb wird der Wechselrichter nicht an Solarzellen, sondern an einer Gleichspannungsquelle betrieben. Um auch mit kleineren Spannungen das Modul zu untersuchen, wurde auf die Vorgabe einer Mindestspannung verzichtet. Bei zu geringer Spannung kann jedoch nicht mehr die gewünschte Ausgangsspannung gestellt werden. Erkannt werden kann dies über den angeforderten Tastgrad der PWM-Ansteuerung. Mehr als 100%, also dauerhaft angeschaltet, ist nicht möglich, daher wird beim ersten Stellen von Werten größer 1 (oder kleiner -1) zunächst eine Variable gesetzt, dass die Gleichspannung nicht ausreichend ist und die Ansteuerung beim vorigen Tastgrad belassen. Erst beim zweiten Übertreten des Grenzwerts in Folge wird der Fehler gesetzt und das Relais geöffnet, um bei kurzen Spitzen nicht sofort zu trennen. Während bis 2018 Photovoltaikanlagen bereits bei einer Netzfrequenz von 50,2 Hz vom Netz getrennt wurden, muss inzwischen – um bei dem großen Anteil von PVA nicht die Stabilität des Netzes bei gleichzeitiger Abschaltung zu gefährden – nur noch die Leistung linear bis 51,5 Hz auf 50% gedrosselt werden. Die Trennung erfolgt nun bei der Überschreitung von 51,5 Hz, wie auch unter 47,5 Hz [8]. Die Frequenz kann über ω ermittelt und eine entsprechende Abschaltung eingeleitet werden.

5.11 Dokumentation

Für die Dokumentation soll nach NFA5 einerseits die vorliegende Arbeit dienen, andererseits auch der kommentierte Quellcode. Die Annotation der eingebundenen *STM32*-Bibliotheken erfolgt mittels *Doxygen*-kompatiblen Kommentaren, welche sich mit selbigem Programm zu einer interaktiven HTML-Dokumentation und einem LaTeX Dokument umwandeln lassen.

Entwickelnde haben mit dem leicht anderen Aufbau der Kommentare mit *Doxygen* nur wenig Mehraufwand. Für Nutzende der resultierenden Dokumentation ergeben sich jedoch diverse Vorteile. Anstatt sowohl die C- als auch gegebenenfalls Header-Dateien nach hilfreichen Kommentaren zu durchsuchen, lassen sich die generierten HTML-Dateien in einem Webbrowser betrachten und enthalten nur die erläuternden Informationen der Kommentare, jedoch nicht die genaue Implementierung selbst. Durch Module werden dateiübergreifend Gruppierungen erstellt oder innerhalb einer Datei aufgeteilt, wodurch die Lesbarkeit zusätzlich gesteigert wird. *Doxygen* verlinkt relevante Funktionen, Variablen, etc. automatisch, sodass Lesende durch Klicken des Hyperlinks direkt zum Referenzierten gelangen. Dadurch entfällt das aktive Öffnen von anderen Dateien oder das Scrollen zu der erwähnten Stelle. Ebenso kann die Dokumentation online ohne Anpassungen bereitgestellt werden.

```
1  /**
2    * @brief Enables the bridges.
3    * @param on Set to 1 to enable, 0 to disable.
4    * @retval None
5    * @note When the \link #init_pwm initialization\endlink is not
       done yet, the function will return without setting the pins.
6    */
7  void enable_bridge(bool on) {
8    /// Sanity check: Skip the enabling, if PWM is not initialized
       yet.
9    if (pwm_initialized < on) return;
10   /// Skip setting registers, if current status equals 'on'.
11   if (on == bridge_enabled) return;
12   /// Sanity check: When grid is still connected when the bridge
       is turned off, make sure, that grid gets disconnected first.
13   if (on < mains_connected) {
14     set_relay(0);
15   }
16   set_pin(&ENABLE_L_LS, on);
17   set_pin(&ENABLE_L_HS, on);
18   set_pin(&ENABLE_R_LS, on);
19   set_pin(&ENABLE_R_HS, on);
20   bridge_enabled = on;
21 }
```

