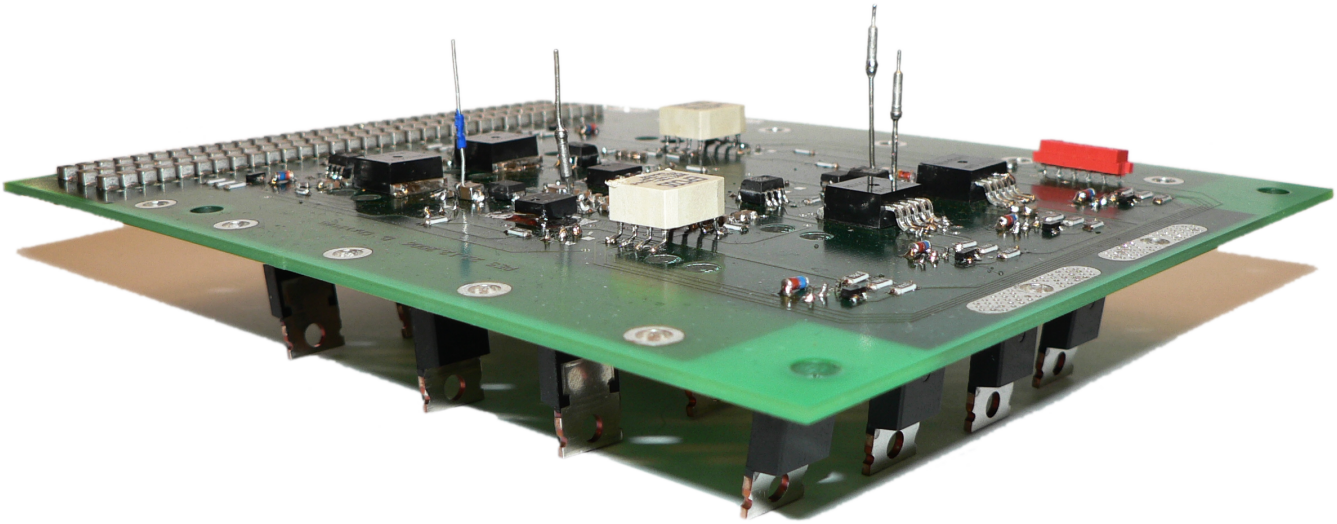
Aufbau und Inbetriebnahme eines Gleichrichters

Hard- und Software einer Dual-Active-Bridge



Semesterarbeit WS 06/07

am Power Electronic Systems Laboratory (PES) der ETH Zürich

von Daniel Christen und Sebastian Ryffel

Betreuung durch Hanna Plesko

23. Februar 2007

Zusammenfassung

Diese Semesterarbeit befasst sich mit einer für Hybridautos optimierten Dual Active Bridge. Durch eine spezielle Integration und Verwendung des Laststernpunktes des Motors können Bauteile und damit potentiell auch Gewicht und Kosten eingespart werden.

Die Semesterarbeit ist in einen Software- und einen Hardware-Teil gegliedert.

Ziel des Softwareteils war, den bestehenden Code der Dreiphasenbrückenschaltung auf das neue System anzupassen und zu optimieren. Der Code für die Steuerung und Regelung der Dual Active Bridge mussten neu implementiert werden.

Ziel im Hardwareteil war die Evaluation und Spezifikation der Ausgangskapazität und der Transistoren der bidirektionalen Gleichrichterschaltung auf der Niederspannungsseite. Es sollte bei einer Ausgangsspannung von 14V eine Ausgangsleistung von $P_2 = 1$kW erreicht werden.

Als Resultat wurde ein Prototyp entworfen, gebaut und getestet. Die oben aufgeführten Ziele wurden erreicht.

