Masterarbeit

Stromzwischenkreis-Pulswechselrichter für Netzbetrieb

Wilfried Rudolf Wiltberger

Betreuer: Ass.Prof. Dipl.-Ing. Dr.techn. Klaus Krischan

Institut für Elektrische Antriebstechnik und Maschinen
Technische Universität Graz

Graz 2013

An dieser Stelle möchte ich allen Personen danken die mich direkt oder indirekt bei dieser Arbeit unterstützt haben, insbesondere gilt dies für Herrn Dr. Krischan, der mir stets mit Rat und Tat zur Seite stand.

EIDESSTATTLICHE ERKLÄRUNG

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

Graz, am …… ……..
(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

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

…… ……..
date (signature)

Inhaltsverzeichnis

1 Aufgabenstellung

Das Ziel dieser Masterarbeit ist die Entwicklung der netzseitigen Stufe eines Pulswechselrichters mit Stromzwischenkreis. Besondere Aufmerksamkeit soll dabei folgenden Punkten gewidmet werden:

- Auswahl der Leistungshalbleiter und deren Ansteuerung
- Entwurf und Simulation einer geeigneten Regelung
- Entwicklung und Aufbau der Schaltung
- Verifikation des Modells und der Simulation am realen Aufbau

Die Leistungsdaten des Umrichters sollen sich dabei an folgenden Eckdaten orientieren:

- Zwischenkreisstrom $I_{out} = 15\text{A}$
- Spannungsfestigkeit $U_{netz,eff} = 400\text{V}$ (verkettet)

Weiters soll auf die Besonderheiten dieser Umrichtertopologie wie z.B.

- Rückspeisefähigkeit
- Möglichkeit zur Blindleistungserzeugung
- sinusförmige Netzströme

eingegangen werden.

2 Hardware

2.1 Prinzipschaltung des Leistungsteiles

Abbildung 2.1 zeigt die prinzipielle Grundschaltung eines vollständigen Pulswechselrichters mit Gleichstromzwischenkreis. Die Schalter S1 bis S6 bilden dabei den netzseitigen Teil, sie übernehmen die Funktion eines Gleichrichters, welcher einen vorgegebenen Strom in die Zwischenkreisinduktivität einprägt und gleichzeitig die Form des aufgenommenen Netzstromes beeinflussen kann. Üblicherweise wird hier versucht einen weitestgehend sinusförmigen Eingangsstrom mit Phasenverschiebung nahe 0 zu erreichen. Der hier vorgestellten Topologie ist es zudem möglich, durch Erzeugung eines vor- oder nacheilenden Netzstromes, eine beispielsweise kapazitive Last zu simulieren, um den Blindleistungsanteil anderer, meist induktiver, Verbraucher gezielt zu kompensieren.

Der Wechselrichter- oder maschinenseitige- Teil besteht aus den Schaltern S7 bis S12 und ermöglicht es an den Ausgangsklemmen ein weitestgehend frei wählbares Wechsel- oder Gleich-Stromsystem zu erzeugen.

Die Hardware der beiden B6 Brückenschaltungen kann vollständig ident sein, die Unterschiede beschränken sich auf Art der Ansteuerung und die zur Regelung des Umrichters notwendigen Messgrößen.

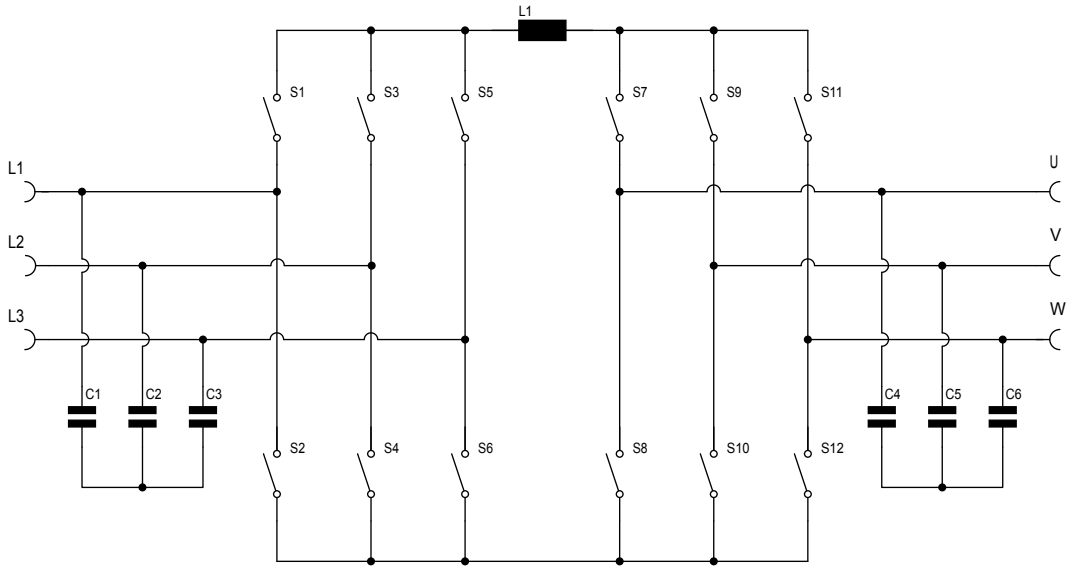


Abbildung 2.1: Prinzipschaltung eines Stromzwischenkreisumrichters

2.2 Bauteilauswahl

2.2.1 Definition der Anforderungen an die Leistungshalbleiter

Zu den grundsätzlichen Anforderungen zählt natürlich die Fähigkeit, den vorgesehenen Zwischenkreisstrom von 15A dauerhaft zu führen und die auftretenden Spannungen sperren zu können. Bei einer verketteten Netzspannung von $400V_{eff}$ beträgt die maximal zu sperrende Spannung:

$$U_{max} = U_{netz,verkettet} * \sqrt{2} = 400V * \sqrt{2} \approx 565V \tag{2.1}$$

Wird dazu noch ein Faktor von 1.3 für Überspannungen, welche bei schnellen Schaltvorgängen auftreten können, berücksichtigt, so scheint es angebracht, auf Transistoren und Dioden mit einer Spannungsfestigkeit von zumindest 800V zurückzugreifen. Dies allein schränkt die Auswahl an geeigneten MOSFETs bereits deutlich ein. Werden hingegen für eine Spannung von 1200V ausgelegte Bauteile verwendet, so wäre je nach Höhe der tatsächlich auftretenden Schaltüberspannungen auch ein Betrieb an einem 690V-Netz möglich.

Ein weiterer Punkt ist die auftretende Verlustleistung sowohl bei Schaltvorgängen als auch im leitenden Zustand, da der kontinuierlich fließende Zwischenkreisstrom über alle Lastfälle hinweg stets Verluste verursacht. Es ist also auf eine möglichst geringe V_{CEsat} im Falle von IGBTs, bzw. einen kleinen R_{DSon} bei Feldeffekttransistoren Wert zu legen.

Besonders zu beachten ist bei der Schaltung nach Abbildung 2.1, dass die Schaltelemente imstande sein müssen, Spannungen beider Polaritäten zu sperren. Leitfähigkeit wird hingegen nur in Vorwärtsrichtung gefordert. Da die Forderung nach Sperrfähigkeit in Rückwärtsrichtung von einem einzelnen abschaltbaren Halbleiter nicht erfüllt werden kann, muss hier je eine Diode in Serie zu den Transistoren geschaltet werden.
Die einzige Ausnahme bilden hier einige IGBTs des Herstellers IXYS, welche für diese Anwendung optimiert sind und daher die Funktion einer integrierten Seriendiode besitzen. Da diese Bauteile jedoch auch auf Nachfrage bei den Distributoren des Herstellers nicht, oder nur mit erheblicher Lieferzeit von mehreren Monaten und in großen Stückzahlen, erhältlich waren, musste von deren Einsatz abgesehen werden.

Eine Alternative zu den üblicherweise für solche Anwendungen eingesetzten IGBTs stellen Bauteile auf Siliziumcarbid-Basis dar. Obwohl die Verfügbarkeit dieser Bauteile zur Zeit dieser Arbeit noch sehr eingeschränkt war, waren verschiedene Siliziumcarbid-JFETs des Herstellers SemiSouth bei mehreren Distributoren verfügbar, welche beim Schaltverhalten, und durch sehr kleine Kanalwiderstände auch insbesondere im Teillastbereich, geringere Verluste versprechen.
Ebenfalls von SouthSemi verfügbar waren Siliziumcarbid- Schottkydioden.

Es wurde beschlossen, diese neuen Bauteile (SiC-JFETs und SiC-Schottky-Dioden) sowie konventionelle IGBTs und Dioden Vergleichsmessungen zu unterziehen. Eine Auflistung der zu genauen Tests ausgewählten Bauteile ist in Tabelle 2.1 zu finden.

Tabelle 2.1: Übersicht der getesteten Leistungshalbleiter

Bauteil	U_{sperr}	I_{nenn}	$V_{CE,sat}$ / $R_{DS,on}$ / V_F	Bemerkung
SGW15N120	1200V	15A	3,1V	IGBT
IXDR30N120	1200V	30A	2,4V	IGBT
IXDR30N120D1	1200V	30A	2,4V	IGBT mit antiparalleler Diode
SJEP120R100	1200V	17A	100mΩ	SiC JFET selbstsperrend
SJDP120R085	1200V	17A	85mΩ	SiC JFET selbstleitend
STTH3012W	1200V	30A	1,3V	ultrafast Diode
20EFT12	1200V	20A	1.3V	fast soft recovery Diode
SDP10S120D	1200V	20A	2,2V	SiC schottky Diode

2.2.2 Leitendverluste

Eines der wichtigsten Kriterien für die Bauteileauswahl sind die Verluste im Durchlasszustand. Alle in Tabelle 2.1 aufgelisteten Halbleiter wurden hierzu von einem konstanten Kollektor- bzw. Drainstrom durchflossen, während eben dieser sowie der Spannungsabfall am Bauteil gemessen wurde.

In Abbildung 2.2 sind die gemessenen Spannungen und die errechneten Verlustleistungen dargestellt (Werte bei Strömen über 12A wurden aus den vorhandenen Daten extrapoliert).
Der IXDR30N120D1 wird hier nicht gesondert aufgeführt, da dieser sich vom IXDR30N120 nur durch das Vorhandensein einer antiparallel zum IGBT geschalteten schnellen Diode unterscheidet, welche auf diese Messung jedoch keinen Einfluss hat.

Insgesamt sind hier deutliche Unterschiede bei den zu erwartenden Verlusten zu sehen, besonders der bereits ältere SGW15N120 kann in dieser Disziplin mit den anderen getesteten Transistoren nicht mehr konkurrieren und wird daher in den weiteren Messungen nicht berücksichtigt.
Das ohmsche Verhalten der Feldeffekttransistoren kann bei einem geplanten Arbeitsbereich von $I_{out} \leq 15\text{A}$ als Vorteil im Teillastbereich gesehen werden, wäre jedoch ein höherer Zwischenkreisstrom angestrebt, so würde die Verlustleistung der FETs (näherungsweise: $P_v \sim \text{I}^2$) bei höheren Strömen schnell jene aktueller IGBTs (näherungsweise: $P_v \sim \text{I}$) übersteigen.

Die Siliziumcarbid-Schottkydiode zeigt, obwohl für diesen Vergleich bereits beide Anoden des Chips parallel geschaltet waren, höhere Verluste als die herkömmlichen Dioden. Erwähnenswert ist aber, dass ihre Flussspannung einen positiven Temperaturkoeffizienten aufweist, was eine problemlose Parallelschaltung von mehrerern Dioden, welche sich nicht auf einem gemeinsamen Chip befinden, ermöglicht.

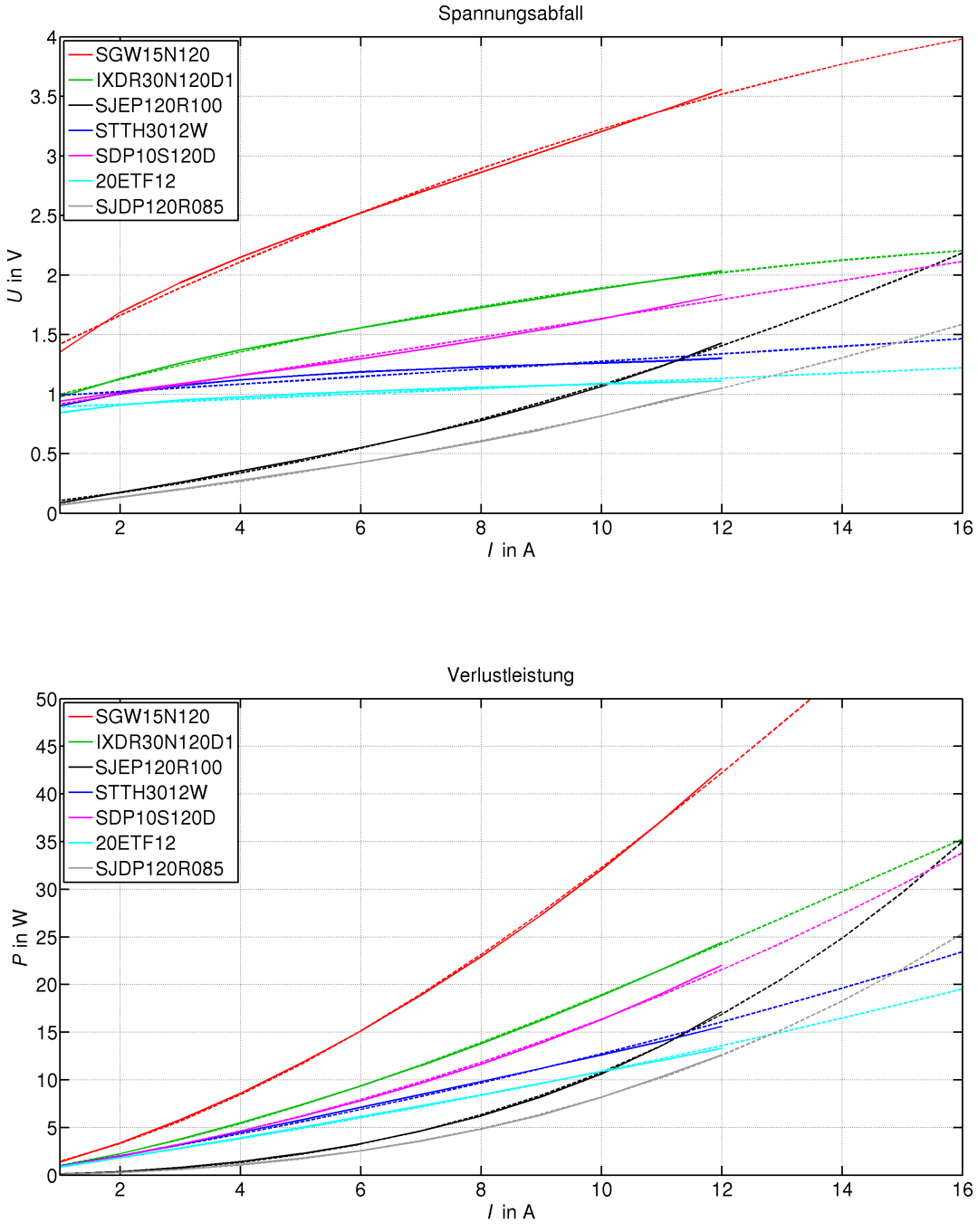


Abbildung 2.2: Im Durchlasszustand anfallende Verluste aller getesteten Bauteile

2.2.3 Testschaltung

Für die folgenden Messungen, welche das Schaltverhalten der verschiedenen Halbleiter bzw. deren Kombinationen betreffen, wurde die in Abbildung 2.3 gezeigte Schaltung verwendet.
Als Stromquelle dient ein Netzgerät mit Stromregelung
Die Induktivität L1001 und die Kapazität C1001 wurden ausreichend groß gewählt, um den Strom I_{test} bzw. die Spannung U_{RC} während einer Periode der Schaltfrequenz konstant zu halten. Die Ansteuerung der Transistoren erfolgte durch einen Signalgenerator, welcher auch die Einhaltung der notwendigen Verriegelungszeit übernimmt und Treiber vom Typ MCP1406 bei einer Versorgungsspannung von +15V.
Den Gates der IGBTs wurden Serienwiderstände von 13Ω vorgeschaltet, bei den JFETs in Serie zum Gatewiderstand zusätzlich eine Parallelschaltung aus 160Ω und 22nF. Dadurch wird der stationäre Strom in das Gate des JFETs auf ein vertretbares Maß reduziert, beim Schaltvorgang hingegen können wie beim IGBT deutlich höhere Ströme fließen.

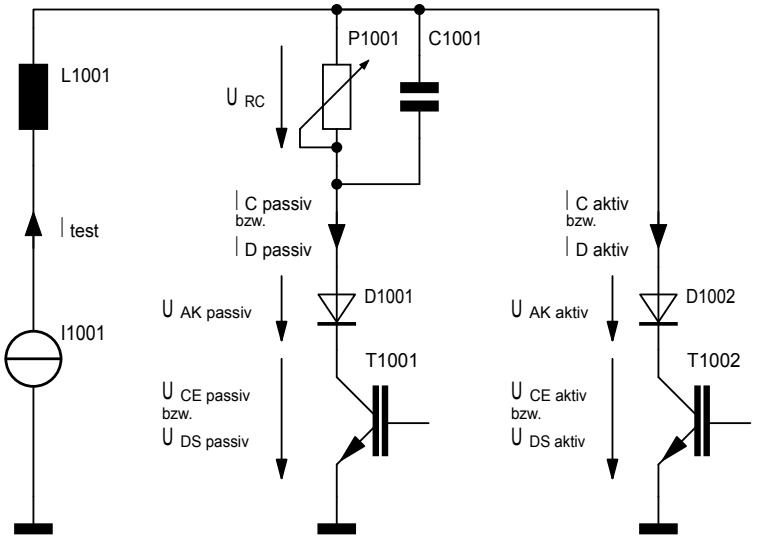


Abbildung 2.3: Aufbau zur Messung des Schaltverhaltens (schematisch)

2.2.4 Vergleiche verschiedener Halbleiterkombinationen

Die Abbildungen 2.4 und 2.5 zeigen das aktive und passive Schaltverhalten folgender Kombinationen von Halbleitern:

Tabelle 2.2: Kombinationen von Si- und SiC-Halbleitern

Transistor \ Diode		Si STTH3012W	SiC SDP10S120D
Si	IXDR30N120	A	B
SiC	SJEP120R100	C	D

Die in den jeweiligen Halbleitern während der abgebildeten Zeitspanne umgesetzten Verluste wurden, wie in den folgenden Kapiteln, aus den Kurvenverläufen errechnet. Dabei wurden mögliche Störeinflüsse wie beispielsweise Offsets in der Strommessung bereits berücksichtigt und kompensiert. Zur besseren Übersicht sind die Schaltverluste sowohl in den Tabellen 2.3 und 2.4, als auch in den Diagrammen eingetragen.
Charakteristisch für die herkömmliche Si-Diode ist dabei eine ausgeprägte Rückstromspitze, welche maßgeblich die Verluste beim aktiven Einschaltvorgang bestimmt. Durch die wesentlich kleinere Sperrverzögerungsladung tritt die Rückstromspitze bei der SiC-Schottky-Diode praktisch nicht auf.
Beim aktiven Ausschalten hingegen sind die auftretenden Verluste in hohem Maße dem, nur langsam abklingenden, Stromschwanz des IGBTs zuzurechnen, der beim Feldeffekttransistor prinzipbedingt nicht vorhanden ist.
Ganz allgemein kann an dieser Stelle festgehalten werden, dass die Verluste bei aktiven Schaltvorgängen im Wesentlichen in den Transistoren anfallen, und deutlich größer sind als bei den passiven Schaltvorgängen, welche hauptsächlich in den Dioden Verluste hervorrufen.

Wie den Tabellen 2.3 und 2.4 entnommen werden kann, scheint allein durch Ersatz der IGBTs und Siliziumdioden durch siliziumcarbidbasierte Halbleiter eine Senkung der Schaltverluste auf etwa $\frac{1}{3}$ der ursprünglichen Werte realistisch, was andererseits eine Steigerung der Schaltfrequenz ermöglicht. In Kapitel 2.2.5 wird näher auf die Ansteuerung dieser Bauteile und Möglichkeiten zur weiteren Senkung der Verluste durch optimierte Treiberschaltungen eingegangen.

Tabelle 2.3: Verluste beim aktiven Schaltvoergang

Verluste in μJ	aktiv EIN			aktiv AUS		
Kombination	Diode	Transistor	Gesamt	Diode	Transistor	Gesamt
A: Si / Si	7.7	157.0	164.7	2.1	142.1	144.2
B: Si / SiC	4.4	56.1	60.5	0.8	136.5	137.3
C: SiC / Si	13.9	168.3	182.2	1.0	57.4	58.4
D: SiC / SiC	2.5	41.2	43.7	1.0	49.4	50.4

Tabelle 2.4: Verluste beim passiven Schaltvoergang

Verluste in μJ	passiv EIN			passiv AUS		
Kombination	Diode	Transistor	Gesamt	Diode	Transistor	Gesamt
A: Si / Si	2.1	4.1	6.2	40.9	18.3	59.2
B: Si / SiC	2.8	4.1	6.9	3.7	0.9	4.6
C: SiC / Si	11.2	3.8	15.0	61.9	0.2	62.1
D: SiC / SiC	6.6	3.4	10.0	8.9	0.8	9.7

Aktives Schaltverhalten

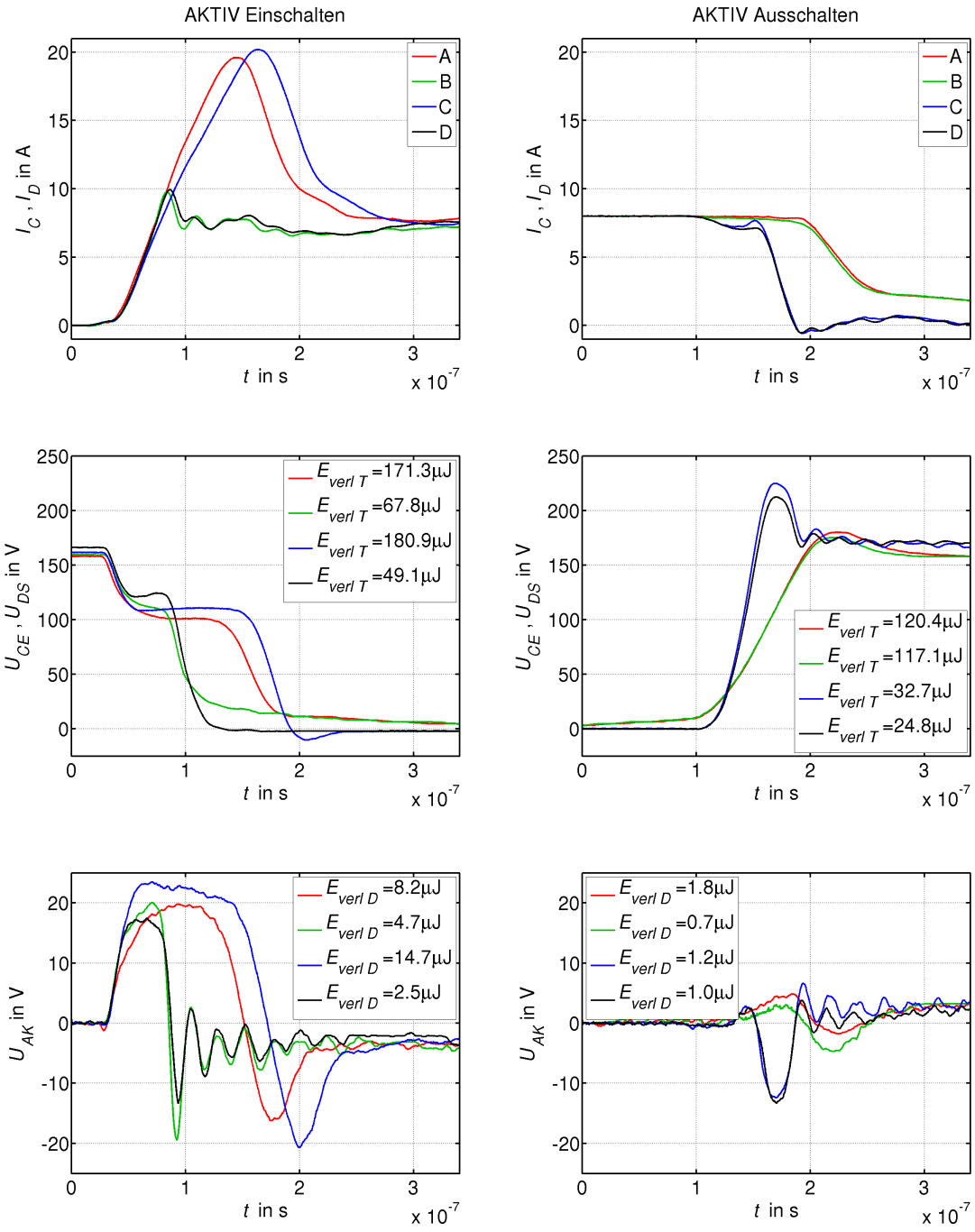


Abbildung 2.4: Aktives Schaltverhalten verschiedenener Halbleiterkombinationen

Passives Schaltverhalten

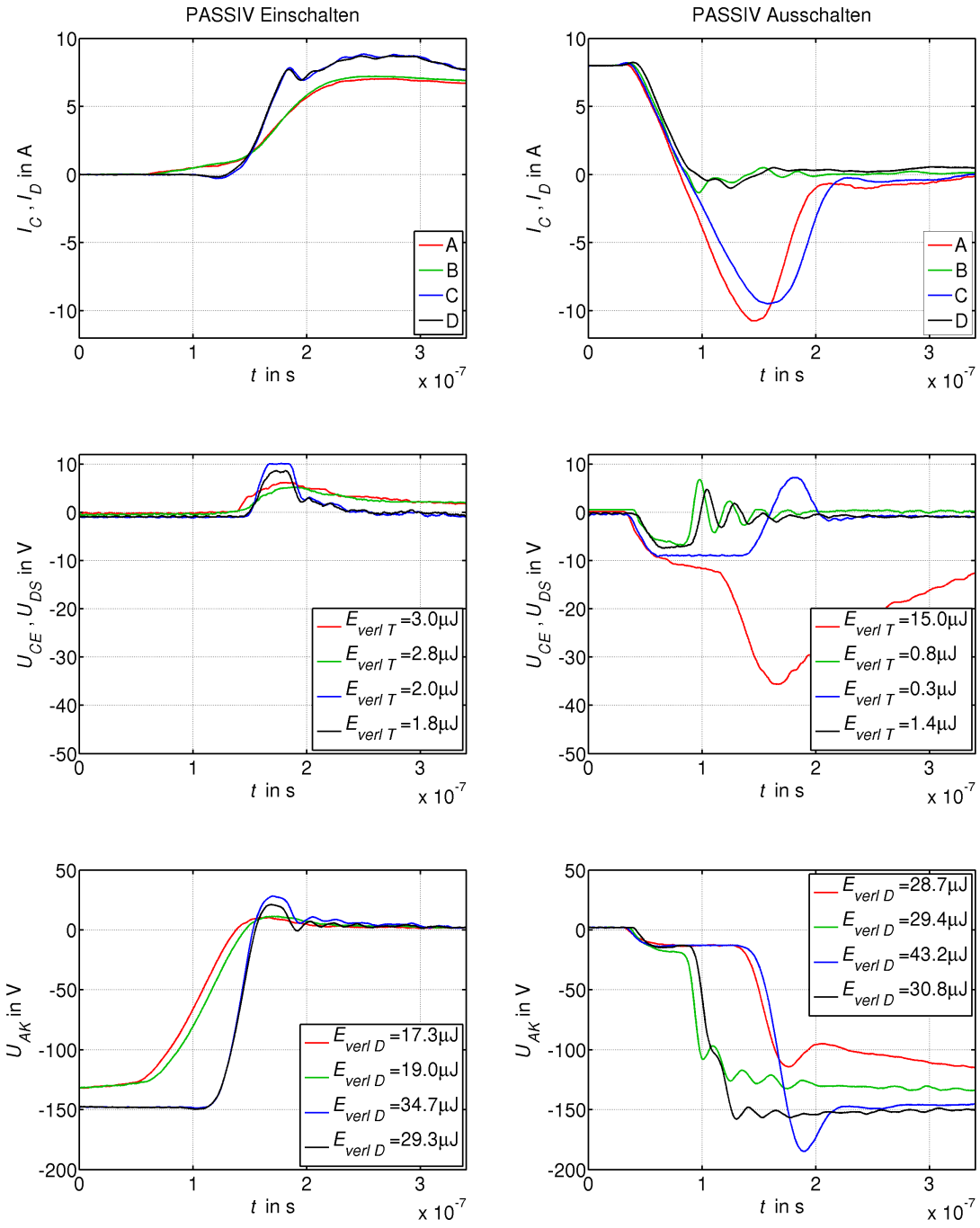


Abbildung 2.5: Passives Schaltverhalten verschiedenener Halbleiterkombinationen

2.2.5 Ansteuerung von Siliziumcarbid-JFETs

U_{DS} in Abhängigkeit des Gate- und Drain- Stromes

Im folgenden Abschnitt wird die Abhängigkeit der Drain-Source-Spannung von Gatestrom und Drainstrom anhand des SJDP0120R085 (selbstleitender JFET) untersucht. Zwar ist dieser Transistor auch bei I_G=*0* leitffähig, doch sollte untersucht werden, ob R_{DSon} bzw. U_{DS} durch Einprägen eines Gatestromes merklich verringert werden können.
Die Verläufe in Abbildung 2.6 legen jedoch nahe, dass auf diese Weise kaum eine Verringerung der Verluste erreicht werden kann.

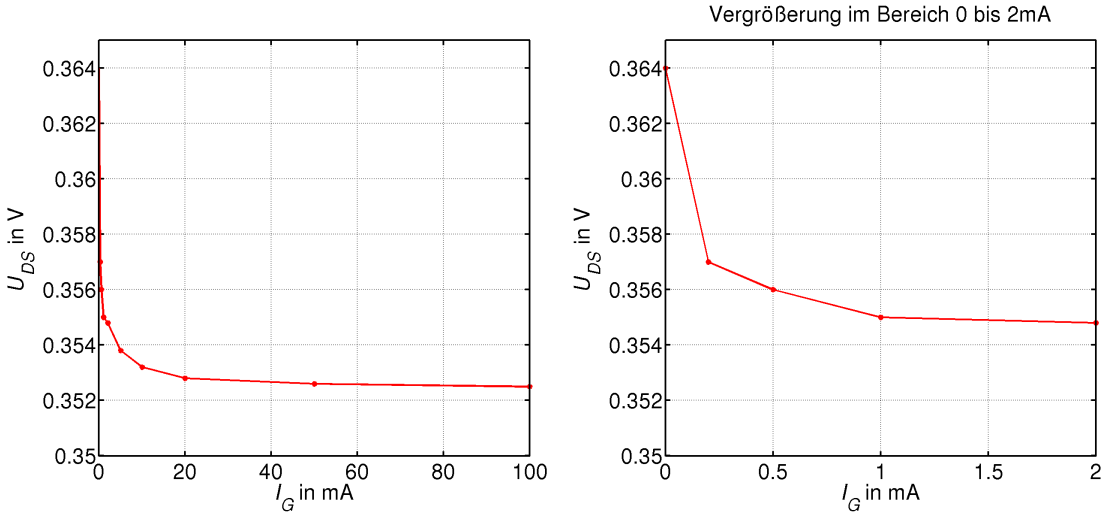


Abbildung 2.6: U_{DS} in Abhängigkeit vom Gatestrom I_G bei I_D=*5A*

Weiters wurde der Verlauf von U_{DS} über I_D bei 2 verschiedenen Gateströmen I_G=*10mA* und I_G=*80mA* ermittelt. Abbildung 2.7 zeigt ein weitestgehend lineares, bzw. ohmsches Verhalten ohne gravierende Abhängigkeit vom Gatestrom und bestätigt den vom Hersteller angegebenen R_{DSon} von $85m\Omega$.

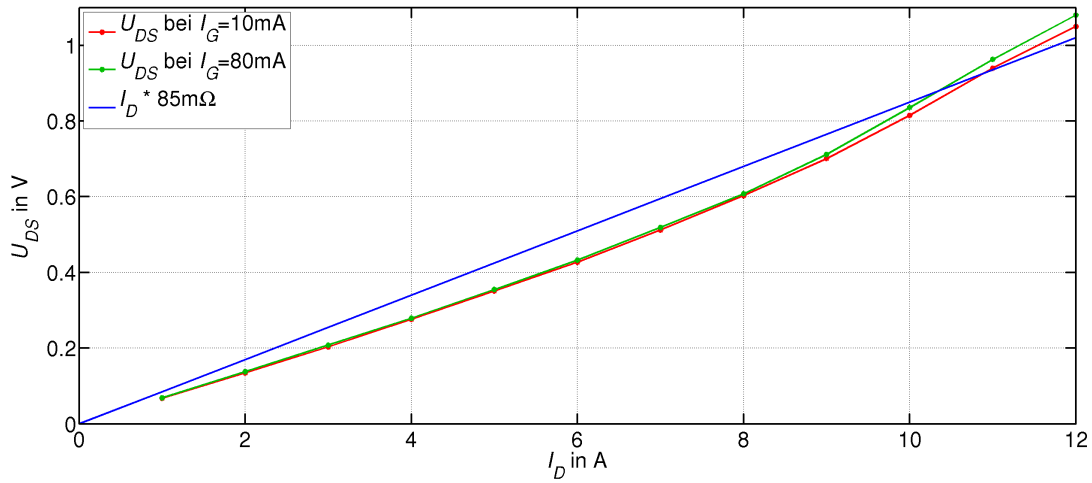


Abbildung 2.7: U_{DS} in Abhängigkeit vom Drainstrom I_G

Schaltverhalten mit unterschiedlichen Treiberschaltungen

Für diese Messung wurde die Testschaltung nach Abbildung 2.3 wie folgt verändert:

- Überbrückung der Diode im aktiv schaltenden Zweig
- Überbrückung des Transistors im passiv schaltenden Zweig
- Ansteuerung des verbleibenden Transistors durch jeweils eine der in Abbildung 2.8 gezeigten Schaltungsvarianten

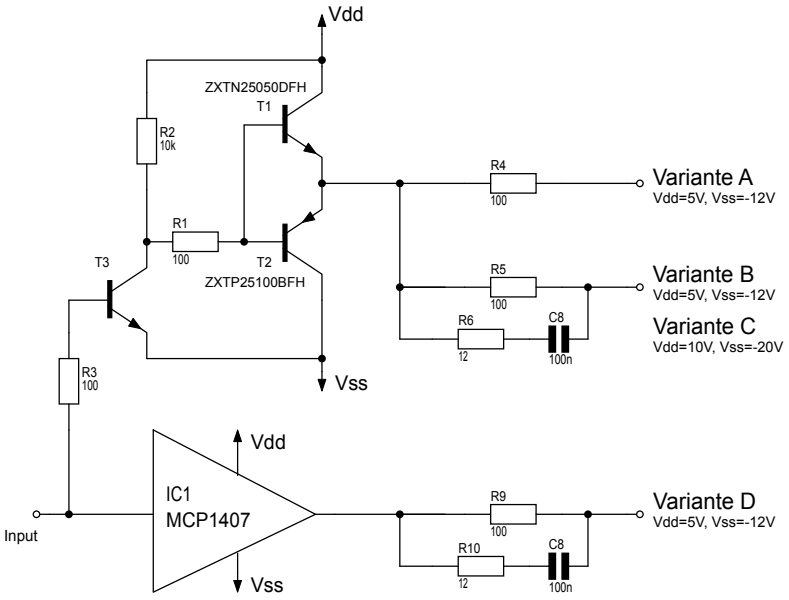


Abbildung 2.8: verschiedene Treiberschaltungen (schematisch)

Die Messergebnisse sind in Abbildung 2.9 dokumentiert. Auch hier sind die in den Leistungshalbleitern umgesetzten Verluste in der Legende eingetragen.
Es ist klar zu sehen, dass eine Ansteuerung per simplem Vorwiderstand nicht ausreicht, um schnelle, möglichst verlustarme, Schaltvorgänge zu gewährleisten.
Während sich die übrigen Versionen im Ausschaltverhalten eher gering unterscheiden, sind beim Einschalten größere Unterschiede vorhanden.
Bemerkenswert ist dabei auch, dass der integrierte Treiber (Variante D), bei identischer Beschaltung, einen merklich schnelleren Einschaltvorgang als der aus diskreten Transistoren aufgebaute Treiber (Variante B), bewirkt.
Weiters ist erkennbar, dass der selbstleitende Typ offenbar Vorteile im Einschaltverhalten gegenüber dem selbstsperrenden Typ aufweist.
Zu beachten ist bei den in die Diagramme eingeblendeten Verlusten, dass sich die mit höherer Flankensteilheit zunehmenden Schwingungen, insbesondere bei sehr kleinen Werten, negativ auf die Genauigkeit der Berechnungen auswirken.

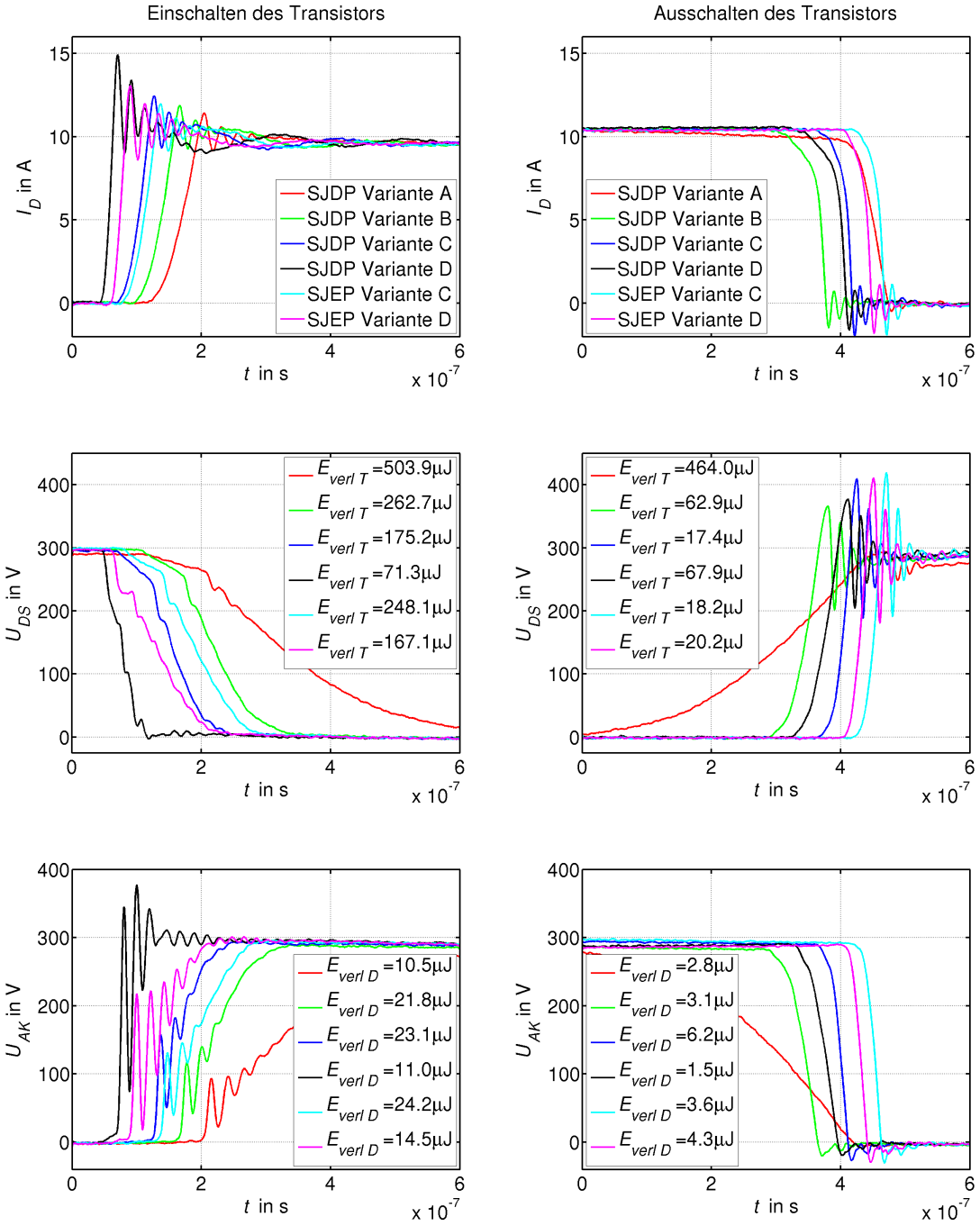


Abbildung 2.9: Schaltverhalten verschiedener SIC-JFETs mit unterschiedlichen Treiberschaltungen

2.2.6 Einfluss einer zum Transistor antiparallel geschalteten Diode

Die beispielsweise bei MOSFETs inhärent vorhandene Invers- bzw. Bodydiode wird bei Umrichtern mit Spannungszwischenkreis meist ohnehin benötigt. Da in der hier gewählten Schaltung des Leistungsteils rückwärts sperrfähige Schalter erforderlich sind und deshalb eine Serienschaltung von Diode/Transistor als Schalter eingesetzt werden muss, hat diese Diode weder im leitenden noch im sperrenden Zustand des Schalters eine Funktion zu erfüllen.

Der Einfluss dieser Inversdiode auf das Schaltverhalten wurde anhand des IXDR30N120 und IXDR30N120*D1* untersucht. Dies sind identische IGBTs, jedoch besitzt der IXDR30N120*D1* eine im Gehäuse integrierte Inversdiode, während der IXDR30N120 ohne diese Inversdiode, in Rückwärtsrichtung kein explizit definiertes Verhalten aufweist.

Unterschiede im Schaltverhalten ergeben sich hier einzig beim 'passiven Ausschalten'.
Abbildung 2.10 zeigt diesen Fall. In das oberste Diagramm eingeblendet sind die insgesamt in einem Schaltelement (Transistor, Seriendiode und ggfs. Inversdiode) beim Schaltvorgang umgesetzten Verlustenergien. Ist die Diode nicht vorhanden, wird ein Teil der negativen Spannung vorübergehend vom Transistor aufgenommen was in einem tendenziell schnelleren Abbau der Rückstromspitze resultiert und dadurch die Verluste minimal senkt.