Quelltext 5.4: Beispiel einer mit Doxygen annotierten Funktion

Die Kommentare für *Doxygen* werden durch `///` oder mehrzeilig durch `/**` (regulär mit

`//` oder `/*`) eingeleitet. Über eine mit dem Befehl `\f$` umschlossene Gleichung wird diese als LaTeX Gleichung interpretiert und eine entsprechende Grafik erzeugt. Die erwähnten Verlinkungen sind in Quellcode 5.4 mit `\link` und `\endlink` ausgezeichnet und verweist auf die Funktion `init_pwm` in der selben Datei. In Abbildung 5.20 ist der Auschnitt aus der von *Doxygen* aus dem Quellcode erzeugten HTML-Seite zu sehen.

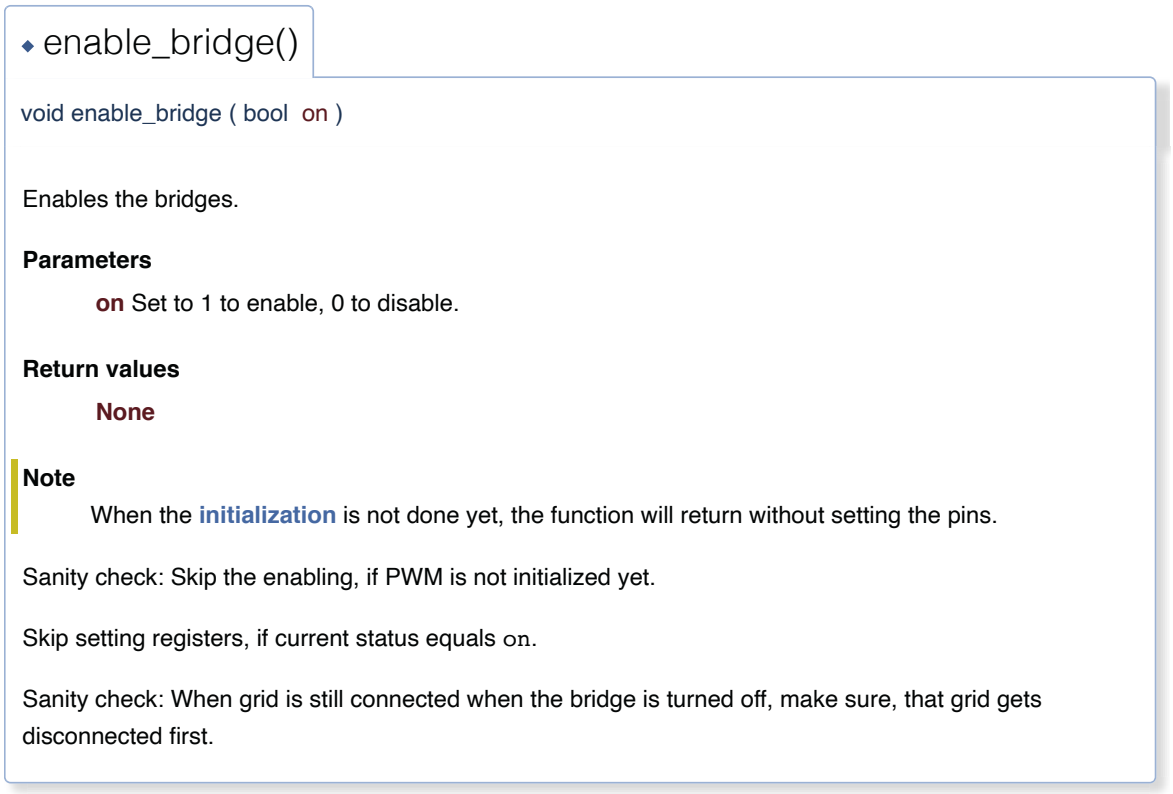

Abbildung 5.20: Ansicht der von *Doxygen* generierten Dokumentation im Webbrowser