Vorwort

Im Rahmen dieser Arbeit wurde eine neuartige Integration einer Dual Active Bridge als DC/DC Konverter in Hybridfahrzeugen ausgelegt und implementiert. Die Arbeit besteht aus einem Software-Teil, welcher von Sebastian Ryffel übernommen wurde und einem von Daniel Christen behandelten Hardware-Teil. Sebastian Ryffel hat den existierenden Code des Back-to-Back Konverters angepasst und darin die neue Regelung und Steuerung der Dual Active Bridge integriert. Daniel Christen hat die Schaltung ausgelegt und das PCB gelayoutet und gebaut.

Als Gefäss dieser Arbeit diente die Semesterarbeit im Wintersemester vom Oktober 2006 bis im Februar 2007. Durchgeführt wurde sie am Institut des Power Electronic Systems Laboratory (PES) an der Eidgenössischen Technischen Hochschule Zürich (ETH).

An dieser Stelle bedanken wir uns beim Power Electronic Systems Laboratory für die zur Verfügung gestellte Infrastruktur wie Geräte, Hardware und Software.

Wir danken auch Hanna Plesko für die fachliche Betreuung und tatkräftige Unterstützung.

Hiermit bestätigen wir, dass wir die vorliegende Arbeit ohne fremde Hilfe und nur unter Zuhilfenahme der angegebenen Quellen verfasst haben.

Zürich, Februar 2007

Daniel Christen und Sebastian Ryffel

Inhaltsverzeichnis

Kapitel 1

Einleitung

Steigende Benzinpreise und eine wachsende Sensibilisierung der Gesellschaft für die Klimaproblematik machen Hybridfahrzeuge zur ernsthafter Konkurrenz für herkömmlich angetriebene Modelle. Die Verkaufszahlen bestätigen dies: Toyota alleine hat schon über eine halbe Million Autos mit Hybridantrieb verkauft und erweitert das Angebot fortlaufend. Mittlerweile kann es sich kein grosser Autohersteller mehr leisten, auf diesem Gebiet nicht aktiv zu sein.

Die meisten Systeme benützen dabei neben einem Verbrennungsmotor einen elektrischen Motor. Zur Minimierung der Verluste werden diese von einem Hochspannungsbus gespiesen. Parallel dazu steht eine 12V-Hilfsspannung für Verbraucher wie Bordelektronik und Klimaanlage zur Verfügung.

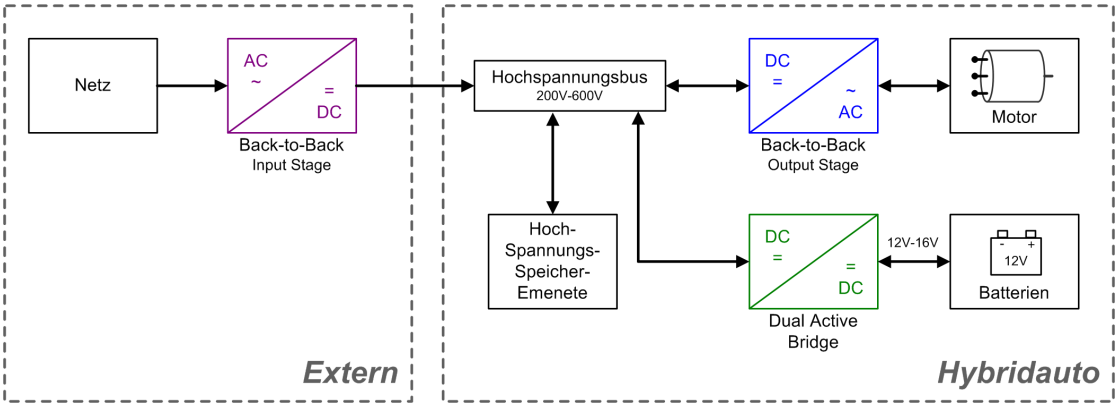


Abbildung 1.1: Leistungselektronik eines Hybridfahrzeuges

Die Abbildung 1.1 zeigt einen möglichen Aufbau des oben vorgestellten Systems. Der Hochspannungsbus liefert Spannungen zwischen 200V und 600V. Er wird von Hochspannungsspeicherelementen wie Supercaps oder Hochspannungsbatterien gespiesen, welche von einem externen Netz über einen AC/DC-Konverter geladen werden können. Der DC/AC-Konverter generiert die von der Motorregelung vorgegebene Eingangsspannung der Maschine.