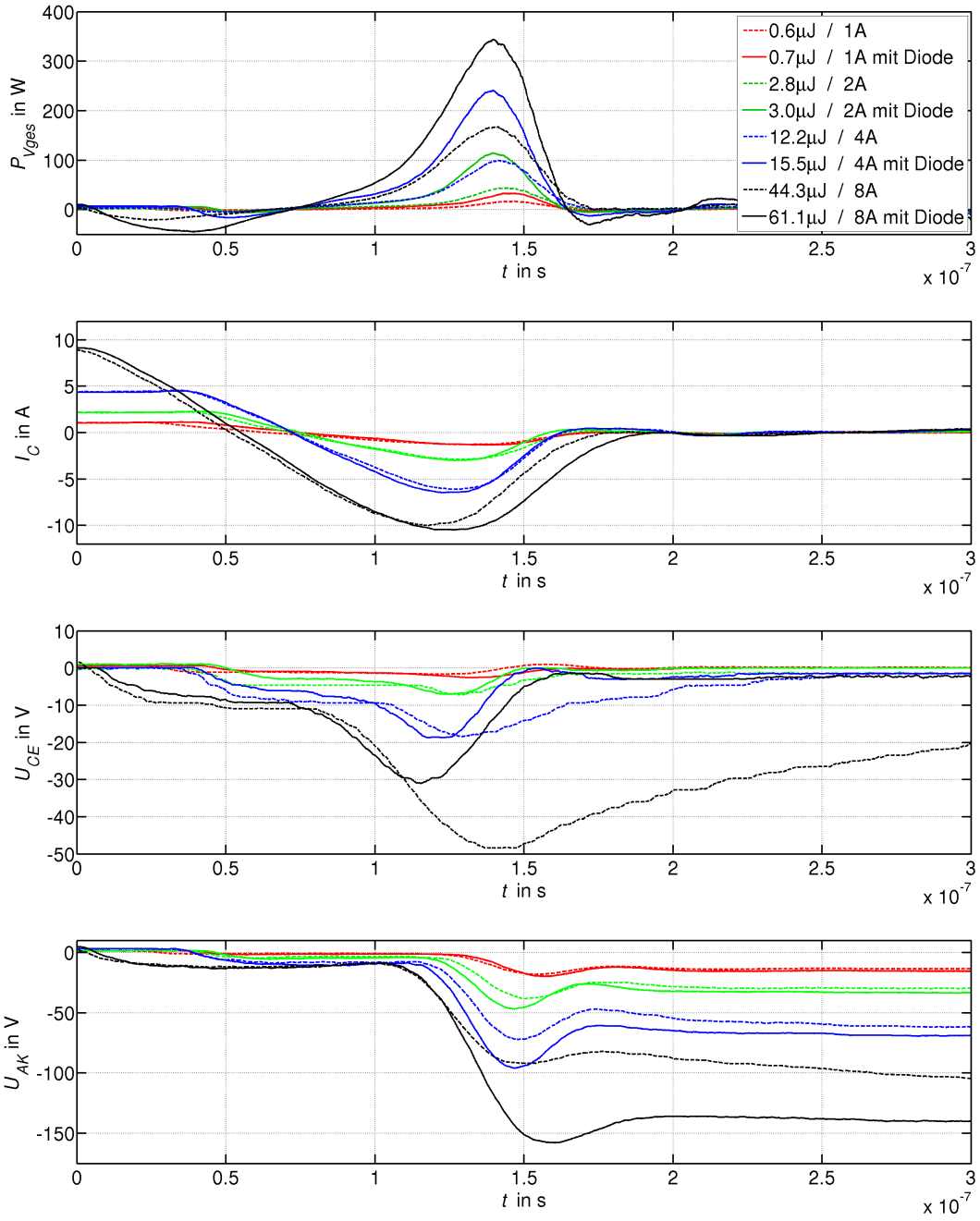


Abbildung 2.10: Einfluss einer zum Transistor antiparallel geschalteten schnellen Diode auf den passiven Ausschaltvorgang

2.2.7 Auswirkungen verschiedener Kommutierungskreisinduktivitäten

Um die Auswirkung von verschieden großen Kommutierungskreisinduktivitäten untersuchen zu können, wurde die Schaltung nach Abbildung 2.3 verwendet. Dazu wurde, wie schon in Kapitel 2.2.5, die Diode des aktiv schaltenden Zweiges sowie der Transistor im passiv schaltenden Zweig überbrückt, und zusätzlich die Zusammensetzung und Verkabelung von C1001 und P1001 (Abbildung 2.3) wie folgt variiert:

Tabelle 2.5: Variation von C1001 und P1001

Beschreibung und Aufbau von C1001	R_{gate}=10Ω	R_{gate}=50Ω
13,2µF Elektrolytkondensator an 350mm Kabel	A	D
obiges + 470nF Folienkondensator an 100mm verdrilltem Kabel	B	E
obiges + 220nF Folienkondensator direkt an die Bauteile gelötet	C	F

Die verwendeten Bauteile sind:

D1001	STTH3012W
T1002	IXDR30N120
L1001	11mH
P1001	variabler Drahtwiderstand, 40Ω

Die Legende in den Diagrammen zeigt jeweils die anhand des Einschaltvorganges angenäherte Kommutierungskreisinduktivität und die im Transistor bei dem betreffenden Schaltvorgang umgesetzte Verlustenergie.

Bemerkenswert ist die geringe Abweichung in der Bestimmung der Kommutierungskreisinduktivität zwischen den Messungen mit $R_G = 10\Omega$ und $R_G = 50\Omega$ (Die hierzu verwendete Näherung ist in den beiden unteren Diagrammen des Einschaltvorganges als Volllinie eingezeichnet.), sowie der äußerst geringe Einfluss auf die Schaltverluste.
Speziell in Hinblick auf das Thema der elektromagnetischen Beeinflussung scheint es dennoch angebracht, die Induktivität des Kommutierungskreises weitestmöglich zu minimieren.

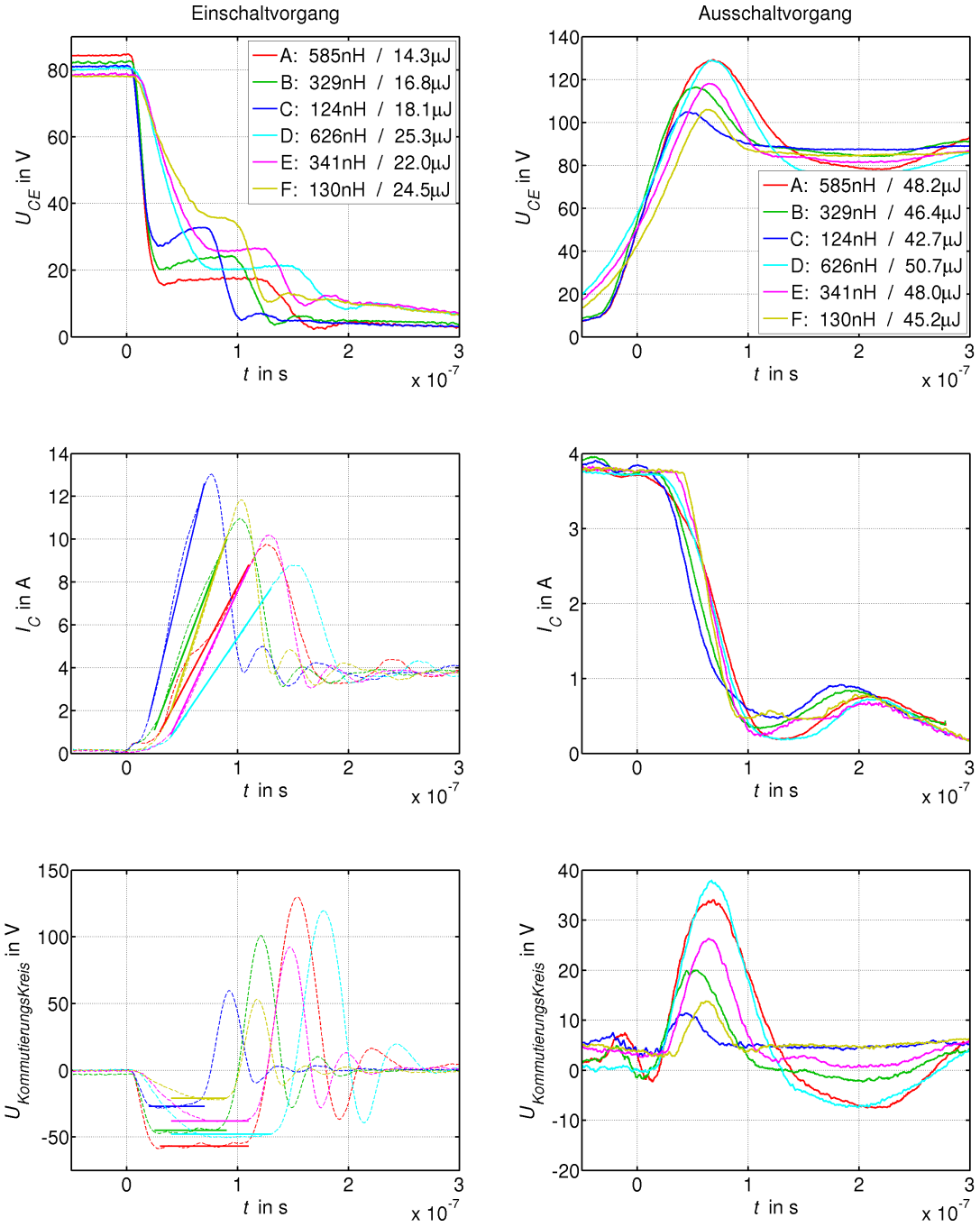


Abbildung 2.11: Auswirkung verschiedener Kommutierungskreisinduktivitäten und Gate-Vorwiderstände auf das Schaltverhalten eines IGBTs

2.2.8 Verwendete Komponenten und gewählte Treiberschaltung

Aufgrund des eindeutig besseren Schaltverhaltens wurde, trotz der gegenüber siliziumbasierten Dioden höheren Flussspannung, entschieden, bei dem Versuchsaufbau Siliziumcarbid- Schottky-Dioden einzusetzen.

Auch bei der Auswahl der Transistoren wurde aus mehreren Gründen für SiC-JFETs entschieden:

- geringere Schaltverluste ermöglichen höhere Schaltfrequenz
- geringere Verluste im leitenden Zustand insbesondere im Teillastbereich
- selbstleitender JFET:
  Wie in Kapitel 2.3.4 erwähnt, muss dem Zwischenkreisstrom stets ein leitfähiger Pfad zur Verfügung stehen, um unkontrollierte Überspannungen zu vermeiden. Beim Einsatz von selbstleitenden Transistoren ist dies sogar beim Totalausfall der Versorgungsspannung gegeben, was in puncto Betriebssicherheit einen großen Vorteil bedeutet.

Abbildung 2.12 zeigt die schließlich gewählte Treiberschaltung. Die Versorgungsspannungen liegen bei $V_{dd} = +5V$ und $V_{ss} = -15V$.
Im Zuge weiterer Versuche konnte bestätigt werden, dass sich Ein- und Ausschaltverhalten, wie in [1] angedeutet, voneinander praktisch unabhängig optimieren lassen, wenn dem Gate zwei getrennte und mittels Dioden entkoppelte Zweige vorgeschaltet werden.

Auf diese Weise konnte der verlustträchtigere Einschaltvorgang weiter optimiert werden ohne beim Ausschalten unnötig hohe Spannungsspitzen hervorzurufen.

Ein kleiner stationärer Gatestrom wird durch R2 erzeugt. Dies dient neben einer geringen Senkung des Kanalwiderstandes auch der Immunität gegen äußere Störungen, da so die Kapazität der Gate-Source-Diode des JFETs bis zur Flussspannung geladen wird, was einen etwas größeren Sicherheitsabstand zur Gateschwellspannung, von ca. $-4V$, bedeutet.
Als Treiber wurde ein IC mit integriertem Optokoppler gewählt um den Schaltungsaufwand zu reduzieren. Der Vorwiderstand der LED ist für den Betrieb am Mikrocontroller (V_{dd}=3V) dimensioniert.

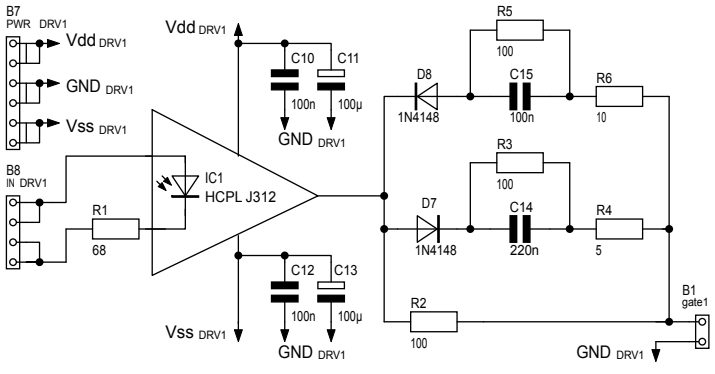


Abbildung 2.12: Treiberschaltung

2.3 Schaltungsentwurf

Im folgenden Kapitel werden die verschiedenen Schaltungsteile des Umrichters beschrieben.

2.3.1 Leistungsteil

Wie schon in Kapitel 2.1 gezeigt, besteht der Leistungsteil des Umrichters (Abbildung 2.13) aus 3 Halbbrücken. Diese sind jeweils aus 2 Siliziumcarbid-JFETs und dazu in Serie geschalteten Siliziumcarbid-Schottky-Dioden aufgebaut, wodurch ein rückwärts sperrfähiges Schaltelement entsteht. Am Eingang befinden sich die Netzfilterkondensatoren, ausgangsseitig ein Überspannungsableiter, die Zwischenkreisinduktivität und eine Steckverbindung zum Einschleifen des Stromwandlers für die Messung des Zwischenkreisstromes. Weiters sind Lastspannung (für motorischen Betrieb) und Lastwiderstand schematisch eingezeichnet. Die Folienkondensatoren C1, C2, C3 und die Keramikkondensatoren C7, C8, C9 sitzen direkt auf der Leiterplatte, um möglichst kleine Kommutierungskreisinduktivitäten zu ermöglichen. Die Elektrolytkondensatoren C4 bis C6 sind aus Platzgründen, wie in Abbildung 2.19 und 2.18 zu sehen, hinter der Leiterplatte angebracht.

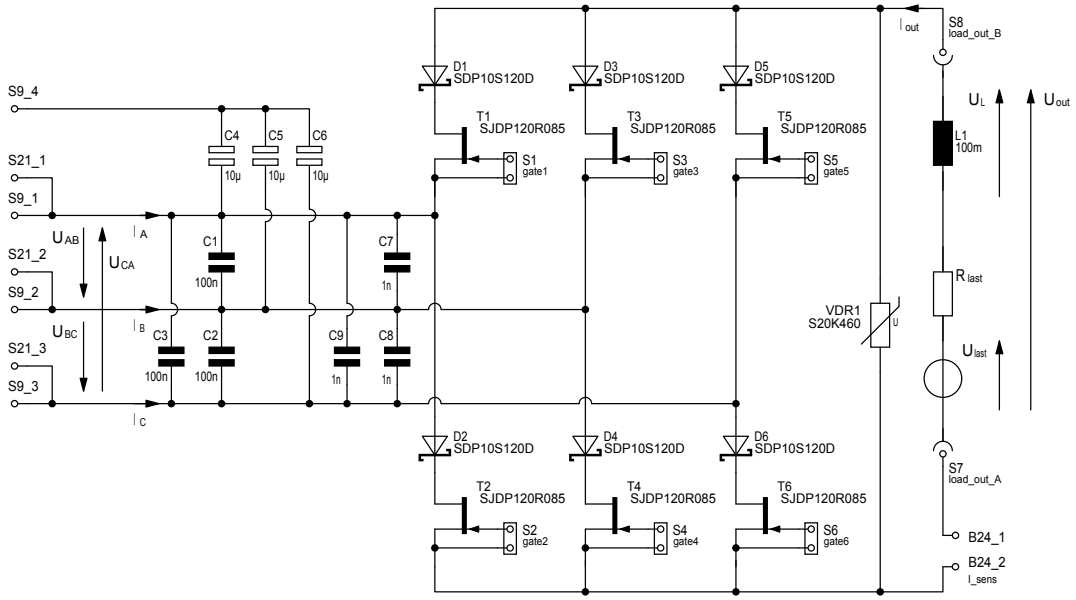


Abbildung 2.13: Schaltung des Leistungsteiles

2.3.2 Treiber

Die Schaltung eines einzelnen Treibers beschränkt sich im Wesentlichen auf den Gate-Drive-Optokoppler selbst und die Beschaltung seines Ausganges, welche in 2.2.8 beschrieben wird.

2.3.3 Spannungsversorgung

Die Spannungsversorgung der Treiber ist in Abbildung 2.14 abgebildet und wird durch jeweils einen Sperrwandler mit 2 Sekundär-Wicklungen realisiert. Um möglichst kleine Übertrager und geringe Windungszahlen verwenden zu können, wurde eine mit 100kHz recht hohe Schaltfrequenz gewählt. Dies ist, abgesehen von der vorteilhafteren Baugröße, auch hilfreich um die kapazitive Kopplung zwischen Primär- und Sekundärseite zu minimieren. Zusätzlich wurden aus diesem Grund Ringkerne verwendet, auf welche an einer Seite die Primärwicklung und an der gegenüberliegenden Seite die Sekundärwicklungen aufgebracht wurden, sodass diese einen maximalen Abstand voneinander aufweisen (Abbildung 2.17).
Die Ausgangsspannungen betragen +5V und -15V. Da die positive Versorgung auch im eingeschalteten Zustand der JFETs kontinuierlich Strom liefern muss, die negative Versorgungsspannung aber allein durch die Umladeströme der Gatekapazität belastet ist, wird die positive Spannung mittels Feedback per Optokoppler geregelt. Die -15V hingegen bleiben ungeregelt und werden allein durch das Windungszahlenverhältnis festgelegt.

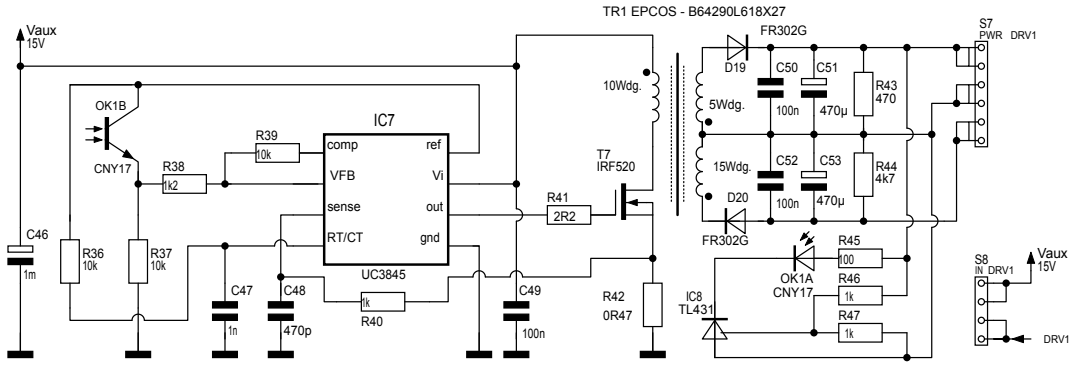


Abbildung 2.14: Spannungsversorgung für die Treiber

2.3.4 Verriegelungslogik

Im Gegensatz zu mit Spannungszwischenkreis betriebenen Brückenschaltungen, wo durch Einhaltung einer Verriegelungszeit sichergestellt werden muss, dass das Speicherelement im Zwischenkreis (im Allgemeinen ein Kondensator) während Umschaltvorgängen nicht kurzgeschlossen wird (break before make), ist beim Betrieb eines Stromzwischenkreises ein Unterbrechen des Stromflusses zu vermeiden (make before break). Hierzu wird das Ausschalten der Leistungshalbleiter um ca. 500ns verzögert, während ohne Verzögerung eingeschaltet wird. Die Verriegelungszeit muss ausreichend groß gewählt werden, damit der einschaltende Transistor voll leitfähig ist, bevor der Ausschaltvorgang beginnt. Diese wurde aus Sicherheitsgründen vorerst sehr groß gewählt, kann aber, wie in Kapitel 2.2 zu sehen, noch deutlich reduziert werden. Realisiert wurde diese Verzögerung durch ein RC-Glied, dessen Widerstand beim Entladevorgang (entspricht dem Einschalten eines Transistors des Leistungsteils) durch eine Diode überbrückt wird.
In Abbildung 2.15 ist eine Hälfte der gesamten Logik dargestellt, welche je einmal für den oberen und einmal für den unteren Teil der B6 Brücke verwendet wird. Alle verwendeten Logikgatter besitzen Eingänge mit Schmitt-Trigger-Verhalten.
Weiters übernimmt diese Schaltung eine Schutzfunktion. Sollte durch eine Fehlfunktion (z.B. des Mikrocontrollers) an keinem der 3 Eingänge ein Singal zum Einschalten des zugehörigen Transistors anliegen, so wird der erste Ausgang eingeschaltet, um einen leitfähigen Pfad für den Zwischenkreisstrom zu garantieren.

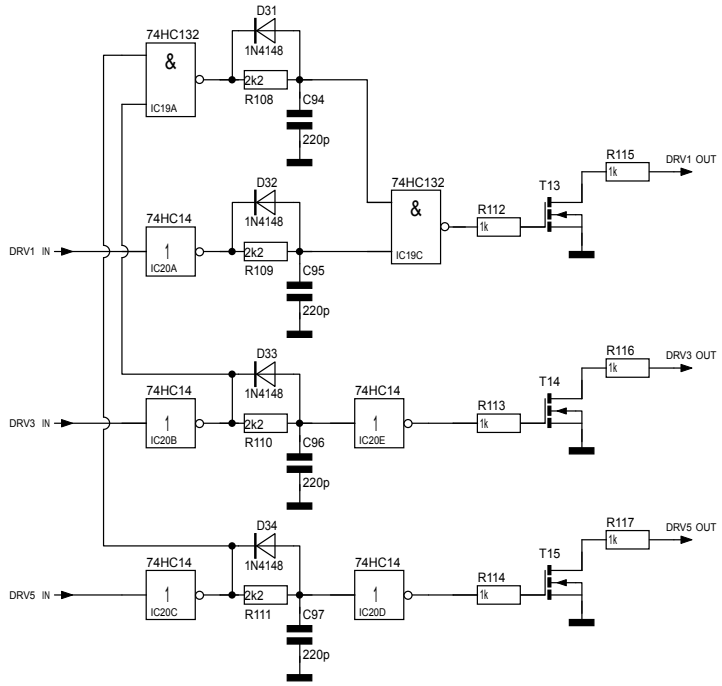


Abbildung 2.15: Verriegelungslogik

2.3.5 Nulldurchgangserkennung

Aufgabe der Nulldurchgangserkennung ist es, bei jedem positiven Nulldurchgang einer Phase der Netzspannung ein Signal an den Mikrocontroller zu senden. Abbildung 2.16 zeigt die hierzu verwendete Schaltung (eine Phase). Die Widerstände R128 - R132 begrenzen den Strom durch die Zenerdiode, welche ihrerseits die Spannung an C103 auf knapp unter 12V limitiert. Bei jedem positiven Nulldurchgang wird C104 umgeladen und steuert dabei kurzzeitig T19, und damit die LED des Optokopplers, an. C102 wird benötigt um Störungen durch der Netzspannung überlagerte Transienten zu unterdrücken. Daraus resultiert bei geringen Eingangsspannungen unter $80V_{eff}$ eine Verzögerung von etwa 500μs welche jedoch bei Bedarf sehr einfach in der Software des Mikrocontrollers kompensiert werden kann. Die Schaltung arbeitet zuverlässig ab einer Netzspannung von ca. $40V_{eff}$, ist bis etwa $600V_{eff}$ einsetzbar und erzeugt dabei etwa 1ms lange Pulse am Ausgang.

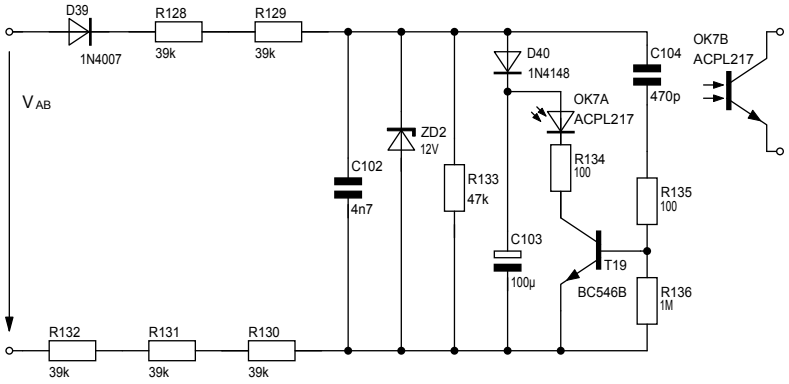


Abbildung 2.16: Nulldurchgangserkennung

2.3.6 Mikrocontrollerboard

Das Mikrocontrollerboard dient als Schnittstelle zwischen dem STM32F4Discovery und den weiteren Komponenten des Umrichters. Abbildung 8.5 zeigt den gesamten Schaltplan. Die wesentlichen Funktionen sind:

- Aufnahme des STM32F4Discovery-Boards
- LC-Display
- Schnittstellen für
  - Analoge Eingangsspannungen bzw. Potentiometer inkl. Pegelkonvertierung von 5V auf 3V
  - Ausgänge zu den Treibern (direkte Ansteuerung der Optokoppler möglich)
  - Logikausgänge um die Treiber mit zwischengeschalteter Verriegelungslogik-Platine anzusteuern
  - Steckerverbindung zu Nulldurchgangsdetektor
- Stromversorgung für Mikrocontroller und LCD aus der 15V Hilfsspannung
- 4 LEDs
- 4 Taster

2.4 Mechanischer Aufbau

Um optimale Voraussetzungen für Verbesserungen, Änderungen und weitergehende Untersuchungen zu schaffen, wurde ein modularer Aufbau des gesamten Umrichters verwirklicht. Weiters wird dadurch die Reparierbarkeit und Wartungsfreundlichkeit deutlich verbessert.
Der Prototyp ist in folgende Module (in Abbildung 2.17 von oben nach unten) unterteilt:

- Mikrocontroller (STM32F4Discovery)
- Mikrocontrollerboard
- Abschirmung
- Verriegelungslogik
- Abschirmung
- Spannungsversorgung der Treiber
- Treiber (2x, stehend hintereinander)
- Leistungsteil mit unterhalb montierten Halbleitern
- Kühlkörper
- rechts seitlich: Stromsensor
- dahinter:
  - Eingangsfilterkondensatoren, Abbildung 2.18 und 2.19
  - Nulldurchgangserkannung, Abbildung 2.19
- im Hintergrund rechts:
  - Zwischenkreisinduktivität, mit Sicherheitslaborkabel am Prototypen angeschlossen

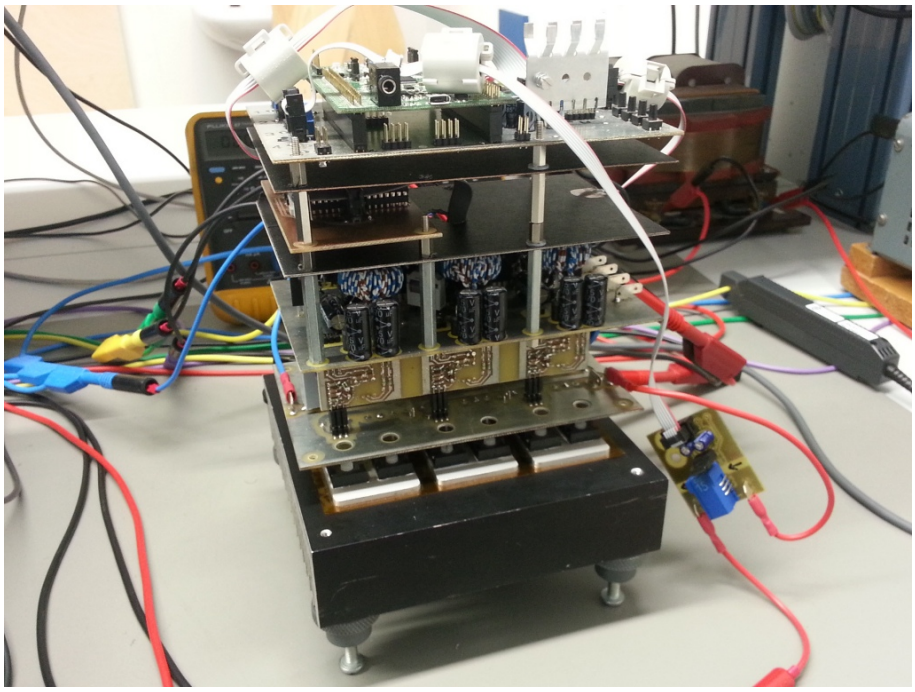


Abbildung 2.17: Ansicht des Umrichters von vorne

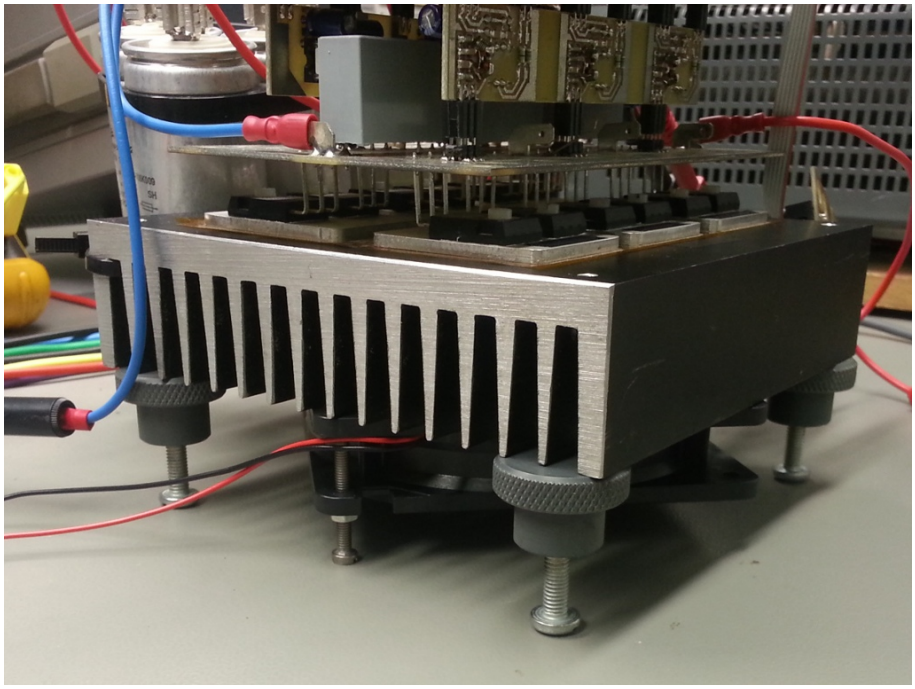


Abbildung 2.18: Anbringung des Lüfters

In Abbildung 2.18 ist der unterhalb des Kühlkörpers angebrachte Ventilator zu sehen, welcher bei übermäßiger Wärmeentwicklung zugeschaltet werden kann.

Um die thermischen Übergangswiderstände von den Leistungshalbleitern zum Kühlkörper zu minimieren, wurde jedes Transistor/Diode- Paar gemeinsam, ohne Isolation und deren zusätzlichen thermischen Widerstand, auf eine Zwischenplatte montiert. Dies ist möglich, da der Drain-Anschluss der Transistoren und die Kathoden der Dioden (diese Anschlüsse sind jeweils mit der Kühlfahne der TO247 Gehäuse verbunden) ohnehin eine leitende Verbindung erfordern. Es muss also nur der Übergang von der Zwischenplatte zum Kühlkörper elektrisch isoliert ausgeführt werden, welcher eine deutlich größere Kontaktfläche besitzt als die Halbleitergehäuse selbst. Daraus folgt ein geringerer thermischer Widerstand der Isolierschicht.

Die Bedienelemente sind in Abbildung 2.19 zu erkennen.
Am linken Rand des Mikrocontrollerboards befinden sich 2 Potentiometer (blau) zur Vorgabe von Sollwerten. Rechts oben befinden sich das Display und 4 LEDs zur Anzeige des Betriebszustandes, darunter 4 Taster mit denen das Gerät bedient wird.

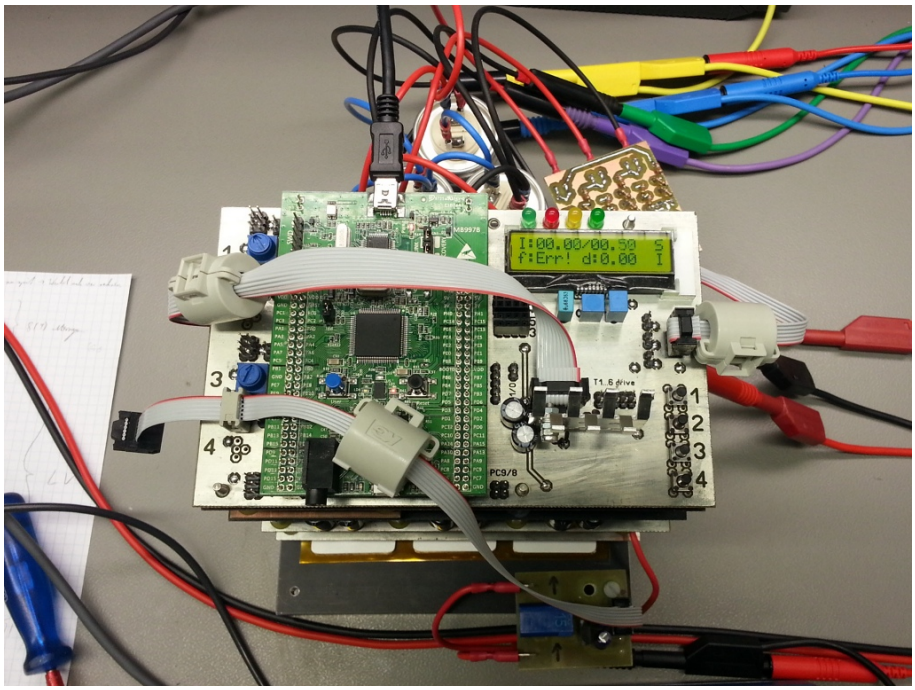


Abbildung 2.19: Ansicht des Umrichters und der Bedienelemente von oben

3 Regelung

Das im Folgenden beschriebene Verfahren zur Regelung des Umrichters arbeitet mit einem PI-Regler und einer Tabelle, welche eine Periode einer Sinusfunktion nachbildet. So müssen abseits des PI-Reglers prinzipiell nur 3 weitere Multiplikationen ausgeführt werden, was eine Implementierung auf Systemen mit geringer Rechenleistung erleichtert, beziehungsweise ermöglicht. Weiters wird die Beeinflussung der Phasenlage des Netzstromes ermöglicht

3.1 Blockschaltbild

In Abbildung 3.1 ist das Blockschaltbild des Umrichters und der Regelung dargestellt. Nach der Netzeinspeisung werden die aktuelle Phasenlage der Netzspannung und die Phasenverschiebung zwischen Eingangsstrom und Netzspannung gemessen. Der Zwischenkreisstrom I_{out} wird aus dem Block B6-Brücke ausgelesen.

Das Tastverhältnis d repräsentiert die an den Ausgang anzulegende Spannung. Es wird in Betrag und Vorzeichen aufgeteilt, da eine negative Ausgangsspannung durch eine geänderte Zuordnung der Schaltzustände erzeugt wird (Block: Auswahl der Schaltzustände).

Das Tastverhältnis wird mit den 3 Momentanwerten der Sinusfunktionen der verketteten Spannungen des Drehstromsystems gewichtet und der Betrag gebildet. Jeder der so erhaltenen 3 Werte stellt die Einschaltdauer für die jeweilige verkettete Spannung dar[1] .

Der so vom Umrichter aufgenommene Strom ist sinusförmig und in Phase mit der Netzspannung. Hinzu addiert sich jedoch der kapazitive Strom der Eingangsfilterkondensatoren. Um vom Netz ausschließlich Wirkstrom zu beziehen, muss die Phasenverschiebung des vom Umrichter aufgenommenen Stromes derart geregelt werden, dass dieser den Blindstrom der Filterkondensatoren aufhebt. Bei dieser Art der Regelung ist das sehr einfach realisierbar indem zum gemessenen Winkel φ ein durch den Phasenverschiebungsregler errechneter Winkel ρ addiert wird. Weiters besteht dabei auch die Möglichkeit, einen weiteren Verbraucher in die Regelung miteinzubeziehen und so dessen Blindstromanteil ebenfalls zu kompensieren.

Die Zuordnung der Schaltzustände wird in Abschnitt 3.2 detailliert beschrieben.

[1] Um die Summe der 3 Einschaltdauern auf den Bereich 0 bis 1 zu skalieren, muss der Faktor 1/2 eingeführt werden, da: $max\ (\ |sin(\varphi)| + |sin(\varphi - 120°)| + |sin(\varphi - 240°)|\ )\ =\ 2$.

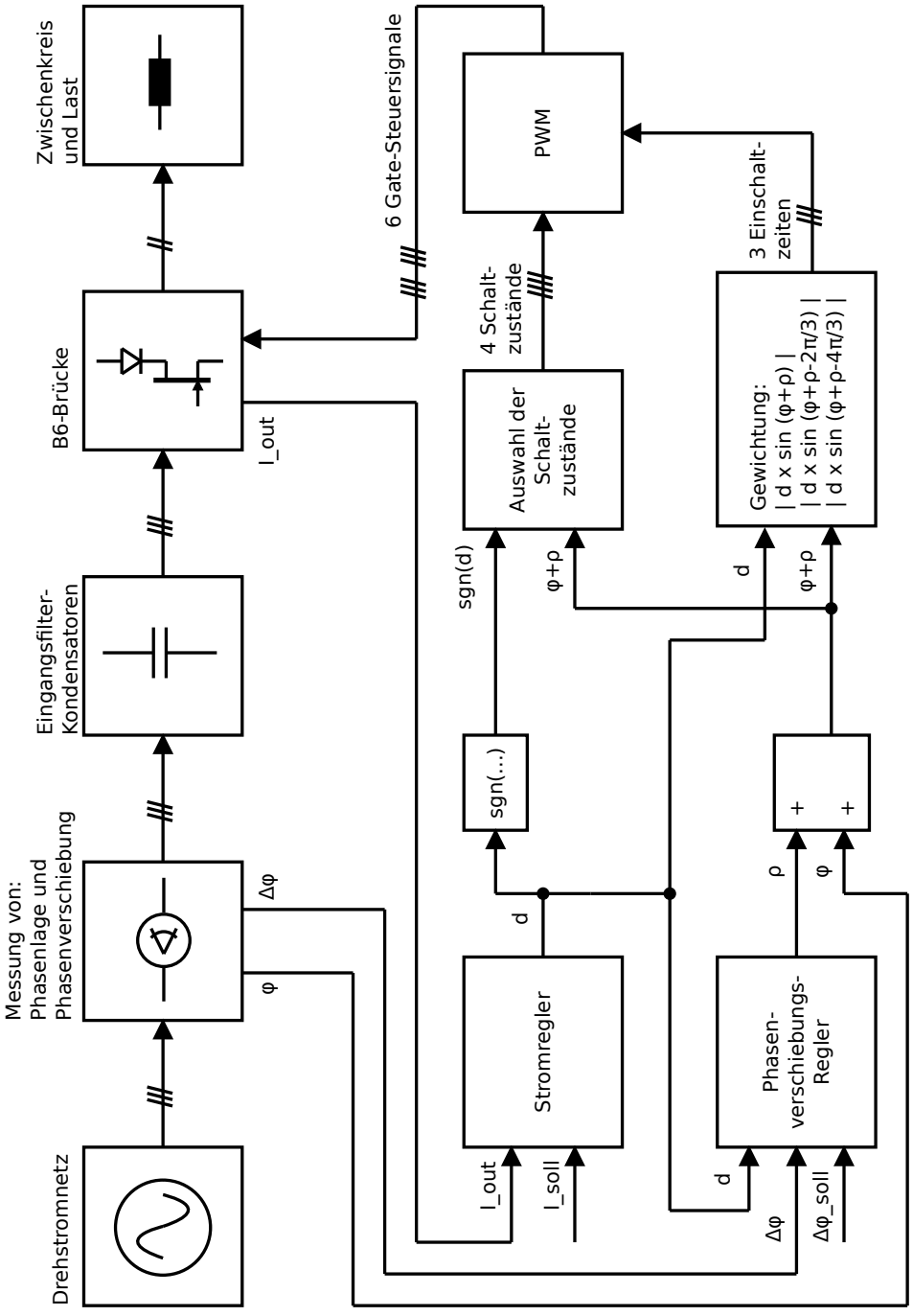


Abbildung 3.1: Blockschaltbild der Regelung des Umrichters

3.2 Schaltzustände und deren Zuordnung

Der Stromzwischenkreisumricher besitzt 9 gültige Kombinationen von Schalterstellungen, Tabelle 3.1 zeigt diese. Die Zustände I bis VI sind aktive Zustände, das heißt sie legen die jeweils angegebene verkettete Spannung an den Zwischenkreis. VII bis IX (Nullzustände) schließen den Zwischenkreis kurz.

Jeder Schaltzustand beinhaltet die Schalterstellungen der Leistungsschalter (Abbildung 3.2) in dem jeweiligen Zustand, für I ist dies beispielsweise: [1,0,0,1,0,0] (= 1. Zeile in Tabelle 3.1).

Tabelle 3.1: Schalterstellungen in den 9 gültigen Zuständen

angelegte Spannung	Schalter / Zustand	S1	S2	S3	S4	S5	S6
$-U_{AB}$	I	1	0	0	1	0	0
$-U_{BC}$	II	0	0	1	0	0	1
$-U_{CA}$	III	0	1	0	0	1	0
U_{AB}	IV	0	1	1	0	0	0
U_{BC}	V	0	0	0	1	1	0
U_{CA}	VI	1	0	0	0	0	1
Nullzustände	VII	1	1	0	0	0	0
	VIII	0	0	1	1	0	0
	IX	0	0	0	0	1	1

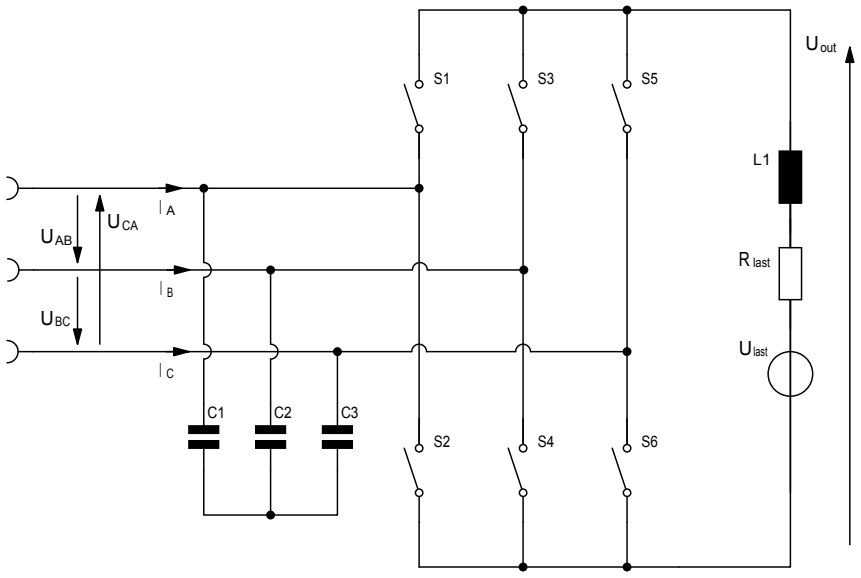


Abbildung 3.2: Skizze zu den Schalterstellungen

In Abbildung 3.3 wird eine Periode der Netzspannung in 6 Abschnitte unterteilt. Die Grenzen zwischen den Abschnitten sind durch die Nulldurchgänge der 3 verketteten Spannungen definiert. Tabelle 3.2 zeigt die nötigen Zustände, um die Spannungen U_{AB}, U_{BC} und U_{CA} in positiver Richtung an den Zwischenkreis zu legen. Für negative Ausgangsspannungen kann entweder die in Tabelle 3.3 gezeigte Zuordnung, oder die Zustände des dritt nächsten Abschnittes verwendet werden.

Der Block 'Auswahl der Schaltzustände' im Blockschaltbild enthält eben diese Tabellen. Aus den Eingangsgrößen $\varphi + \rho$ (aktuelle Phasenlage + Verschiebung durch Regelung) und Vorzeichen des Tastverhältnisses werden die aktuell benötigten Schaltzustände (3 aktive Zustände, 1 Nullvektor) ermittelt.

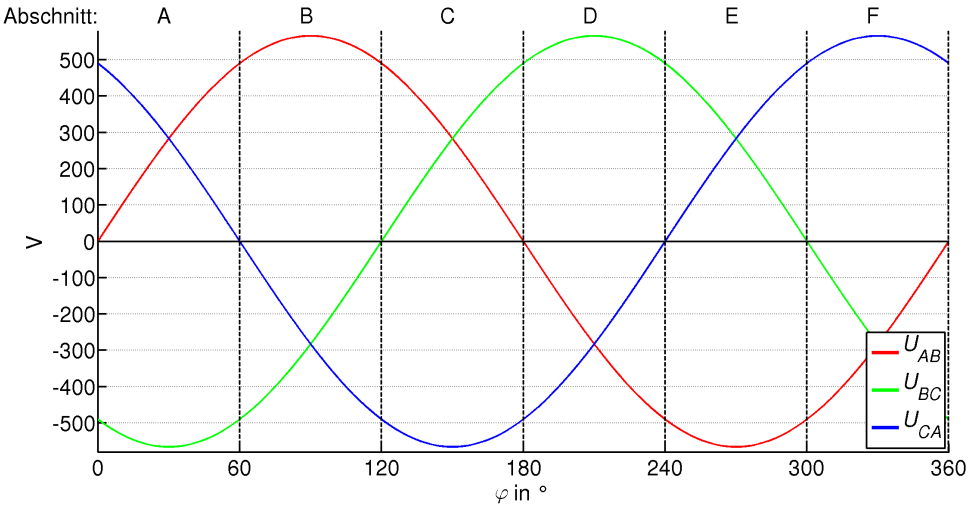


Abbildung 3.3: Abschnitte der Netzspannung

Tabelle 3.2: aktive Schaltzustände für *positive* Ausgangsspannung in den jeweiligen Abschnitten

Spannung \ Abschnitt	A	B	C	D	E	F
U_{AB}	IV			I		
U_{BC}	II		V			II
U_{CA}	VI	III			VI	

Tabelle 3.3: aktive Schaltzustände für *negative* Ausgangsspannung in den jeweiligen Abschnitten

Spannung \ Abschnitt	A	B	C	D	E	F
$-U_{AB}$	I			IV		
$-U_{BC}$	V		II			V
$-U_{CA}$	III	VI			III	

3.3 Verringerung der Schaltverluste

Werden die Schaltzustände in der Reihenfolge der zugehörigen Phasenspannungen (U_{AB}, U_{BC}, U_{CA}) ausgegeben, so müssen bei einem Übergang zum nächsten Zustand oftmals mehr als 2 Schalter umgeschaltet werden.