Zur Generierung der entsprechenden Dateien mit *Doxygen* wird ein *Doxyfile* benötigt, in dem die Parameter, wie beispielsweise Programmiersprache und Zielverzeichnis vermerkt sind. Dieses ist im Ordner *Core* des Projekts abgelegt. Mittels der Ausführung von `doxygen Doxyfile` in dem Verzeichnis wird die Dokumentation im übergeordneten Verzeichnis unter *Doc* erzeugt. Aufrufen lässt sich die erzeugte HTML-Dokumentation im Unterorder *HTML/index.html* mit jedem modernen Webbrowser.

Auf eine separate Dokumentation mittels *Doxygen* für die grafische Benutzeroberfläche wird aufgrund des geringen Umfangs des Programms verzichtet.

6 Auswertung

Das entwickelte Softwareframework dient als Basis für die weitere Forschung und Entwicklung im Rahmen von *Voyager-PV*. Die Erfüllung der eingangs gestellten Anforderungen wird nachfolgend überprüft, sowie ein Ausblick für zukünftige Arbeiten mit dem System gegeben.

6.1 Erfüllung der Anforderung

Eine zentrale Anforderung an das Framework ist die effiziente, auf die Architektur abgestimmte Implementierung der PLL und Stromregelung, um die Ausführungsfrequenz entsprechend steigern zu können. Durch die Optimierung für den Microcontroller konnte die mögliche Ausführungsfrequenz um mehr als Faktor zehn gesteigert werden. Die Regelung funktioniert dabei erwartungsgemäß und erreicht bereits jetzt mit der noch nicht optimal konfigurierten Wechselrichterplatine und nicht final eingestellten Regelparametern ein nahezu normkonformes Verhalten.

Die Robustheit des Frameworks erwies sich während der Evaluationsphase als ausreichend. Durch die Änderung der Verkabelung wurde die Flussrichtung des Stromsensors invertiert. Die Regelung vergrößerte infolgedessen die Abweichung von der Sollvorgabe. Im weiteren Verlauf wurde der Grenzwert für den Stromfluss überschritten. Die Software erkannte dieses, leitete die Sicherheitsabschaltung automatisch ein und vermied die Zerstörung der Hardware.

Die GUI wird von den Projektbearbeitenden mit unterschiedlichen Informatikkenntnissen als sehr komfortabel nutzbar bewertet. Innerhalb der bisherigen Nutzungsphase konnten darüber hinaus keine grenzwertverletzenden Werte eingestellt werden. Somit ist es bei korrektem Anschluss der Hardware zu keiner Fehlfunktion durch Eingaben der Nutzenden gekommen.

Der Quellcode der vorherig genutzten Software ist schwer lesbar und dadurch ebenfalls nur mit erhöhtem Aufwand wartbar. Durch die starke Kongruenz des neuen Quelltextes mit den *Simulink*-Modellen lassen sich die Funktionen leicht identifizieren. Die erstellte Dokumentation erleichtert zudem die Weiterentwicklung. Aufgrund des trivialen Aufbaus der für die Dokumentation genutzten Kommentare, kann diese auch simpel ergänzt werden. Eine abschließende Bewertung kann jedoch erst nach Erweiterung des Frameworks durch andere Projektbearbeitende erfolgen.

Trotz der Corona-Pandemie und der anhaltenden Chipkrise konnten alle Anforderungen erfüllt und vor Ort validiert werden. Mit dem Wechselrichter und dem entwickelten Software-Framework konnte im Rahmen von *Voyager-PV* erstmalig Strom eingespeist werden.

6.2 Ausblick

Wie bereits in Kapitel 5.3 erwähnt, liegt die THD des Stroms noch über dem Grenzwert der entsprechenden Norm. In folgenden Versuchen muss daher überprüft werden, welche Änderungen an der Hardware vorgenommen werden müssen, um die Grenzwerte einzuhalten. Die unternommenen Versuche, durch Veränderung der Parameter für die Stromregelung die THD zu senken, hatten nicht genügend Auswirkung.