Diese zwei Konverter werden durch einen Back-to-Back-Konverter realisiert. Dieser besteht aus zwei Dreiphasenbrückenschaltungen: der Input Stage (AC/DC) und der Output Stage

(DC/AC) (Abb. 1.2).

Der Hochspannungsbus und das 12V-16V-Netz der Hilfsspannung sind über einen bidirektionalen potentialgetrennten DC/DC-Konverter verbunden. Eine gute Wahl für diesen Konverter ist die Dual Active Bridge (DAB). Sie besitzt einige Features, die sie für eine hochfrequente Leistungswandlung in elektrischen Fahrzeugen spannend machen. Dazu gehören: eine hohe Leistungsdichte, niedrige Belastung der Komponenten, kleine Filter-Komponenten, hohe Effizienz, wenige Bauteile, Buck- und Boost-Moden und der geforderte bidirektionale Leistungsfluss.

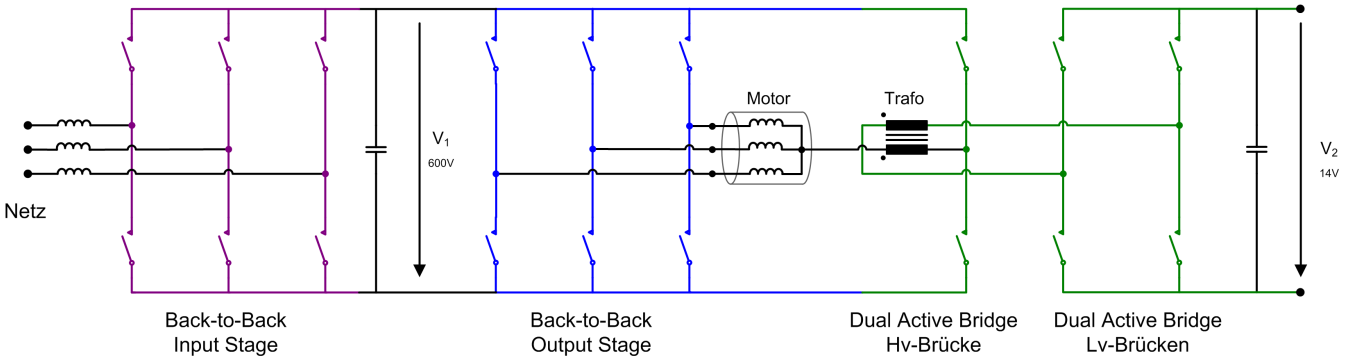


Abbildung 1.2: Schaltplan der neuartigen Integration

In dieser Arbeit geht es um eine sehr spezielle Implementation der Dual Active Bridge. Die Idee ist, die Spannung im Laststernpunkt des Motors als Hochspannungseingang zu verwenden (Abb. 1.2). Durch diese Bauweise können Bauteile gespart werden. Das System kann so bezüglich den für die Autonindustrie wichtigen Grössen wie Gewicht, Kosten und Volumen optimiert werden.

Dies ist ein erster Schritt in Richtung einer weitergehenden Integration. Letztendlich sollte es möglich sein, den Transformator in den Motor zu integrieren.

Kapitel 2

Dual Active Bridge

Die geforderten Spezifikationen unserer Dual Active Bridge sind wie folgt: Die maximale Eingangsspannung beträgt 600V und die Ausgangsspannung 14V. Sie muss für eine Leistung von 1kW ausgelegt sein und eine Schaltfrequenz von 20kHz besitzen.