Beispielsweise bedeutet der Übergang von II (aktive Schalter: S6, S3) auf III (aktive Schalter: S2, S5) in Abschnitt B, einen Schaltvorgang in 4 Schaltelementen.

Zur Verringerung der Schaltverluste ist es zweckmäßig, die Reihenfolge derart festzulegen, dass bei jedem Übergang nur ein Schaltelement ein- und ein anderes ausgeschaltet wird. In Tabelle 3.4 ist eine Möglichkeit dies zu erreichen dargestellt.

Tabelle 3.4: Zugunsten geringerer Schaltverluste optimierte Reihenfolge der Schaltzustände (für positive Ausgangsspannung)

Zustände \ Abschnitt	A	B	C	D	E	F
1. aktiver Zustand:	IV	III	V	I	VI	II
Spannung:	U_{AB}	$-U_{CA}$	U_{BC}	$-U_{AB}$	U_{CA}	$-U_{BC}$
aktive Schalter:	2/3	2/5	4/5	4/1	6/1	6/3
2. aktiver Zustand:	II	IV	III	V	I	VI
Spannung:	$-U_{BC}$	U_{AB}	$-U_{CA}$	U_{BC}	$-U_{AB}$	U_{CA}
aktive Schalter:	6/3	2/3	2/5	4/5	4/1	6/1
3. aktiver Zustand:	VI	II	IV	III	V	I
Spannung:	U_{CA}	$-U_{BC}$	U_{AB}	$-U_{CA}$	U_{BC}	$-U_{AB}$
aktive Schalter:	6/1	6/3	2/3	2/5	4/5	4/1
Nullzustand:	VII	IX	VIII	VII	IX	VIII
aktive Schalter:	2/1	6/5	4/3	2/1	6/5	4/3

3.4 optimiertes Modulationsverfahren

Wie in Abbildung 3.4 zu sehen und in Fußnote 1 auf Seite 31 erwähnt, erreicht die Summe der Absolutbeträge der 3 um 120° versetzten Sinusfunktionen ein Maximum von 2 (Kurve 'sum'). Um also die gesamte Einschaltzeit auf den Bereich 0 bis 1 (0-100%) zu skalieren, muss ein Vorfaktor von 1/2 eingeführt werden.

Wie beim Spannungszwischenkreisumrichter besteht jedoch auch hier die Möglichkeit die Aussteuerung, bei weiterhin sinusförmigem Eingangsstrom, zu vergößern. Dazu wird zu jedem Zeitpunkt der Medianwert der 3 Sinusfunktionen (a,b,c) bestimmt und von selbigen subtrahiert. Es ergeben sich die Verläufe A, B und C. Die Summe der Beträge von A, B und C (SUM) hat ein kleineres Maximum als zuvor, die Differenzen (A-B, B-C, C-A) jedoch verändern sich dabei gegenüber den rein sinusförmigen Funktionen nicht.
Der Vorfaktor kann nun auf $1/\sqrt{3}$ erhöht werden, was eine um ca. 15% höhere maximale Ausgangsspannung bedeutet.

Weiters hat dieses Vorgehen den positiven Nebeneffekt, dass immer eine der 3 Einschaltzeiten null wird und somit nurmehr 2 aktive Schaltzustände pro Periode ausgegeben werden, was die Schaltverluste weiter verringert.

Bei der Realisierung ist eine Berechnung des Medians zur Laufzeit nicht erforderlich, es wird schlicht die Funktion |A| anstatt |a| als Lookuptable verwendet.

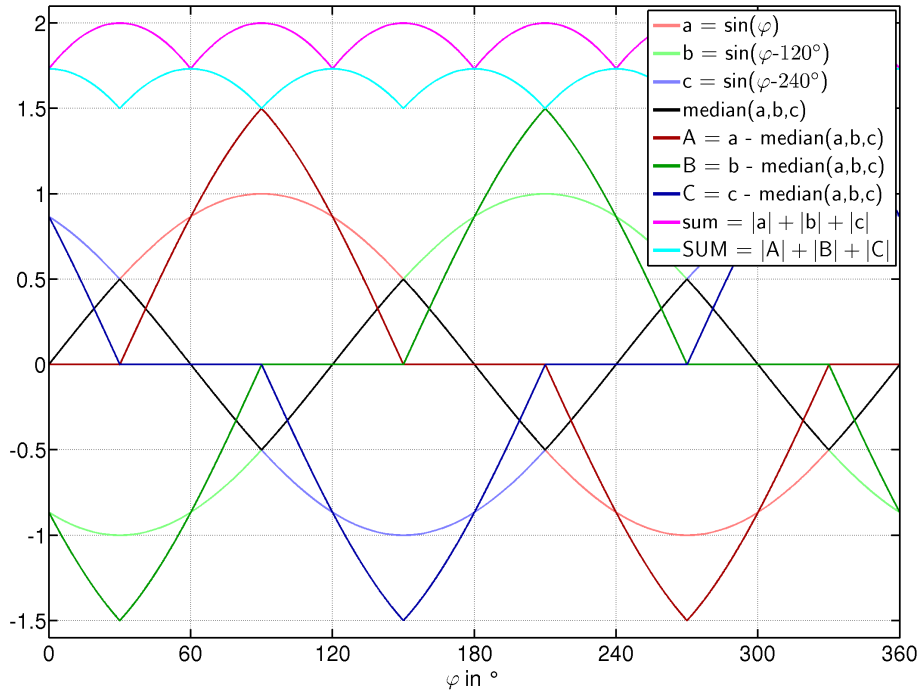


Abbildung 3.4: Signalverläufe für optimiertes Modulationsverfahren

4 Simulation

4.1 MATLAB®/Simulink® -Modell des Umrichters

Um das Verhalten der Hardware, und insbesondere der entworfenen Regelstrategie, bereits vor dem Aufbau eines Prototypen testen zu können, wurde mit Hilfe von MATLAB® /Simulink® unter Verwendung der SimPowerSystems-Toolbox ein Modell des gesamten Umrichters erstellt. Dies ermöglichte es, Modifikationen an der Regelung des Systems gefahrlos und mit verhältnismäßig geringem Aufwand vorzunehmen, anstatt die Programmierung des Mikrocontrollers verändern zu müssen.

4.1.1 Gesamtübersicht

In Abbildung 4.1 ist eine Übersicht des verwendeten Modells abgebildet.
In dieser Abbildung enthalten ist unter anderem auch der Block 'Winkelregelung', welcher durch die Konstante 'Winkelregelung ein' (=0) umgangen wird. Die Winkelregelung dient der in Kapitel 3 erwähnten Möglichkeit den vom Umrichter aufgenommenen Strom in Relation zur Netzspannung so zu verschieben, dass sich zusammen mit dem voreilenden Strom der Eingangsfilterkondensatoren eine bestimmte Verschiebung (meist ist hier $\varphi = 0$, bzw. ein Leistungsfaktor von 1 gewünscht) ergibt. Da eben diese Funktion jedoch in starker Wechselwirkung mit der Stromregelung steht[1],ist hier eine sehr feine Abstimmung erforderlich. Aus zeitlichen Gründen wurde daher lediglich die prinzipielle Funktion getestet, aber darauf verzichtet, diese Möglichkeit bis zu einem unter allen Bedingungen funktionierenden System zu entwickeln. Bei den Ergebnissen der Simulation in Kapitel 4.2 wird daher auf eine Darstellung verzichtet.
Die Diskretisierungszeit wurde nach den jeweiligen Bedürfnissen angepasst, für längere Simulationen hat sich ein Wert von 500ns als gut geeignet erwiesen.
Der Block Kompensation an der Netzzuleitung enthält (für jede der 3 Phasen) ein R-C-L-Glied mit dem Schwingungen aufgrund steilflankiger Transienten, welche hier von der sprunghaften Addition der Oberschwingungen herrühren, (s. Kapitel 6.3) verringert werden konnten. Am realen Objekt konnte damit jedoch keine wesentliche Verbesserung erreicht werden.

[1] Aufgrund der mit steigender Verschiebung sinkenden mittleren Ausgangsspannung sinkt auch die Verstärkung der Stromregelung.

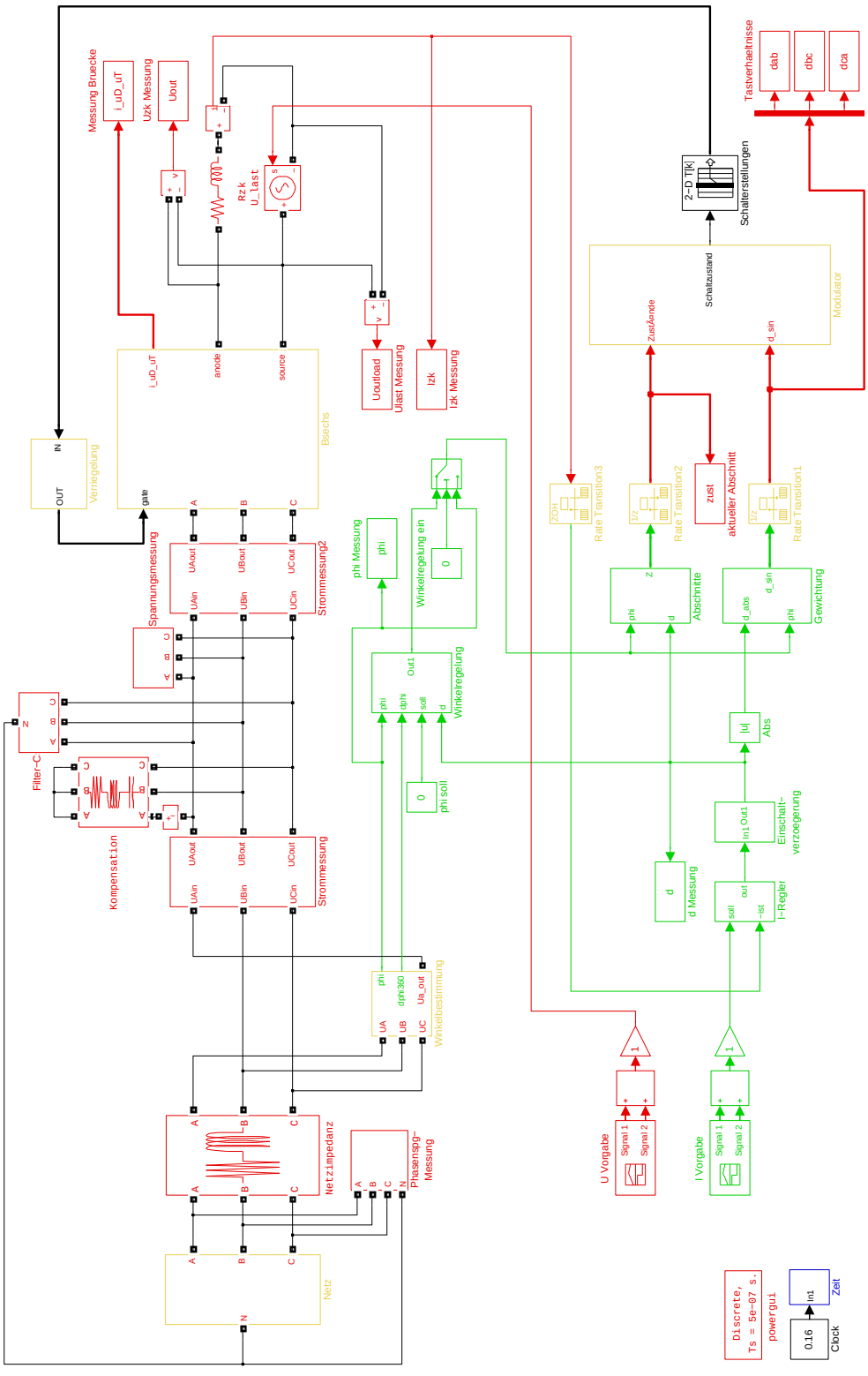


Abbildung 4.1: Simulink®-Modell des Gesamtsystems

4.1.2 Stromregelung

Die Stromregelung konnte aufgrund des in Kapitel 3 beschriebenen Verfahrens sehr einfach gehalten werden. Sie besteht lediglich aus einem PI-Regler mit Anti-Windup-Maßnahme um das Überschwingen bei Sollwertänderungen zu verringern.

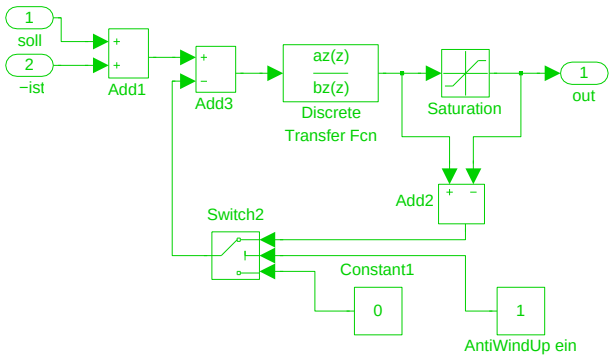


Abbildung 4.2: Simulinkmodell der Stromregelung

4.1.3 Gewichtung der Einschaltzeiten

Der Block Gewichtung errechnet aus dem vom Stromregler ausgegebenen Tastverhältnis und der aktuellen Phasenlage der Netzspannung die 3 relativen Einschaltzeiten für die 3 Phasenströme. Dazu werden aus dem aktuellen Winkel die Augenblickswerte der 3 zueinander um 120° versetzten Sinusfunktionen berechnet, mit dem Tastverhältnis multipliziert, entsprechend skaliert und der Absolutbetrag davon ausgegeben.

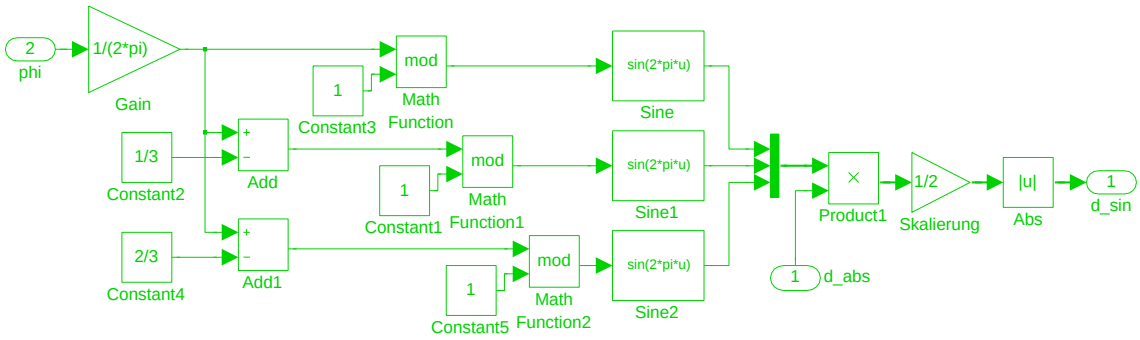


Abbildung 4.3: Gewichtung der Einschaltzeiten

4.1.4 Winkelbestimmung

Der Simulationsblock 'Winkelbestimmung' wurde in einer Art aufgebaut, die die Implementierung am Mikrocontroller mittels Interrupts bei den Nulldurchgängen nachbildet. Am Eingang werden die Nulldurchgänge bestimmt und damit Zähler zurückgesetzt. Sample/Hold-Glieder speichern jeweils den Maximalwert der Zähler. Aus dem Quotienten des aktuellen Wertes und dem zugehörigen Maximalwert wird der aktuelle Winkel bestimmt und in Radiant umgerechnet. Dies erfolgt bei allen 3 verketteten Spannungen. Ausgegeben wird jeweils derjenige Wert, der zuletzt durch einen Nulldurchgang zurückgesetzt wurde.
Weiters ist in Abbildung 4.4, im unteren Teil, auch der Entwurf zur Messung der Phasenlage des Netzstromes der Phase A zu sehen. Wie in 4.1.1 beschrieben, wurde diese Option jedoch nicht weiter optimiert und nur die grundsätzliche Funktion getestet.

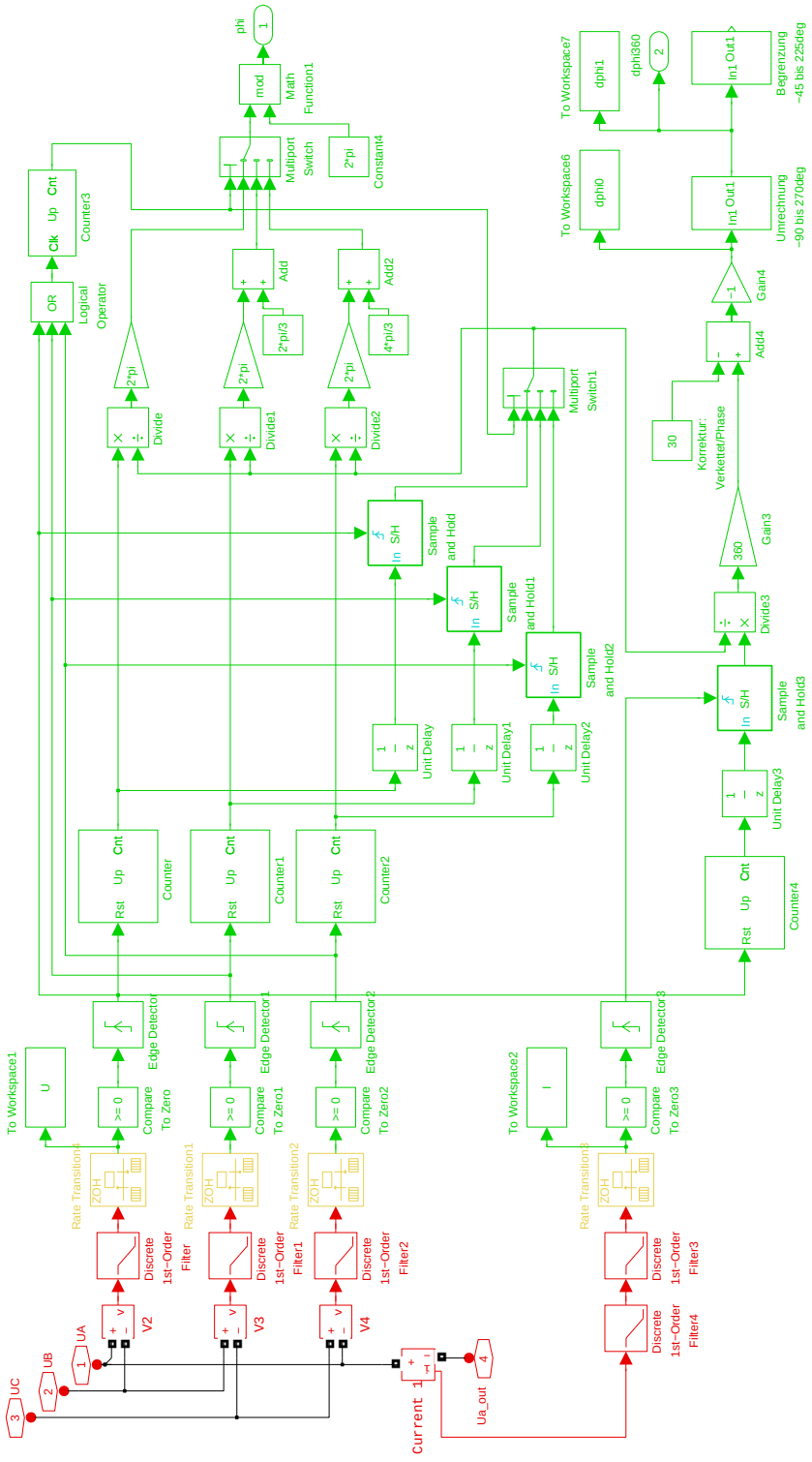


Abbildung 4.4: Bestimmung der aktuellen Phasenlage der Netzspannung und Phasenverschiebung des Netzstromes

4.2 Simulationsergebnisse

Die in der Simulation betrachteten Lastfälle und Ereignisse sind in Abbildung 4.5 dargestellt. Tabelle 4.1 zeigt die zu den jeweiligen Zeitpunkten auftretenden Ereignisse.

Tabelle 4.1: Betriebszustände in der Simulation

$t = 0.01s$	Sollwertsprung auf 15A
$t = 0.02s$	Sprung der Lastspannung auf +200V; motorischer Betrieb
$t = 0.04s$	Beginn Netzoberschwingung (5.harm.) mit 10% der Grundschwingungsamplitude
$t = 0.06s$	Beginn Netzoberschwingung (5.harm.) mit 20% der Grundschwingungsamplitude
$t = 0.08s$	Ende Netzoberschwingungen
$t = 0.09s$	Sprung der Lastspannung auf -200V; generatorischer Betrieb
$t = 0.11s$	Sprung der Lastspannung auf 0V; rein ohmsche Last
$t = 0.12s$	Sollwertsprung auf 0A

Im Vergleich zu den Messungen am Prototypen (Kapitel 6) zeigt die Simulation etwas größere Oberschwingungsanteile in den Netzströmen.
Dennoch verspricht die Simulation auch bei hohem Oberschwingungsanteil ein beherrschbares Verhalten des Umrichters.
Sowohl Sollwert- als auch Lastspannungssprünge werden problemlos verarbeitet.

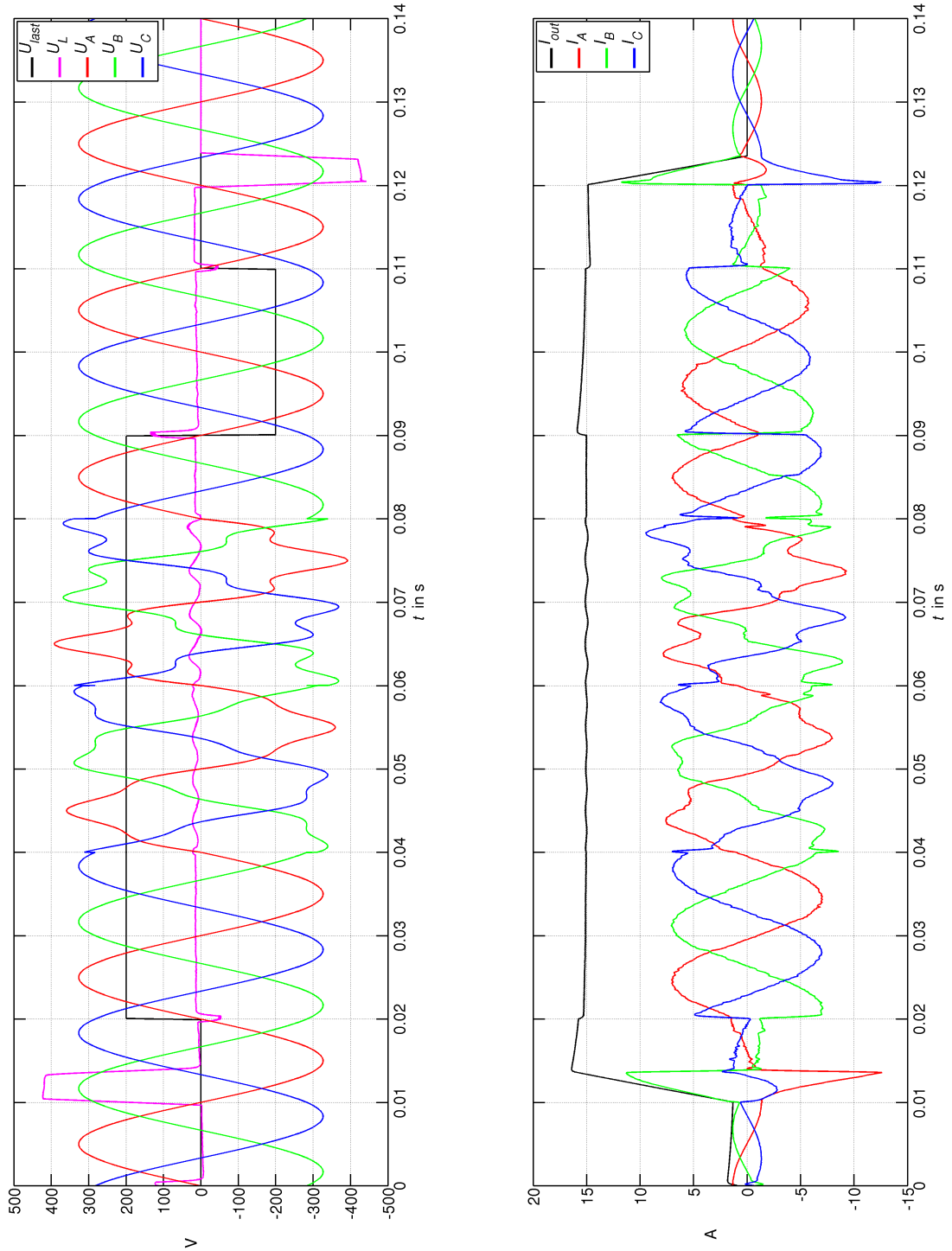


Abbildung 4.5: Ergebnis der Simulation

5 Software

Diesem Kapitel soll vorausgeschickt werden, dass im Anhang dieser Arbeit der vollständig kommentierte Quellcode des Mikrocontrollerprogrammes enthalten ist. Es wird hier ein Überblick über die Struktur und Funktion des Programms und der einzelnen Funktionen gegeben. Zudem wird detailliert auf die Interruptroutinen, welche die Hauptaufgaben übernehmen, eingegangen.

5.1 Anzeigen und Bedienelemente

Die Bedienelemente und Anzeigen des Umrichters bzw. des Mikrocontrollerboards sind in Abbildung 5.1 zu sehen. Ihre Funktion ist in Tabelle 5.1 beschrieben.

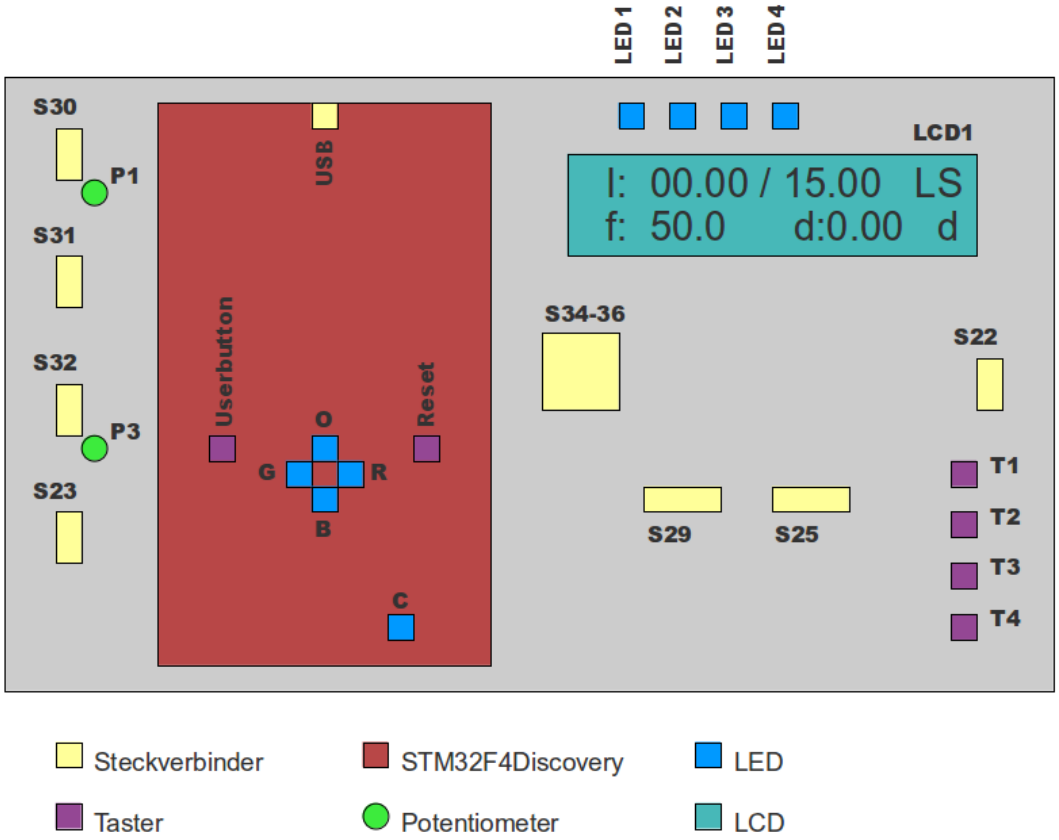


Abbildung 5.1: Bedienelemente und Stecker an der Oberseite des Umrichters

Tabelle 5.1: Bedienelemente und Stecker an Mikrocontrollerboard und STM32F4Discovery

Bezeichnung	Funktion
S30 / P1	Analog Input Nr. 1, mit P1 wird der Sollwert für I_{out} vorgegeben.
S31	Analog Input Nr. 2, hier nicht verwendet.
S32 / P3	Analog Input Nr. 3, mit P3 wird der alternative Zwischenkreisstrom für Sollwert-Sprünge vorgegeben.
S23	Analog Input Nr. 4, der Istwert von I_{out} vom Stromwandler.
USB	Mini-USB-Buchse, dient der Programmierung des Mikrocontrollers.
S34-36	Stromversorgung für eventuelle Erweiterungen (Masse, 3V, 5V).
S29	Verbindung zur Verriegelungslogik
S25	Direkt-Verbindung zu den Eingängen der Treiber-ICs, falls die Verriegelungszeiten in der Software des Mikrocontrollers realisiert werden und daher die Verriegelungslogik entfallen kann.
S22	Verbindung zur Nulldurchgangserkennung
LED1	Signalisiert, dass die Netzspannung in korrekter Phasenfolge anliegt und der Umrichter darauf synchronisiert ist.
LED2	Deutet auf einen Fehler bei der Netzsynchronisation (z.B. falsche Phasenfolge, eine fehlende Phase, usw...) hin.
LED3	Blinkt auf, wenn ein Sollwertsprung ausgelöst wird.
LED4	Zeigt, dass der Umrichter eingeschaltet ist.
R	Rote LED des STM32F4Discovery, keine besondere Funktion.
G	Grüne LED des STM32F4Discovery, keine besondere Funktion.
B	Blaue LED des STM32F4Discovery, keine besondere Funktion.
O	Orange LED des STM32F4Discovery, keine besondere Funktion.
C	USB-OTG-Overcurrent LED des STM32F4Discovery
Userbutton	Hiermit kann in den Betriebsmodus 'konstantes Tastverhältnis' umgeschaltet werden. Das Tastverhältnis wird dann mit P1 im Bereich von -0,99 bis 0,99 vorgegeben.
Reset	Setzt den Mikrocontroller zurück.
T1	Schaltet den Umrichter ein und aus.
T2	Bewirkt eine Verschiebung des aufgenommenen Netzstromes gegenüber der Netzspannung (Blindleistungserzeugung) um einen festen Winkel.
T3	Löst einen Sprung des Zwischenkreisstromsollwertes aus.
T4	Schaltet in den Modus mit optimierter Aussteuerung.
LCD	• erste Zeile – Soll- und Ist-Wert des Zwischenkreisstromes – Blindleistungserzeugung (L) – Normale (S) bzw. optimierte Aussteuerung(Λ) • zweite Zeile – Netzfrequenz – aktuelles Tastverhältnis – Stromregelung aktiv (I) bzw. konstantes Tastverhältnis (d)

5.2 Struktur des Programmes

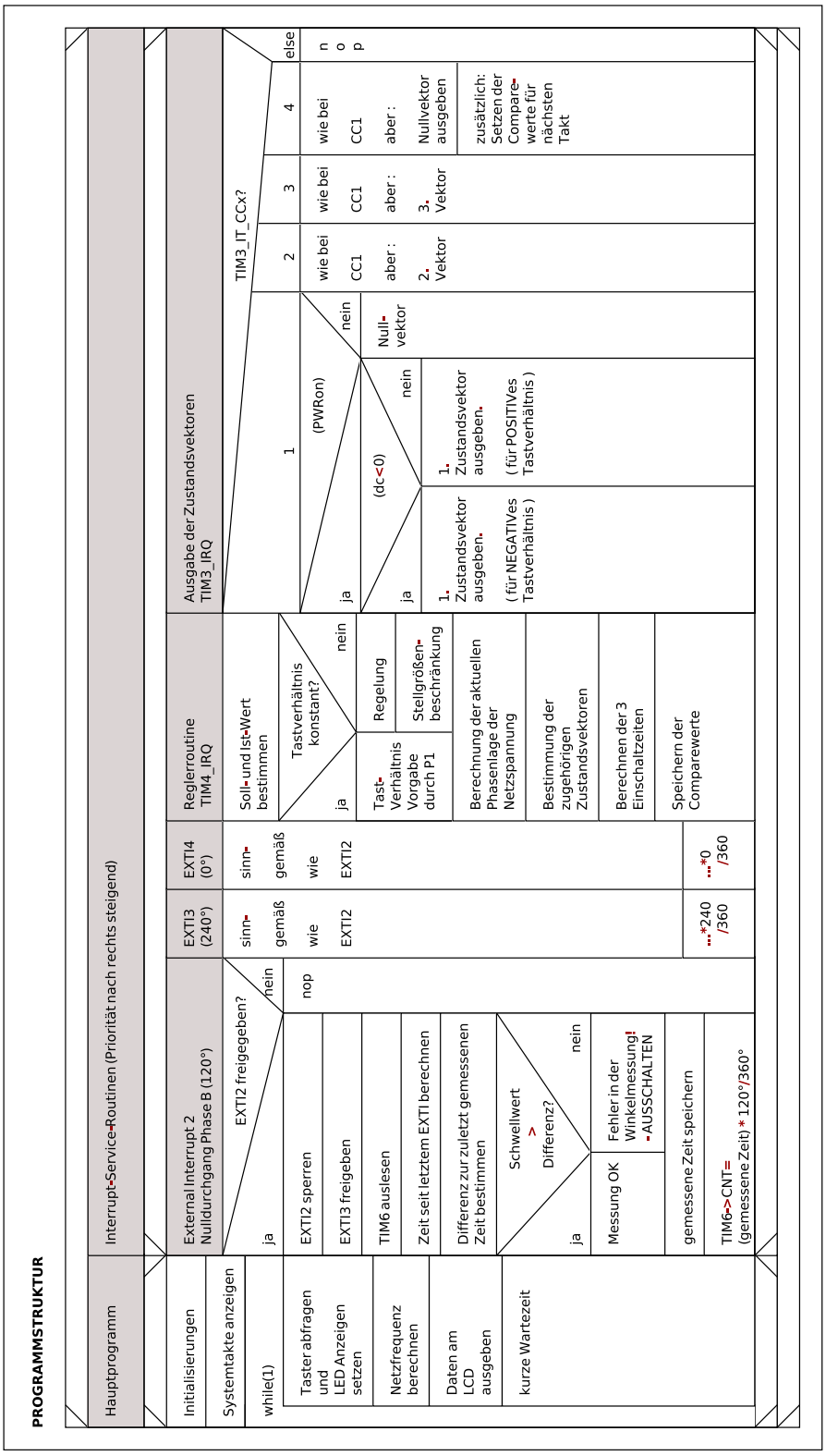


Abbildung 5.2: Struktur des Mikrocontrollerprogrammes

5.3 Übersicht der Funktionen

5.3.1 functions.h

Der Header 'functions.h' enthält die Funktionen zum Ansteuern der LEDs am Mikrocontrollerboard, zum Auslesen und Anzeigen der aktuellen Systemtakte und zum konvertieren von Integerzahlen in ACSII-Character, um sie am LCD anzeigen zu können.

void disc_leds(char, uint8_t);

Diese Funktion dient dem Ansteuern der 5 LEDs des STM32F4Discovery.
Der erste Parameter bestimmt welche LED angesprochen werden soll:

- 'R': Rot
- 'G': Grün
- 'B': Blau
- 'O': Orange
- 'C': Overcurrent-LED am USB-OTG Anschluss

Mit dem 2. Parameter wird bestimmt welche Aktion ausgeführt werden soll:

- 0: ausschalten
- 1: einschalten
- 2: toggeln

void board_leds(uint8_t, uint8_t);

Entspricht der Funktion 'void disc_leds(char, uint8_t);' steuert jedoch die 4 LEDs am Mikrocontrollerboard. Der erste Parameter kann hier den Wert 1-4 annehmen.

void int2str(char*, int32_t);

Konvertiert Integer (2.Parameter) nach String (1.Parameter), wird für Ausgabe von Werten am LCD benötigt.

void int2strST(char*, int32_t, uint8_t);

Wie 'int2str', jedoch wird mindestens die im dritten Parameter angegebene Anzahl an Stellen ausgegeben und gegebenenfalls mit vorangestellten Nullen aufgefüllt.

void show_clocks(void);

Diese Funktion wird beim Systemstart aufgerufen und zeigt die Taktfrequenzen von SYSCLK, HCLK, AHB1 und AHB2 (detaillierte Erklärung in [2] Seite 82 ff.) in MHz am LCD.

5.3.2 i2clcd.h

Diese Datei enthält alle Funktionen, welche zur Ansteuerung des LCDs gebraucht werden, sowie dessen I^2C-Adresse.

void init_I2C1(void);

Initialisiert die I^2C Schnittstelle 1 (I2C1) mit einem Takt von 100kHz an PB8 (SCL) undPB9 (SDA).

void I2C_start(I2C_TypeDef*, uint8_t, uint8_t);

Startet einen Datentransfer an I2C1.

1. Parameter: betreffende I^{2}C-Schnittstelle, hier: I2C1
2. Parameter: Slave-Adresse
3. Parameter: Datentransferrichtung aus Sicht des Masters (I2C_Direction_Transmitter oder I2C_Direction_Receiver)

void I2C_stop(I2C_TypeDef*);

Stoppt einen Datentransfer an betreffender Schnittstelle.

void I2C_write(I2C_TypeDef*, uint8_t);

Sendet ein Datenbyte an betreffende Schnittstelle.

void i2clcd_command(uint8_t);

Diese Funktion startet eine I^{2}C-Übertragung an das LCD, sendet ein Kommando und stoppt die Verbindung danach. Es werden hierfür die Funktionen

- void I2C_start(I2C_TypeDef*, uint8_t, uint8_t);
- void I2C_stop(I2C_TypeDef*);
- void I2C_write(I2C_TypeDef*, uint8_t);

verwendet.

void i2clcd_databyte(uint8_t);

Wie 'void i2clcd_command(uint8_t);' allerdings wird hier ein Datenbyte, also ein am LCD sichtbares Zeichen, gesendet.

void i2clcd_setcur(uint8_t, uint8_t);

setzt den Cursor des LCDs an eine bestimmte Stelle.

1. Parameter: Zeile
2. Parameter: Spalte

void i2clcd_string(const char*);

Sendet einen String an das LCD. Übergabeparameter ist hier ein Zeiger auf das Char-Array.

void i2clcd_init(void);

Enhält alle nötigen Kommandos um das LCD zu initialisieren. Diese Funktion wird nur einmal nach dem Einschalten aufgerufen.

5.3.3 inits.h

Hier sind alle Funktionen zum Initialisieren der verschiedenen Module und Ports des Mikrocontrollers deklariert.

void outputs_init(void);

Initialisiert die Pins welche als Ausgänge zu den Treibern für die 6 Transistoren des Leistungsteils (PD1-4 und PD6-7) verwendet werden.

void taster_init(void);

Initialisiert PA0 damit der 'Userbutton' der STM32F4Discovery verwendet werden kann.

void leds_init(void);

Initialisiert die zugehörigen Pins (PD5 und PD12-15) sodass die LEDs des STM32F4Discovery verwendet werden können.

void EXTI_init_angdet(void);

PE2-4 werden als External-Interrupt-Pin konfiguriert.
Bei jeder fallenden Signalflanke an einem dieser Pins wird der jeweils zugehörige Interrupt (EXTIx) ausgelöst. An diesen Eingängen wird die Schaltung zur Nulldurchgangserkennung der Netzspannung angeschlossen, womit es möglich ist sowohl die aktuelle Phasenlage als auch die Frequenz der Netzspannung zu erfassen.

void ADC_DMA_init_curr(uint32_t);

Hier wird ADC1 und DMA2 des STM32F407 so konfiguriert, dass periodisch die Analogwerte von PA1 und PA4 (Soll- und Istwert des Zwischenkreisstromes) eingelesen und per DMA_Stream0 in den Speicher geschrieben werden. Der Übergabeparameter ist die Anfangsadresse des Speicherbereiches der per DMA beschrieben werden soll. Die Wandlung wird zyklisch mit einer Abtastrate von ca. 260kS/s durchgeführt.

void ADC2_init(void);

ADC2 liest kontinuierlich den Analogwert von PA3 ein.

void TIM_init_ang(void);

Initialisiert TIM6 für die Phasenwinkel- und Netzfrequenz- Messung als Up-Counter mit einem Vorteiler von 46. Daraus ergibt sich eine Zählerfrequenz von ca. 913kHz und eine Durchlaufzeit von knapp 72ms.

void TIM_init_clk(void);

TIM3 und der zugehörige Output-Compare-Interrupt sind für die Ausgabe der jeweiligen Schaltzustände zuständig. Es wird mit einer Durchlaufzeit von 100μs (entspricht 10kHz Taktfrequenz des Umrichters) bis 200 gezählt, die Auflösung der PWM beträgt also 0,5%. Die Output-Compare-Werte werden von der Reglerroutine errechnet und stellen jeweils den Umschaltzeitpunkt in den nächsten Zustand dar.

void TIM_init_sinPWM(void);

Hier wird TIM4 auf eine Durchlaufzeit von ebenfalls 100μs eingestellt. In der Interruptroutine des TIM3 wird TIM4 mit der PWM synchronisiert. Der Wert im Compare-Capture-Register 1 dieses Timers (0-100) bestimmt zu welchem Zeitpunkt innerhalb eines Taktes (hier bei 80%) die Reglerroutine aufgerufen wird.

void FPU_on(void);

Diese Funktion aktiviert die Floating-Point-Unit des STM32F407.

void STM_Board_init(void);

Mit dieser Funktion werden die jeweiligen Pins für die am Mikrocontrollerboard platzierten LEDs 1-4 und Taster 1-4 konfiguriert.

void SysTick_init(void);

Der Systick-Timer des Cortex-M4 Kernes wird so eingestellt, dass jede Millisekunde ein Systick-Interrupt ausgelöst wird, dies ist hilfreich, wenn Wartezeiten realisiert werden sollen.

void INIT(uint32_t);

Die Funktion ruft alle notwendigen Initialisierungen in der korrekten Reihenfolge auf, sodass im Hauptprogramm nur ein einziger Aufruf erfolgen muss. Der Parameter wird an die Funktion'void ADC_DMA_init_curr(uint32_t);' weitergegeben (Speicheradresse für DMA-Zugriff)

5.3.4 lookuptable.h

'lookuptable.h' beinhaltet Lookuptables für die 2 verschiedenen Modulationsarten, die Zuordnung der Schaltzustände zu den Abschnitten einer Netzperiode (für positive und negative Ausgangsspannungen) und das Mapping der Portpins zu den jeweiligen Schaltzuständen.

5.3.5 main.h

In 'main.h' befinden sich die Deklarationen der Funktionen für Wartezeiten und Offsetabgleich sowie Defines, um den Zugriff auf die Taster zu erleichtern. Weiters werden verschiedene C-Libraries und weitere Header-Dateien für Peripheriekomponenten des Mikrocontrollers aus den 'StandardPeriperalLibraries' von STMicroelectronics eingebunden.

void wait_ms (uint16_t);

Eine Warteschleife, Übergabeparameter ist die Wartezeit in Millisekunden.

void I_offset_abgleich (void);

Der Stromsensor besitzt einen Offset von ca. 2,5V, diese Funktion liest den Analogwert bei I_{out}=0 ein und berechnet den in der Regelungsroutine benötigten Offset.

5.4 Hauptprogramm und Initialisierungen

Hauptprogramm

Die Aufgaben des Hauptprogrammes 'int main(void)' beschränken sich, wie in Abbildung 5.2 zu sehen, im Wesentlichen auf das Auslesen der Tasten zur Bedienung des Umrichters und die Ausgabe von Informationen am LCD. Da diese Programmteile weitestgehend selbsterklärend und nicht wesentlich für die Funktion des Umrichters sind, wird für weitere Informationen auf den ausführlich kommentierten Quellcode im Anhang (main.c) verwiesen.

Initialisierungen

Wie das Hauptprogramm bedürfen die Initialisierungen der verschiedenen Module des Mikrocontrollers kaum weiterer Erklärung als ohnehin in den Kommentaren des Quellcodes enthalten ist. Auch hier wird daher auf den Quellcode im Anhang verwiesen.