Darüber hinaus schließt sich an diese Arbeit die Anbindung des in der Analyse erwähnten *WirePas*-Moduls an. Dieses ersetzt die bisherige serielle Schnittstelle und dient dem vernetzten Betrieb. Zum Anschluss muss eine neue Platine entwickelt werden, die die bisherige Lösung von mehreren ineinander gesteckten Modulen ablöst. Der Microcontroller kann zusammen mit der Sensorik und der Wechselrichter-Hardware auf einer gemeinsamen Platine untergebracht werden. Dadurch lassen sich elektromagnetische Störeinflüsse weiter reduzieren. Zur präziseren Erfassung des Stroms kann zudem das Sampling von einem Messwert auf 16 über die Periode T_{PWM} äquidistante Samples angepasst werden. Der Microcontroller unterstützt Oversampling mit nachfolgendem Bitshift, sodass eine Mittelwertbildung der 16 Messwerte in Hardware erfolgen kann.

Eine Entspannung der Chipkrise würde zu einer wieder besserer Verfügbarkeit von Bauteilen führen. Der Wechsel der Stromsensorik auf den in Abschnitt 4.1.2 vorgeschlagenen Baustein oder einen anderen Sensor muss dann noch einmal evaluiert werden.

Die beispielhaft ausgeführte Umwandlung eines *Simulink*-Modells erzeugte nur bedingt verständlichen, aber ausführbaren Quelltext. Für zukünftige vergleichbare Softwareprojekte wird daher empfohlen, weiterhin Personen mit einem starken Informatikbezug einzubinden, um die Softwarequalität zu steigern.