2.1 Ersatzschaltbild

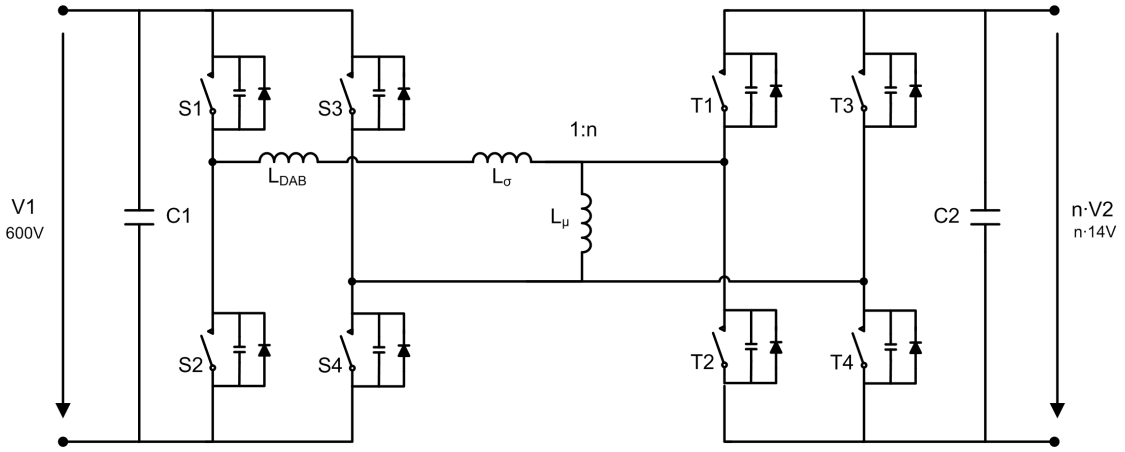


Abbildung 2.1: Ersatzschaltbild der DAB

Die Abbildung 2.1 zeigt das Schaltbild der DAB. Der Transformator wurde durch ein einfaches, auf die Primärseite bezogenes, Ersatzschaltbild ersetzt. Es besteht aus der Streuinduktivität L_σ und der Magnetisierungsinduktivität L_μ. Das Windungsverhältnis beträgt 1:n, weshalb im Ersatzschaltbild die Ströme und Spannungen der Sekundärseite entsprechend mit n skaliert werden müssen.

Da die Streuinduktivität L_σ des Trafos meist sehr klein ist, wird seriell dazu eine externe Induktivität L_{DAB} verwendet.

Bei den primärseitigen Schaltern S_{1-4} handelt es sich um die Ersatzschaltbilder von MOSFETS und bei den sekundärseitigen T_{1-4} um diejenigen von IGBTs.

Die Eingangs- und Ausgangsspannungen V_1 und V_2 werden näherungsweise als konstant angenommen.

2.2 Phase-shift

Es gibt diverse Modulationsverfahren für den Leistungstransfer, die sich für verschiedene Anwendungen und bei verschiedenen Auslastungen anbieten. Mit Phase-shift haben wir uns am intensivsten auseinandergesetzt und dieses Verfahren haben wir auch implementiert. Deshalb wollen wir hier exemplarisch auf dieses Modulationsverfahren eingehen. Wir wollen hier ein 'ideales' System betrachten. Zur qualitativen Beschreibung der Schaltung wollen wir von Nichtidealitäten, wie den endlichen Schaltzeiten der Schalter und den dadurch nötigen Interlocking Delays, absehen und die Elemente durch einfache Ersatzschaltbilder ersetzen.

Phase-shift ist eine ebenso einfache wie effektive Methode. Das Verfahren eignet sich gleichermassen für alle Auslastungen und Transportrichtungen.

Der Ablauf in einer Schaltperiode lässt sich in vier Phasen unterteilen. Sie werden nun nacheinander präsentiert und ihre charakteristischen Eigenschaften aufgezeigt. Die Strom- und Spannungsverläufe sind in der Abbildung 2.6 angegeben.