5.5 Interruptroutinen (in main.c)

5.5.1 Reglerroutine

Aufgabe der Reglerroutine, welche im Interrupthandler von Timer 4 untergebracht wurde, ist es, die Soll- und Ist-Werte des Zwischenkreisstromes einzulesen, die Einschaltdauer der 3 aktiven Zustände zu berechnen und die dafür notwendigen Comparewerte zu speichern.
Wie in den anderen Interrupt-Service-Routinen wird auch hier ein Pin des Mikrocontrollers zur Messung der Ausführungszeit verwendet. Die Variable sinA, sinB und sinC dienen als Zwischenspeicher für die aus der Lookuptable gelesenen Werte. Durch die erste Abfrage wird ermittelt, ob der Sollwert von P1 (Normalbetrieb) oder P3 (alternativ, z. B. für Sprungfunktionen) eingelesen werden soll.

```
void TIM4_IRQHandler(void){
    TIM_ClearITPendingBit(TIM4, TIM_IT_CC1);

    GPIOD->BSRRL = GPIO_Pin_14;

    float sinA, sinB, sinC;
    int32_t tmp;

     // Sollwert einlesen
     if (I_soll_2) {
         I_soll_reg = (float)ADC_GetConversionValue(ADC2) * (float)faktor_Sollwert; }
     else {
         I_soll_reg = (float)ADC_curr[0]                  * (float)faktor_Sollwert; }
```

Danach wird der aktuelle Istwert berechnet. Vom Wert 'ADC_curr[1]' wird dazu, der am Programmstart von der Funktion 'void I_offset_abgleich (void)' berechnete Offset abgezogen und das Ergebnis mit 'faktor_Istwert' skaliert. Da sich bei Werten nahe Null durch Ungenauigkeiten des ADC negative Werte ergeben könnten (welche in der Realität aufgrund der Schaltungstopologie nicht auftreten können), wird dies vorher geprüft und in einem solchen Falle das Ergebnis direkt mit Null ausgegeben. Folgend wird eine exponentielle Mittelwertbildung der Messwerte durchgeführt.

```
     // Istwert berechnen
     tmp = (int32_t)ADC_curr[1];
     if (tmp > I_offset) {
         tmp = (int32_t)( (float)(tmp-I_offset) * (float)faktor_Istwert );  }
     else { tmp = 0; }
     I_ist_reg = ( (float)( tmp + ((mittelwert_mult)*I_ist_reg) )
                 / (float)(mittelwert_div) );
```

Die Variable 'dc_fixed' wird verwendet, um zu Testzwecken die Regelung des Zwischenkreisstromes umgehen und ein konstantes Tastverhältnis ausgeben zu können. Ist diese ungleich Null, so wird die Stellung von P1 als Tastverhältnis von -0,99 bis 0,99 interpretiert. Andernfalls wird im folgenden Abschnitt die Regelung ausgeführt. Die Division durch 1000 rechnet die Soll- und Istströme von mA in A um. 'windup_max' wird so angepasst, dass der Integralanteil des Reglers nicht wesentlich größer als 1 werden kann, dies minimiert Windup-Effekte, behält dabei aber die Fähigkeit des Reglers, eine statische Abweichung von Null zu erreichen. Sollte der Umrichter deaktiviert sein, wird danach sowohl Tastverhältnis als auch der Integralanteil auf Null gesetzt. Nachfolgend wird das Tastverhältnisses auf kleiner 1 beschränkt (dies ist notwendig da die Interruptroutinen zur Ausgabe der Schaltzustände in der korrekten Reihenfolge abgearbeitet werden können).

```
// dc-konstant
if (dc_fixed) { dc = (float)(ADC_curr[0]-2048) / 2048; }
// Regelalgorithmus
else
{
    err =( I_soll_reg- I_ist_reg ) / 1000;
    errsum += err;
    if ( errsum >   windup_max  ) { errsum =  windup_max; }
    if ( errsum < (-windup_max) ) { errsum = -windup_max; }
    //dc=(float)( ( KP * err ) + ( KI * TD * errsum ) + ( KD * (err-erralt) ) );
    dc=( ( KP * err ) + ( KI * TD * errsum )  );
    erralt=err;
    if ( !PWRON ) { dc=0; errsum=0; }
}
if ( dc >  0.98 ) { dc= 0.98; }
if ( dc < -0.98 ) { dc=-0.98; }
```

Die aktuelle Phasenlage der Netzspannung wird aus dem momentanen Wert des Timer6 und dem 3-fachen Abstand der vorangegangenen beiden Nulldurchgänge (entspricht der Dauer einer Periode) berechnet und mit der Größe der Lookuptable multipliziert. Um die Multiplikation von 'TIM6_CNT_diff' mit '3' nicht extra berechnen zu müssen, wird mit dem um Faktor 3 kleineren (konstanten) Wert 'sizeLUTdiv3' statt 'sizeLUT' multipliziert. Da die Lookuptable den Bereich von 0 bis 360° abdeckt, kann mittels 'sinus[winkel]' der Absolutbetrag der Sinusfunktion beim momentanen Phasenwinkel der Netzspannung ausgelesen werden.
Mit der Variablen 'winkelversch' kann der zur Bestimmung der Einschaltzeiten für die aktiven Zustände verwendete Winkel verschoben werden. Daraus resultiert eine Aufnahme bzw. Erzeugung von Blindleistung.

```
winkel = (float)( ( (float)TIM_GetCounter(TIM6)
                  * (float)sizeLUTdiv3) / (float)TIM6_CNT_diff );

// Funktion zum Verschieben d. Netzstromes
if (winkelversch)
    {
    winkel += 70;      // sizeLUT=300 => 75=90deg; 70=84deg;
    }
```

Sollte sich durch die Addition einer Verschiebung aus dem letzten Schritt ein Wert von 'winkel' ergeben, welcher größer als die Länge der Lookuptable (eine Periode) ist, so muss eine Modulo-Operation ausgeführt werden. Da 'winkel' vom Typ 'float' ist, wird diese Operation hier nachgebildet.

```
if(winkel>sizeLUT)    {winkel-=sizeLUT;}
```

Folgend wird bestimmt, welchem Abschnitt der Netzperiode der aktuelle Winkel angehört. Die verschachtelte Anordnung wurde in Hinblick auf maximale Ausführungsgeschwindigkeit gewählt. Auch hier werden zu Kontrollzwecken während bestimmter Abschnitte LEDs geschaltet. Alternativ wäre es hier auch möglich, den Wert von 'winkel' durch ein Sechstel der Lookuptable-Länge zu dividieren. Der um eins erhöhte ganzzahlige Anteil des Ergebnisses ergibt dann den gesuchten Abschnitt. Diese Lösung benötigt anstatt der bis zu 3 Vergleiche lediglich eine Division.

```
if (winkel >  (3*sizeLUT/6) ){
    if (winkel >  (5*sizeLUT/6) )            {    abschnitt=6;    }
    else    {
            if (winkel <  (4*sizeLUT/6) )    {    abschnitt=4;    }
            else                             {    abschnitt=5;    }
            }
}
else{
    if (winkel >  (2*sizeLUT/6) )            {    abschnitt=3;    }
    else    {
```

```
                if (winkel <  (1*sizeLUT/6) )    {    abschnitt=1;    }
                else                             {    abschnitt=2;    }
                }
     }
    if((abschnitt==1)||(abschnitt==4)||(abschnitt==5))    {disc_leds('B',1);}
    else                                                  {disc_leds('B',0);}
```

Nun werden die Absolutbeträge der Sinusfunktionen für alle 3 Phasen aus der Lookuptable gelesen, zwischengespeichert, mit dem Tastverhältnis multipliziert, und so skaliert (comp_MULT_sin bzw. comp_MULT_max), dass ihre Summe bei einem Tastverhältnis von 1 den Wert 200 (= Maximum des Timer3) hat. Schließlich werden diese 3 Werte noch aneinandergereiht, um die Comparewerte für den nächsten Takt zu ergeben. Durch die Variable sin_Nmax kann auf die Verwendung der alternativen Lookuptable, welche es ermöglicht den Umrichter weiter auszusteuern, umgeschaltet werden. Eine detailierte Beschreibung dieser Möglichkeit ist in Kapitel 3.4 zu finden.
Zu diesem Programmabschnitt sei noch angemerkt, dass die Operation 'winkel % sizeLUT-1' im Zusammenspiel mit der hier verwendeten Lookuptable nicht korrekt ist. Da die Lookuptable mit dem Wert $0.021 = sin(2\pi * 299/300)$ endet, also den Wert für einen Winkel von 2π nicht mehr enthält, müsste korrekt 'winkel % sizeLUT' gerechnet werden. Sinngemäß gilt dies natürlich auch für die Lookuptable der optimierten Modulationsart.

```
// Comparewerte berechnen
if (sin_Nmax) {
 sinA=sinus[ (int32_t)  winkel %   (sizeLUT-1)  ];// %-Operanden muessen INT sein!
 sinB=sinus[ (int32_t)( winkel + (2*sizeLUTdiv3) ) % (sizeLUT-1) ]; // +240deg, = -120deg
 sinC=sinus[ (int32_t)( winkel +    sizeLUTdiv3 )  % (sizeLUT-1) ]; // +120deg, = -240deg
 comparewerte[1]=                  (uint16_t)( (float)comp_MULT_sin * sinA * fabsf(dc) );
 comparewerte[2]=comparewerte[1] + (uint16_t)( (float)comp_MULT_sin * sinB * fabsf(dc) );
 comparewerte[3]=comparewerte[2] + (uint16_t)( (float)comp_MULT_sin * sinC * fabsf(dc) );
}
else{
 sinA=maxtable[ (int32_t)  winkel                     % (sizeLUT-1) ];
 sinB=maxtable[ (int32_t)( winkel + (2*sizeLUTdiv3) ) % (sizeLUT-1) ];
 sinC=maxtable[ (int32_t)( winkel +    sizeLUTdiv3  ) % (sizeLUT-1) ];
 comparewerte[1]=                  (uint16_t)( (float)comp_MULT_max * sinA * fabsf(dc) );
 comparewerte[2]=comparewerte[1] + (uint16_t)( (float)comp_MULT_max * sinB * fabsf(dc) );
 comparewerte[3]=comparewerte[2] + (uint16_t)( (float)comp_MULT_max * sinC * fabsf(dc) );
}
GPIOD->BSRRH = GPIO_Pin_14;
```

5.5.2 Ausgaberoutine

Der Interrupt-Handler des Timer3 dient der Ausgabe der Schaltzustände.
Hierfür zählt TIM3 mit einem Takt von 2MHz von 0 bis 200, was eine Durchlaufzeit von 100μs ergibt. Die 4 Compare-Register stellen die Umschaltzeitpunkte auf den jeweils nächsten Schaltzustand dar.
Als erstes wird ein Pin des Mikrocontrollers 'high' gesetzt, dies ermöglicht eine Messung der zur Ausführung der Interruptroutine benötigen Zeit.

```
 void TIM3_IRQHandler(void){
    GPIOD->BSRRL = GPIO_Pin_12;
```

Zuerst muss geprüft werden welcher der 4 compare-Matches den Interrupt ausgelöst hat. Dazu wird in den nächsten Zeilen geprüft ob das 'TIM_IT_CC1'-Bit gesetzt wurde. Wenn dies der Fall ist, wurde der Wert in Compare-Register 1 vom Timer überschritten und es muss der 2. aktive

Zustand für diese Periode ausgegeben werden.
Danach wird das 'TIM_IT_CC1'-Bit des Timer3 Statusregisters zurückgesetzt und abgefragt ob der Umrichter eingeschaltet ist. Falls nein, wird der Nullvektor ausgegeben. Ansonsten muss geprüft werden, ob das Tastverhältnis positiv oder negativ ist und der zugehörige Schaltzustand ausgegeben werden.
Da nur die 6 Pins, welche die Schaltzustände ausgeben, aber nicht die anderen 10 Pins des PORTD, verändert werden sollen, wird der Port nicht direkt beschrieben, sondern zuerst BSSRH, also das Reset-Register mit der Bitweisen &-Verknüpfung von zvekt[0] (maskiert die unbeteiligten Pins) und dem invertierten nächsten Schaltzustand beschrieben. Dadurch werden alle Pins, welche in diesem Schaltzustand den Wert 0 haben auf 0 gesetzt. In einem zweiten Schritt werden die notwendigen Pins per BSSRL ('Set-Register') eingeschaltet. Ein Maskieren ist hier nicht erforderlich da in 'zvekt' alle unbeteiligten Pinnummern ohnehin den Wert 0 haben und somit nicht gesetzt werden.
Diese Vorgehensweise stellt prinzipiell ein 'break-before-make'-Verhalten dar welches aber durch die sehr geringe Zeit zwischen den beiden Befehlen keine Auswirkungen zeigt zumal die Verriegelungszeiten ohnehin mittels einer gesonderten Baugruppe sichergestellt werden.

```
if( (TIM3->SR) & (TIM_IT_CC1) ) {                          // 2.Zustand setzen
   TIM_ClearITPendingBit(TIM3, TIM_IT_CC1);
   if(PWRON){
     if   (dc>=0)  { GPIOD->BSRRH = (zvekt[0] & ~(zvekt[zustaende_pos[2][abschnitt]]));
                     GPIOD->BSRRL =  zvekt[zustaende_pos[2][abschnitt]];}
     else          { GPIOD->BSRRH = (zvekt[0] & ~(zvekt[zustaende_neg[2][abschnitt]]));
                     GPIOD->BSRRL =  zvekt[zustaende_neg[2][abschnitt]];}}
   else            { GPIOD->BSRRH =  zvekt[0];}
}
```

Wird in der Abfrage des Timer3 Statusregisters festgestellt, dass Compare-Match 1 nicht der Auslöser war, muss im folgenden Abschnitt geprüft werden, ob Compare-Match 2 den Interrupt ausgelöst hat und die ansonsten gleiche Prozedur wie zuvor für Compare-Match 2 (Ausgeben des 3. aktiven Zustandes) ausgeführt werden. Die weiteren Fälle verhalten sich dazu analog.

```
else{
   if((TIM3->SR) & (TIM_IT_CC2)){                          // 3.Zustand setzen
      TIM_ClearITPendingBit(TIM3, TIM_IT_CC2);
      if(PWRON){
        if  (dc>=0)  { GPIOD->BSRRH = (zvekt[0] & ~(zvekt[zustaende_pos[3][abschnitt]]));
                       GPIOD->BSRRL =  zvekt[zustaende_pos[3][abschnitt]];}
        else         { GPIOD->BSRRH = (zvekt[0] & ~(zvekt[zustaende_neg[3][abschnitt]]));
                       GPIOD->BSRRL =  zvekt[zustaende_neg[3][abschnitt]];}}
      else           { GPIOD->BSRRH =  zvekt[0];}
   }
   else{
      if((TIM3->SR) & (TIM_IT_CC3)){
          TIM_ClearITPendingBit(TIM3, TIM_IT_CC3);
      if(PWRON){
        if  (dc>=0)  { GPIOD->BSRRH = (zvekt[0] & ~(zvekt[zustaende_pos[4][abschnitt]]));
                       GPIOD->BSRRL =  zvekt[zustaende_pos[4][abschnitt]];}
        else         { GPIOD->BSRRH = (zvekt[0] & ~(zvekt[zustaende_neg[4][abschnitt]]));
                       GPIOD->BSRRL =  zvekt[zustaende_neg[4][abschnitt]];}}
      else           { GPIOD->BSRRH =  zvekt[0];}
        }
```

Im Fall von Compare-Match 4 (beim Wert von 200) kommt hinzu, dass die neuen Comparewerte für den nächsten Takt geschrieben, und Timer 3 selbst, sowie Timer 4 (dient zum Aufruf der Reglerroutine) auf 0 gesetzt werden müssen.
Als letztes wird PD12 (zur Zeitmessung) wieder rückgesetzt.

```
      else{
              TIM_ClearITPendingBit(TIM3, TIM_IT_CC4);
     if(PWRON){
        if  (dc>=0) { GPIOD->BSRRH = (zvekt[0] & ~(zvekt[zustaende_pos[1][abschnitt]]));
                      GPIOD->BSRRL =  zvekt[zustaende_pos[1][abschnitt]];  }
        else        { GPIOD->BSRRH = (zvekt[0] & ~(zvekt[zustaende_neg[1][abschnitt]]));
                      GPIOD->BSRRL =  zvekt[zustaende_neg[1][abschnitt]];  }  }
     else           { GPIOD->BSRRH =  zvekt[0];}

        // neue Compare-Werte setzen
        TIM_SetCompare1(TIM3, comparewerte[1]);
        TIM_SetCompare2(TIM3, comparewerte[2]);
        TIM_SetCompare3(TIM3, comparewerte[3]);

        TIM_SetCounter(TIM3, 0);
        TIM_SetCounter(TIM4, 0);
        }
    }
}
GPIOD->BSRRH = GPIO_Pin_12;
}
```

5.5.3 External Interrupts

Bei jedem positiven Nulldurchgang der Netzspannung wird die zugehörige External- Interrupt-Routine aufgerufen. Diese 3 Interruptroutinen messen die Periodendauer der Netzspannung und erkennen, ob ein Fehler, beispielsweise eine fehlende Phase oder starke Unsymmetrie, vorliegt. Beispielhaft ist hier der Handler von External-Interrupt Line 2 (Nulldurchgang Phase B) gezeigt, die Handler zu Line 3 und 4 verhalten sich analog dazu, werden aber beim Nulldurchgang von Phase C bzw. A aufgerufen.
Weiters soll hier noch erwähnt werden, dass die Interrupt-Routinen der EXTI-Lines in unregelmäßigen Abständen offenbar mehrfach aufgerufen werden, obwohl das Rücksetzen des 'Interrupt-Pending-Bits' sofort nach Aufruf der Routine geschieht und nachweislich nur eine einzige Flanke am betreffenden Pin auftritt. Daher wurde das Programm um eine 'Verriegelung' erweitert, welche die Ausführung verhindert, wenn der nächste Nulldurchgang in einer anderen Phase erwartet wird.

Am Beginn der Funktion wird das zugehörige 'Interrupt-Pending-Bit' zurückgesetzt. Die beiden Variablen 'tmp' und 'abweichung' werden innerhalb der Routine für Berechnungen verwendet.

```
void EXTI2_IRQHandler(void){
    EXTI_ClearITPendingBit(EXTI_Line2);
    int32_t tmp;
    int32_t abweichung;
```

Es folgt die 'Verriegelung', dabei wird geprüft, ob der gerade aufgerufene Interrupt tatsächlich als nächster erwartet wird. Danach wird abgefragt ob der zu diesem External Interrupt gehörige Pin tatsächlich auf 'low' gehalten wird. Diese Abfrage verhindert ein ungewolltes Ausführen, auch im Falle von transienten Störungen an den betreffenden Pins. In der folgenden Zeile wird die Verriegelung für die aktuell laufende Routine gesetzt, damit diese nicht fälschlicherweise ein weiteres Mal ausgeführt wird, und danach die Verriegelung für die Routine des nächsten Nulldurchganges freigegeben. Das Toggeln der LED dient auch hier nur der Zeitmessung und Kontrollzwecken.

```
    if(!(lock_exti_2)){
        if(!(GPIOE->IDR & GPIO_Pin_2)){
          lock_exti_2=1;
          disc_leds('O',2);
          lock_exti_3=0;
```

Nun wird der aktuelle Wert von TIM6 ausgelesen, der vormals aktuelle Differenzwert (Differenz der Timerwerte zwischen 2 Nulldurchgängen) als 'TIM6_CNT_diff_alt' gespeichert, und die 'neue' Differenz in 'TIM6_CNT_diff' abgelegt. Weichen diese beiden Werte weiter als 'winkel_diff_max' voneinander ab (d. h. die zeitlichen Abstände zwischen den Nulldurchgängen unterscheiden sich um mehr als einen Schwellwert), kann davon ausgegangen werden, dass ein Fehler in der Spannungsversorgung vorliegt. Die Variable 'winkel_OK_CNT' wird in diesem Fall auf 0 gesetzt, andernfalls bis zu einem Grenzwert erhöht. Zuletzt wird der ausgelesene Timerwert gespeichert und der Timer auf den aktuell korrekten Wert gesetzt (hier: Nulldurchgang von B entspricht 120° = 1/3 einer Periode, eine gesamte Periode besteht aus: 3x'Abstand 2er Nulldurchgänge', d. h. der Timer muss auf: 'Abstand zweier Nulldurchgänge' gesetzt werden.)

```
        tmp=TIM_GetCounter(TIM6);
        TIM6_CNT_diff_alt=TIM6_CNT_diff;
        TIM6_CNT_diff=tmp-TIM6_CNT_last;
        abweichung=TIM6_CNT_diff-TIM6_CNT_diff_alt;
        if ( abs(abweichung) > winkel_diff_max ) {
            if ( winkel_OK_CNT > 0 ) { winkel_OK_CNT=0; board_leds(2,1);} }
        else {
            if ( winkel_OK_CNT < winkel_OK_lim*2 ) { winkel_OK_CNT++; } }
        TIM6_CNT_last=TIM6_CNT_diff;
        TIM_SetCounter(TIM6, TIM6_CNT_diff);}
        }
    }
```

6 Messungen am Umrichter

Aus Sicherheitsgründen wurden die folgenden Messungen mit einer (wenn nicht anders angegeben), durch einen Stelltrenntransformator erzeugten, verringerten Netzspannung von

$$U_{netz,phase,eff} = 115V$$

bzw.

$$U_{netz,verkettet,eff} = U_{netz,phase,eff} * \sqrt{3} = 200V$$

durchgeführt.

Weiters wurde darauf geachtet, die Leistungshalbleiter bei den Tests nicht zu zerstören, da diese gegen Ende des Projektes nicht mehr erhältlich waren und nur wenige Ersatzteile zur Verfügung standen. Im Falle eines Defektes hätte der Leistungsteil somit auf IGBTs umgerüstet und die Spannungsversorgung der Treiber angepasst werden müssen.

6.1 Sollwertsprung

In Abbildung 6.1 und 6.2 ist das Verhalten des Umrichters bei sprungförmigem Sollwert des Zwischenkreisstromes zu sehen. Abbildung 6.1 zeigt Sprünge von $I_{out} = 1A$ auf $4/6/8/10/12A$ bei konstantem Lastwiderstand von 10Ω.
Vor $t = 0s$ wird der Netzstrom maßgeblich von den Eingangsfilterkondensatoren bestimmt und ist daher weitestgehend kapazitiv. Bei $t = 0s$ legt der Umrichter maximale Ausgangsspannung an den Zwischenkreis, naturgemäß werden dadurch Stromspitzen in allen 3 Phasen erzeugt. Nach Erreichen des geforderten Zwischenkreisstromes nehmen die Ströme der Netzzuleitungen Sinusform an.

Obiges gilt auch für Abbildung 6.2, welche einen Sprung von 1A auf 6A Zwischenkreisstrom bei verschiedenen Lastwiderständen zeigt, wo das Überschwingen der Regelung mit zunehmendem Widerstand nahezu verschwindet.

Die höherfrequenten Anteile des Netzstromes rühren von, durch Schaltvorgänge angeregte, Resonanzen der Streuinduktivität des Trenntransformators mit den Eingangsfilterkondensatoren her. Es ist auch zu erkennen, dass diese Anteile bei steigendem Netzstrom nicht wachsen und mit steigendem Lastwiderstand zunehmend gedämpft werden.

Anzumerken ist hierbei, dass im regulären Betrieb die Vorgabe einer sprungförmigen Führungsgröße wenig zweckmäßig ist. Wird als Führungsgröße eine vergleichsweise langsam steigende Rampe verwendet, so bleiben die bei Sprüngen erzeugten Stromspitzen und die damit verbundenen Oberschwingungen im Netzstrom aus.

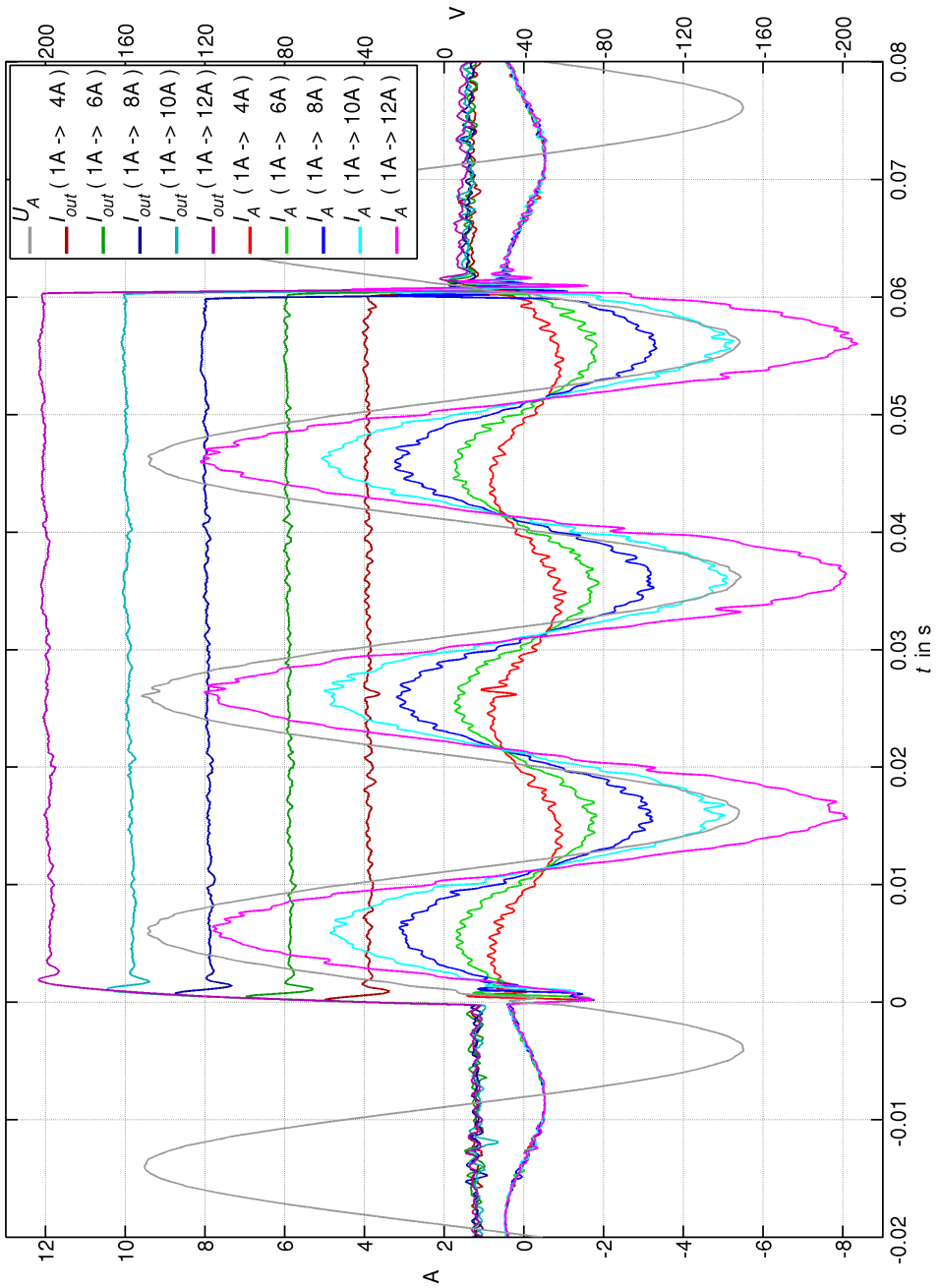


Abbildung 6.1: Reaktion des Umrichters auf verschiedene sprungförmige Sollwertvorgaben ($R_{last} = 10\Omega$)

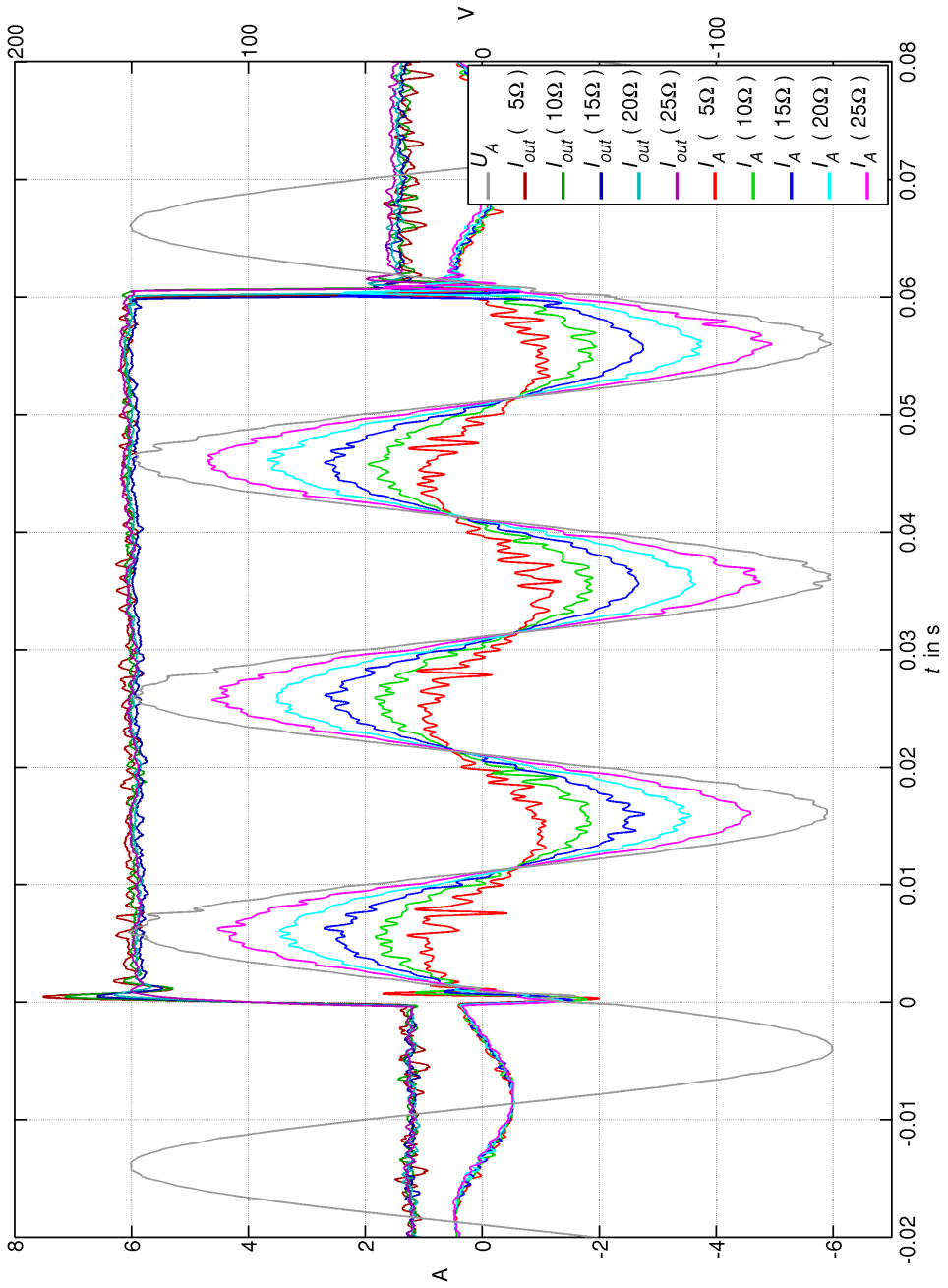


Abbildung 6.2: Reaktion des Umrichters auf sprungförmige Sollwertvorgabe bei verschiedenen Lastwiderständen

6.2 Lastspannungssprung und generatorischer Betrieb

Für diesen Versuch wurde ein 4-Quadranten-Leistungsverstärker in den Zwischenkreis des Umrichters eingeschleift, der eine sprungförmige Lastspannung simuliert.
Deutlich zu sehen ist dabei in Abbildung 6.3 wie der Netzstrom im generatorischen Betrieb sein Vorzeichen ändert und die Ausgangsspannung des Umrichters negativ wird.
Um eine bestmögliche Darstellung zu ermöglichen, wurde die Eingangsspannung für diesen Versuch weiter gesenkt.

Tabelle 6.1: Ausgangsspannung des Leistungsverstärkers (Lastspannungssprung)

Abschnitt	U_{last}	Betriebsart des Umrichters
$t < 0s$	0V	motorisch
$0s < t < 0.025s$	+30V	motorisch
$0.025s < t < 0.050s$	-40V	generatorisch
$t > 0.050s$	0V	motorisch

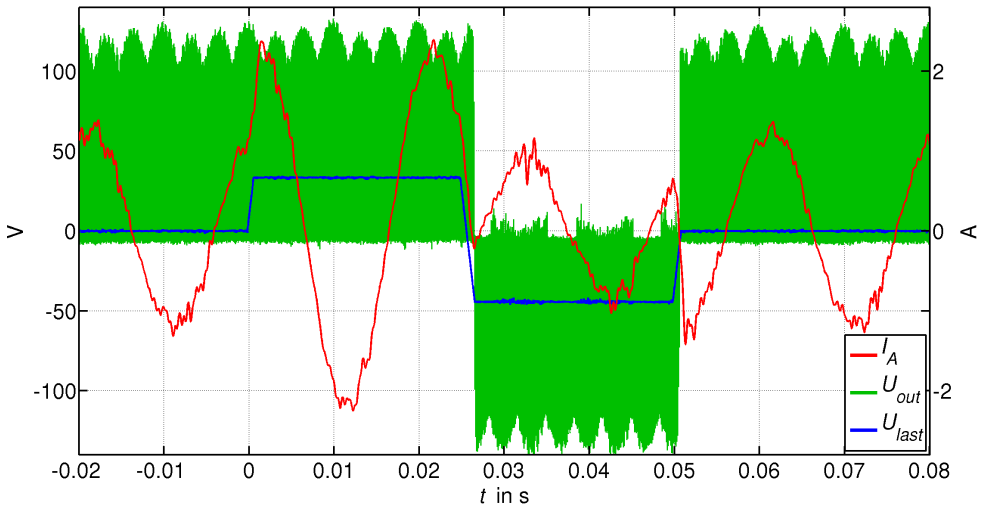


Abbildung 6.3: Motorischer und generatorischer Betrieb des Umrichters

6.3 Verhalten bei Netzoberschwingungen

Um das Verhalten des Umrichters bei oberschwingungsbehafteter Netzspannung zu untersuchen, wurde mit Hilfe des Leistungsverstärkers ein Drehstromsystem mit überlagerter 5. harmonischer Oberschwingung (250Hz) erzeugt und wie in Kapitel 6.1 ein Sollwertsprung vorgegeben. Der Oberschwingungsgehalt wurde in 5%- Schritten bis 20% erhöht und jeweils Strom/Spannung in der Phase A sowie im Zwischenkreis gemessen. Das Ergebnis zeigt Abbildung 6.4, wobei zur besseren Übersicht die Kurven entlang der Ordinate in 100V bzw. 2A Schritten, verschoben wurden.
Die Stromspitzen in den positiven Nulldurchgängen werden durch transiente Störungen in der Netzspannung, welche in den Spannungsverläufen deutlich zu sehen sind, hervorgerufen.
Auffällig ist hier auch die zunehmende Unsymmetrie des Netzstromes der Phase A. Verursacht wird dies durch ein Schwingen des Reglers in Folge der schwankenden Eingangsspannung. Um die Regelung für diesen Betriebsfall zu optimieren wäre eine Anpassung der Reglerparameter erforderlich.

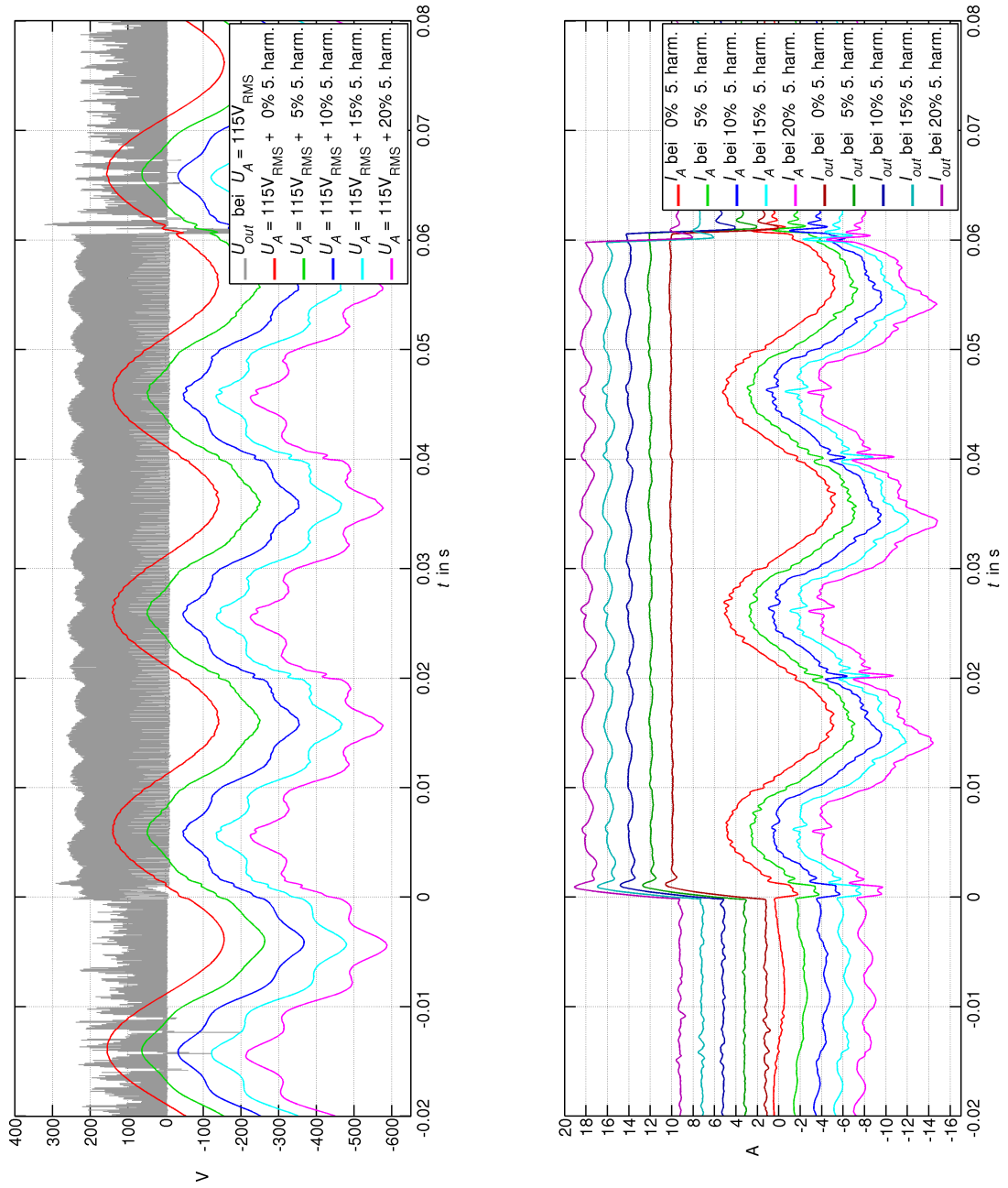


Abbildung 6.4: Verhalten des Umrichters bei oberschwingungsbehafteter Netzspannung (Verläufe entlang der Ordinate verschoben)

6.4 Blindleistungsgenerierung (Betrieb mit Phasenverschiebung)

Aufgrund der Eingangsfilterkondensatoren fließt im Leerlauf des Umrichters ein kapazitiver Strom von ca. $350mA_{eff}$. Der Phasenwinkel des vom Umrichter in den Zwischenkreis gespeisten Stromes kann durch Verschiebung der Pulsmuster in weitem Bereich variiert werden. Zu beachten ist dabei, dass mit zunehmender Verschiebung der Mittelwert der gleichgerichteten Spannung sinkt und schließlich bei 90° den Wert 0 erreicht.

Beispielhaft ist in Abbildung 6.5 eine Verschiebung um 84° gezeigt, was einen nahezu rein kapazitiven Netzstrom bedeutet. Zum Vergleich sind Netzstrom, Zwischenkreisspannung und Zwischenkreisstrom, für die gleiche Last, auch ohne Verschiebung des aufgenommenen Stromes eingezeichnet.

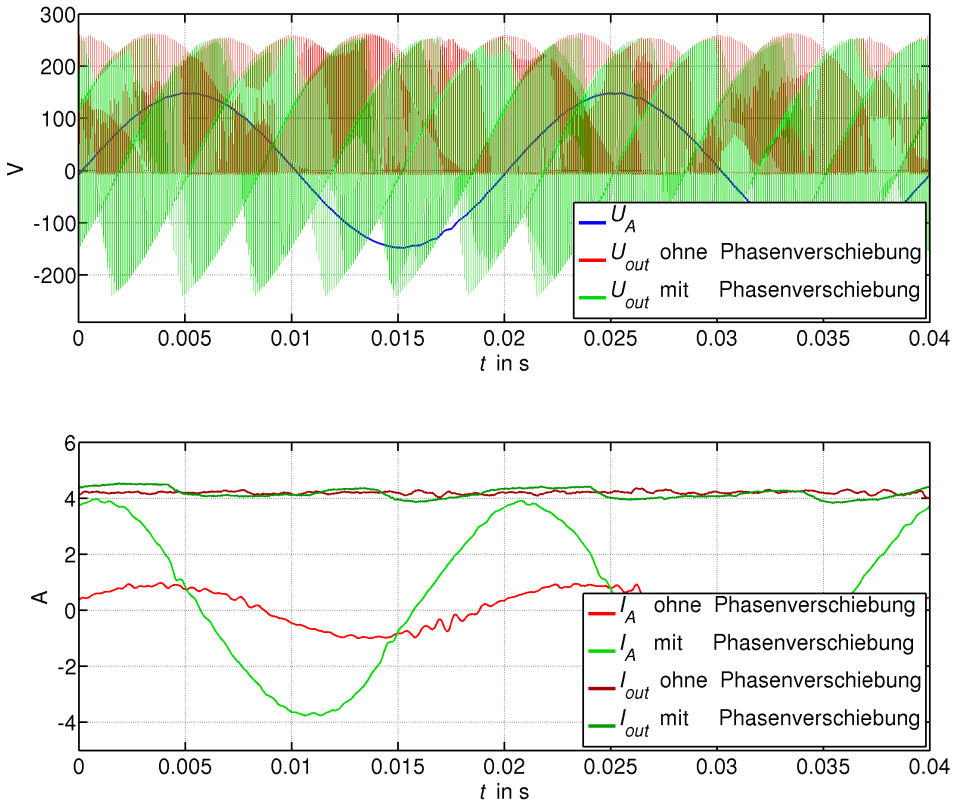


Abbildung 6.5: Betrieb des Umrichters mit Phasenverschiebung

In Abbildung 6.6 ist der Nulldurchgang U_A (entspricht $\varphi = 30°$, Abschnitt A) vergrößert dargestellt. Die Spannung U_{out} setzt sich aus 3 Teilen zusammen:

- $u_{AB} \approx 180$ bis $120V$
- $u_{BC} \approx -260V$
- $u_{CA} \approx 100$ bis $150V$

Durch die aktivierte Verschiebung werden die Schaltzustände aus Abschnitt F verwendet, wodurch U_{AB} und U_{BC} in gedrehter Polarität an den Zwischenkreis gelegt werden. Deshalb liefert U_{AB} einen negativen Beitrag zum Mittelwert der Ausgangsspannung.

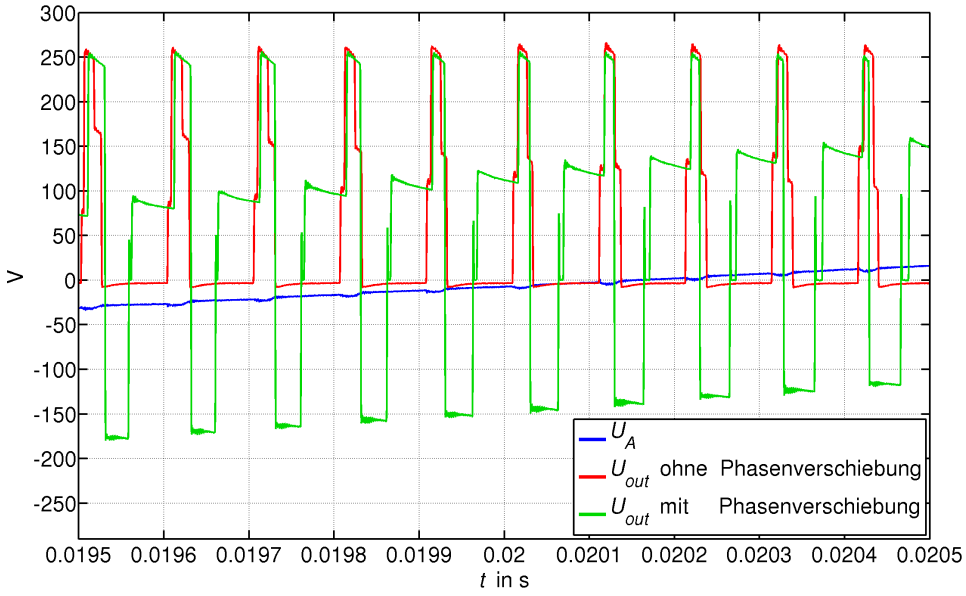


Abbildung 6.6: Betrieb des Umrichters mit Phasenverschiebung, Ausschnitt

6.5 Vergleich zwischen sinusförmiger- und optimierter Modulation

Wie in Kapitel 3.4 beschrieben, kann der Umrichter mit optimiertem Modulationsmuster betrieben werden, wobei in jeder Schaltperiode nurmehr 2 aktive Zustände und ein Nullvektor ausgegeben werden.