A Schaltplan und Platinenlayout

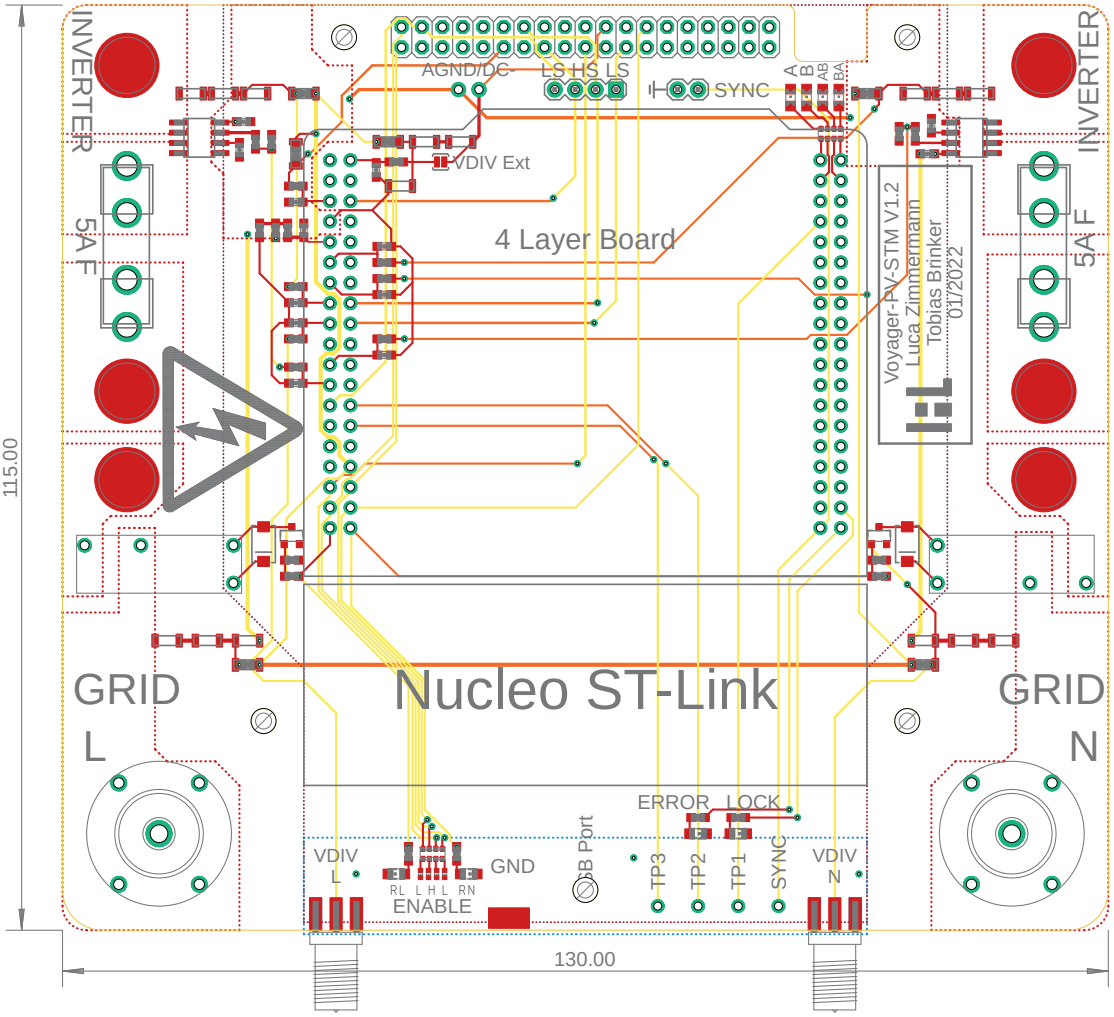

Abbildung A.1: Layout der entwickelten Adapterplatine

No semiconductors were harmed during this thesis.