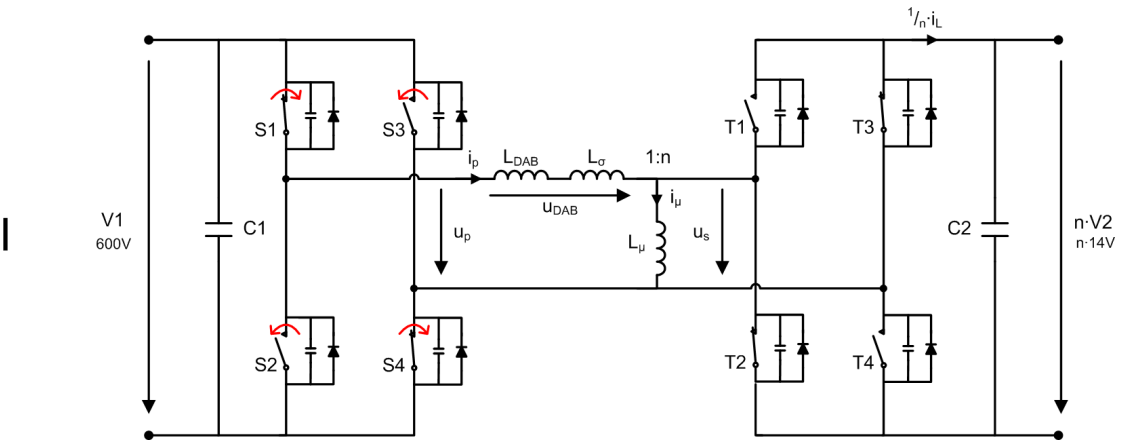


Abbildung 2.2: Phase-shift: Phase 1

Zu Beginn einer Periode bei $t = 0$ werden die Schalter $S_{1,4}$ geschlossen und gleichzeitig $S_{2,3}$ geöffnet (Abb. 2.2). Die Primärspannung u_p des Transformators ändert dadurch schlagartig das Vorzeichen von -V_1 auf +V_1. u_s ist nach wie vor gleich $-nV_2$. Der primärseitige Strom i_p ist zu Beginn dieser, respektive am Ende der 4-ten Phase negativ. Die Induktivitäten L_σ und L_{DAB} erzwingen einen kontinuierlichen Stromfluss. Dieser fliesst anfänglich über die Dioden der Schalter $S_{1,4}$ in C_1. Die an den Induktivitäten anliegende positive Spannung $u_{DAB} = V_1 + nV_2$ bewirkt einen schnellen Anstieg des Stromes. Er wird schliesslich positiv und erreicht in $t = t_1$ seinen Maximalwert $\hat{i}_p$ (sofern $V_1 < nV_2$). Er fliesst dann über die geschlossenen

Schalter $S_{1,4}$.

Der Strom i_μ durch L_μ wird durch die Sekundärspannung bestimmt. Da $L_\mu \gg L_\sigma$ gilt, ist auch $i_p \gg i_\mu$. Deshalb wird hier i_μ in erster Näherung vernachlässigt: $i_L = i_p - i_\mu \approx i_p$.