Abbildung 6.7 zeigt vergrößerte Ansichten beider Möglichkeiten im motorischen Betrieb an einem Lastwiderstand von 10Ω. Neben den 2 bzw. 3 Stufen im Verlauf der Zwischenkreisspannung, welche die verketteten Spannungen darstellen, ist die Spannungsänderung an den Eingangskondensatoren während einer Schaltperiode gut zu erkennen.

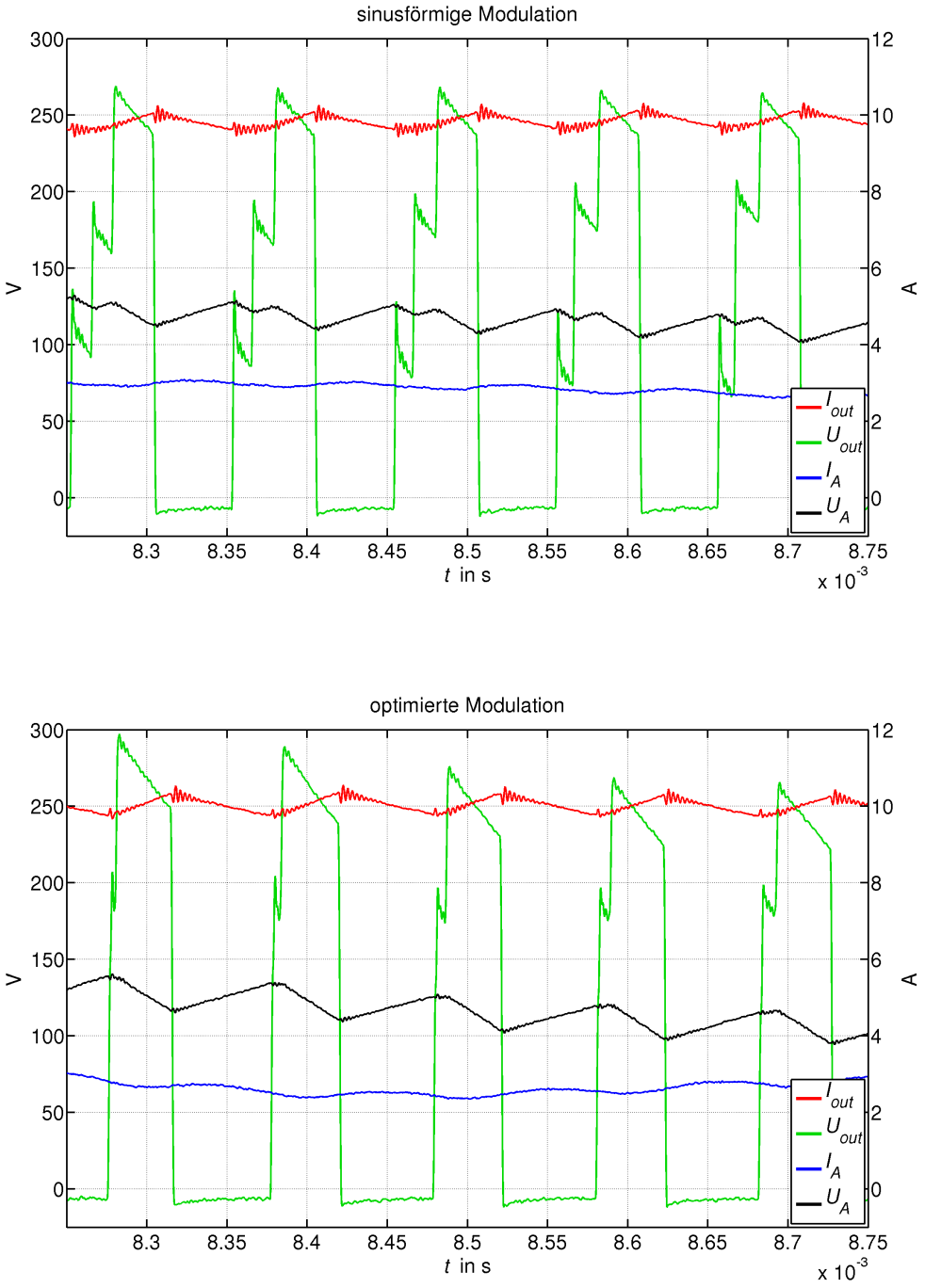


Abbildung 6.7: unterschiedliche Modulationsarten

6.6 Wirkungsgradmessungen

Zur Bestimmung des Wirkungsgrades wurden mit Hilfe eines Norma N5000 Power Analyzers die Netzströme und Netzspannungen, Zwischenkreisstrom und Zwischenkreisspannung, sowie aufgenommene und abgegebene Leistung gemessen. Daraus konnten die in Tabelle 6.2 und 6.3 gezeigten Wirkungsgrade errechnet werden.

Tabelle 6.2: Wirkungsrad bei sinusförmiger Modulation

R_{last} \ I_{out}	2A	4A	6A	8A	10A
5Ω	0.72	0.84	0.87	0.89	0.90
10Ω	0.85	0.91	0.93	0.94	0.95
15Ω	0.90	0.94	0.96	0.96	0.96
20Ω	0.92	0.96	0.97	0.97	x
25Ω	0.94	0.96	0.97	x	x

Tabelle 6.3: Wirkungsrad bei optimierter Modulation

R_{last} \ I_{out}	2A	4A	6A	8A	10A
5Ω	0.73	0.83	0.87	0.89	0.90
10Ω	0.85	0.92	0.93	0.94	0.95
15Ω	0.90	0.94	0.96	0.96	0.96
20Ω	0.92	0.96	0.97	0.97	x
25Ω	0.94	0.96	0.97	x	x

In dieser Berechnung nicht berücksichtigt ist die zur Versorgung des Mikrocontrollers und der Treiber benötigte Leistung, welche bei insgesamt ca. 6W liegt.

Wenig überraschend ist der geringere Wirkungsgrad bei kleiner Last sowie die deutliche Verbesserung bei höherer Ausgangsspannung. Insgesamt wurde ein durchaus recht hohes Niveau erreicht, das keinen merklichen Abfall zu hohen Strömen hin zeigt.

Aufgrund der verhältnismäßig geringen Schaltverluste sind zwischen den Messungen mit sinusförmigem und optimiertem Modulationsverfahren, hinsichtlich des Wirkungsgades nur marginale Unterschiede festzustellen.

6.7 Temperaturmessungen

Um einen Überblick über die Erwärmung der Bauteile des Umrichters zu erhalten und in Folge Aussagen zur Belastbarkeit des Gerätes machen zu können, wurde eine Wärmebildkamera (FLIR SC620) eingesetzt.
Die Umgebungstemperatur betrug etwa 22°C und der Emissionsgrad wurde auf 0,96 eingestellt. Da der gesamte Versuchsaufbau aus vielen verschiedenen Oberflächen und Materialien besteht wurde der Wert von 0,96 laut [3] als Kompromiss gewählt, welcher die relevanten Bereiche (schwarz eloxierter Kühlkörper, FR4 Leiterplatte, Halbleitergehäuse und schwarzes Isolierband auf ansonsten glatten Oberfläche) bestmöglich abdeckt.
Abbildung 6.8 zeigt den Umrichter im Leerlauf. Die beiden Hotspots im linken Bereich zeigen die Schaltregler IC's für die Spannungsversorgung der Treiber, welche weitgehend unabhängig vom Ausgangsstrom eine Temperatur von etwa 45°C erreichen. Im rechten Bereich sind die beiden Mikrocontroller und ein Spannungsregler des STM32F4Discovery sowie etwas unterhalb der 5V-Spannungsregler des Mikrocontrollerboards zu erkennen.

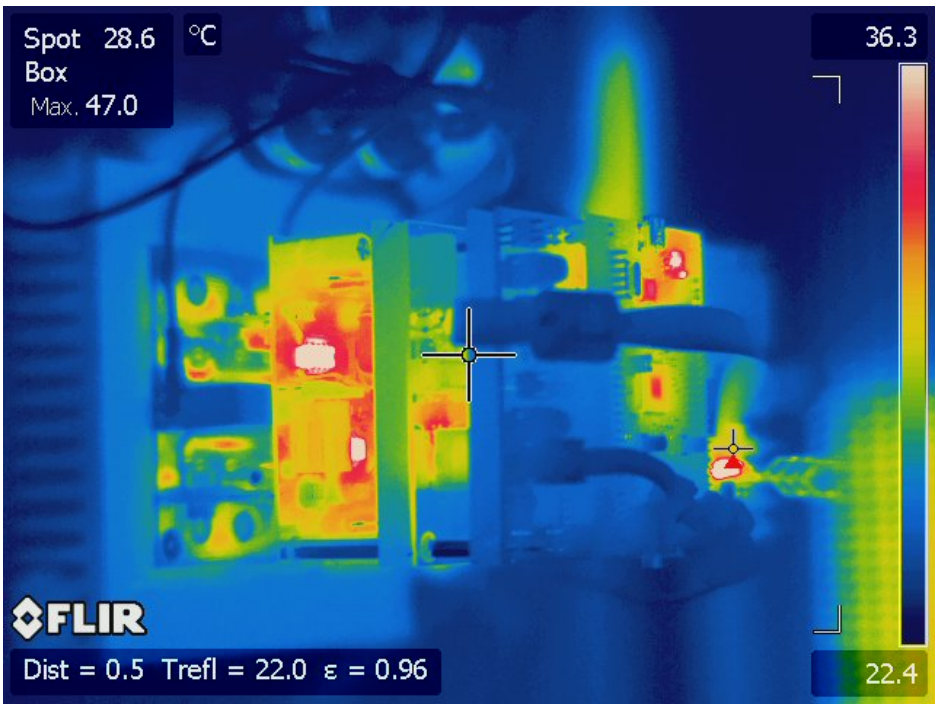


Abbildung 6.8: Erwärmung des Umrichters, Aufnahme von der Seite

Im Verlauf der Messungen wurde der Zwischenkreisstrom schrittweise erhöht und das Erreichen eines stationären thermischen Zustandes abgewartet bis schließlich bei I_{out}=12A eine Temperatur von 80 °C an den Gehäusen der Leistungshalbleiter erreicht wurde (Abbildung 6.9).
Die etwas höhere Temperatur der Bauteile der rechten Halbbrücke ist darin begründet, dass diese den Strom im Nullzustand im Mittel eine etwas längere Zeit führen. In späteren Revisionen der Mikrocontrollersoftware wurde dies bereits bereinigt.

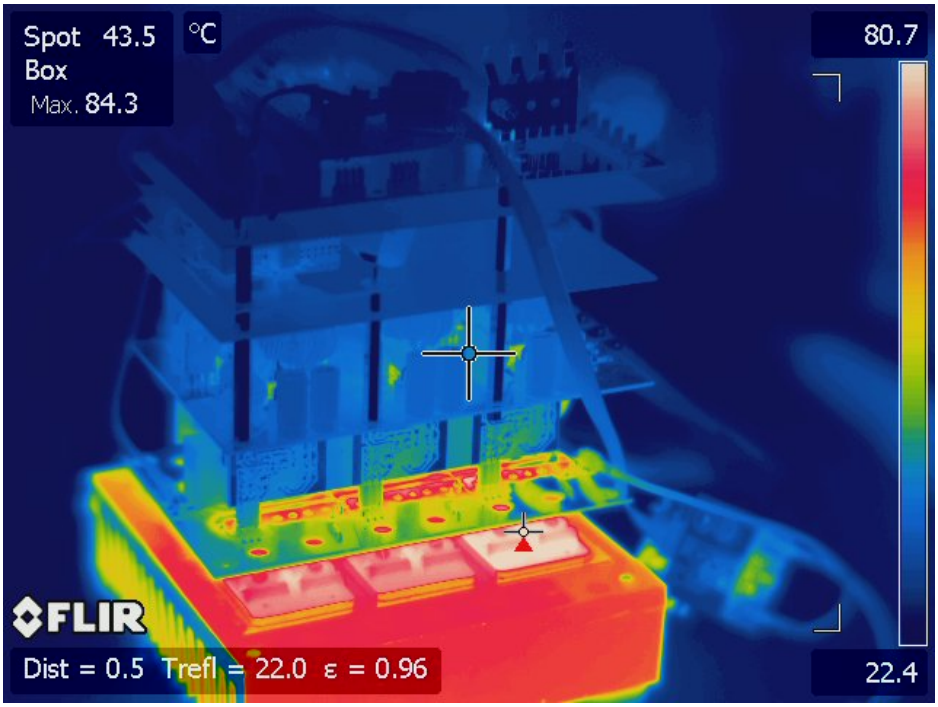


Abbildung 6.9: Erwärmung der Leistungshalbleiter **ohne** Zwangsbelüftung (I_{out}=12A)

Mit Hilfe eines Ventilators zur aktiven Belüftung des Kühlkörpers konnte der Zwischenkreisstrom, bei gleichzeitig wesentlich kühleren Bauteilen, bis zum Nennwert von 15A erhöht werden. Da hier die Temperaturen aller Leistungshalbleiter noch deutlich unter 70 °C liegen (Abbildung 6.10) ist zu erwarten, dass auch ohne weitere Kühlmaßnahmen eine nennenswerte Steigerung des maximalen Ausgangsstromes möglich wäre.

Abbildung 6.11 zeigt einen Ausschnitt der Leiterplatte des Leistungsteiles. Im Bereich des ‘Maximum-Markers‘ ist eine starke Erwärmung der Leiterplatte zu erkennen. Es empfiehlt sich daher, bei einer Weiterentwicklung die Leiterbahnen in diesem Bereich breiter zu gestalten oder dickere Kupferauflage (hier: 70μm) zu verwenden, bzw. die Leiterplatte des Prototypen durch Auflöten von Kupferdraht oder Entlötlitze zu verstärken.

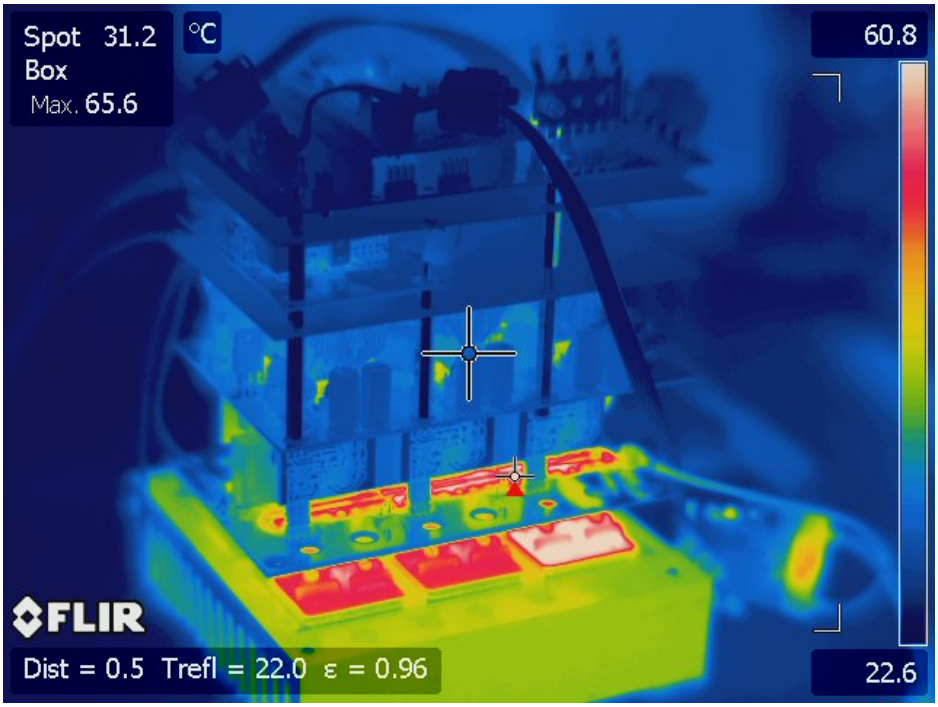


Abbildung 6.10: Erwärmung der Leistungshalbleiter **mit** Zwangsbelüftung (I_{out}=15A)

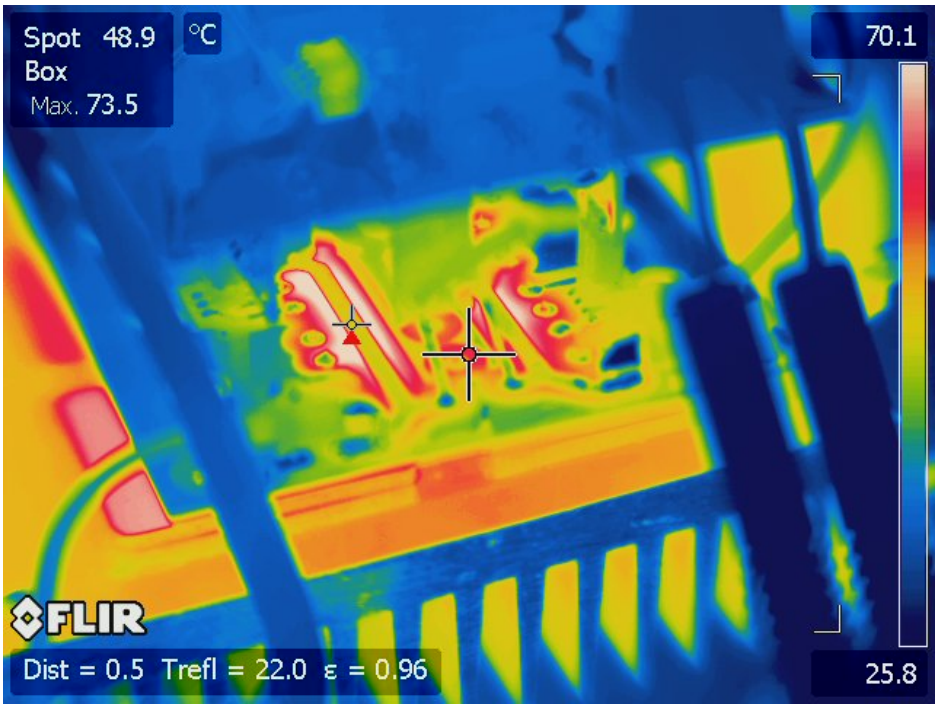


Abbildung 6.11: Erwärmung der Leiterbahnen im Leistungsteil (I_{out}=12A)

7 Schlussbemerkungen

7.1 erreichte Ziele

Die in dieser Arbeit erreichten Ziele sind:

- Einarbeitung in die Materie der Stromzwischenkreisumrichter und des Schaltverhaltens von Leistungshalbleitern.
- Detailierte Analyse und Vergleich der zu erwartenden Verluste beim Einsatz verschiedener Leistungshalbleiter.
- Entwicklung einer geeigneten Ansteuerung für Siliziumcarbid-JFETs im Leistungsteil.
- Entwurf einer geeigneten Regelung zum Betrieb des Umrichters.
- Simulation des Gesamtsystems aus Hardware und Regelung am PC.
- Aufbau eines funktionsfähigen Prototypen.
- Implementierung der Regelung auf einem autark arbeitenden Mikrocontroller.
- Messungen am Prototypen unter Laborbedingungen und deren Dokumentation.

7.2 mögliche Verbesserungen

Im Zuge der Arbeit wurden weitere Verbesserungsmöglichkeiten erschlossen, welche aus Zeitgünden nicht realisiert werden konnten und daher hier genannt werden sollen.

- Realisierung der Verriegelungszeit in der Software des Mikrocontrollers. Die ursprünglich aus Sicherheitsgründen beim Testen der Software vorgesehene Baugruppe 'Verriegelungslogik' kann somit entfallen.
- Implementierung einer Regelung für die Phasenverschiebung des Netzstromes, um den kapazitiven Netzstrom der Eingangsfilterkondensatoren kompensieren zu können. Weiters wäre es damit auch möglich, durch gezielt phasenverschobene Netzströme die Blindleistungsaufnahme anderer Verbraucher zu kompensieren.
- Veränderung der Ausgaberoutine, um die in Kapitel 3.4 beschriebene Optimierung der Reihenfolge der Schaltzustände zu nutzen.
- Erhöhung der Stromtragfähigkeit der Leiterbahnen im Leistungsteil.
- Ersatz des Stromsensors durch einen unipolaren Typ höherer Empfindlichkeit, um die Genauigkeit der Strommessung zu verbessern.
- Steigerung der Schaltfrequenz, um eine Verkleinerung der netzseitigen Filterkondensatoren zu ermöglichen.

8 Anhang

8.1 Schaltungen

Leistungsteil

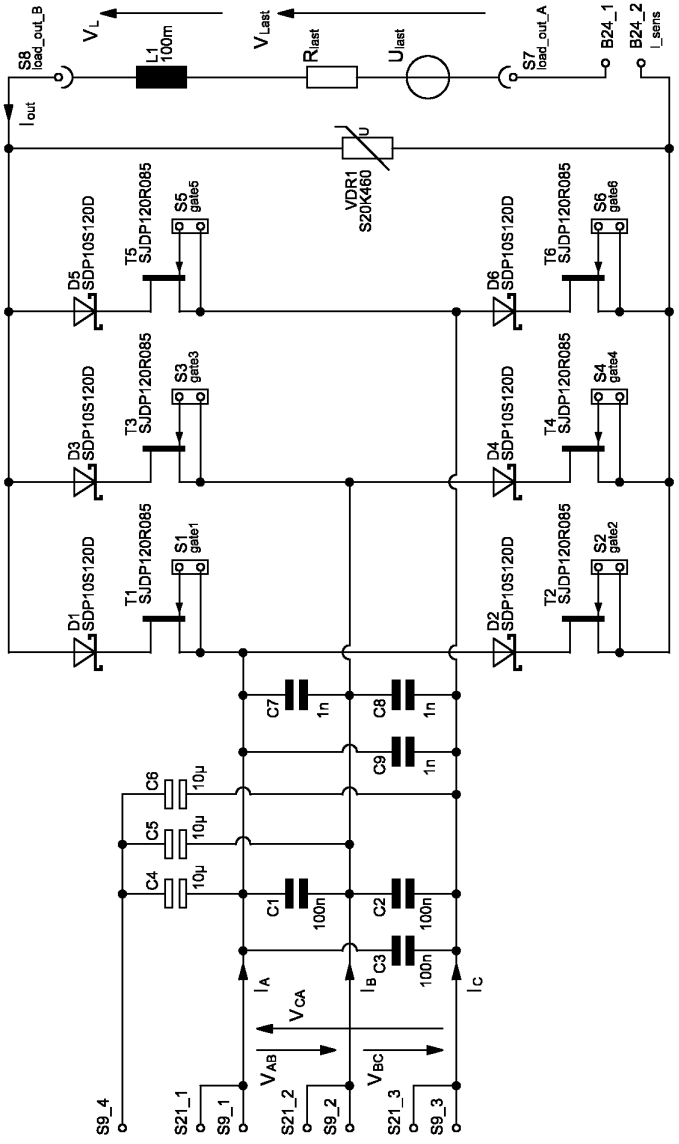


Abbildung 8.1: Schaltplan, Leistungsteil

Treiber

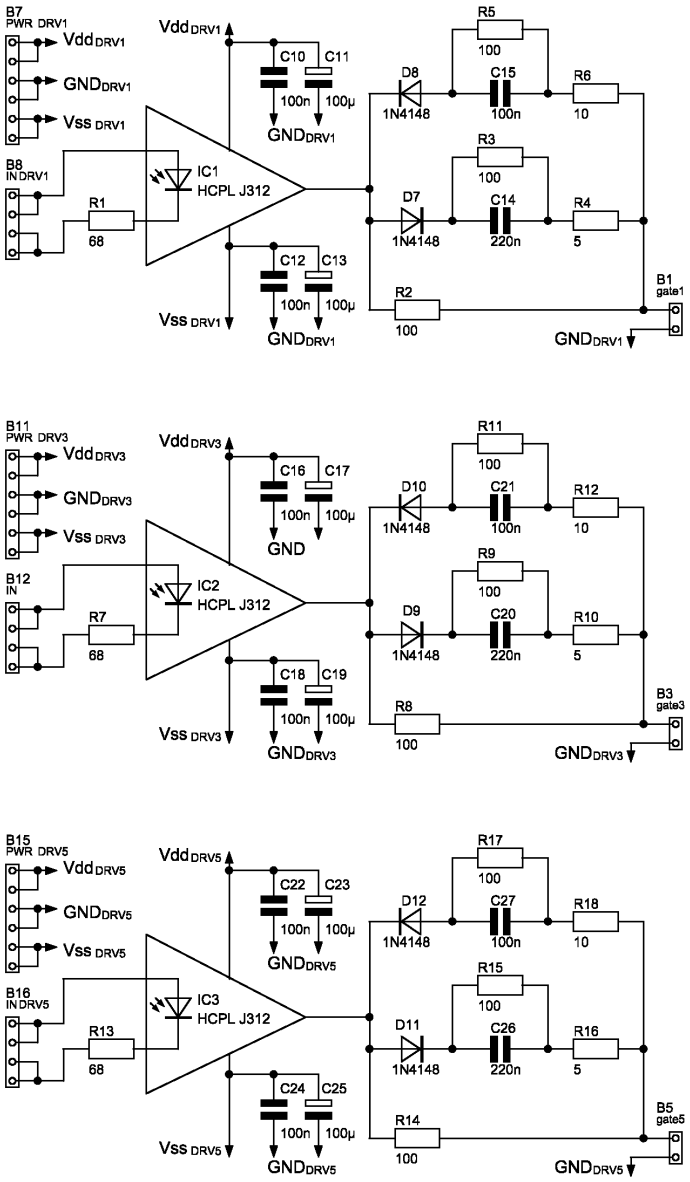


Abbildung 8.2: Schaltplan, Treiber (2x)