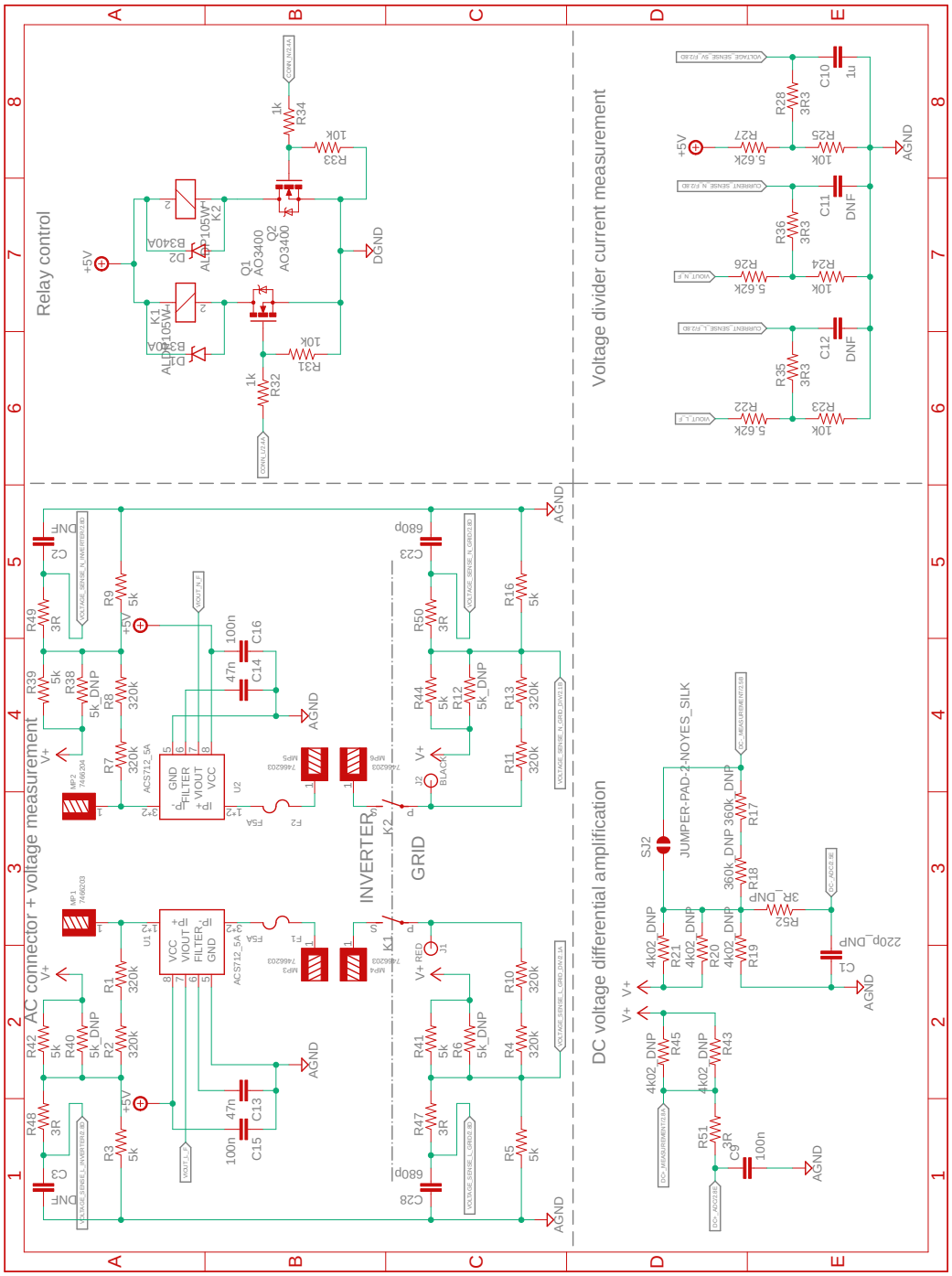

Abbildung A.2: Schaltplan Seite 1 der entwickelten Adapterplatine

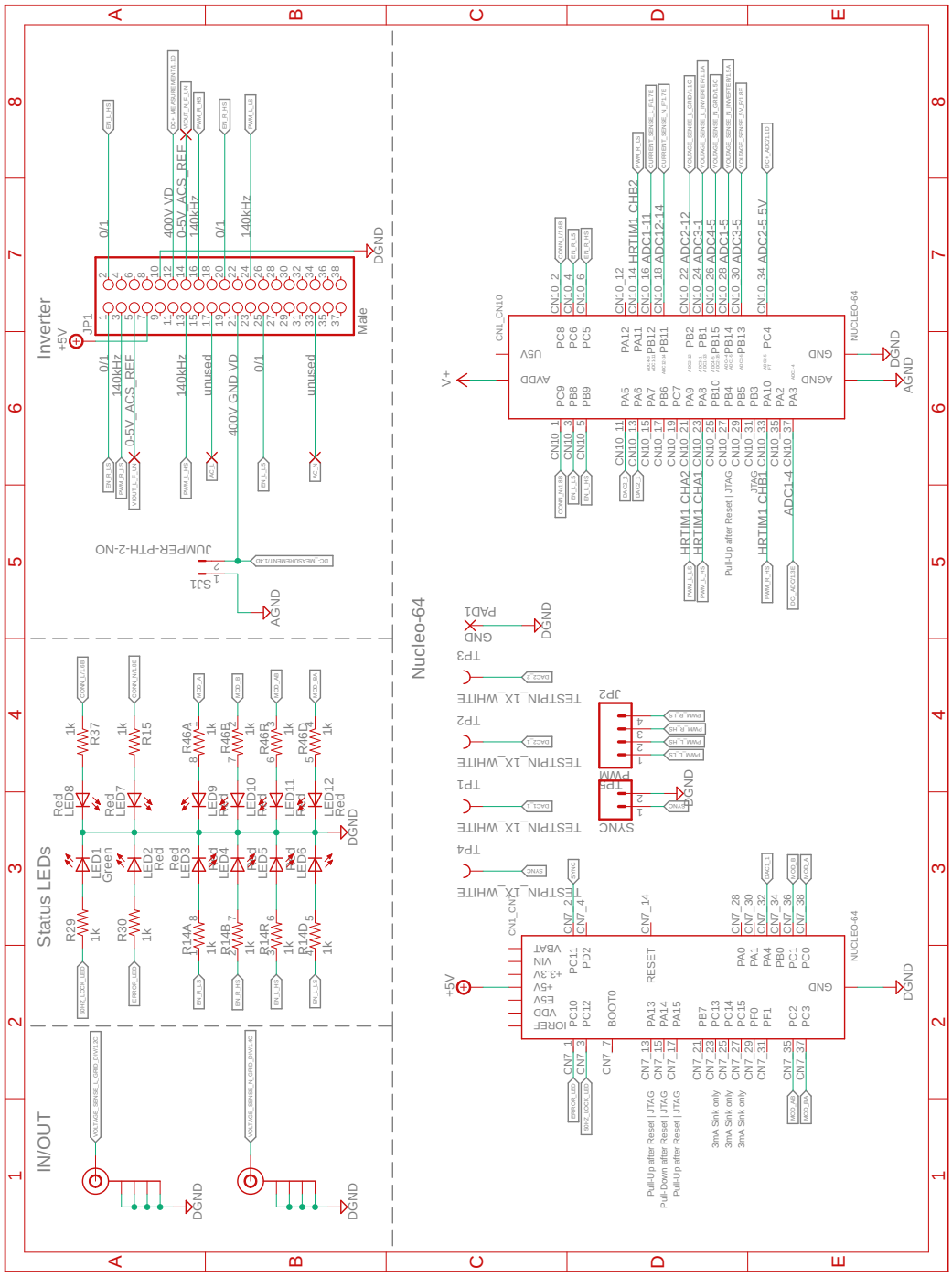

Abbildung A.3: Schaltplan Seite 2 der entwickelten Adapterplatine

Literaturverzeichnis

[1] HOSSEINI, Seyed E.: Transition away from fossil fuels toward renewables: lessons from Russia-Ukraine crisis. In: *Future Energy* 1 (2022), Nr. 1

[2] BR24: *Genehmigung von Nord Stream 2 gestoppt*. https://www.tagesschau.de/wirtschaft/weltwirtschaft/scholz-nordstream-101.html, Abruf: 20.03.2022

[3] TAGESSCHAU24: *Moskau liefert Polen und Bulgarien kein Gas mehr*. https://www.tagesschau.de/ausland/europa/gas-polen-bulgarien-101.html, Abruf: 27.04.2022

[4] MCWILLIAMS, Ben ; SGARAVATTI, Giovanni ; TAGLIATPIETRA, Simone ; ZACHMANN, Georg: *Can Europe survive painlessly without Russian gas?* https://www.bruegel.org/2022/01/can-europe-survive-painlessly-without-russian-gas/, Abruf: 20.04.2022

[5] APOSTOLERIS, Harry ; SGOURIDIS, Sgouris ; STEFANCICH, Marco ; CHIESA, Matteo: Utility solar prices will continue to drop all over the world even without subsidies. In: *Nature Energy* 4 (2019), Nr. 10, S. 833–834

[6] OECD NUCLAR ENERGY AGENCY: *Ukraine: Current status of nuclear power installations*. https://www.oecd-nea.org/jcms/pl_66130/current-status-of-nuclear-power-installations-in-ukraine, Abruf: 15.04.2022

[7] XUE, Yaosuo ; CHANG, Liuchen ; KJAER, Sren B. ; BORDONAU, J. ; SHIMIZU, T.: Topologies of single-phase inverters for small distributed power generators: an overview. In: *IEEE Transactions on Power Electronics* 19 (2004), Nr. 5, S. 1305–1314. http://dx.doi.org/10.1109/TPEL.2004.833460. – DOI 10.1109/TPEL.2004.833460

[8] VDE: *VDE-AR-N 4015 Anwendungsregel:2018-11, Erzeugungsanlagen am Niederspannungsnetz – Technische Mindestanforderungen für Anschluss und Parallelbetrieb von Erzeugungsanlagen am Niederspannungsnetz*. VDE, 2018