II

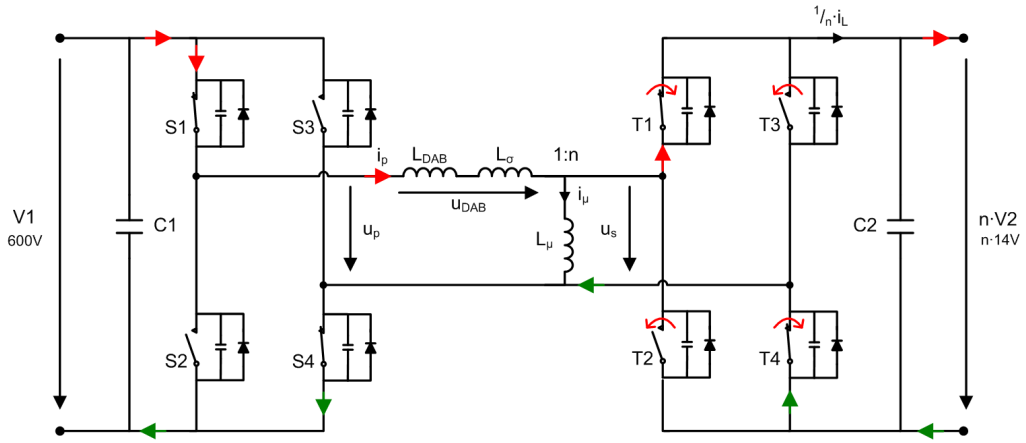


Abbildung 2.3: Phase-shift: Phase 2

Bei $t = t_1$ wird durch Schalten auf der Sekundärseite auch das Vorzeichen von u_s gekehrt: $u_s = +nV_2$. Der Strom fliesst nun über die geschlossenen Schalter $S_{1,4}$ und $T_{1,4}$ (Abb. 2.3). An den Induktivitäten liegt eine leicht negative Spannung $u_{DAB} = V_1 - nV_2$ an und lässt i_p langsam wieder abnehmen. In dieser Phase ist i_L positiv, es wird also Leistung auf die Sekundärseite transportiert.

III

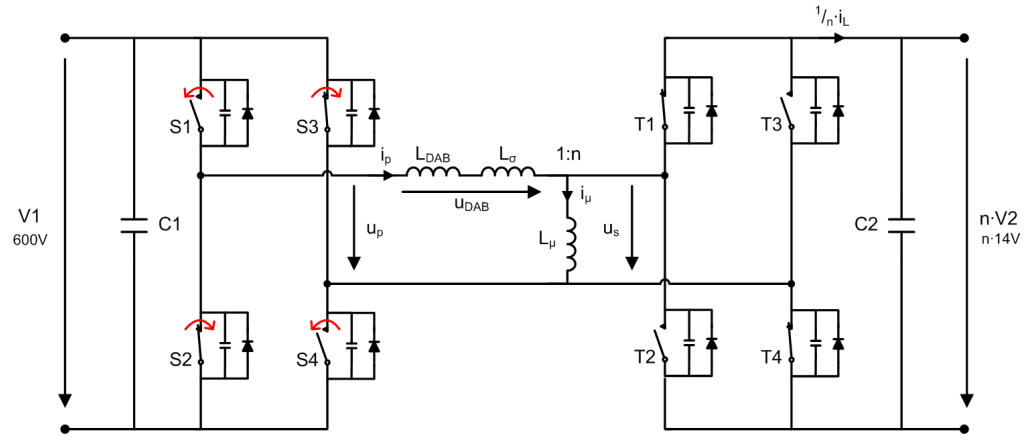


Abbildung 2.4: Phase-shift: Phase 3

Die Phase 3 beginnt in $t = t_2$ mit dem Schaltvorgang auf der Primärseite (Abb. 2.4). Es laufen nun die gleichen Vorgänge mit umgekehrten Vorzeichen wie in Phase 1 ab. i_p wird negativ und muss in t_3 den Wert $-\hat{i}_p$ erreichen. Der Grund dafür liegt in der Symmetrie der Schaltung. Dies muss so sein, damit weder der Transformator noch die Kondensatoren sättigen.