Versorgung

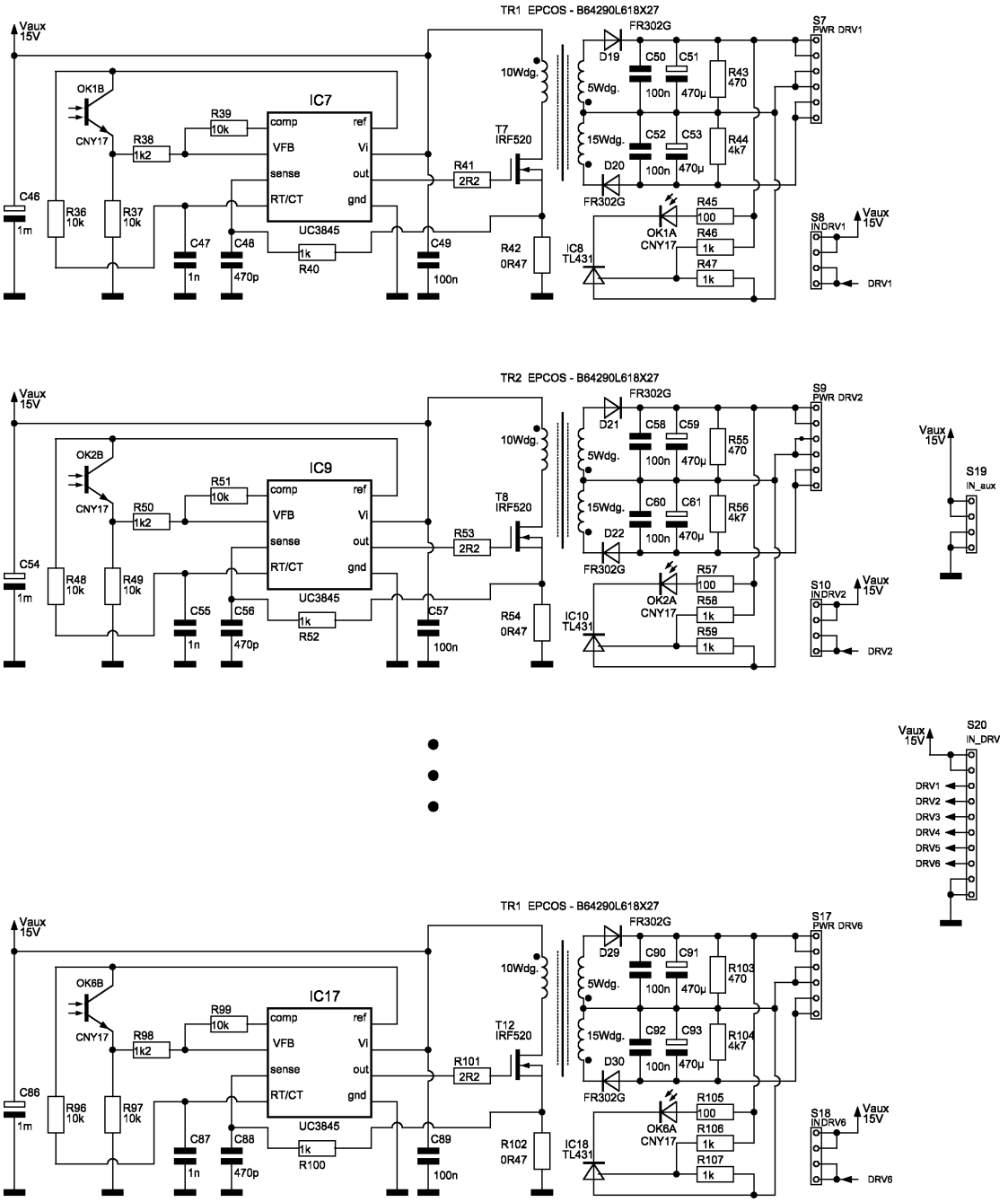


Abbildung 8.3: Schaltplan, Treiber-Spannungsversorgung

Verriegelungslogik

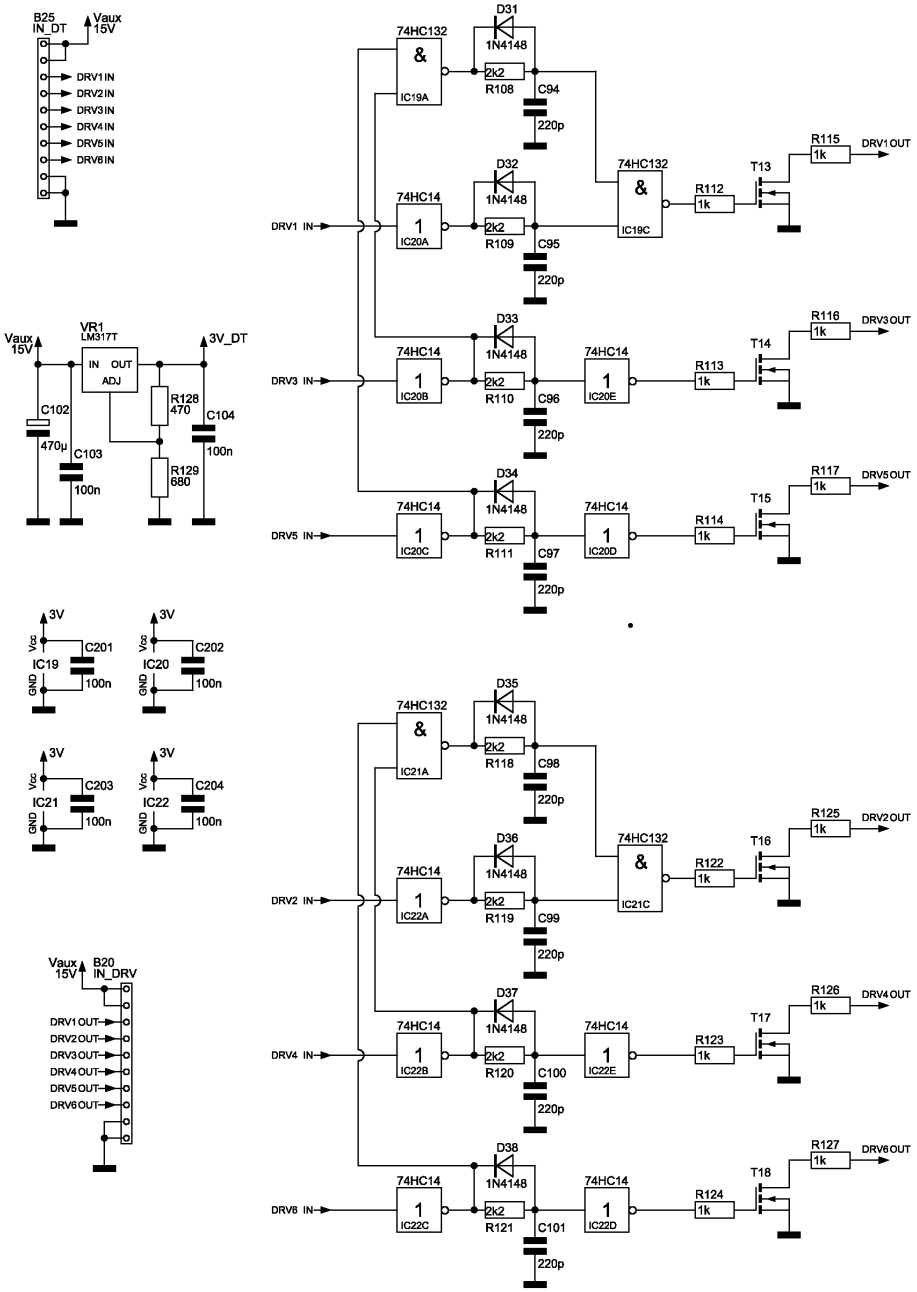


Abbildung 8.4: Schaltplan, Verriegelungslogik

Mikrocontrollerboard

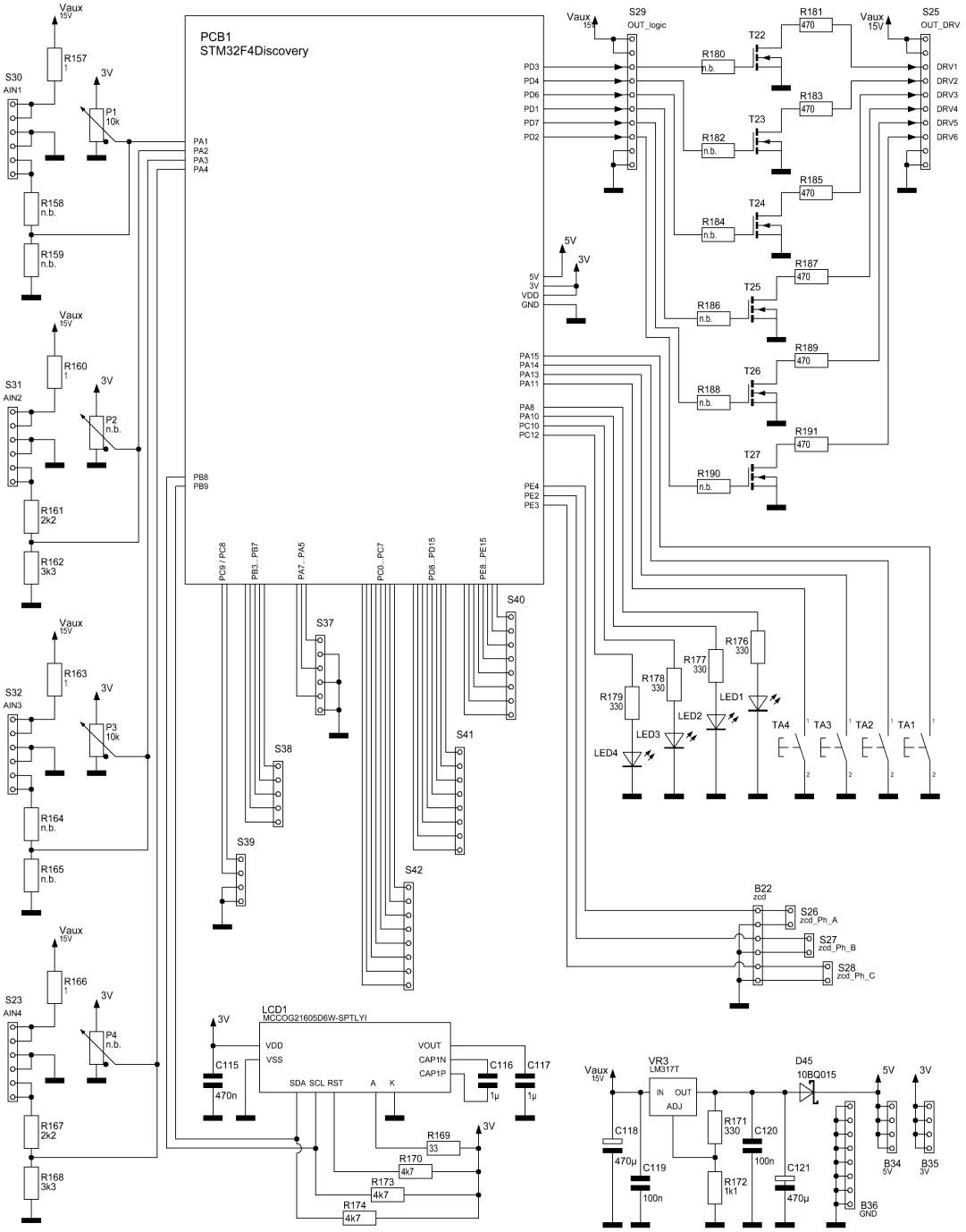


Abbildung 8.5: Schaltplan, Mikrocontrollerboard

Stromsensor

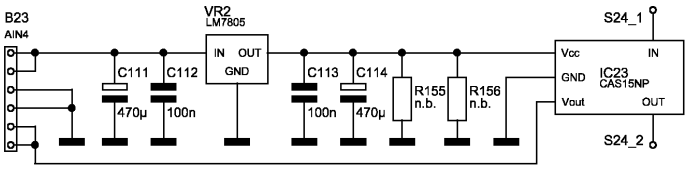


Abbildung 8.6: Schaltplan, Stromsensor

Nulldurchgangsdetektor

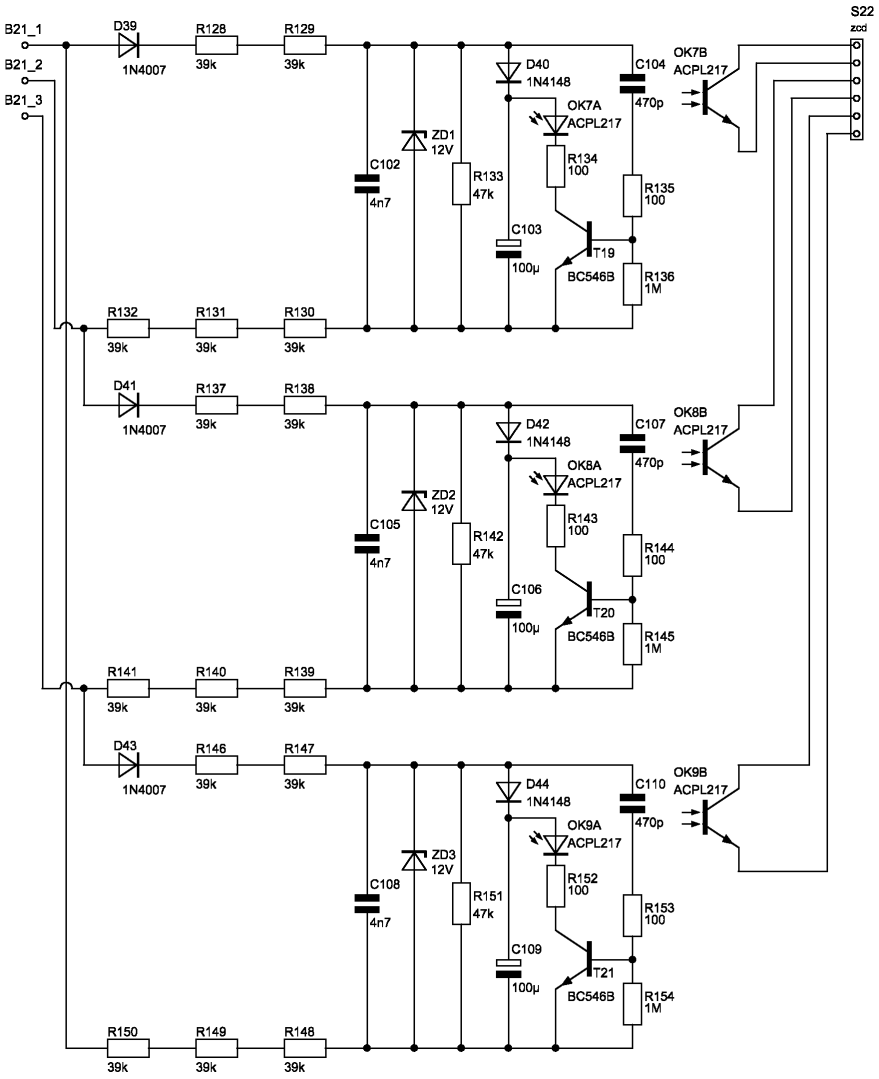


Abbildung 8.7: Schaltplan, Nulldurchgangsdetektor

8.2 Layouts

Leistungsteil

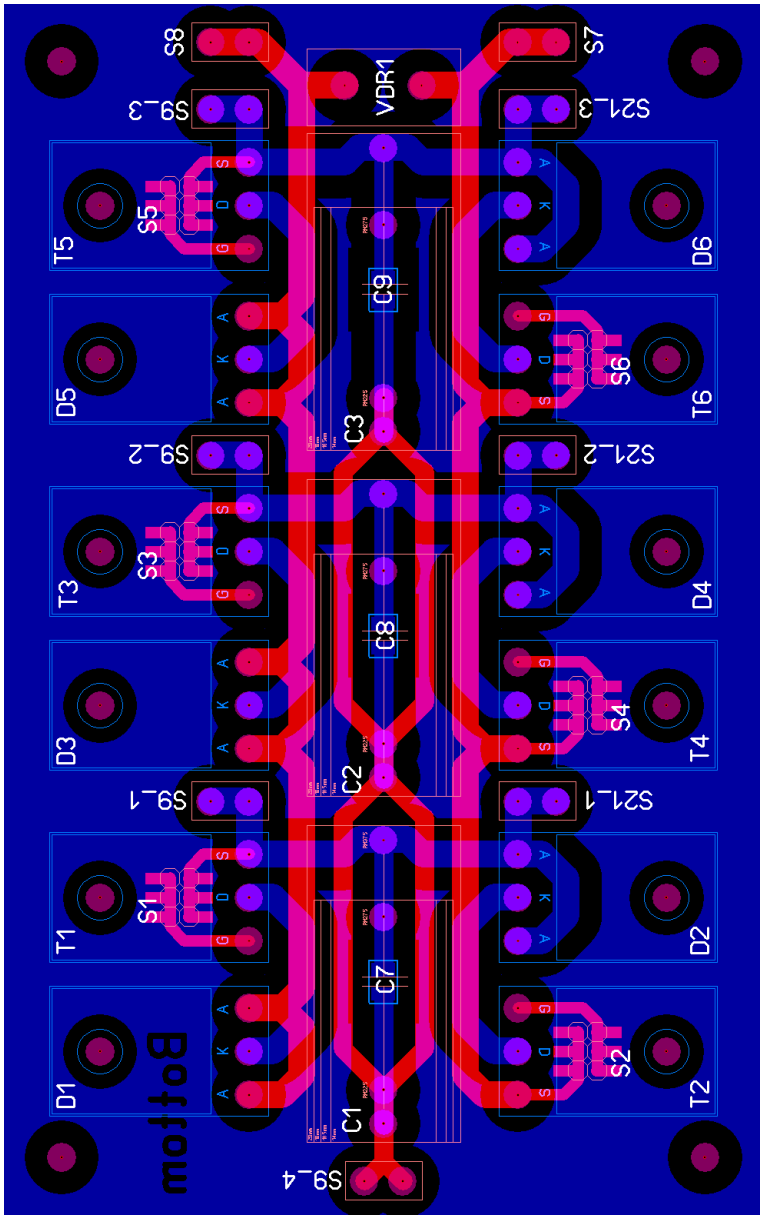


Abbildung 8.8: Leistungsteil, Bestückung

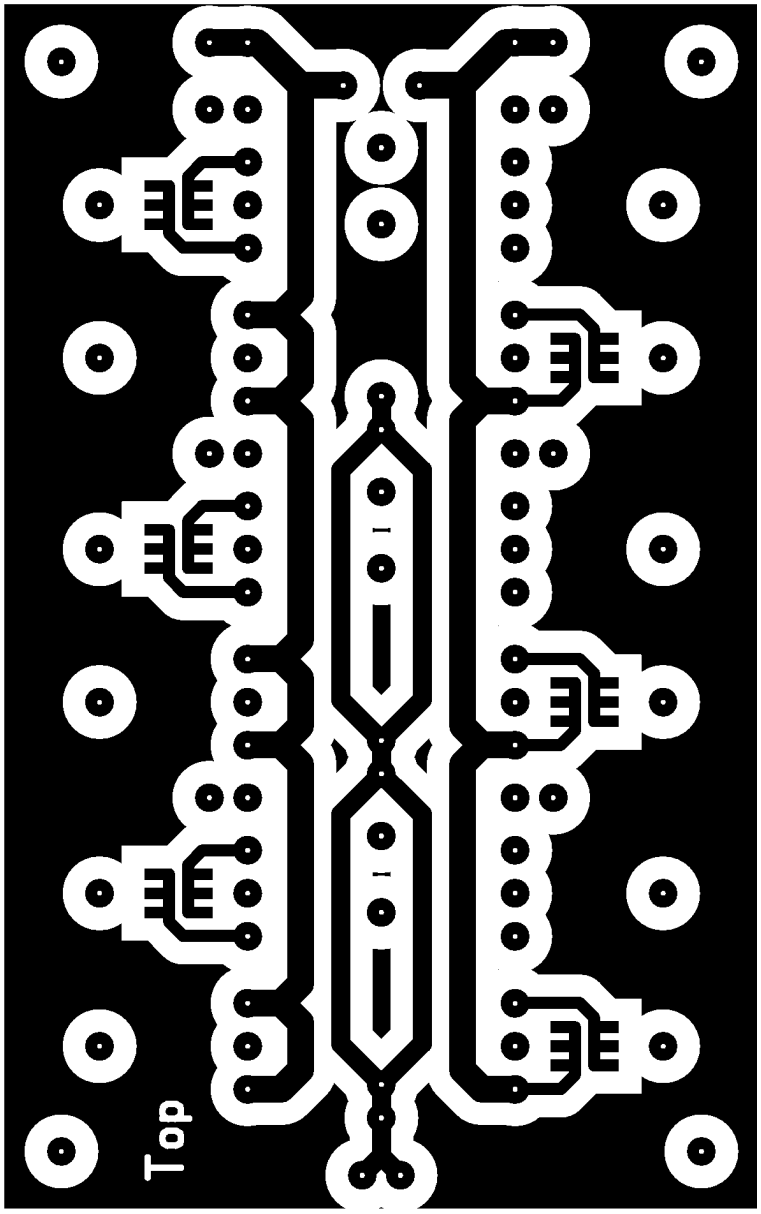


Abbildung 8.9: Leistungsteil, Top

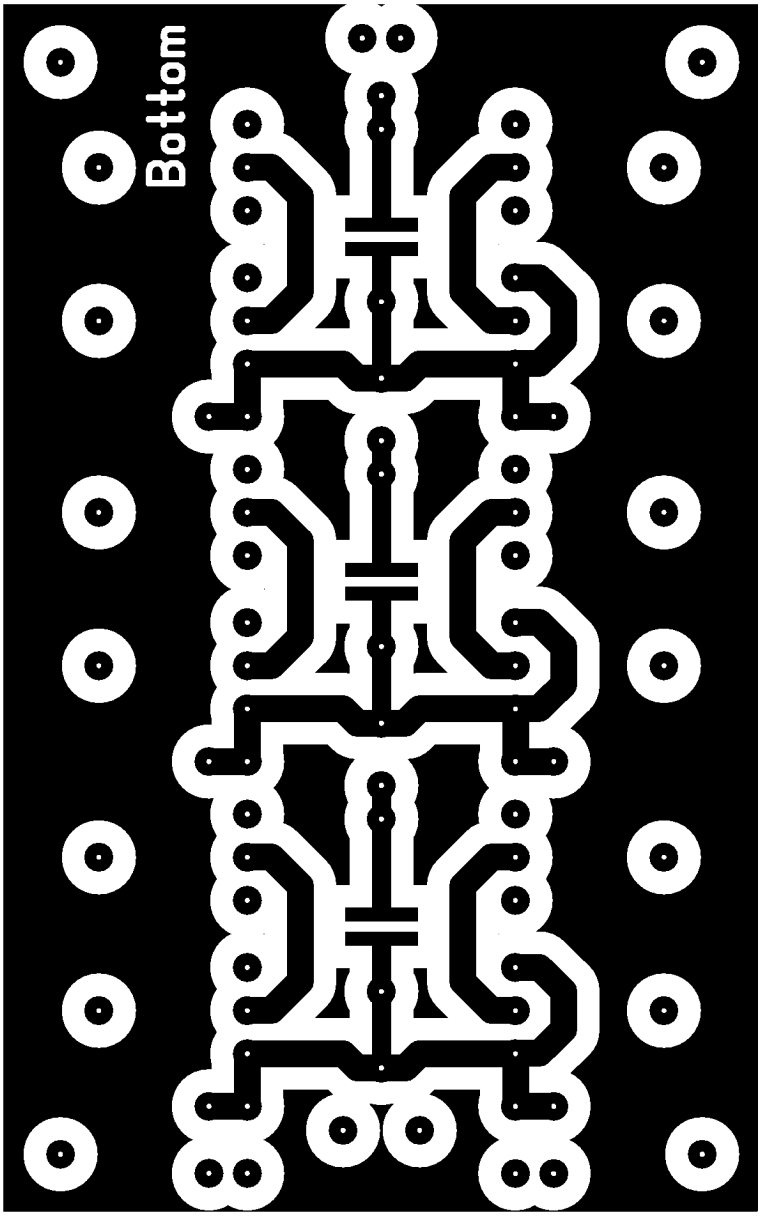


Abbildung 8.10: Leistungsteil, Bottom

Treiber

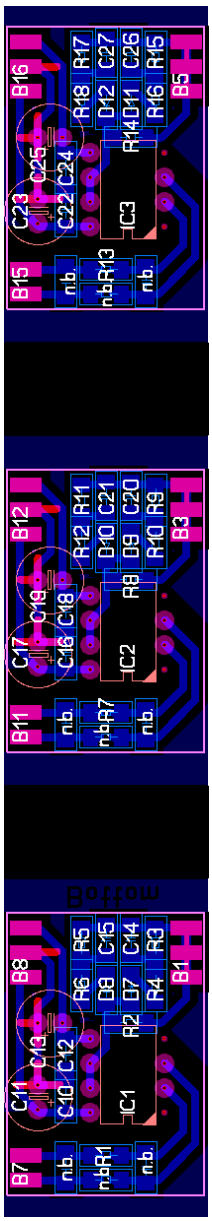


Abbildung 8.11: Treiber, Bestückung

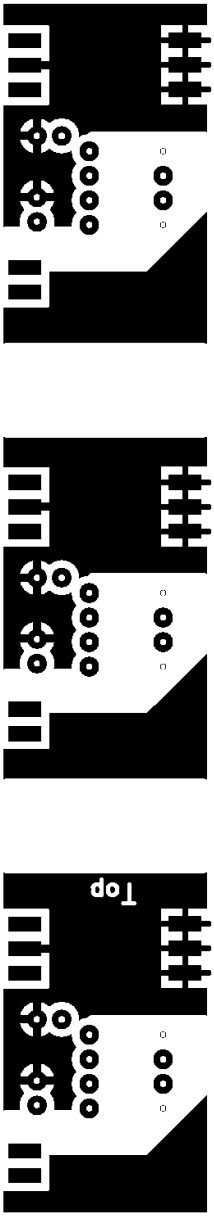


Abbildung 8.12: Treiber, Top

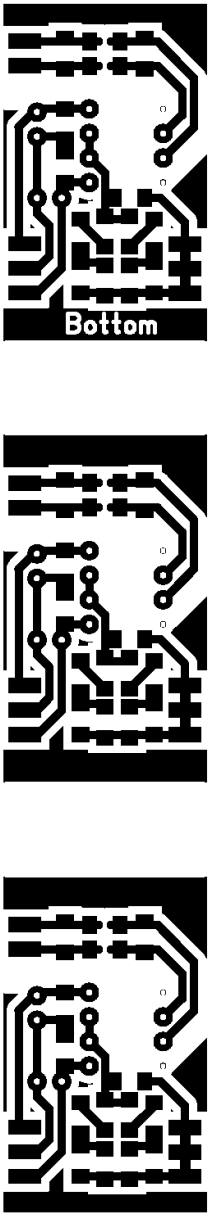


Abbildung 8.13: Treiber, Bottom

Versorgung

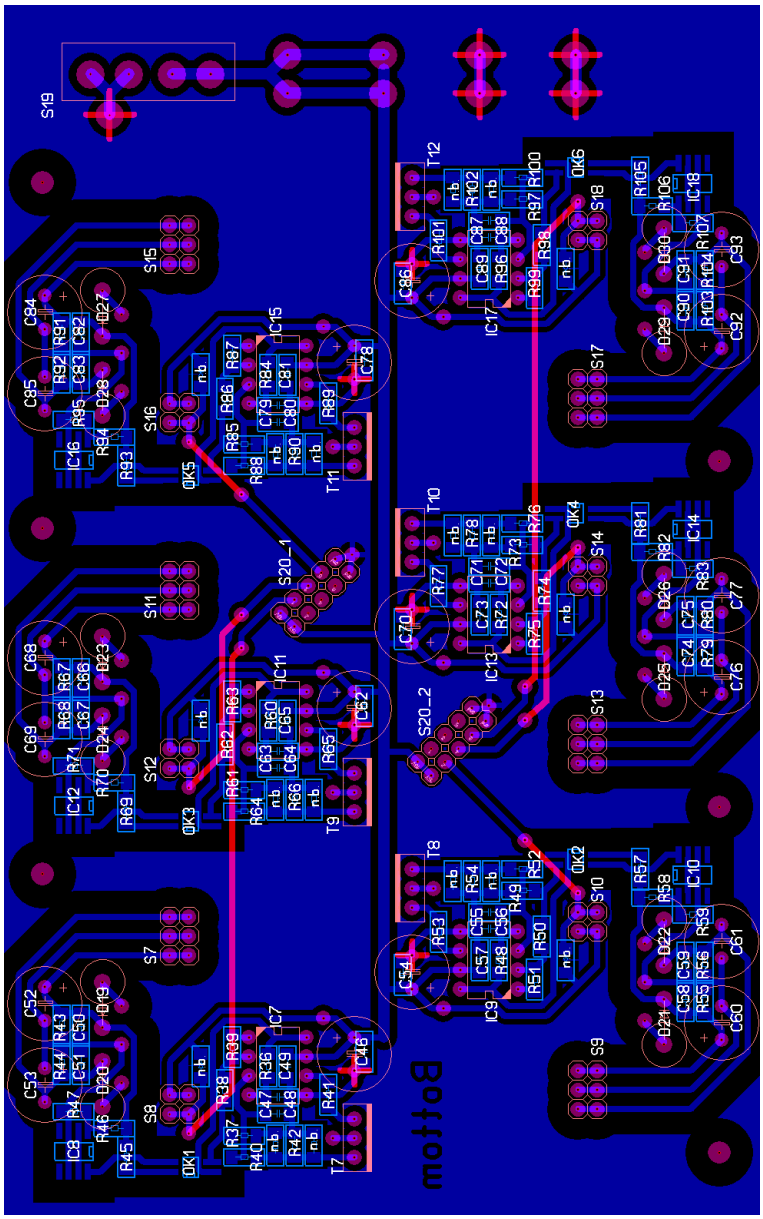


Abbildung 8.14: Spannungsversorgung der Treiber, Bestückung

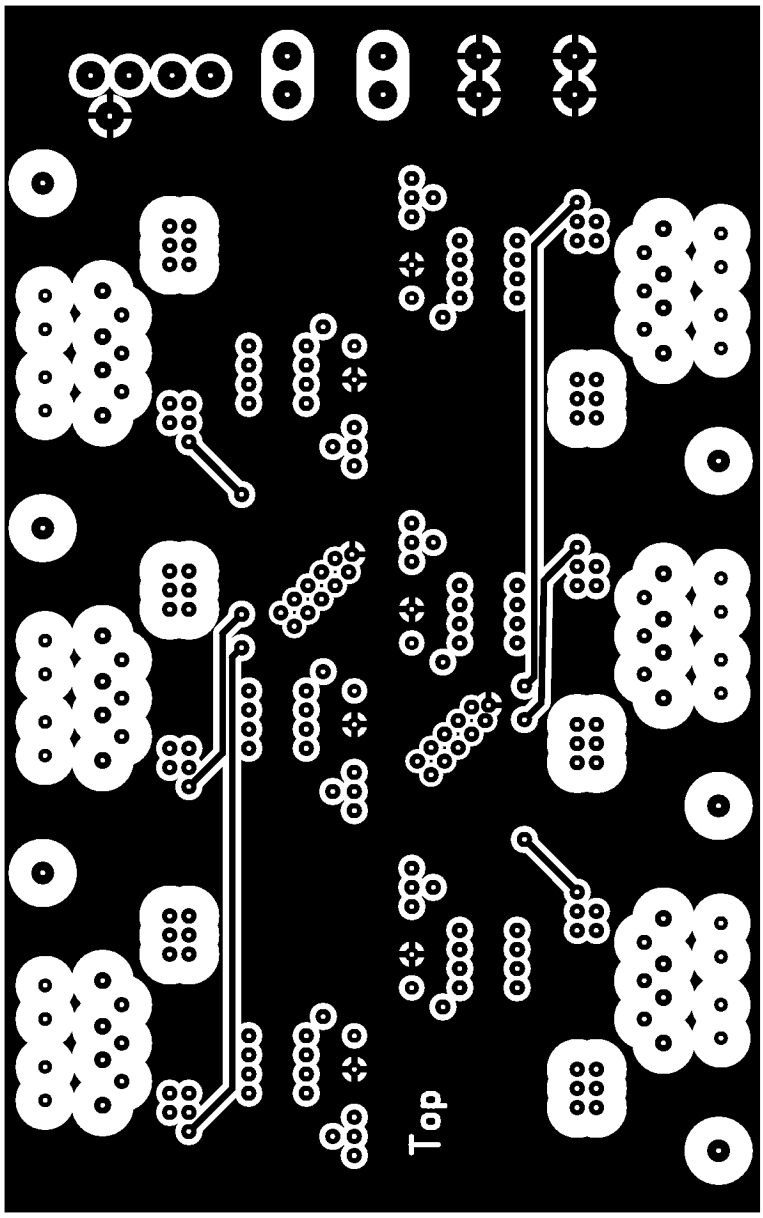


Abbildung 8.15: Spannungsversorgung der Treiber, Top

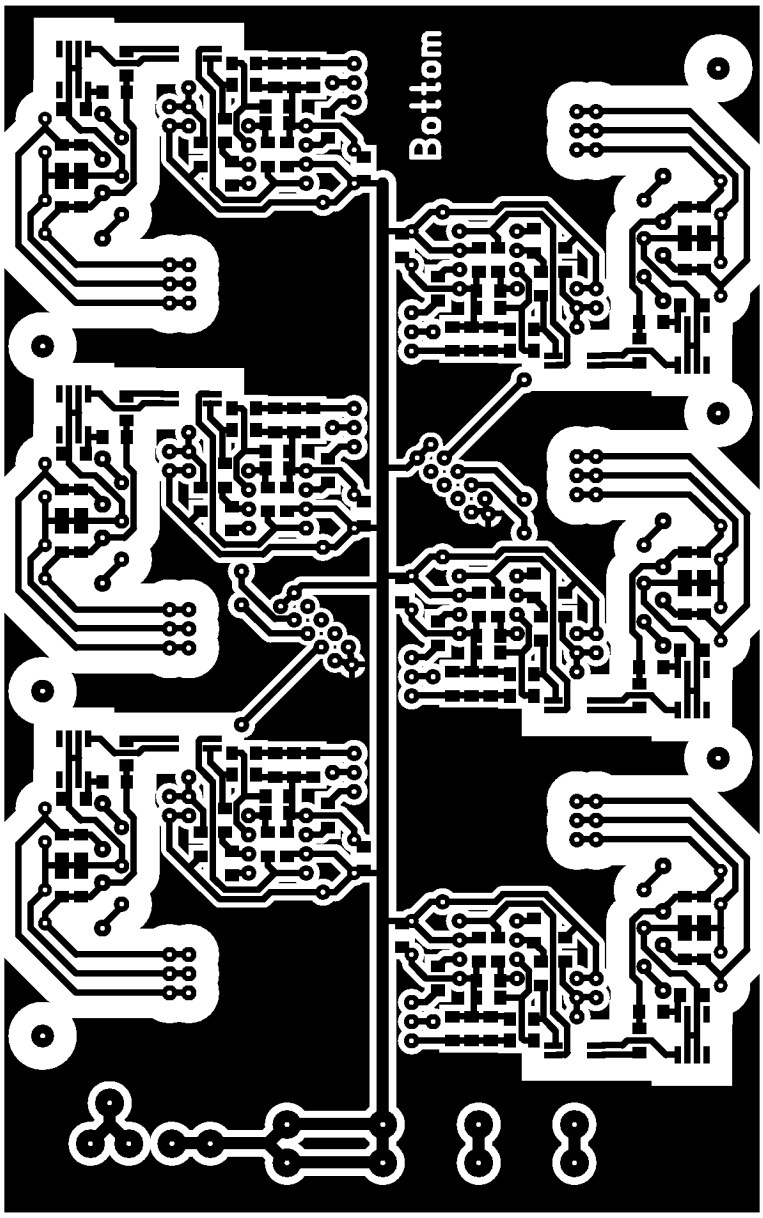


Abbildung 8.16: Spannungsversorgung der Treiber, Bottom

Verriegelungslogik

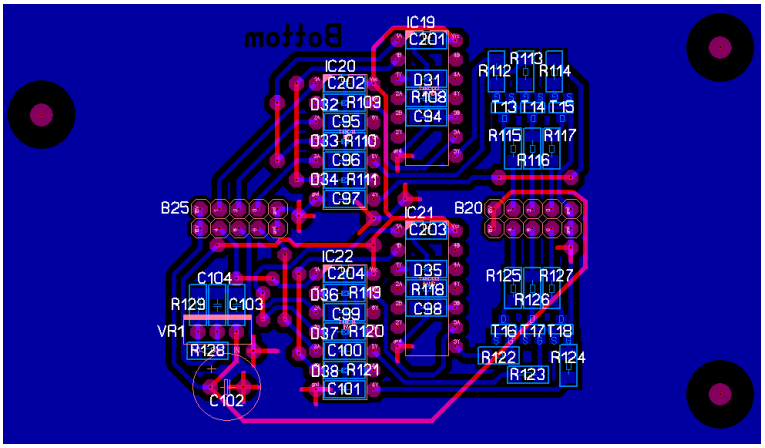


Abbildung 8.17: Verriegelungslogik, Bestückung

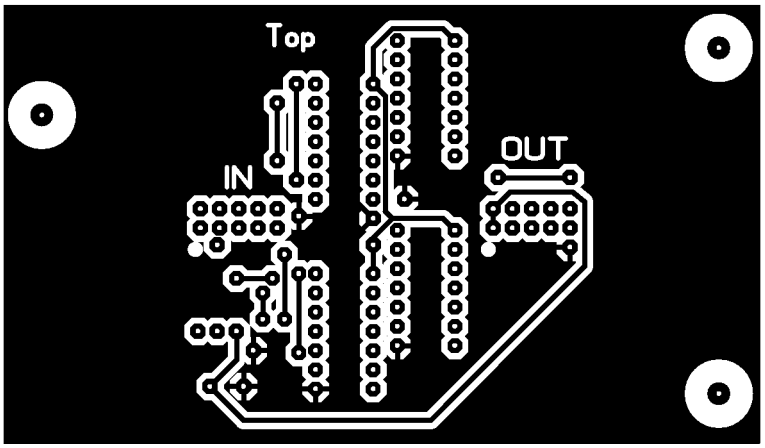


Abbildung 8.18: Verriegelungslogik, Top

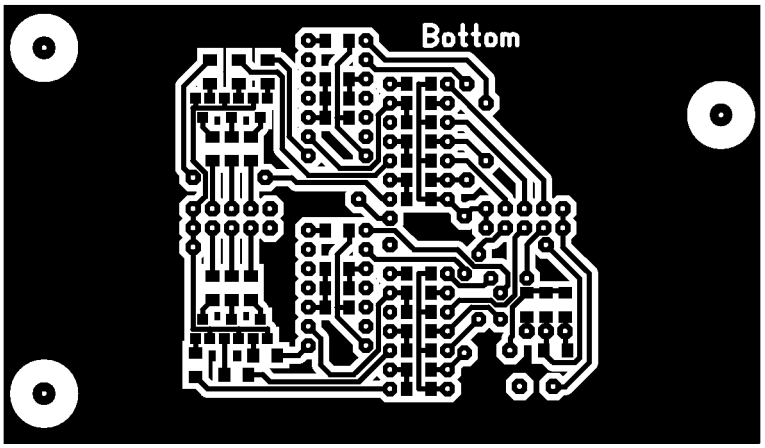


Abbildung 8.19: Verriegelungslogik, Bottom

Mikrocontrollerboard

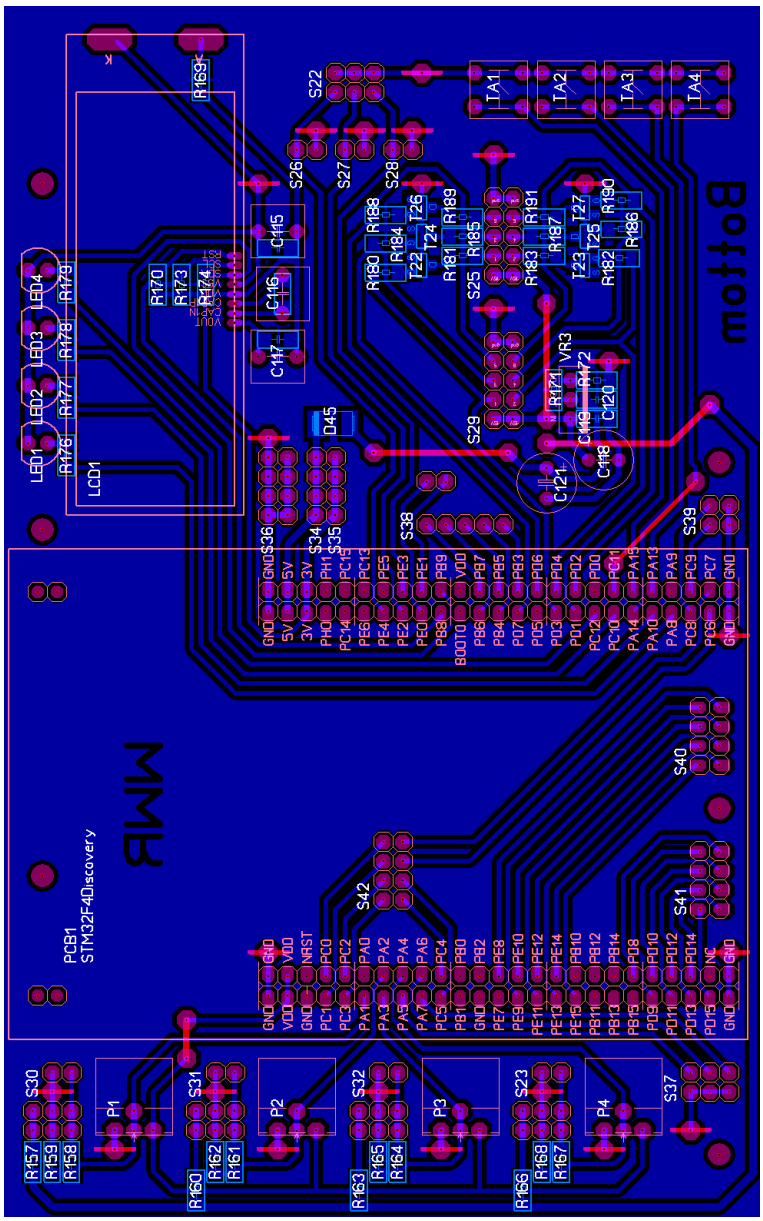


Abbildung 8.20: Mikrocontrollerboard, Bestückung

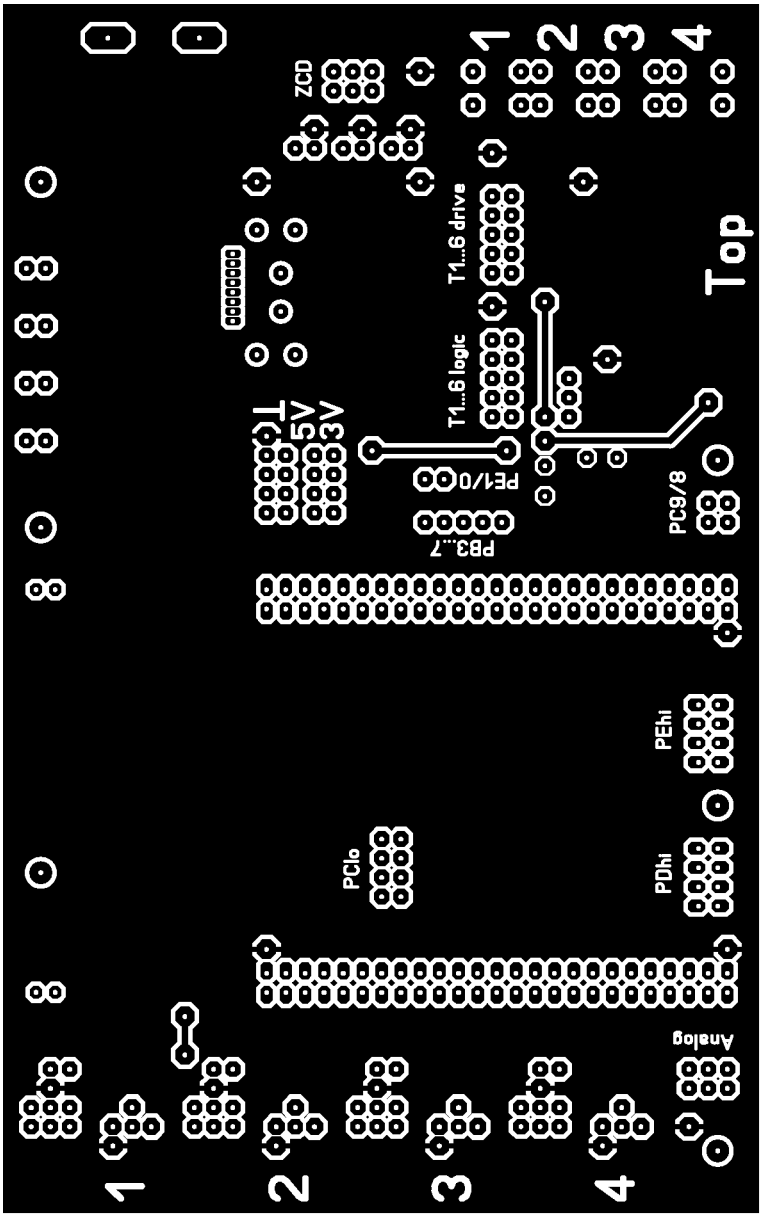


Abbildung 8.21: Mikrocontrollerboard, Top

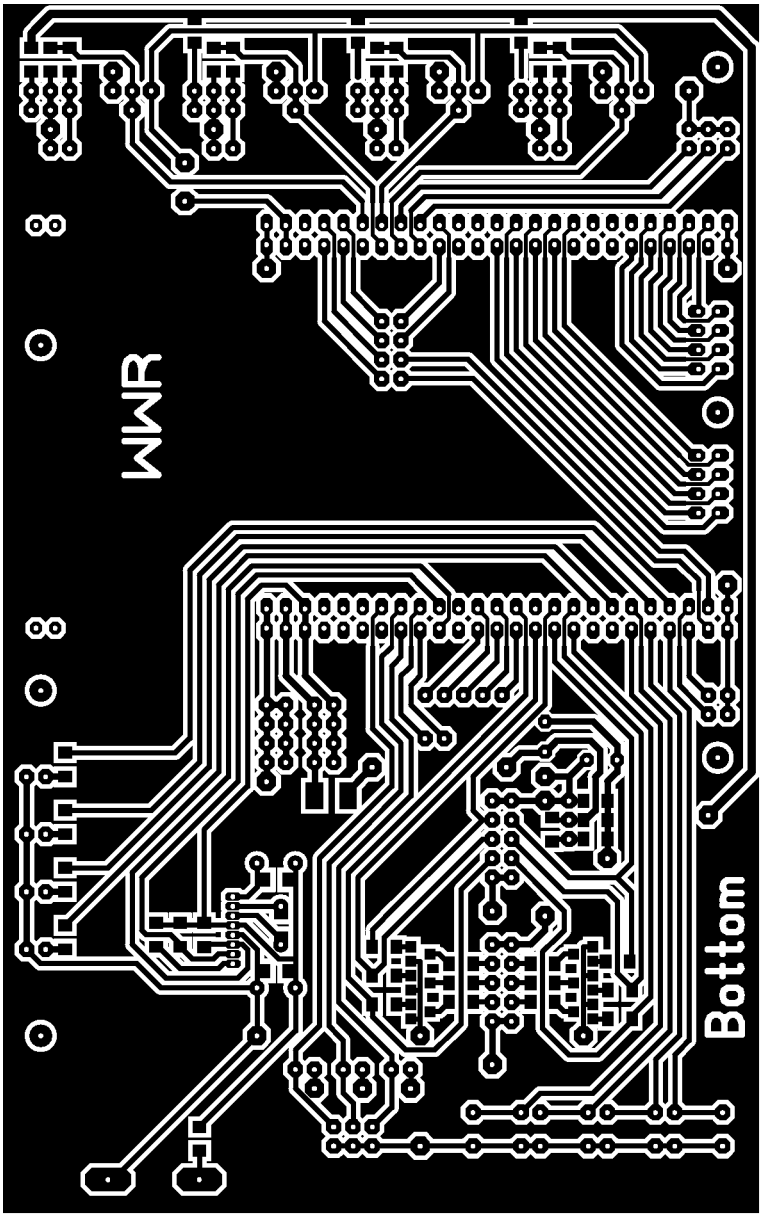


Abbildung 8.22: Mikrocontrollerboard, Bottom

Stromsensor

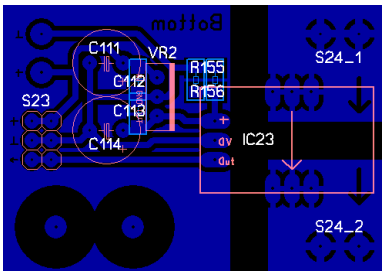


Abbildung 8.23: Stromsensor, Bestückung

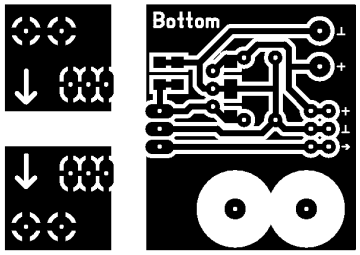


Abbildung 8.24: Stromsensor, Bottom

Nulldurchgangsdetektor

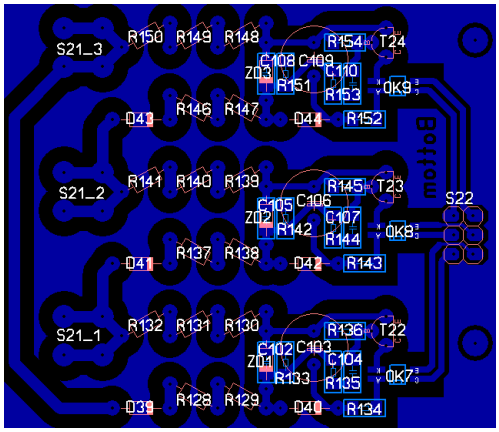


Abbildung 8.25: Nulldurchgangsdetektor, Bestückung

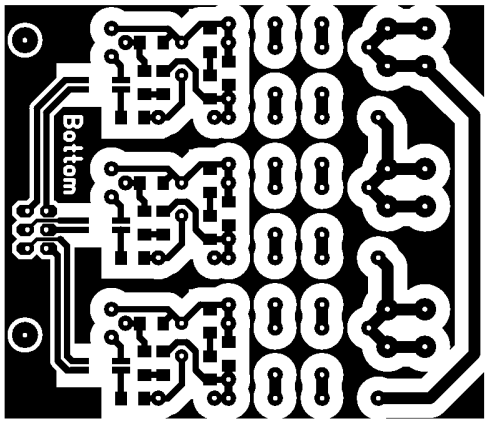


Abbildung 8.26: Nulldurchgangsdetektor, Bottom

8.3 Quellcode

functions

Listing 8.1: functions.h

```
#ifndef FUNCTIONS_H_
#define FUNCTIONS_H_

void disc_leds(char, uint8_t);              // Zugriff auf die 4 LEDs des Discovery-Boards;
                                            // Parameter: char R/G/B/O = Farbe; C = USB-OC_LED;
                                            // 0: aus, 1: ein, 2: toggle

void board_leds(uint8_t, uint8_t);          // s. "disc_leds"; fuer die LEDs am "Breakout-Board"

//void delay(volatile uint32_t);            // Warteschleife; nicht verwendet -> loeschen!

void int2str(char*, int32_t);               // konvertiert Integer nach String

void int2strST(char*, int32_t, uint8_t);    // -||-    mit vorgegebener Stellenzahl

void show_clocks(void);                     // schreibt die Taktfrequenzen auf das LCD

#endif
```

Listing 8.2: functions.c

```
/*
 * functions.c
 *
 *  Created on: 11.12.2012
 *      Author: wwr
 */


#include "main.h"
#include "functions.h"

void disc_leds(char farbe, uint8_t off_on_tog)          // Funktion f. LEDs am STM32F4Discovery
{

    if(off_on_tog==0)          // LED AUSschalten
    {
    switch (farbe)
        {
        case 'R': GPIOD->BSRRH = GPIO_Pin_14;break;     // BSSRH ist das Reset-Register
        case 'G': GPIOD->BSRRH = GPIO_Pin_12;break;
        case 'B': GPIOD->BSRRH = GPIO_Pin_15;break;
        case 'O': GPIOD->BSRRH = GPIO_Pin_13;break;
        case 'C': GPIOD->BSRRL = GPIO_Pin_5;break;      // USB-OC-LED (invertiert)
        }
    }
    if (off_on_tog==1)          // LED EINschalten
    {
        switch (farbe)
        {
        case 'R': GPIOD->BSRRL = GPIO_Pin_14;break;     // BSSRL ist das Set-Register
        case 'G': GPIOD->BSRRL = GPIO_Pin_12;break;
        case 'B': GPIOD->BSRRL = GPIO_Pin_15;break;
        case 'O': GPIOD->BSRRL = GPIO_Pin_13;break;
        case 'C': GPIOD->BSRRH = GPIO_Pin_5;break;
        }
    }
    if (off_on_tog==2)          // LED TOGGLEn
    {
        switch (farbe)
        {
        case 'R': GPIOD->ODR ^= GPIO_Pin_14;break;     // XOR des betr. ODR-bits
        case 'G': GPIOD->ODR ^= GPIO_Pin_12;break;
        case 'B': GPIOD->ODR ^= GPIO_Pin_15;break;
        case 'O': GPIOD->ODR ^= GPIO_Pin_13;break;
        case 'C': GPIOD->ODR ^= GPIO_Pin_5;break;
        }
    }

}

//void delay(volatile uint32_t nCount)     // einfache Delay-funktion; nicht benutzen!
//{while(nCount--) {}}

void int2str(char* string, int32_t zahl)  // Umwandlung von INT nach ASCII f. LCD
{
  uint32_t i, Div=1e9, j=0,  det=0;       // Div ausreichend gross fuer alle geg Zahlen

  for (i = 0; i < 10; i++)
  {
    string[j++] = (zahl / Div) + 48;      // Stelle bestimmen (/Div), + in ASCII wandeln (+48)
```

```
    zahl %= Div;                                // bereits bestimmte Stelle entfernen
    Div /= 10;                                  // Divisor verkleinern (Dezimalsystem: Faktor 1/10)
    if ( (string[j-1] == '0') && (det == 0) )  // letzte Stelle & alle vorher = null -> Anfang
    {j=0;}                                      // = Abschneiden der fuehrenden Nullen
    else
    {det=1;}
  }
  string[j++]='\0';                             // ASCII "NUL" ; end of string
}

void int2strST(char* string, int32_t zahl, uint8_t stellen)   // s.o. aber fixe Stellenanzahl
{
  uint32_t i, Div=1e9, j=0,  det=0;

  for (i = 0; i < 10; i++)
  {
    string[j++] = (zahl / Div) + 48;
    zahl %= Div;
    Div /= 10;
    if ( i >= (10-stellen) )                    // wenn Anzahl d. gew. Stellen ereicht,...
    {det=1;}                                    // ... j nichtmehr 0-setzen
    if ( (string[j-1]=='0') && (det==0) )
    {j=0;}
    else
    {det=1;}
  }
  string[j++]='\0';
}

void show_clocks()    // Funktion um die Taktfrequenzen am LCD anzuzeigen
{
        RCC_ClocksTypeDef RCC_Clocks;                       // Struct zum speichern d Clocks
        RCC_GetClocksFreq(&RCC_Clocks);                     // Clocks auslesen
        char out[2];                                        // zum Speichern d. Strings f. d. LCD
        int2str(out, RCC_Clocks.SYSCLK_Frequency/1e6);      // SYSCLK div 1M und in ASCII wandeln
        i2clcd_setcur(1,1);
        i2clcd_string("xy");
        i2clcd_setcur(1,1);                                 // Cursor positionieren
        i2clcd_string("␣S␣");                               // Strings schreiben
        i2clcd_string(out);
        int2str(out, RCC_Clocks.HCLK_Frequency/1e6);        // HCLK ...
        i2clcd_setcur(1,10);
        i2clcd_string("H␣");
        i2clcd_string(out);
        int2str(out, RCC_Clocks.PCLK1_Frequency/1e6);       // AHB1 ...
        i2clcd_setcur(2,2);
        i2clcd_string("1␣");
        i2clcd_string(out);
        int2str(out, RCC_Clocks.PCLK2_Frequency/1e6);       // AHB2 ...
        i2clcd_setcur(2,10);
        i2clcd_string("2␣");
        i2clcd_string(out);
}

void board_leds(uint8_t nummer, uint8_t off_on_tog)      // analog "disc_leds"
{
    if(off_on_tog==0)
    {
    switch (nummer)
        {
        case 1: GPIOA->BSRRH = GPIO_Pin_8;break;
        case 2: GPIOA->BSRRH = GPIO_Pin_10;break;
        case 3: GPIOC->BSRRH = GPIO_Pin_10;break;
        case 4: GPIOC->BSRRH = GPIO_Pin_12;break;
        }
    }
    if (off_on_tog==1)
    {
        switch (nummer)
        {
        case 1: GPIOA->BSRRL = GPIO_Pin_8;break;
        case 2: GPIOA->BSRRL = GPIO_Pin_10;break;
        case 3: GPIOC->BSRRL = GPIO_Pin_10;break;
        case 4: GPIOC->BSRRL = GPIO_Pin_12;break;
        }
    }
    if (off_on_tog==2)
    {
        switch (nummer)
        {
        case 1: GPIOA->ODR ^= GPIO_Pin_8;break;
        case 2: GPIOA->ODR ^= GPIO_Pin_10;break;
        case 3: GPIOC->ODR ^= GPIO_Pin_10;break;
        case 4: GPIOC->ODR ^= GPIO_Pin_12;break;
        }
    }

}
```

i2clcd

Listing 8.3: i2clcd.h

```
#ifndef I2CLCD_H_
#define I2CLCD_H_

#define SLAVE_ADDRESS 0x7C                  // I2C-Addresse des LCDs

GPIO_InitTypeDef GPIO_InitStruct;           // Typedef f. Initialisierungs-struct d. GPIOs
I2C_InitTypeDef I2C_InitStruct;             // Typedef f. Initialisierungs-struct d. I2C-Komm.

void init_I2C1(void);                       // initialisert I2C1 an PINB8/9 (SCL/SDA)mit 100kHz

void I2C_start(I2C_TypeDef*, uint8_t, uint8_t);     // startet I2C transfer

void I2C_stop(I2C_TypeDef*);                // stoppt I2C transfer

void I2C_write(I2C_TypeDef*, uint8_t);      // sendet Datenbyte

void i2clcd_command(uint8_t);               // sendet Kommando an das LCD

void i2clcd_databyte(uint8_t);              // sendet Datenbyte an das LCD

void i2clcd_setcur(uint8_t, uint8_t);       // setzt den Cursor am LCD (Param.: Zeile, Spalte)

void i2clcd_string(const char*);            // sendet String an das LCD

void i2clcd_init(void);                     // initialisiert das LCD

#endif
```