[9] LAMO, Paula ; PIGAZO, Alberto ; AZCONDO, Francisco J.: Evaluation of Quadrature Signal Generation Methods with Reduced Computational Resources for Grid Synchronization of Single-Phase Power Converters through Phase-Locked Loops. In: *Electronics* 9 (2020), Nr. 12. http://dx.doi.org/10.3390/electronics9122026. – DOI 10.3390/electronics9122026. – ISSN 2079–9292

[10] *Xilinx FPGA - Field Programmable Array, Mouser Electronics*. https://www.mouser.de/c/semiconductors/integrated-circuits-

ics/embedded-processors-controllers/fpga-field-programmable-gate-array/?m=Xilinx&number%20of%20logic%20elements=10476%20LE~~783300%20LE&sort=pricing&rp=semiconductors%2Fintegrated-circuits-ics%2Fembedded-processors-controllers%2Ffpga-field-programmable-gate-array%7C~Number%20of%20Logic%20Elements, Abruf: 26.04.2022

[11] AHMED, Nasim ; KHAN, Ziaur R.: Microcontroller Based Pure Sine Wave Inverter. In: *2021 IEEE International Conference in Power Engineering Application (ICPEA)*, 2021, S. 173–177

[12] IKKEN, Naima ; BOUKNADEL, Abdelahdi ; HADDOU, Ahmed ; TARIBA, Nour-Eddine ; EL OMARI, Hafsa ; EL OMARI, Hamid: PLL Synchronization Method Based on Second-Order Generalized Integrator for Single Phase Grid Connected Inverters Systems during Grid Abnormalities. In: *2019 International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS)*, 2019, S. 1–5

[13] IEC: *International Standard IEC 61727, Photovoltaic (PV) systems – Characteristics of the untility Interface*. International Electrotechnical Commission, 2004

[14] TEICHERT, Tilo: *Untersuchung und Umsetzung geeigneter Regelungskonzepte für die Verwendung kostengünstiger PV-Modulwechselrichter*, Leibniz Universität Hannover, Bachelorarbeit, 2021

[15] MICROELECTRONICS, ST: *STM32 Digital Power Ecosystem*. https://www.st.com/content/st_com/en/ecosystems/stm32-digital-power.html, Abruf: 03.11.2021

[16] ALLEGRO MICROSYSTEMS: *Datenblatt ACS724*. https://www.allegromicro.com/-/media/files/datasheets/acs724-datasheet.pdf, Abruf: 05.11.2022

[17] ALLEGRO MICROSYSTEMS: *Datenblatt ACS725*. https://www.allegromicro.com/-/media/files/datasheets/acs725-datasheet.pdf, Abruf: 05.11.2022

[18] J.P. MORGAN: *Supply Chain Issues and Autos: When Will the Chip Shortage End?* https://www.jpmorgan.com/insights/research/supply-chain-chip-shortage. Version: 12 2021

[19] TEODORESCU, Remus ; LISERRE, Marco ; RODRÍGUEZ, Pedro: *Wiley - IEEE*. Bd. 29: *Grid Converters for Photovoltaic and Wind Power Systems*. 1. Aufl. Hoboken : Wiley-IEEE Press, 2010. – ISBN 0470057513

[20] DIN: *DIN EN 50160:2020-11, Merkmale der Spannung in öffentlichen Elektrizitätsversorgungsnetzen; Deutsche Fassung EN 50160:2010 + Cor.:2010 + A1:2015 + A2:2019 + A3:2019*. Beuth Verlag GmbH, 2020

[21] MICROELECTRONICS, ST: *Application Note 5325, Getting started with the CORDIC accelerator using STM32CubeG4 MCU Package*, 2021. https://www.st.com/resource/en/application_note/dm00614795-

getting-started-with-the-cordic-accelerator-using-stm32cubeg4-mcu-package-stmicroelectronics.pdf

[22] ARM LIMITED: Cortex-M4 Technical Reference Manual. https://documentation-service.arm.com/static/5f19da2a20b7cf4bc524d99a, Abruf: 22.11.2020