IV

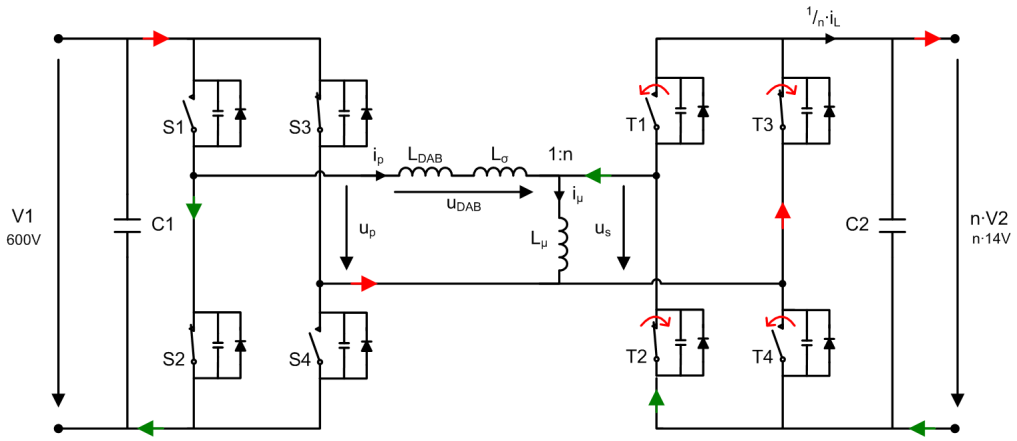


Abbildung 2.5: Phase-shift: Phase 4

Wenn nun bei $t = t_3$ die Sekundärseite schaltet, wechselt damit das Vorzeichen von u_s (Abb. 2.5). i_p wird in der Phase 4 negativ und geht über $S_{2,3}$. Damit fliesst analog zur Phase 2 ein positiver Strom i_L über $T_{2,3}$ (Abb. 2.5).

Nun ist eine komplette Schaltperiode durchlaufen.

2.2.1 Leistungscharakteristik

Die Ausgangsleistung einer Dual Active Bridge ist gemäss [10]:

$$P = \frac{V_1^2}{f L_\sigma} q\phi \left(1 - 2|\phi|\right)$$

Dabei ist V_1 die Eingangs-, V_2 die Ausgangsspannung und $f = \frac{1}{T}$ die Schaltfrequenz in Hertz.

ϕ ist der Phase-shift. Er bezeichnet die Position d der steigenden Flanke von u_s als Bruchteil der Periode T, relativ zum Beginn einer Schaltperiode. Die Position ist dann $d = \phi \cdot T$. ϕ ist im Betrieb der einzige Freiheitsgrad mit welchem der Leistungstransfer variiert werden kann.

$q = \frac{V_1}{nV_2}$ bezeichnet, ob man im Buck- ($q < 1$) oder Boost-Modus ($q > 1$) operiert. n ist das Windungsverhältnis des Transformators. In diesem Beispiel ist $q < 1$ oder anders ausgedrückt $V_1 < nV_2$. In der Praxis ist eine wichtige Überlegung bei der Wahl von q, die Grösse des dadurch definierten Operationsbereichs unter Soft-Switching-Bedingungen zu betrachten. [10]

L_σ ist die, auf die Primärseite bezogene, Streuinduktivität.

Es ist offensichtlich, dass die übertragene Leistung für $\phi = 0$ null wird. In diesem Fall hätten die Phasen 1 und 3 die Länge 0. In den übrigen Phasen fliesst i_p und damit i_L zu gleichen Teilen in beide Richtungen, womit netto keine Leistung übertragen wird. Analoges gilt auch für $\phi = \pm\frac{1}{2}$. So hätten die Phasen 2 und 4 die Länge 0. An L_σ liegen somit gleich lang die Spannungen $\pm(V_1 + nV_2)$ an. Über eine Periode gemittelt fliesst also immer noch kein Strom i_L.