Listing 8.4: i2clcd.c

```
#include <stm32f4xx.h>
#include <stm32f4xx_i2c.h>
#include "main.h"
#include "i2clcd.h"


// Initialisierung d. I2C-Buses
void init_I2C1(void)
{
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE);  // Clock fuer I2C Schnittstelle
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE); // Clock fuer GPIOB

    // Portpins initialisieren
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_9;      // PB8/PB9
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;                // Verw. als Alternate Function
    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;           // Geschwindigkeit
    GPIO_InitStruct.GPIO_OType = GPIO_OType_OD;              // Open Drain
    GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;                // Pull-Ups einschalten
    GPIO_Init(GPIOB, &GPIO_InitStruct);
    // die internen Pull-Ups sind fuer 100kHz I2C-Takt nicht ausreichend -> externe Rs!

    // Pins an Alternate Function verknuepfen
    GPIO_PinAFConfig(GPIOB, GPIO_PinSource8, GPIO_AF_I2C1); // SCL
    GPIO_PinAFConfig(GPIOB, GPIO_PinSource9, GPIO_AF_I2C1); // SDA

    I2C_InitStruct.I2C_ClockSpeed = 100000;                  // 100kHz
    I2C_InitStruct.I2C_Mode = I2C_Mode_I2C;                  // I2C mode
    I2C_InitStruct.I2C_DutyCycle = I2C_DutyCycle_2;          // d=0.5 (standard)
    I2C_InitStruct.I2C_OwnAddress1 = 0x00;                   // Eigene Adresse
    I2C_InitStruct.I2C_Ack = I2C_Ack_Disable;                // kein Ack beim Lesen
    I2C_InitStruct.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit; // Addresslaenge
    I2C_Init(I2C1, &I2C_InitStruct);
    I2C_Cmd(I2C1, ENABLE);                                   // I2C1 EINschalten
}

// Starten einer I2C-Uebertragung
void I2C_start(I2C_TypeDef* I2Cx, uint8_t address, uint8_t dir)
{
    while(I2C_GetFlagStatus(I2Cx, I2C_FLAG_BUSY));           // warten bis Bus frei
    I2C_GenerateSTART(I2Cx, ENABLE);                         // Startbedingung
    while(!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_MODE_SELECT))
        {    }                                               // warten bis ACK
    I2C_Send7bitAddress(I2Cx, address, dir);                 // Slaveaddresse senden
    if        (dir == I2C_Direction_Transmitter)             // Slave bestaetigt Mode
        {
        while(!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED)) {}
        }                                                    // Transmitter
    else
    {
        if    (dir == I2C_Direction_Receiver)
            {
            while(!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED)) {}
            }                                                // Receiver
    }
```

```
}

// Beenden d. Uebertragung
void I2C_stop(I2C_TypeDef* I2Cx)
{
    I2C_GenerateSTOP(I2Cx, ENABLE);                                  // Stopbedingung
}

// Daten per I2C senden
void I2C_write(I2C_TypeDef* I2Cx, uint8_t data)
{
    I2C_SendData(I2Cx, data);                                                  // Daten senden
    while(!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_BYTE_TRANSMITTED)) {}  // warten bis fertig
}

// Kommando an LCD senden
void i2clcd_command(uint8_t command)
{
    I2C_start(I2C1, SLAVE_ADDRESS, I2C_Direction_Transmitter);  // Start
    I2C_write(I2C1, 0x80);                      // "Code" fuer: es folgen Kommandos
    I2C_write(I2C1, command);                   // Kommando senden
    I2C_stop(I2C1);                             // Stop
}

// Daten an LCD senden
void i2clcd_databyte(uint8_t data)
{
    I2C_start(I2C1, SLAVE_ADDRESS, I2C_Direction_Transmitter);
    I2C_write(I2C1, 0x40);              // "Code" fuer: es folgen Daten
    I2C_write(I2C1, data);              // Daten senden
    I2C_stop(I2C1);
}

// Cursor d. LCDs setzen
void i2clcd_setcur(uint8_t zeile, uint8_t spalte)
{
    uint8_t ddramaddr;
    if(zeile==1){ddramaddr=0x00;}       // Zeilenaddresse im DDRAM setzen
    if(zeile==2){ddramaddr=0x40;}       // 2. Zeile
    ddramaddr+=spalte;                  // Spaltenaddresse hizufuegen
    ddramaddr-=0x01;                    // Korrektur wegen Indexierung v. 0
    ddramaddr+=0x80;                    // MSB setzen = Befehl f. schreiben in DDRAM
    I2C_start(I2C1, SLAVE_ADDRESS, I2C_Direction_Transmitter);
    I2C_write(I2C1, 0x80);              // es folgt: Kommando
    I2C_write(I2C1, ddramaddr);         // Speicheraddresse(=Position) senden
    I2C_stop(I2C1);                     // stop
}

// String auf LCD ausgeben
void i2clcd_string(const char *data)
{
    I2C_start(I2C1, SLAVE_ADDRESS, I2C_Direction_Transmitter);
    I2C_write(I2C1, 0x40);              // es folgen: Daten
    while (*data!='\0')                 // wenn != Stringende
    {I2C_write(I2C1, *data++);}         // sende naechstes Zeichen
    I2C_stop(I2C1);                     // stop
}

// Initialisierung d LCDs
void i2clcd_init()
{
    I2C_start(I2C1, SLAVE_ADDRESS, I2C_Direction_Transmitter);
    I2C_write(I2C1, 0x0);
    I2C_write(I2C1, 0x38);              // 8bit mode, 2lines, normal set
    I2C_write(I2C1, 0x39);              // 8bit mode, 2lines, ext set
    I2C_write(I2C1, 0x14);              // CursorShift Rechts
    I2C_write(I2C1, 0x7C);              // Set CGRAM to 110100
    I2C_write(I2C1, 0x5D);              // Set CGRAM to 010100
    I2C_write(I2C1, 0x6D);              // Set CGRAM to 101111
    I2C_write(I2C1, 0x0F);              // LCD ON, Cursor OFF, Blink OFF
    I2C_write(I2C1, 0x01);              // Clear LCD
    I2C_write(I2C1, 0x02);              // Return Home
    I2C_write(I2C1, 0x06);              // LCD ON, Cursor ON, Blink ON
    I2C_stop(I2C1);                     // stop
    //i2clcd_string(" ");               // zu Testzwechen; entfernt
    //i2clcd_setcur(1,1);
}
```

inits

Listing 8.5: inits.h

```
#include <stdint.h>

#ifndef INITS_H_
#define INITS_H_

void taster_init(void);             // initialisiert den Taster am Discovery-Board als Eingang
void leds_init(void);               // initialisiert die LEDs am Board als Ausgaenge
void EXTI_init_angdet(void);        // init v. PE4/2/3 als ext-Interrupt, zur Winkelbestimmung
void ADC_DMA_init_curr(uint32_t);   // liest PA1/2 per DMA2 in Array (Param: Addr. d. Arrays)
void outputs_init(void);            // initialisiert die 6 Ausgaenge zur B6-Bruecke

// init fuer Timer zur Winkelbestimmung; Timertakt:84MHz
#define TIM6_clk            84e6                    // Takt v. TIM6
#define TIM_ang_presc       45                      // Vorteiler f. TIM6 (33)
#define TIM_ang_presc_x3    TIM_ang_presc * 3       // wird im Programm verwendet
//#define TIM_ang_max_50      TIM6_clk * 0.02 / TIM_ang_presc//Timertakt*Netzper./Presc.(33)
//#define TIM_ang_max_40      TIM_ang_max_50 * 50 / 40  // bis hierher zaehlt der Timer bei 40Hz

void TIM_init_ang(void);            // Initialisierung d. Timers zur Winkelbestimmung

void TIM_init_clk(void);            // Initialisierung d. Timers f. Ausgabe d. Zustandsvektoren

void TIM_init_sinPWM(void);         // Timer wird f. PWM an PD13 (nur f. Kontrollen)

void FPU_on(void);                  // Explizites Aktivieren d. FloatingPointUnit

void STM_Board_init(void);          // Init der Taster u. LEDs am "Breakout-Board"

void SysTick_init(void);            // Init. d. Systick-Timers; jede ms;

void INIT(uint32_t);                // Zusammenfassung aller Initialisierungen am Programmstart

void ADC2_init(void);

#endif
```

Listing 8.6: inits.c

```
#include "inits.h"
#include "main.h"

// initialisiert den Taster am Discovery-Board als Eingang
void taster_init(void)
{
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);  // Clock fuer GPIOA
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_0;                 // Konfiguriert wird: PIN0
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN;              // als: EINgang
    GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;            // Output-type: egal, weil: Input
    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;         // Geschwindigkeit
    GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_DOWN;            // interner Pull-down aktiviert
    GPIO_Init(GPIOA, &GPIO_InitStruct);
}

// initialisiert die LEDs am Board als Ausgaenge
void leds_init(void)
{
    // Pin 5 ist die USB-OC-LED; an open-drain-out vom USB-versorgungs-IC angeschlossen
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);  // Clock fuer GPIOD
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_12|GPIO_Pin_13|GPIO_Pin_14|GPIO_Pin_15|GPIO_Pin_5;
    // Konfiguriert werden: PIN 5 / 12 / 13 / 14 / 15
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT;             // als: AUSgang
    GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;            // in Push-Pull-Konfig
    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_100MHz;        // Geschwindigkeit
    GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;          // keine Pull-Up/Down-Widerstaende
    GPIO_Init(GPIOD, &GPIO_InitStruct);
}

// initialisiert PE4/2/3 als ext-Interrupt, zur Winkelbestimmung
void EXTI_init_angdet(void)
{
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);     // notw. fuer:"SYSCFG_EXTILine.."

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOE, ENABLE);      // Pins initialisieren
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_4 | GPIO_Pin_2 | GPIO_Pin_3;
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN;
    GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_2MHz;
    GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_Init(GPIOE, &GPIO_InitStruct);

    SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOE , GPIO_PinSource4);     // EXTI Lines an GPIOE
    SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOE , GPIO_PinSource2);
    SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOE , GPIO_PinSource3);
```

```
    EXTI_InitStruct.EXTI_Line = EXTI_Line4;                          // Line 4 konfigurieren
    EXTI_InitStruct.EXTI_LineCmd = ENABLE;                           // Einschalten
    EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;                 // Mode: Interrupt
    EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Falling;             // fallende Flanke
    EXTI_Init(&EXTI_InitStruct);

    EXTI_InitStruct.EXTI_Line = EXTI_Line2;                          // Line 2 konfigurieren
    EXTI_InitStruct.EXTI_LineCmd = ENABLE;
    EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;
    EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Falling;
    EXTI_Init(&EXTI_InitStruct);

    EXTI_InitStruct.EXTI_Line = EXTI_Line3;                          // Line 3 konfigurieren
    EXTI_InitStruct.EXTI_LineCmd = ENABLE;
    EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;
    EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Falling;
    EXTI_Init(&EXTI_InitStruct);

    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);
    NVIC_InitStruct.NVIC_IRQChannel = EXTI4_IRQn;                    // EXTI4_IRQ konfig
    NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;                     // Einschalten
    NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0x04;        //
    NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0x00;
    NVIC_Init(&NVIC_InitStruct);

    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);                  // EXTI2_IRQ
    NVIC_InitStruct.NVIC_IRQChannel = EXTI2_IRQn;
    NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
    NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0x04;
    NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0x00;
    NVIC_Init(&NVIC_InitStruct);

    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);                  // EXTI3_IRQ
    NVIC_InitStruct.NVIC_IRQChannel = EXTI3_IRQn;
    NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
    NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0x04;
    NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0x00;
    NVIC_Init(&NVIC_InitStruct);

}

// Init d. AD-Wandlers (2 Kanaele wandeln + per DMA in Array uebertragen)
void ADC_DMA_init_curr(uint32_t memaddr)
{
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA2, ENABLE);               // Clock f. AHB1
  DMA_InitStruct.DMA_Channel = DMA_Channel_0;                        // DMA Kanal 0
  DMA_InitStruct.DMA_PeripheralBaseAddr = (uint32_t)&(ADC1->DR);     // Quellreg: ADC-Data-Reg
  DMA_InitStruct.DMA_Memory0BaseAddr = memaddr;                      // Zieladdr != Variablenname!
  DMA_InitStruct.DMA_DIR = DMA_DIR_PeripheralToMemory;               // Richtung: Periph. -> Mem.
  DMA_InitStruct.DMA_BufferSize = 2;                                 // 2 Werte
  DMA_InitStruct.DMA_PeripheralInc = DMA_PeripheralInc_Disable;      // Quellregister !=increment
  DMA_InitStruct.DMA_MemoryInc = DMA_MemoryInc_Enable;               // Zielregister increment
  DMA_InitStruct.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;     // 16bit-daten
  DMA_InitStruct.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;             // 16bit-daten
  DMA_InitStruct.DMA_Mode = DMA_Mode_Circular;                       // zyklischer Durchlauf
  DMA_InitStruct.DMA_Priority = DMA_Priority_Medium;                 // mittlere Prioritaet
  DMA_InitStruct.DMA_FIFOMode = DMA_FIFOMode_Disable;                // FIFO Mode aus
  DMA_InitStruct.DMA_FIFOThreshold = DMA_FIFOThreshold_HalfFull;
  DMA_InitStruct.DMA_MemoryBurst = DMA_MemoryBurst_Single;           // jeder Transf genau 1 Wert
  DMA_InitStruct.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;// -||-
  DMA_Init(DMA2_Stream0, &DMA_InitStruct);                           // DMA2-Stream0 init
  DMA_Cmd(DMA2_Stream0, ENABLE);                                     // DMA2-Stream0 ein

  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);               // Clock fuer ADC1 ein
  GPIO_InitStruct.GPIO_Pin = GPIO_Pin_1 | GPIO_Pin_4;                // Pin PA1 u. PA2
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AIN;                         // als analog-in
  GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(GPIOA, &GPIO_InitStruct);

  ADC_CommonInitStruct.ADC_Mode = ADC_Mode_Independent;              // ADC-Mode: unabhaengig
  ADC_CommonInitStruct.ADC_Prescaler = ADC_Prescaler_Div8;           // Clock-teiler: 8
  ADC_CommonInitStruct.ADC_DMAAccessMode = ADC_DMAAccessMode_Disabled;       // kein DMA-Access
  ADC_CommonInitStruct.ADC_TwoSamplingDelay = ADC_TwoSamplingDelay_5Cycles; // 5 cycles delay
  ADC_CommonInit(&ADC_CommonInitStruct);

  ADC_InitStruct.ADC_Resolution = ADC_Resolution_12b;                // Genauigkeit: 12bit
  ADC_InitStruct.ADC_ScanConvMode = ENABLE;                          // mehrere Pins auslesen
  ADC_InitStruct.ADC_ContinuousConvMode = ENABLE;                    // kontinuierlich wandeln
  ADC_InitStruct.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_None; // notw. f. SW-trig
  ADC_InitStruct.ADC_DataAlign = ADC_DataAlign_Right;                // Daten rechtsbuendig
  ADC_InitStruct.ADC_NbrOfConversion = 2;                            // 2 Kanaele wandeln
  ADC_Init(ADC1, &ADC_InitStruct);

  ADC_RegularChannelConfig(ADC1, ADC_Channel_1, 1, ADC_SampleTime_15Cycles);  // 1.: CH1(PA1)
  ADC_RegularChannelConfig(ADC1, ADC_Channel_4, 2, ADC_SampleTime_15Cycles);  // 2.: CH4(PA4)

  ADC_DMARequestAfterLastTransferCmd(ADC1, ENABLE);                // nach Wandlung: DMA Request
  ADC_DMACmd(ADC1, ENABLE);                                        // DMA-Request aktivieren
  ADC_Cmd(ADC1, ENABLE);                                           // ADC einschalten
  ADC_SoftwareStartConv(ADC1);                                     // starten (SW-triggerung)
  // Kalibrierung des ADCs ist beim F4xx hinfaellig!
}
```

```
// initialisiert die 6 Ausgaenge zur B6-Bruecke
void outputs_init(void)
{
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
  GPIO_InitStruct.GPIO_Pin = GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3|GPIO_Pin_4|GPIO_Pin_6|GPIO_Pin_7;
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStruct.GPIO_Speed = GPIO_Speed_2MHz;
  GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_Init(GPIOD, &GPIO_InitStruct);
}

// Initialisierung d. Timers zur Winkelbestimmung
void TIM_init_ang(void)
{
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM6, ENABLE);            // Takt fuer TIM6 ein

    TIM_TimeBase_InitStruct.TIM_ClockDivision = TIM_CKD_DIV1;       // nur relevant f. ext. Takt
    TIM_TimeBase_InitStruct.TIM_CounterMode = TIM_CounterMode_Up; // Aufwaertszaehler
    TIM_TimeBase_InitStruct.TIM_Period = 65000; //TIM_ang_max_40-1;// Max.@40Hz; nicht relev.
    TIM_TimeBase_InitStruct.TIM_Prescaler = TIM_ang_presc-1;        // Vorteiler
    TIM_TimeBaseInit(TIM6, &TIM_TimeBase_InitStruct);

    TIM_ITConfig(TIM6, TIM_IT_Update, ENABLE);
    // Prioritaet einstellen
    NVIC_InitStruct.NVIC_IRQChannel = TIM6_DAC_IRQn;                // IRQ f. TIM6 konfig
    NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;                    // einschalten
    NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0x04;       // Preemption priority 2
    NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0x00;              // keine sub-priority
    NVIC_Init(&NVIC_InitStruct);

    TIM_Cmd(TIM6, ENABLE);                                          // Timer starten
}

// Initialisierung d. Timers f. Ausgabe d. Zustandsvektoren
void TIM_init_clk(void)
{

RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);            // Takt fuer TIM3 ein

TIM_TimeBase_InitStruct.TIM_ClockDivision = TIM_CKD_DIV1;       // nur relevant falls ext. Takt
TIM_TimeBase_InitStruct.TIM_CounterMode = TIM_CounterMode_Up;   // Aufwaertszaehler
TIM_TimeBase_InitStruct.TIM_Period = 300-1;                     // nicht relevant; bei CC4:reset
TIM_TimeBase_InitStruct.TIM_Prescaler = 42-1;                   // Vorteiler f. 10kHz @ max=200
TIM_TimeBaseInit(TIM3, &TIM_TimeBase_InitStruct);

TIM_OC_InitStruct.TIM_OCMode = TIM_OCMode_Inactive;             // OC_Mode wird nicht verwendet
TIM_OC_InitStruct.TIM_Pulse = 24;                               // wird spaeter per SW gesetzt
TIM_OC1Init(TIM3, &TIM_OC_InitStruct);

TIM_OC_InitStruct.TIM_OCMode = TIM_OCMode_Inactive;
TIM_OC_InitStruct.TIM_Pulse = 48;
TIM_OC2Init(TIM3, &TIM_OC_InitStruct);

TIM_OC_InitStruct.TIM_OCMode = TIM_OCMode_Inactive;
TIM_OC_InitStruct.TIM_Pulse = 72;
TIM_OC3Init(TIM3, &TIM_OC_InitStruct);

TIM_OC_InitStruct.TIM_OCMode = TIM_OCMode_Inactive;
TIM_OC_InitStruct.TIM_Pulse = 200;                              // f. 10kHz; s.O.
TIM_OC4Init(TIM3, &TIM_OC_InitStruct);

// Hier wird ein interrupt beim Compare-match konfiguriert
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);

// Interrupts bei allen 4 Compare-Matches
TIM_ITConfig(TIM3, TIM_IT_CC1 | TIM_IT_CC2 | TIM_IT_CC3 | TIM_IT_CC4, ENABLE);

// Prioritaet einstellen
NVIC_InitStruct.NVIC_IRQChannel = TIM3_IRQn;                    // IRQ f. TIM3 konfigurieren
NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;                    // einschalten
NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0x02;       // Preemption priority 2
NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0x00;              // keine sub-priority
NVIC_Init(&NVIC_InitStruct);

TIM_Cmd(TIM3, ENABLE);                                          // Timer 3 starten
}

void FPU_on(void)
{
    // unterstuetzt nur 32bit float! -> Suffix f
    SCB->CPACR |= ((3UL << 10*2)|(3UL << 11*2)); /* set CP10 and CP11 Full Access */
}

// Zusammenfassung aller Initialisierungen am Programmstart
void INIT(uint32_t ADC_curr_memaddr)
{
    SystemInit();                           // Standard-system-initialisierung
    FPU_on();                               // FPU einschalten
    leds_init();                            // Pins d. LEDs konfigurieren
    taster_init();                          // Pins d. Userbutton am F4Discovery-Board
```

```
    init_I2C1();                                    // I2C-Schnittstelle konfigurieren
    i2clcd_init();                                  // I2C-LCD initialisieren
    EXTI_init_angdet();                             // externe IRQs fuer Winkelbestimmung
    ADC_DMA_init_curr(ADC_curr_memaddr);            // Analogwerte einlesen + per DMA in Array
    outputs_init();                                 // Ausgaenge zu den Treibern konfigurieren
    TIM_init_ang();                                 // Timer zur Winkelbestimmung
    TIM_init_clk();                                 // Timer f. Ausgabe d Zustandsvektoren
    TIM_init_sinPWM();                              // Timer f. Kontrollzwecke (OC1 an PD13)
    STM_Board_init();                               // Init d. Taster/LEDs am "Breakoutboard"
    SysTick_init();                                 // Systick-Timer; jede ms; hier f. Wartefkt.
    ADC2_init();
}

// Timer wird f. PWM an PD13 initialisiert (nur f. Kontrollzwecke)
void TIM_init_sinPWM(void)
{
    // TIM4 wird fuer 10kHz bei CNT: 0->99 initialisiert PD13(orange LED) an CC2
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);            // Takt fuer TIM4 ein
/*
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;                       // AlternateFunction (TIM4)
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_13;                         // fuer PD13
    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOD, &GPIO_InitStruct);
    GPIO_PinAFConfig(GPIOD, GPIO_PinSource13, GPIO_AF_TIM4);
*/
    TIM_TimeBase_InitStruct.TIM_ClockDivision = TIM_CKD_DIV1;       // nur f. ext clock rel.
    TIM_TimeBase_InitStruct.TIM_CounterMode = TIM_CounterMode_Up;   // Aufwaertszaehler
    TIM_TimeBase_InitStruct.TIM_Period = 100-1;                     // Maximalwert 100
    TIM_TimeBase_InitStruct.TIM_Prescaler = 84-1;                   // Prescaler 84 f. 10kHz
    TIM_TimeBaseInit(TIM4, &TIM_TimeBase_InitStruct);
/*
    TIM_OC_InitStruct.TIM_OCMode = TIM_OCMode_PWM1;                 // PWM
    TIM_OC_InitStruct.TIM_OCIdleState = TIM_OCIdleState_Reset;      // OC-Pin =0 im Idle
    TIM_OC_InitStruct.TIM_OCPolarity = TIM_OCPolarity_High;         // Polaritaet
    //TIM_OC_InitStruct.TIM_OCNIdleState = TIM_OCNIdleState_Set;    // -||- f. neg Ausgang
    //TIM_OC_InitStruct.TIM_OCNPolarity = TIM_OCNPolarity_High;     // -||- f. neg Ausgang
    TIM_OC_InitStruct.TIM_OutputState = TIM_OutputState_Enable;     // nicht inv. Ausgang akt
    TIM_OC_InitStruct.TIM_OutputNState = TIM_OutputNState_Disable;  // inv. Ausgang
    TIM_OC_InitStruct.TIM_Pulse = 0;                                // Pulsdauer, exemplarisch
    TIM_OC2Init(TIM4, &TIM_OC_InitStruct);
*/
    TIM_OC_InitStruct.TIM_OCMode = TIM_OCMode_Inactive;             // s.O.
    TIM_OC_InitStruct.TIM_Pulse = 80;
    TIM_OC1Init(TIM4, &TIM_OC_InitStruct);

    TIM_ITConfig(TIM4, TIM_IT_CC1, ENABLE);

    //NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);  // in "TIM_init_clk" bereits aufgerufen!

    NVIC_InitStruct.NVIC_IRQChannel = TIM4_IRQn;                    // s.O.
    NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
    NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0x06;
    NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0x00;
    NVIC_Init(&NVIC_InitStruct);

/*
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);                 // nichtmehr benoetigt!
    TIM_ITConfig(TIM3, TIM_IT_CC2, ENABLE);

    NVIC_InitStruct.NVIC_IRQChannel = TIM3_IRQn;
    NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
    NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0x0F;
    NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0x0F;
    NVIC_Init(&NVIC_InitStruct);
*/
    TIM_Cmd(TIM4, ENABLE);                                          // s.O.
}

// Initialisiert die am "Breakout-Board" als Taster u. LEDs
void STM_Board_init(void)                                           // s."LED- u. Taster-Init"
{
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_10;
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT;
    GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;
    GPIO_Init(GPIOA, &GPIO_InitStruct);

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_10 | GPIO_Pin_12;
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT;
    GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;
    GPIO_Init(GPIOC, &GPIO_InitStruct);

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15;
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN;
    GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
```

```
    GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_Init(GPIOA, &GPIO_InitStruct);

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_11;
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN;
    GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_Init(GPIOC, &GPIO_InitStruct);
}

// Init. d. Systick-Timers; jede ms; hier fuer Wartefunktion
void SysTick_init(void)
{
    RCC_ClocksTypeDef    Clocks;
    RCC_GetClocksFreq(&Clocks);
    SysTick_Config( Clocks.HCLK_Frequency/1000 - 1 );  // Resetvalue=1000Hz; + akt. Systemtimer
    //NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);  // in "TIM_init_clk" bereits aufgerufen
    NVIC_SetPriority(SysTick_IRQn, 8);                 // Prioritaet 8
}

void ADC2_init(void)
{
    // DMA2-Stream0 einschalten

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC2, ENABLE);      // Clock fuer ADC1 einschalten
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_3;                    // Pin PA1 u. PA2
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AIN;                // als analog-in
    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStruct);

    // folgender Block wurde schon in ADC_DMA... ausgefuehrt
    /*
    ADC_CommonInitStruct.ADC_Mode = ADC_Mode_Independent;
    ADC_CommonInitStruct.ADC_Prescaler = ADC_Prescaler_Div8;
    ADC_CommonInitStruct.ADC_DMAAccessMode = ADC_DMAAccessMode_Disabled;
    ADC_CommonInitStruct.ADC_TwoSamplingDelay = ADC_TwoSamplingDelay_5Cycles;
    ADC_CommonInit(&ADC_CommonInitStruct);
    */

    ADC_InitStruct.ADC_Resolution = ADC_Resolution_12b;       // s. "ADC_DMA_init"
    ADC_InitStruct.ADC_ScanConvMode = ENABLE;
    ADC_InitStruct.ADC_ContinuousConvMode = ENABLE;
    ADC_InitStruct.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_None;
    ADC_InitStruct.ADC_DataAlign = ADC_DataAlign_Right;
    ADC_InitStruct.ADC_NbrOfConversion = 1;
    ADC_Init(ADC2, &ADC_InitStruct);

    ADC_RegularChannelConfig(ADC2, ADC_Channel_3, 1, ADC_SampleTime_15Cycles);

    ADC_DMARequestAfterLastTransferCmd(ADC2, ENABLE);         // nach Wandlung DMA Request
    ADC_Cmd(ADC2, ENABLE);                                    // ADC einschalten
    ADC_SoftwareStartConv(ADC2);                              // starten (SW-triggerung)
}
```

lookuptable

Listing 8.7: lookuptable.h

```
#ifndef LOOKUPTABLE_H_
#define LOOKUPTABLE_H_

// Werte fuer "sin([2*PI*x]/300)"
static const float sinus[300]=
{
        0.000, 0.021, 0.042, 0.063, 0.084, 0.105, 0.125, 0.146, 0.167, 0.187,
        0.208, 0.228, 0.249, 0.269, 0.289, 0.309, 0.329, 0.349, 0.368, 0.388,
        0.407, 0.426, 0.445, 0.463, 0.482, 0.500, 0.518, 0.536, 0.553, 0.571,
        0.588, 0.605, 0.621, 0.637, 0.653, 0.669, 0.685, 0.700, 0.714, 0.729,
        0.743, 0.757, 0.771, 0.784, 0.797, 0.809, 0.821, 0.833, 0.844, 0.855,
        0.866, 0.876, 0.886, 0.896, 0.905, 0.914, 0.922, 0.930, 0.937, 0.944,
        0.951, 0.957, 0.963, 0.969, 0.974, 0.978, 0.982, 0.986, 0.989, 0.992,
        0.995, 0.996, 0.998, 0.999, 1.000, 1.000, 1.000, 0.999, 0.998, 0.996,
        0.995, 0.992, 0.989, 0.986, 0.982, 0.978, 0.974, 0.969, 0.963, 0.957,
        0.951, 0.944, 0.937, 0.930, 0.922, 0.914, 0.905, 0.896, 0.886, 0.876,
        0.866, 0.855, 0.844, 0.833, 0.821, 0.809, 0.797, 0.784, 0.771, 0.757,
        0.743, 0.729, 0.714, 0.700, 0.685, 0.669, 0.653, 0.637, 0.621, 0.605,
        0.588, 0.571, 0.553, 0.536, 0.518, 0.500, 0.482, 0.463, 0.445, 0.426,
        0.407, 0.388, 0.368, 0.349, 0.329, 0.309, 0.289, 0.269, 0.249, 0.228,
        0.208, 0.187, 0.167, 0.146, 0.125, 0.105, 0.084, 0.063, 0.042, 0.021,
        0.000, 0.021, 0.042, 0.063, 0.084, 0.105, 0.125, 0.146, 0.167, 0.187,
        0.208, 0.228, 0.249, 0.269, 0.289, 0.309, 0.329, 0.349, 0.368, 0.388,
        0.407, 0.426, 0.445, 0.463, 0.482, 0.500, 0.518, 0.536, 0.553, 0.571,
        0.588, 0.605, 0.621, 0.637, 0.653, 0.669, 0.685, 0.700, 0.714, 0.729,
        0.743, 0.757, 0.771, 0.784, 0.797, 0.809, 0.821, 0.833, 0.844, 0.855,
        0.866, 0.876, 0.886, 0.896, 0.905, 0.914, 0.922, 0.930, 0.937, 0.944,
        0.951, 0.957, 0.963, 0.969, 0.974, 0.978, 0.982, 0.986, 0.989, 0.992,
        0.995, 0.996, 0.998, 0.999, 1.000, 1.000, 1.000, 0.999, 0.998, 0.996,
        0.995, 0.992, 0.989, 0.986, 0.982, 0.978, 0.974, 0.969, 0.963, 0.957,
        0.951, 0.944, 0.937, 0.930, 0.922, 0.914, 0.905, 0.896, 0.886, 0.876,
        0.866, 0.855, 0.844, 0.833, 0.821, 0.809, 0.797, 0.784, 0.771, 0.757,
        0.743, 0.729, 0.714, 0.700, 0.685, 0.669, 0.653, 0.637, 0.621, 0.605,
        0.588, 0.571, 0.553, 0.536, 0.518, 0.500, 0.482, 0.463, 0.445, 0.426,
        0.407, 0.388, 0.368, 0.349, 0.329, 0.309, 0.289, 0.269, 0.249, 0.228,
        0.208, 0.187, 0.167, 0.146, 0.125, 0.105, 0.084, 0.063, 0.042, 0.021

};

// modifizierte Table, f. max moeglichen Modulationsgrad
static const float maxtable[300]=
{
        0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000,
        0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000,
        0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.036, 0.073, 0.109, 0.145,
        0.181, 0.217, 0.253, 0.289, 0.325, 0.360, 0.396, 0.431, 0.466, 0.501,
        0.535, 0.570, 0.604, 0.638, 0.671, 0.704, 0.737, 0.770, 0.802, 0.834,
        0.866, 0.897, 0.928, 0.959, 0.989, 1.018, 1.047, 1.076, 1.104, 1.132,
        1.159, 1.186, 1.212, 1.238, 1.263, 1.287, 1.311, 1.335, 1.357, 1.380,
        1.401, 1.422, 1.443, 1.462, 1.482, 1.500, 1.482, 1.462, 1.443, 1.422,
        1.401, 1.380, 1.357, 1.335, 1.311, 1.287, 1.263, 1.238, 1.212, 1.186,
        1.159, 1.132, 1.104, 1.076, 1.047, 1.018, 0.989, 0.959, 0.928, 0.897,
        0.866, 0.834, 0.802, 0.770, 0.737, 0.704, 0.671, 0.638, 0.604, 0.570,
        0.535, 0.501, 0.466, 0.431, 0.396, 0.360, 0.325, 0.289, 0.253, 0.217,
        0.181, 0.145, 0.109, 0.073, 0.036, 0.000, 0.000, 0.000, 0.000, 0.000,
        0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000,
        0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000,
        0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000,
        0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.036, 0.073, 0.109, 0.145,
        0.181, 0.217, 0.253, 0.289, 0.325, 0.360, 0.396, 0.431, 0.466, 0.501,
        0.535, 0.570, 0.604, 0.638, 0.671, 0.704, 0.737, 0.770, 0.802, 0.834,
        0.866, 0.897, 0.928, 0.959, 0.989, 1.018, 1.047, 1.076, 1.104, 1.132,
        1.159, 1.186, 1.212, 1.238, 1.263, 1.287, 1.311, 1.335, 1.357, 1.380,
        1.401, 1.422, 1.443, 1.462, 1.482, 1.500, 1.482, 1.462, 1.443, 1.422,
        1.401, 1.380, 1.357, 1.335, 1.311, 1.287, 1.263, 1.238, 1.212, 1.186,
        1.159, 1.132, 1.104, 1.076, 1.047, 1.018, 0.989, 0.959, 0.928, 0.897,
        0.866, 0.834, 0.802, 0.770, 0.737, 0.704, 0.671, 0.638, 0.604, 0.570,
        0.535, 0.501, 0.466, 0.431, 0.396, 0.360, 0.325, 0.289, 0.253, 0.217,
        0.181, 0.145, 0.109, 0.073, 0.036, 0.000, 0.000, 0.000, 0.000, 0.000,
        0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000,
        0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000,
};

// Stellungen der 6 Schalter in jedem der 9 gueltigen Zustaende
// 0:reset, 1-6:akt.Zust, 7-9:Kurzschluss
static const uint16_t zvekt [10]=
{
        //FEDCBA9876543210          Bitnummer
        0b0000000011011110, //      alle resetten
        0b0000000000001010,     //      T4/T1 -> PD1/3      Z1
        0b0000000001000100,     //      T6/T3 -> PD2/6      Z2
        0b0000000010010000,     //      T2/T5 -> PD4/7      Z3
        0b0000000001010000,     //      T2/T3 -> PD4/6      Z4
        0b0000000010000010,     //      T4/T5 -> PD1/7      Z5
        0b0000000000001100,     //      T6/T1 -> PD2/3      Z6
        0b0000000000011000,     //      T2/T1 -> PD4/3      Z7
```

```
        0b0000000001000010,     //      T4/T3 -> PD1/6      Z8
        0b0000000010000100,     //      T6/T5 -> PD2/7      Z9
};

// Zustaende in den jeweiligen abschnitten zustand[1.-4.zustand][abschnitt1-6]
// 2.Zustand in abschnitt 3: zustand[2][3]
// 1.Zeile/Spalte = dummy,
// damit Zahlen trotz indexierung ab 0 direkt lesbar verwendet werden koennen

static const uint16_t zustaende_pos [5][7]=
{
        {0,0,0,0,0,0,0},
        {0,4,4,4,1,1,1},
        {0,2,2,5,5,5,2},
        {0,6,3,3,3,6,6},
        {0,7,8,9,7,8,9},
};

static const uint16_t zustaende_neg [5][7]=
{
        {0,0,0,0,0,0,0},
        {0,1,1,1,4,4,4},
        {0,5,5,2,2,2,5},
        {0,3,6,6,6,3,3},
        {0,7,8,9,7,8,9},
};

#endif /* LOOKUPTABLE_H_ */
```

main

Listing 8.8: main.h

```
#ifndef MAIN_H_
#define MAIN_H_

/* Includes */
// STM32
#include <stm32f4xx.h>                          // allg Def f STM32F4xx
#include <stm32f4xx_conf.h>                     // config
#include <stm32f4xx_i2c.h>                      // f I2C Schnittstelle
#include <stm32f4xx_gpio.h>                     // f GPIO Ports
#include <stm32f4xx_rcc.h>                      // init d Clock-trees
#include <stm32f4xx_tim.h>                      // Timer des STM32F4
#include <stm32f4xx_adc.h>                      // Analog to Digital Converter
#include <stm32f4xx_dma.h>                      // Direct Memory Access
#include <stm32f4xx_exti.h>                     // External Interrupts
#include <stm32f4xx_syscfg.h>                   // sytemkonfig
// C
#include <math.h>                               // mathematische Funktionen
#include <stdlib.h>                             // Standardfunktionen
#include <misc.h>                               // NVIC
#include <stdint.h>                             // Standard Integer Types
#include <string.h>                             // Funktionen f Strings
#include <float.h>                              // Floating-Point-Operationen
// eigene
#include "functions.h"                          // diverse Funktionen
#include "i2clcd.h"                             // ansteuerung d LCDs
#include "inits.h"                              // saemtliche Initialisierungen
#include "lookuptable.h"                        // LUTs, Zustandsvektoren, Zuordnungen

// Defines/Macros
#define disc_button GPIOA->IDR & 0x0001         // "Kurzzugriff" um "Userbutton" abzufragen
#define F_HSE    8000000L                       // Freq ext Osc
#define F_CPU 168000000L                        // Frequenz der CPU

#define button_1 (GPIOA->IDR & GPIO_Pin_15)     // Taster 1 am STM_Board
#define button_2 (GPIOA->IDR & GPIO_Pin_14)     // Taster 2
#define button_3 (GPIOA->IDR & GPIO_Pin_13)     // Taster 3
#define button_4 (GPIOC->IDR & GPIO_Pin_11)     // Taster 4


// Init-Struct-Typedefs
GPIO_InitTypeDef GPIO_InitStruct;
TIM_TimeBaseInitTypeDef TIM_TimeBase_InitStruct;
TIM_OCInitTypeDef TIM_OC_InitStruct;
NVIC_InitTypeDef NVIC_InitStruct;
ADC_InitTypeDef       ADC_InitStruct;
ADC_CommonInitTypeDef ADC_CommonInitStruct;
DMA_InitTypeDef DMA_InitStruct;
EXTI_InitTypeDef EXTI_InitStruct;

void wait_ms (uint16_t);                // Wartezeit (Param: t in ms)
void I_offset_abgleich (void);          // Stromsensor am Programmanfang nullen

#endif
```

Listing 8.9: main.c

```
#include "main.h"

// ---------- Volatile-Variablen ----------
volatile uint32_t SysTickCnt=0;                 // fuer Wartezeit-Funktion benoetigt
volatile uint16_t comparewerte[4]={0,0,0,0};    // Comparewerte f. Umschaltzeitpunkte
volatile uint16_t ADC_curr[2]={0x0000, 0x0000}; // enthaelt soll u. Istwert v. Izk
volatile int32_t TIM6_CNT_last=0;               // Wert v. TIM6 zu Beginn voriger EXTI
volatile int32_t TIM6_CNT_diff=0;               // zul. gemessener Abstand zw. 2 EXTI
volatile int32_t TIM6_CNT_diff_alt=0;           // voriger. gem. Abstand zw. ...
volatile int16_t winkel_OK_CNT=0;               // increment bei jedem korrekten EXTI
volatile uint16_t winkel_OK=0;                  // Indikator ob Netz korrekt erkannt
volatile uint16_t I_offset=0;                   // speichert Offset d. Stromsensors
volatile uint16_t PWRON=0;                      // "Ein- Aus- Schalter"
volatile uint16_t dc_fixed=0;                   // "Schalter" f. fixes Tastverhaeltnis
volatile uint16_t abschnitt=0;                  // Akt. Abschnitt d Netzspannung
volatile uint16_t sin_Nmax=1;                   // Schalter f. modifizierte LUT
volatile uint16_t winkelversch=0;               // schalter f. Blindleistungserz.
volatile uint16_t netztrigger;                  // zum synchronisieren d Sollwertspr.
volatile float winkel=0.0;                      // Phasenwinkel d. Netzspannung
volatile float I_soll_reg;                      // Sollwert f. Stromregelung (in mA)
volatile float I_ist_reg;                       // Strom-Istwert (mA)
volatile uint16_t I_soll_2=0;                   // Schalter f. Sollwert v. P3
volatile uint16_t lock_exti_4=0;                // Verriegelung v. exti4
volatile uint16_t lock_exti_2=0;                // Verriegelung v. exti2
volatile uint16_t lock_exti_3=0;                // Verriegelung v. exti3

// ---------- Defines ----------
#define tastensperrzeit  40             // Verzoegerung nach tastendruck
```

```
#define winkel_OK_lim     100             // Anzahl d. korr. exti bis winkel OK
#define faktor_Sollwert   3.6621          // zur Umrechnung d Sollwertes
#define faktor_Istwert    9.765           // zur Umrechnung d Istwertes
#define ZweiPIdiv3        2.0944          // Konstante
#define winkel_diff_max   5000            // max. Differenz bei exti-Zeiten
#define sizeLUT           300             // Groesse der LookUpTable
#define mittelwert_mult   3               // Faktor zur mittelwertbildung des gem. Stromes
#define I_soll_2_dauer    60              // dauer der Sollwertspruenge (in ms)
// -----berechnete Defines ------
#define sizeLUTdiv3       sizeLUT/3
#define mittelwert_div    mittelwert_mult+1

// Reglerparameter/Variablen
volatile float err=0;
volatile float erralt=0;
volatile float errsum=0;
volatile float dc=0;
// fuer kleine (Netz-)Spannungen (<70V verk.): KP 2 KI 5
#define KP            0.3
#define KI            0.5
#define windup_max    20000     // TD*KI*windup_max=1 ( windup_max = 1 / ( KI * TD ) )
#define TD            0.0001
// bei "MAX-Modulation" darf der jew. Comparewert um Faktor "2/(sqrt3)" groesser sein
#define comp_MULT_sin 100       // Faktor f. skal. damit bei dc=1 Dauer d. Nullzustandes =0
#define comp_MULT_max 115       // -||- bei alternativer LUT f. verbesserte Aussteuerbarkeit


int main(void){                 // ------------ Hauptprogramm --------------------

char lcdout[17];                // Array f. Ausgaben an das LCD
uint16_t tmp=0;                 // temp va. f. Berechnungen f. LCD-Augabe
uint16_t buttonlock=0;          // Tasten freigeben

INIT((uint32_t)&ADC_curr);      // Initialisierungen ausfuehren
show_clocks();                  // Systemtakte anzeigen
wait_ms(600);
i2clcd_setcur(1,1);             // "Logo" anzeigen
i2clcd_string("␣WWR-CSI␣␣␣␣␣␣␣␣");
i2clcd_setcur(2,1);
i2clcd_string("11-2011␣␣04-2013");

I_offset_abgleich();            // Stromsensor-offset

while (1) {                     // ENDLOSSCHLEIFE

// Ueberpruefung d. korrekten winkels + LEDs (1/2) schalten
if ( winkel_OK_CNT >= winkel_OK_lim ) {  // wenn schwellwert ueberschritten: OK
    board_leds(2,0);                     // rote LED aus
    board_leds(1,1);                     // gruene LED ein
    winkel_OK=1; }                       // Indikator setzen
else {                                   // andernfalls:
    board_leds(2,1);                     // rote LED ein
    board_leds(1,0);                     // gruene LED aus
    winkel_OK=0; }                       // Indikator ruecksetzen
if (winkel_OK!=1) {PWRON=0;}             // falls nicht OK: ausschalten
board_leds(4,PWRON);                     // PWR-LED entsprechend Zustand schalten

// Tastenabfragen anfang
if(!buttonlock){
    if (!button_1) {PWRON^=1;}           // ON/OFF
    if (disc_button) {                   // Modus umschalten (fixer dc / Regelung)
        dc_fixed^=1;
        PWRON=0; }                       // bei Modusumschaltung: OFF
    if (!button_3) {                     // Sprung im Sollwert
        netztrigger=0;                   // Netztrigger ruecksetzen
        while(netztrigger==0);           // auf Trigger warten; ev. Zeitbegr!?
        I_soll_2^=1;                     // auf anderen Sollwert umschalten
        board_leds(3,2);                 // Signallampe ein
        wait_ms(I_soll_2_dauer);         // vorgegebene Zeit warten
        I_soll_2^=1;                     // Stromsollwert wieder zurueckschalten
        board_leds(3,2);}                // Signallampe aus
    /*
    if(!button_3){                       // Umschalten auf lternativen Stromsollwert
        I_soll_2^=1; }
    board_leds(3,I_soll_2);
    */
    if (!button_4) {sin_Nmax^=1;}        // Modulation umschalten (sin / max)
    /*
    if(disc_button){                     // Nullabgleich des Stromsensors
        tmp=PWRON;                       // akt. Zustand (ON/OFF) speichern
        PWRON=0;                         // ausschalten
        disc_leds('C',1);                // signallampe
        wait_ms(1000);                   // warten bis Strom=0
        I_offset_abgleich();             // Abgleich-Fkt. aufrufen
        PWRON=tmp; }                     // vorigen Zustand wiederherstellen
    */
    if (!button_2) {winkelversch^=1;}    // Phasenverschiebung aktivieren
    buttonlock=tastensperrzeit;          // Tastensperre setzen
}                                        // Tastenabfragen ende

// Strom Soll- und Ist-werte ausgeben
i2clcd_setcur(1,1);
```

```
i2clcd_string("I:");
int2strST(lcdout, (int32_t)I_ist_reg/1000, 2);
i2clcd_string(lcdout);
i2clcd_string(".");
int2strST(lcdout, ((int32_t)I_ist_reg%1000)/10, 2);
i2clcd_string(lcdout);
i2clcd_string("/");
int2strST(lcdout, (int32_t)I_soll_reg/1000, 2);
i2clcd_string(lcdout);
i2clcd_string(".");
int2strST(lcdout, ((int32_t)I_soll_reg%1000)/10, 2);
i2clcd_string(lcdout);
// Anzeigen ob Blindleistungsgenerierung eingeschaltet
if (winkelversch)      { i2clcd_string("␣\25"); }
else                   { i2clcd_string("␣␣"); }
// Anzeigen ob erweiterte Aussteuerbarkeit aktiviert
if (sin_Nmax)          { i2clcd_string("S"); }
else                   { i2clcd_string("\27"); }
/*
// errsum ausgeben
i2clcd_setcur(2,1);
if (errsum<0) {
     i2clcd_string("esum:-");
     int2strST(lcdout, errsum, 6);}
else {
     i2clcd_string("esum: ");
     int2strST(lcdout, errsum, 6);}
i2clcd_string(lcdout);
i2clcd_string("     ");
*/
// Frequenz (*10) berechnen + ausgeben ( +0.5 bewirkt "Rundung")
tmp=(uint16_t)( (float)(TIM6_clk*10) / (TIM_ang_presc_x3*TIM6_CNT_diff) ) + 0.5;
i2clcd_setcur(2,1);
i2clcd_string("f:");
if ( (tmp>=700)||(tmp<=400) ) {        // Fehler falls Frequenz zu hoch / niedrig
     i2clcd_string("Err!"); }          // "Fehler-String" ausgeben
else {
     int2strST(lcdout, tmp/10, 2);     // Zehner- und Einer-Stelle
     i2clcd_string(lcdout);            // ausgeben
     i2clcd_string(".");               // Dezimalpunkt
     int2strST(lcdout, tmp%10, 1);     // Kommastelle
     i2clcd_string(lcdout); }          // ausgeben
i2clcd_string("␣");                    // Leerzeichen
// Tastverhaeltnis ausgeben
if (dc<0) {                            // Unterscheidung ob groesser od. kleiner 0
     i2clcd_string("d:-.");            // falls kleiner 0 : Vorzeichen ausgeben
     int2strST(lcdout, (uint16_t)(dc*-100), 2); }
else {
     i2clcd_string("d:0.");
     int2strST(lcdout, (uint16_t)(dc*100), 2); }
i2clcd_string(lcdout);
i2clcd_string("␣␣␣␣");
// Betriebsmodus anzeigen (Regelung/fixer dc)
i2clcd_setcur(2,16);
if (dc_fixed) { i2clcd_string("d"); }  // fixes Tastverhaeltnis
else          { i2clcd_string("I"); }  // Stromregelung aktiv
/*
// Comparewerte ausgeben
akt_comparewerte[1]=comparewerte[1];
akt_comparewerte[2]=comparewerte[2];
akt_comparewerte[3]=comparewerte[3];
i2clcd_setcur(2,1);
i2clcd_string(" ");
int2strST(lcdout, akt_comparewerte[1], 4);
i2clcd_string(lcdout);
i2clcd_string(" ");
int2strST(lcdout, akt_comparewerte[2], 4);
i2clcd_string(lcdout);
i2clcd_string(" ");
int2strST(lcdout, akt_comparewerte[3], 4);
i2clcd_string(lcdout);
i2clcd_string(" ");
*/
wait_ms(25);                           // kurze Wartezeit
if (buttonlock) {buttonlock--;}        // decrementieren bis 0
    }                                  // ENDLOSSCHLEIFE ende
}                                      // Hauptprogramm ende

// ------------------------------Interrupt Handler Timer 3 -------------------------
void TIM3_IRQHandler(void){                    // Ausgabe der Zustandsvektoren
GPIOD->BSRRL = GPIO_Pin_12;                    // fuer Zeitmessung

if( (TIM3->SR) & (TIM_IT_CC1) ) {              // wenn 1. Compare-Match: 2. Zustand setzen
   TIM_ClearITPendingBit(TIM3, TIM_IT_CC1);    // InterruptPendingBit fuer CC1 ruecksetzen
   if(PWRON){                                  // wenn "ON": Zustandsvektoren ausgeben
      if (dc>=0) {
          GPIOD->BSRRH = (zvekt[0] & ~(zvekt[zustaende_pos[2][abschnitt]]) );
          GPIOD->BSRRL =  zvekt[zustaende_pos[2][abschnitt]]; }
      else {
          GPIOD->BSRRH = (zvekt[0] & ~(zvekt[zustaende_neg[2][abschnitt]]) );
          GPIOD->BSRRL =  zvekt[zustaende_neg[2][abschnitt]]; } }
   else { GPIOD->BSRRH =  zvekt[0]; } }        // ansonsten: Nullvektor
```

```
else{
   if((TIM3->SR) & (TIM_IT_CC2)){              // CC2 analog zu CC1
      TIM_ClearITPendingBit(TIM3, TIM_IT_CC2);
      if(PWRON){
         if (dc>=0) {
            GPIOD->BSRRH = (zvekt[0] & ~(zvekt[zustaende_pos[3][abschnitt]]) );
            GPIOD->BSRRL =  zvekt[zustaende_pos[3][abschnitt]];  }
         else {
            GPIOD->BSRRH = (zvekt[0] & ~(zvekt[zustaende_neg[3][abschnitt]]) );
            GPIOD->BSRRL =  zvekt[zustaende_neg[3][abschnitt]];  } }
      else { GPIOD->BSRRH =  zvekt[0]; } }

   else{
      if((TIM3->SR) & (TIM_IT_CC3)){           // CC3 analog zu CC1
         TIM_ClearITPendingBit(TIM3, TIM_IT_CC3);
         if(PWRON){
            if (dc>=0) {
               GPIOD->BSRRH = (zvekt[0] & ~(zvekt[zustaende_pos[4][abschnitt]]) );
               GPIOD->BSRRL =  zvekt[zustaende_pos[4][abschnitt]];  }
            else {
               GPIOD->BSRRH = (zvekt[0] & ~(zvekt[zustaende_neg[4][abschnitt]]) );
               GPIOD->BSRRL =  zvekt[zustaende_neg[4][abschnitt]];  } }
         else { GPIOD->BSRRH =  zvekt[0]; } }

      else{ // keine Abfrage mehr noetig!     // CC4 analog zu CC1
         TIM_ClearITPendingBit(TIM3, TIM_IT_CC4);
         if(PWRON){
            if (dc>=0) {
               GPIOD->BSRRH = (zvekt[0] & ~(zvekt[zustaende_pos[1][abschnitt]]) );
               GPIOD->BSRRL =  zvekt[zustaende_pos[1][abschnitt]];  }
            else {
               GPIOD->BSRRH = (zvekt[0] & ~(zvekt[zustaende_neg[1][abschnitt]]) );
               GPIOD->BSRRL =  zvekt[zustaende_neg[1][abschnitt]];  } }
         else { GPIOD->BSRRH =  zvekt[0]; }
        // bei CC4 zusaetzlich:
        // neue Compare-Werte setzen
        TIM_SetCompare1(TIM3, comparewerte[1]);
        TIM_SetCompare2(TIM3, comparewerte[2]);
        TIM_SetCompare3(TIM3, comparewerte[3]);
        // Timer3/4 ruecksetzen
        TIM_SetCounter(TIM3, 0);
        TIM_SetCounter(TIM4, 0); } } }

GPIOD->BSRRH = GPIO_Pin_12; }

// ------------------------------Interrupt Handler EXTI 4 ------------------------
void EXTI4_IRQHandler(void){                          // Nulldurchgang von L1
EXTI_ClearITPendingBit(EXTI_Line4);                   // reset Int.Pend.Bit
int32_t tmp;                                          // Zwischenspeicher
int32_t abweichung;                                   // Abweichung v d letzen Dauer
if (!(lock_exti_4)) {                                 // Verriegelung gesetzt?
   if (!(GPIOE->IDR & GPIO_Pin_4)) {                  // pruefen ob Pin wirklich LOW
      lock_exti_4 = 1;                                // verriegeln
      disc_leds('O',2);                               // LED-toggeln->Diagnosezwecke
      lock_exti_2 = 0;                                // Verriegelung f. naechst AUS
      netztrigger = 1;                                // f synchr d Sollwertspruenge
      tmp = TIM_GetCounter(TIM6);                     // Timerwert auslesen
      TIM6_CNT_diff_alt = TIM6_CNT_diff;              // alte Differenz speichern
      TIM6_CNT_diff = tmp-TIM6_CNT_last;              // neuen Differenz bilden
      abweichung = TIM6_CNT_diff - TIM6_CNT_diff_alt; // Abweichung d Differenzen
      if ( abs(abweichung) > winkel_diff_max ) {      // Abweichung zu gross
         winkel_OK_CNT = 0; }                         // Winkelcounter nullsetzen
      else {                                          // Abweichung OK
         if ( winkel_OK_CNT < winkel_OK_lim*2 ) {     // Winkelcounter erhoehen falls
            winkel_OK_CNT++; } }                      // Schwelle noch nicht erreicht
      TIM6_CNT_last = 0;                              // Setzwert speichern
      TIM_SetCounter(TIM6, 0); } } }                  // Timer setzen

// ------------------------------Interrupt Handler EXTI 2 ------------------------
void EXTI2_IRQHandler(void){                          // s. EXTI4
EXTI_ClearITPendingBit(EXTI_Line2);
int32_t tmp;
int32_t abweichung;
if (!(lock_exti_2)) {
   if (!(GPIOE->IDR & GPIO_Pin_2)) {
      lock_exti_2 = 1;
      disc_leds('O',2);
      lock_exti_3 = 0;
      tmp = TIM_GetCounter(TIM6);
      TIM6_CNT_diff_alt = TIM6_CNT_diff;
      TIM6_CNT_diff = tmp-TIM6_CNT_last;
      abweichung = TIM6_CNT_diff-TIM6_CNT_diff_alt;
      if ( abs(abweichung) > winkel_diff_max ) {
         winkel_OK_CNT = 0; }
      else {
         if ( winkel_OK_CNT < winkel_OK_lim*2 ) {
            winkel_OK_CNT++; } }
      //TIM6_CNT_last = tmp;
      TIM6_CNT_last = TIM6_CNT_diff;                  // geaendert 2013-04-16
      TIM_SetCounter(TIM6, TIM6_CNT_diff); } } }
```

```
// ------------------------------Interrupt Handler EXTI 3 -------------------------
void EXTI3_IRQHandler(void){                                    // s. EXTI4
EXTI_ClearITPendingBit(EXTI_Line3);
int32_t tmp;
int32_t abweichung;
if(!(lock_exti_3)){
   if(!(GPIOE->IDR & GPIO_Pin_3)){
      lock_exti_3 = 1;
      disc_leds('O',2);
      lock_exti_4 = 0;
      tmp = TIM_GetCounter(TIM6);
      TIM6_CNT_diff_alt = TIM6_CNT_diff;
      TIM6_CNT_diff = tmp-TIM6_CNT_last;
      abweichung = TIM6_CNT_diff-TIM6_CNT_diff_alt;
      if ( abs(abweichung) > winkel_diff_max ) {
         if ( winkel_OK_CNT > 0 ) {
            winkel_OK_CNT = 0;
            board_leds(2,1); } }
      else {
         if ( winkel_OK_CNT < winkel_OK_lim*2 ) {
            winkel_OK_CNT++; } }
      //TIM6_CNT_last = tmp;
      TIM6_CNT_last = TIM6_CNT_diff << 1;                       // geaendert 2013-04-16
      TIM_SetCounter(TIM6,(TIM6_CNT_diff << 1)); } } }

// ------------------------------Unterprogramm f Wartezeit -------------------------
void wait_ms (uint16_t t_in_ms){
  uint32_t temp;                                    // temporaere Variable
     temp = SysTickCnt;                             // Zaehlerstand zwischenspeichern
  while((SysTickCnt - temp ) <= t_in_ms );}         // warten bis zeit abgelaufen
// steht hier wegen volatile Variablen
// ein einzelner Ueberlauf muss hier nicht beruecksichtigt werden! -> Zweierkomplement
// ein Durchlauf dauert (24 bit/1ms) ca. 4h40min => normal kein Problem

// ------------------------------Interrupt Handler Systick-Timer -------------------------
void SysTick_Handler(void){
  SysTickCnt++;}                                    // Increment d Zaehlervariable

// ------------------------------Unterprogramm f Offsetabgleich -------------------------
void I_offset_abgleich (void){
int32_t i=0;                                        // Zaehlervariable (=0)
disc_leds('C',1);                                   // Signal-LED ein
for (i=0;i<30;i++){                                 // schleife 30x ausfuehren
    I_offset+=ADC_curr[1]; wait_ms(50); }           // Messwert einlesen, aufsummieren, warten
I_offset/=30;                                       // MIttelwert bilden
disc_leds('C',0);}                                  // Signal-LED aus

// ------------------------------Interrupt Handler Timer 4 -------------------------
void TIM4_IRQHandler(void){                         // Regler-Routine
TIM_ClearITPendingBit(TIM4, TIM_IT_CC1);            // reset Pentding-Bit
GPIOD->BSRRL = GPIO_Pin_14;                         // f. Zeitmessung
float sinA, sinB, sinC;                             // Zwischenspeicher f. Werte aus LUT
int32_t tmp;                                        // temporaere Variable

if (I_soll_2) {                                     // alternativen Stromsollwert einlesen
    I_soll_reg = (float)ADC_GetConversionValue(ADC2) * (float)faktor_Sollwert; }
else {                                              // regulaeren Stromsollwert einlesen
    I_soll_reg = (float)ADC_curr[0]                     * (float)faktor_Sollwert; }

tmp = (int32_t)ADC_curr[1];                         // Strom-Istwert einlesen
if (tmp > I_offset) {                               // wenn Strom > 0: berechnen
    tmp = (int32_t)( (float)(tmp-I_offset) * (float)faktor_Istwert );  }
else { tmp = 0; }                                   // ansonsten: =0!
I_ist_reg = ( (float)( tmp + ((mittelwert_mult)*I_ist_reg) )
            / (float)(mittelwert_div) );            // exp. Glaettung

if (dc_fixed) {                                     // wenn: konst-dc mode: berechnen+setzen
    dc = (float)(ADC_curr[0]-2048) / 2048; }
else {                                              // sonst:  Regelalgorithmus
    err =( I_soll_reg- I_ist_reg ) / 1000;          // Abweichung
    errsum += err;                                  // aufsummieren
    if ( errsum >    windup_max  ) { errsum =  windup_max; }   // Anti-WindUp
    if ( errsum < (-windup_max) ) { errsum = -windup_max; }   // Anti-WindUp
    dc=( ( KP * err ) + ( KI * TD * errsum ) );               // dc berechnen
    erralt=err;                                     // Fehlersumme speichern
    if ( !PWRON ) { dc=0; errsum=0; } }             // nullsetzen falls Ausgeschaltet
if ( dc >  0.98 ) { dc= 0.98; }                     // Stellgroessenbeschraenkung
if ( dc < -0.98 ) { dc=-0.98; }                     // Stellgroessenbeschraenkung

winkel = (float)( ( (float)TIM_GetCounter(TIM6)     // akt Winkel berechnen
                  * (float)sizeLUTdiv3) / (float)TIM6_CNT_diff );

// Funktion zum Verschieben d. Netzstromes = Blindleistungserzeugung
if (winkelversch) { winkel += 70; }                 // sizeLUT=300 => 70=84deg;

if (winkel>sizeLUT) {winkel-=sizeLUT;}              // = modulo LUT-Groesse

if          (winkel > (3*sizeLUT/6) ) {             // best. d. derz. Abschnittes v. Unetz
   if       (winkel > (5*sizeLUT/6) )   { abschnitt=6; }
   else {
       if   (winkel < (4*sizeLUT/6) )   { abschnitt=4; }
       else                             { abschnitt=5; } } }
```

```
else {
   if       (winkel >  (2*sizeLUT/6) )  { abschnitt=3; }
   else {
        if   (winkel < (1*sizeLUT/6) )   { abschnitt=1; }
        else                             { abschnitt=2; } } }

// schaltet waehrend bestimmter Abschnitte eine LED ein; nur zu Diagnosezwecken
if((abschnitt==1)||(abschnitt==4)||(abschnitt==5))     {disc_leds('B',1);}
else                                                  {disc_leds('B',0);}

// Comparewerte berechnen
if (sin_Nmax) {                                                          // normale Modulation
  sinA=sinus[ (int32_t)  winkel                        % (sizeLUT-1) ];    // %-Operanden = INT!
  sinB=sinus[ (int32_t)( winkel + (2*sizeLUTdiv3) ) % (sizeLUT-1) ];    // +240deg, = -120deg
  sinC=sinus[ (int32_t)( winkel +    sizeLUTdiv3 )  % (sizeLUT-1) ];    // +120deg, = -240deg
  comparewerte[1] =                  (uint16_t)((float)comp_MULT_sin * sinA * fabsf(dc));
  comparewerte[2] = comparewerte[1] + (uint16_t)((float)comp_MULT_sin * sinB * fabsf(dc));
  comparewerte[3] = comparewerte[2] + (uint16_t)((float)comp_MULT_sin * sinC * fabsf(dc)); }

else {                                                                   // opt. Modulation
  sinA=maxtable[ (int32_t)  winkel                      % (sizeLUT-1) ];// s. oben.
  sinB=maxtable[ (int32_t)( winkel + (2*sizeLUTdiv3) ) % (sizeLUT-1) ];
  sinC=maxtable[ (int32_t)( winkel +    sizeLUTdiv3  ) % (sizeLUT-1) ];
  comparewerte[1] =                  (uint16_t)((float)comp_MULT_max * sinA * fabsf(dc));
  comparewerte[2] = comparewerte[1] + (uint16_t)((float)comp_MULT_max * sinB * fabsf(dc));
  comparewerte[3] = comparewerte[2] + (uint16_t)((float)comp_MULT_max * sinC * fabsf(dc)); }

GPIOD->BSRRH = GPIO_Pin_14;}

// ------------------------------Interrupt Handler Timer 6 -------------------------
void TIM6_DAC_IRQHandler(void){
    TIM_ClearITPendingBit(TIM6, TIM_IT_Update);
    winkel_OK_CNT=0;}
// wird nur im Falle eines Overflows aufgerufen: Fehler in Netzspannung liegt vor!
```

8.4 weitere Blöcke des Simulink-Modells

B6-Brücke

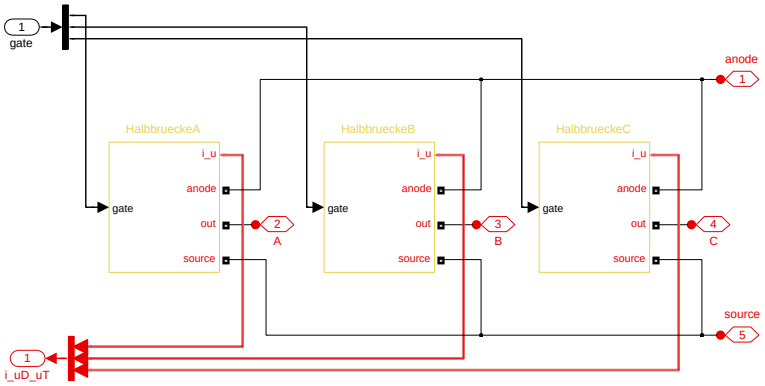


Abbildung 8.27: Simulinkmodell der B6-Brücke

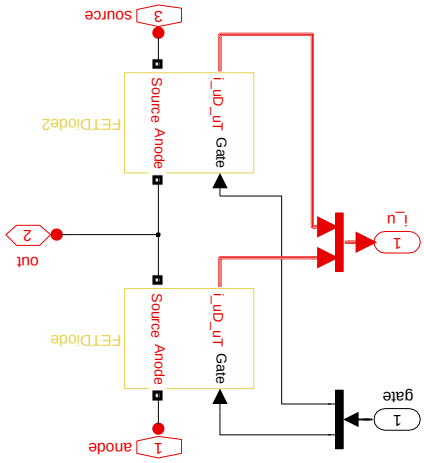


Abbildung 8.28: Simulinkmodell einer Halbbrücke

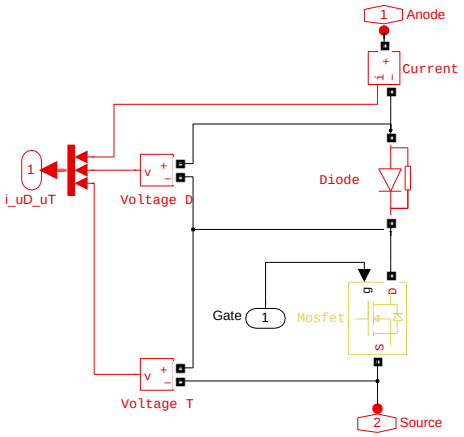


Abbildung 8.29: Simulinkmodell eines Schaltelements (Transistor, Diode)

Winkelregelung

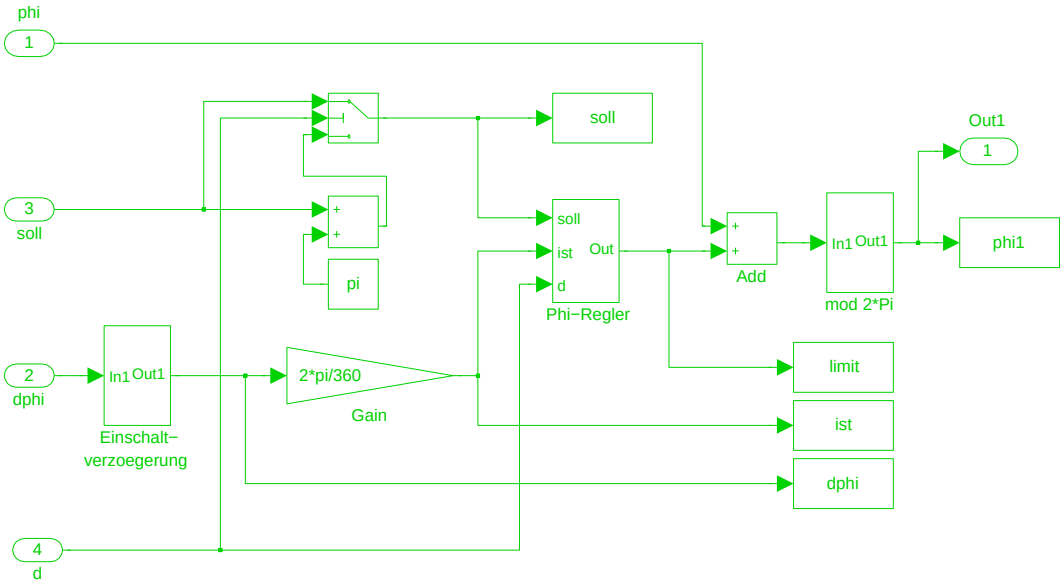


Abbildung 8.30: Regelung der Phasenverschiebung des Netzstromes

Verriegelungslogik

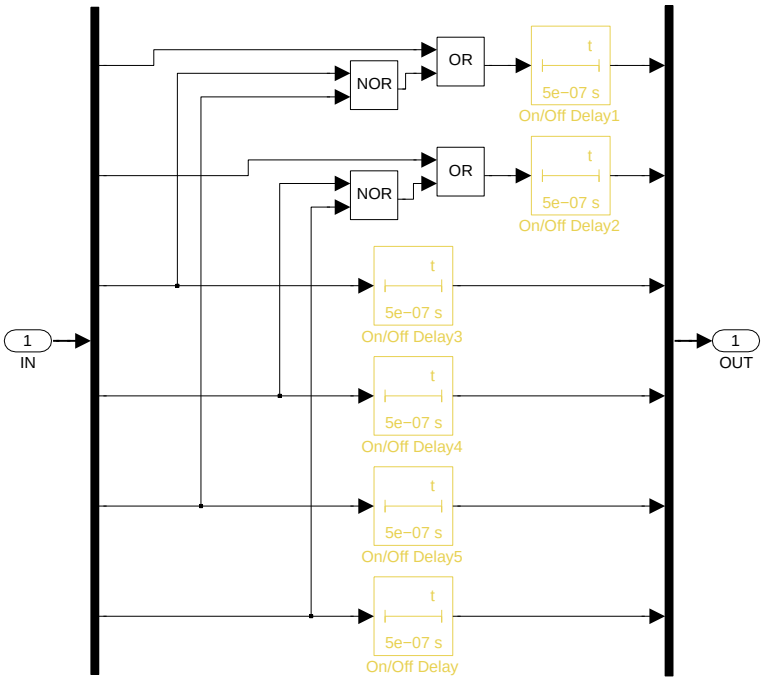
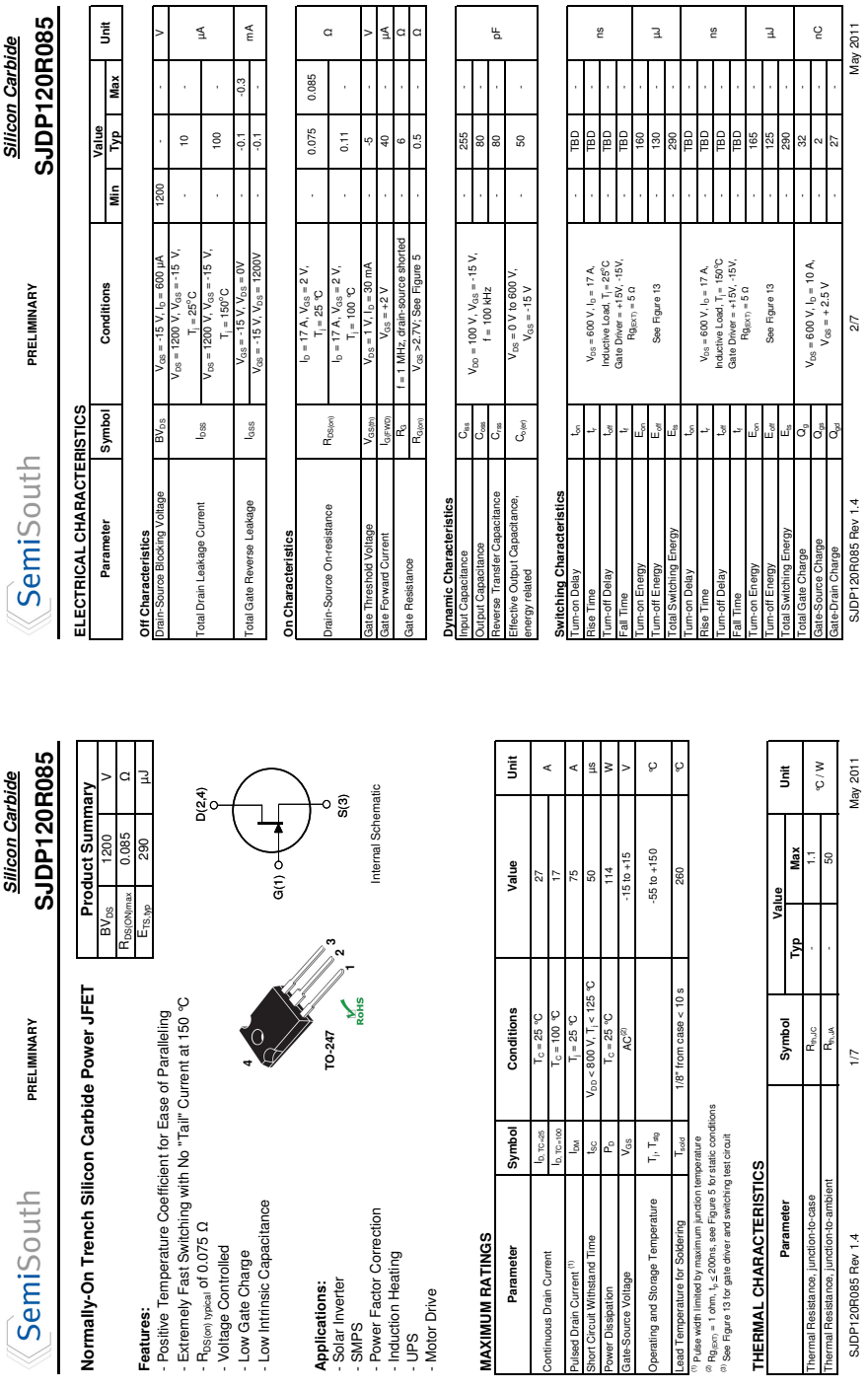


Abbildung 8.31: Modell der Verriegelungslogik

8.5 Ausschnitte der Datenblätter



SemiSouth — Silicon Carbide — PRELIMINARY — SJDP120R085

Normally-On Trench Silicon Carbide Power JFET

Product Summary		
BV_{DS}	1200	V
$R_{DS(ON)max}$	0.085	Ω
$E_{TS,typ}$	290	µJ

Features:
- Positive Temperature Coefficient for Ease of Paralleling
- Extremely Fast Switching with No "Tail" Current at 150 °C
- $R_{DS(on)}$ typical of 0.075 Ω
- Voltage Controlled
- Low Gate Charge
- Low Intrinsic Capacitance

Applications:
- Solar Inverter
- SMPS
- Power Factor Correction
- Induction Heating
- UPS
- Motor Drive

TO-247 — RoHS

Internal Schematic

MAXIMUM RATINGS

Parameter	Symbol	Conditions	Value	Unit
Continuous Drain Current	$I_{D, TC=25}$	T_C = 25 °C	27	A
	$I_{D, TC=100}$	T_C = 100 °C	17	
Pulsed Drain Current[1]	I_{DM}	T_C = 25 °C	75	A
Short Circuit Withstand Time	t_{SC}	V_{DD} < 800 V, T_J < 125 °C	50	µs
Power Dissipation	P_D	T_C = 25 °C	114	W
Gate-Source Voltage	V_{GS}	AC[2]	-15 to +15	V
Operating and Storage Temperature	T_J, T_{stg}		-55 to +150	°C
Lead Temperature for Soldering	T_{sold}	1/8" from case < 10 s	260	°C

[1] Pulse width limited by maximum junction temperature
[2] $R_{G(ext)}$ = 1 ohm, $t_p \leq$ 200ns, see Figure 5 for static conditions
[3] See Figure 13 for gate driver and switching test circuit

THERMAL CHARACTERISTICS

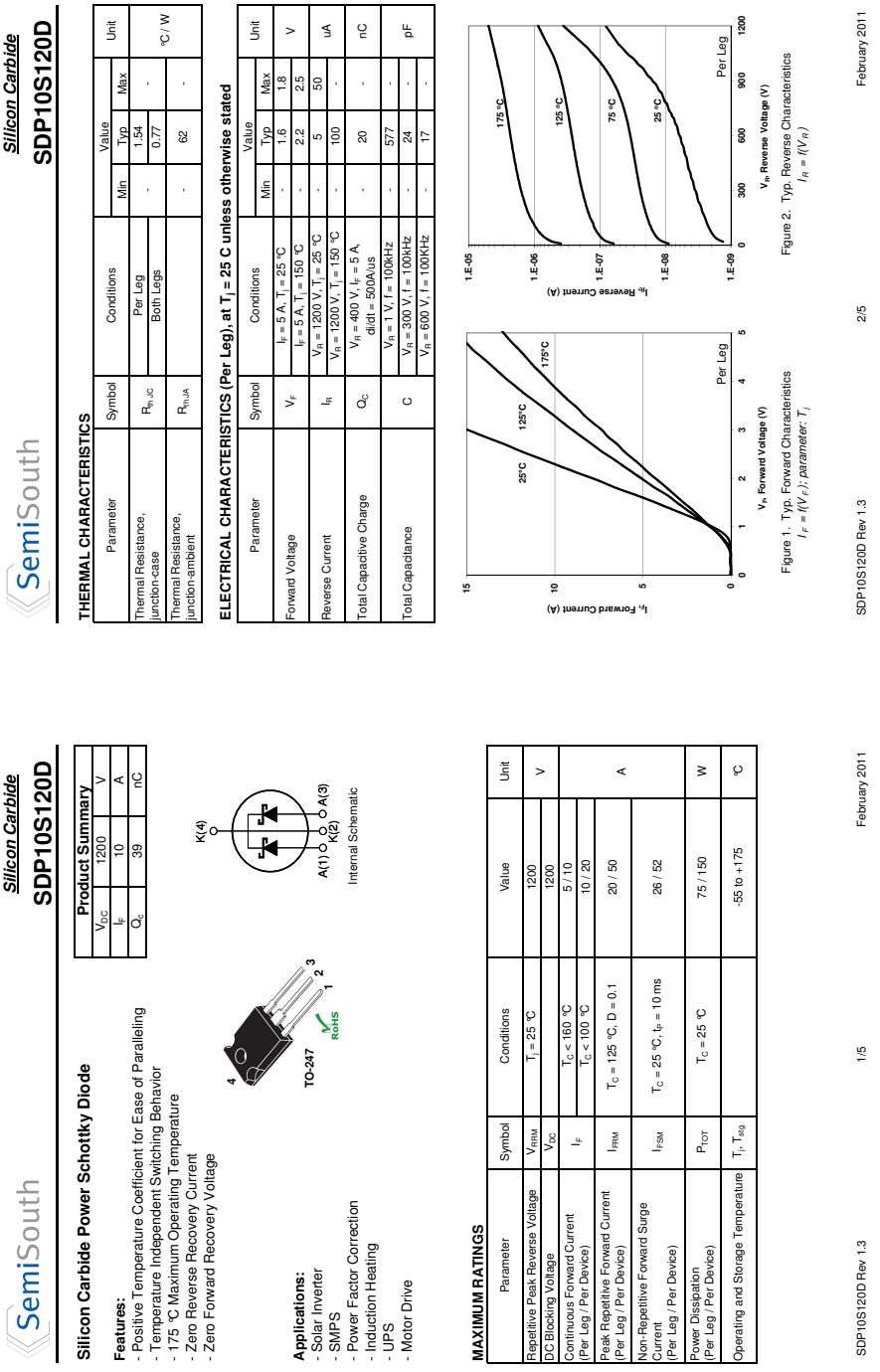
Parameter	Symbol	Value			Unit
		Min	Typ	Max	
Thermal Resistance, junction-to-case	$R_{th,JC}$		-	1.1	°C / W
Thermal Resistance, junction-to-ambient	$R_{th,JA}$		-	50	

SemiSouth — Silicon Carbide — PRELIMINARY — SJDP120R085

ELECTRICAL CHARACTERISTICS

Parameter	Symbol	Conditions	Value			Unit
			Min	Typ	Max	
Off Characteristics						
Drain-Source Blocking Voltage	BV_{DS}	V_{GS} = -15 V, I_D = 600 µA	1200	-	-	V
Total Drain Leakage Current	I_{DSS}	V_{GS} = 1200 V, V_{GS} = -15 V, T_J = 25°C	-	10	-	µA
		V_{DS} = 1200 V, V_{GS} = -15 V, T_J = 150°C	-	100	-	
Total Gate Reverse Leakage	I_{GSS}	V_{GS} = -15 V, V_{DS} = 0V	-	-0.1	-0.3	mA
		V_{GS} = -15 V, V_{DS} = 1200V	-	-0.1	-	
On Characteristics						
Drain-Source On-resistance	$R_{DS(on)}$	I_D = 17 A, V_{GS} = 2 V, T_J = 25 °C	-	0.075	0.085	Ω
		I_D = 17 A, V_{GS} = 2 V, T_J = 100 °C	-	0.11	-	
Gate Threshold Voltage	$V_{GS(th)}$	V_{DS} = 1 V, I_D = 30 mA	-	-5	-	V
Gate Forward Current	I_{GFWD}	V_{GS} = +2 V	-	40	-	µA
Gate Resistance	R_G	f = 1 MHz, drain-source shorted	-	6	-	Ω
	$R_{G(ext)}$	V_{GS} >2.7V: See Figure 5	-	0.5	-	Ω
Dynamic Characteristics						
Input Capacitance	C_{iss}	V_{DD} = 100 V, V_{GS} = -15 V, f = 100 kHz	-	255	-	pF
Output Capacitance	C_{oss}		-	80	-	
Reverse Transfer Capacitance	C_{rss}		-	80	-	
Effective Output Capacitance, energy related	$C_{o(er)}$	V_{DS} = 0 V to 600 V, V_{GS} = -15 V	-	50	-	
Switching Characteristics						
Turn-on Delay	t_{on}	V_{DS} = 600 V, I_D = 17 A, Inductive Load, T_J = 25°C Gate Driver = +15V, -15V, $R_{G(EXT)}$ = 5 Ω See Figure 13	-	TBD	-	ns
Rise Time	t_r		-	TBD	-	
Turn-off Delay	t_{off}		-	TBD	-	
Fall Time	t_f		-	TBD	-	
Turn-on Energy	E_{on}		-	160	-	µJ
Turn-off Energy	E_{off}		-	130	-	
Total Switching Energy	E_{ts}		-	290	-	
Turn-on Delay	t_{on}	V_{DS} = 600 V, I_D = 17 A, Inductive Load, T_J = 150°C Gate Driver = +15V, -15V, $R_{G(EXT)}$ = 5 Ω See Figure 13	-	TBD	-	ns
Rise Time	t_r		-	TBD	-	
Turn-off Delay	t_{off}		-	TBD	-	
Fall Time	t_f		-	TBD	-	
Turn-on Energy	E_{on}		-	165	-	µJ
Turn-off Energy	E_{off}		-	125	-	
Total Switching Energy	E_{ts}		-	290	-	
Total Gate Charge	Q_g	V_{DS} = 600 V, I_D = 10 A, V_{GS} = +2.5 V	-	32	-	nC
Gate-Source Charge	Q_{gs}		-	2	-	
Gate-Drain Charge	Q_{gd}		-	27	-	

Abbildung 8.32: Datenblatt des SJDP120R085 (Ausschnitt)



SemiSouth — Silicon Carbide — SDP10S120D

Silicon Carbide Power Schottky Diode

Product Summary		
V_{DC}	1200	V
I_F	10	A
Q_C	39	nC

Features:
- Positive Temperature Coefficient for Ease of Paralleling
- Temperature Independent Switching Behavior
- 175 °C Maximum Operating Temperature
- Zero Reverse Recovery Current
- Zero Forward Recovery Voltage

Applications:
- Solar Inverter
- SMPS
- Power Factor Correction
- Induction Heating
- UPS
- Motor Drive

TO-247 — RoHS

Internal Schematic: K(4), A(1), K(2), A(3)

MAXIMUM RATINGS

Parameter	Symbol	Conditions	Value	Unit
Repetitive Peak Reverse Voltage	V_{RRM}	T_j = 25 °C	1200	V
D.C Blocking Voltage	V_{DC}		1200	
Continuous Forward Current (Per Leg / Per Device)	I_F	T_C < 160 °C	5 / 10	A
		T_C < 100 °C	10 / 20	
Peak Repetitive Forward Current (Per Leg / Per Device)	I_{FRM}	T_C = 125 °C, D = 0.1	20 / 50	
Non-Repetitive Forward Surge Current (Per Leg / Per Device)	I_{FSM}	T_C = 25 °C, t_P = 10 ms	26 / 52	
Power Dissipation (Per Leg / Per Device)	P_{TOT}	T_C = 25 °C	75 / 150	W
Operating and Storage Temperature	T_j, T_{stg}		-55 to +175	°C

SemiSouth — Silicon Carbide — SDP10S120D

THERMAL CHARACTERISTICS

Parameter	Symbol	Conditions	Value			Unit
			Min	Typ	Max	
Thermal Resistance; junction-case	R_{th_JC}	Per Leg	-	1.54	-	°C / W
		Both Legs		0.77	-	
Thermal Resistance; junction-ambient	R_{th_JA}		-	62	-	

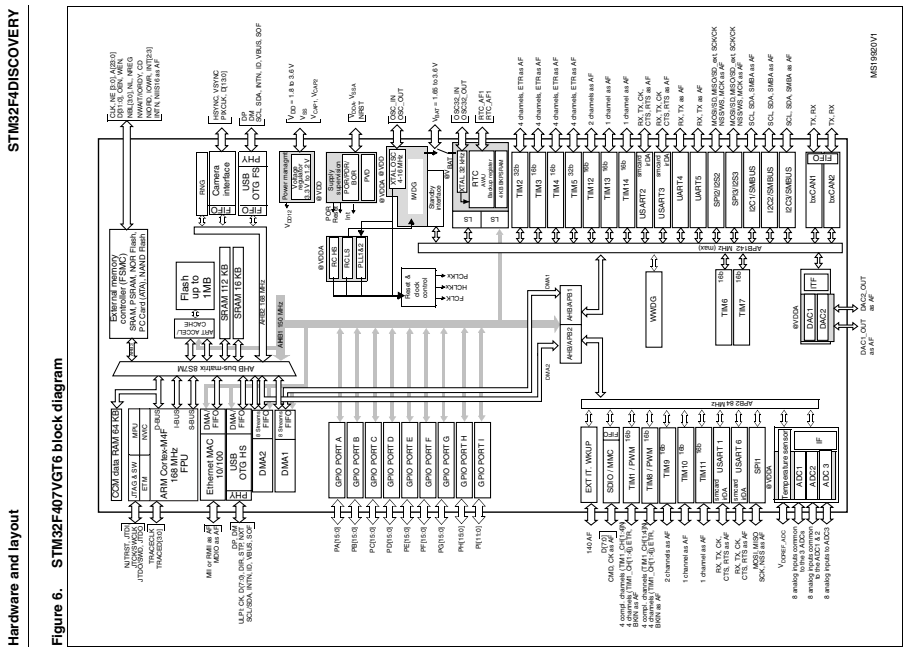
ELECTRICAL CHARACTERISTICS (Per Leg), at T_j = 25 C unless otherwise stated

Parameter	Symbol	Conditions	Value			Unit
			Min	Typ	Max	
Forward Voltage	V_F	I_F = 5 A, T_j = 25 °C	-	1.6	1.8	V
		I_F = 5 A, T_j = 150 °C	-	2.2	2.5	
Reverse Current	I_R	V_R = 1200 V, T_j = 25 °C	-	5	50	uA
		V_R = 1200 V, T_j = 150 °C	-	100	-	
Total Capacitive Charge	Q_C	V_R = 400 V, I_F = 5 A, di/dt = 500A/us	-	20	-	nC
Total Capacitance	C	V_R = 1 V, f = 100kHz	-	577	-	pF
		V_R = 300 V, f = 100kHz	-	24	-	
		V_R = 600 V, f = 100KHz	-	17	-	

Figure 1. Typ. Forward Characteristics
$I_F = f(V_F)$, parameter: T_j

Figure 2. Typ. Reverse Characteristics
$I_R = f(V_R)$

Abbildung 8.33: Datenblatt der SDP10S120D (Ausschnitt)



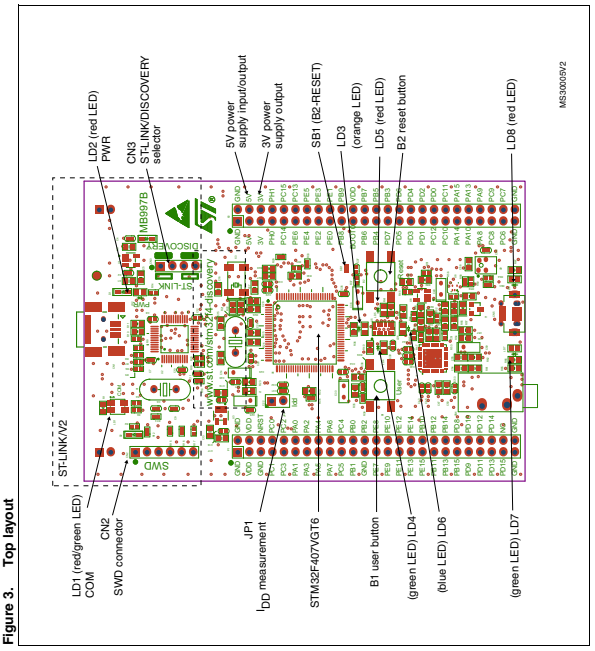


Abbildung 8.34: Datenblatt des STM32F4Discovery (Ausschnitt)

Abbildungsverzeichnis

Literatur

[1] SemiSouth Laboratories Inc. *Datenblatt des SJDP120R085 Siliziumcarbid- JFETs*. 2011.

[2] STMicroelectronics. *RM0090 Reference Manual, advanced ARM-based 32-bit MCUs*. 2011.

[3] Fluke Deutschland GmbH. *Emissionsgrade gebräuchlicher Werkstoffe*. 2013. URL: `http://www.fluke.eu/comx/show_product.aspx?locale=dede&pid=37822`.

[4] Dierk Schröder. *Leistungselektronische Bauelemente*. (2. Auflage). Springer-Verlag, 2006.

[5] Dierk Schröder. *Leistungselektronische Schaltungen*. (2. Auflage). Springer-Verlag, 2008.

[6] Dierk Schröder. *Elektrische Antriebe – Regelung von Antriebssystemen*. (3. Auflage). Springer-Verlag, 2009.

[7] Franz Zach. *Leistungselektronik*. (4. Auflage). SpringerWIenNewYork, 2010.

[8] The MathWorks Inc. *Documentation Center*. 2013. URL: `http://www.mathworks.de/de/help/index.html`.

[9] Dominique Bergogne und Damien Risaletto und Fabien Dubois und Asif Hammoud und Hervé Morel und Pascal Bevilacqua und Bruno Allard und Olivier Berry und Farid MeibodyTabar und Stéphane Raël und Régis Meuret und Sonia Dhokkar. *Normally-On SiC JFETs in Power Converters: Gate Driver and Safe Operation*. VDE VERLAG GMBH, 2010.

[10] F. Hinrichsen und I. Koch und W.R.Canders. *Current Source IGBT-Inverter for Low Inductive Synchronous Machines*. IEEE, 2004.

[11] Christian Klumpner und Frede Blaabjerg. *Using Reverse-Blocking IGBTs in Power Converters for Adjustable-Speed Drives*. IEEE, 2006.

[12] SemiSouth Laboratories Inc. *AN-SS3: SDGR600P1 – 6A Gate Driver Reference Design and Demoboard*. 2011.

[13] SemiSouth Laboratories Inc. *AN-SS5: Operation and intended use of the SGDR2500P2 dual-stage driver board*. 2011.

[14] SemiSouth Laboratories Inc. *Datenblatt der SDP10S120D Siliziumcarbid- Schottkydiode*. 2011.

[15] Avago Technologies. *Datenblatt des HCPL J312 Gate Drive Optocoupler*. 2008.

[16] LEM. *Datenblatt des LEM CAS 15 Current Transducer*. 2008.

[17] STMicroelectronics. *Datenblatt des UC3845BN Current Mode Controller*. 1999.

[18] Moritz Diller. *STM32 Tutorial*. 2012. URL: `http://www.diller-technologies.de/stm32.html`.

[19] STMicroelectronics. *STM32F4 DSP and standard peripherals library*. URL: `http://www.st.com`.

[20] Wikimedia Foundation Inc. *C-Programmierung*. 2012. URL: `http://de.wikibooks.org/wiki/C-Programmierung`.

[21] Supratim Basu und Tore. M. Undeland. *On Understanding and Driving SiC Power JFETs*. Bose Research PVT. LTD. und Norwegian University of Science und Technology,

[22] Ass. Prof Dr.techn. Klaus Krischan. *Skriptum zur Vorlesung Stromrichtertechnik 1 und 2*. Institut für elektrische Maschinen und Antriebe, 2009.

[23] Siegfried Rainer. Diplomarbeit. *Pulswechselrichter mit Stromzwischenkreis*. TU Graz, 2007.

[24] STMicroelectronics. *Datenblatt des STM32F407xx*. 2012.