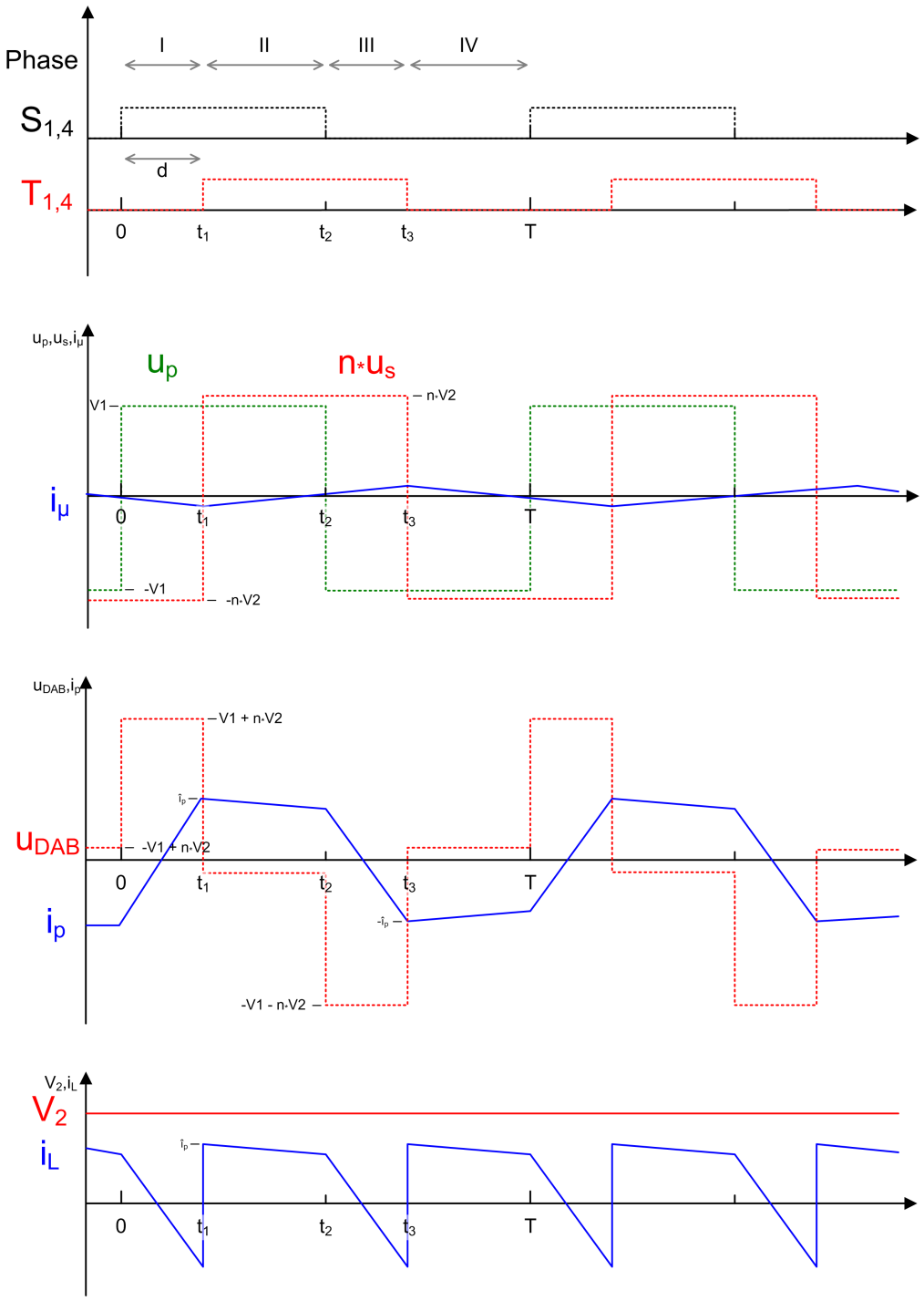


Abbildung 2.6: Charakteristische Strom- und Spannungsverläufe für Phase-shift

Dazwischen muss also jeweils ein Maximum liegen. Durch ableiten bekommt man

$$\frac{\partial}{\partial\phi}P = \frac{V_1^2}{fL_\sigma}q\left(1-4|\phi|\right)$$

Die Nullstellen sind $\phi = \pm\frac{1}{4}$. Für $\phi = \frac{1}{4}$ bekommt man einen maximalen Leistungstransfer zur Sekundärseite, für $\phi = -\frac{1}{4}$ einen maximalen Leistungstransfer in die andere Richtung. Dazwischen gilt die quadratische Abhängigkeit von P und ϕ aus der obigen Formel.

ϕ liegt also für diesen Aufbau im Interval $-\frac{1}{4} \leq \phi \leq \frac{1}{4}$.

2.2.2 Diskussion

Der Vorteil von Phase-shift liegt in seiner Einfachheit. Die Strategie lässt sich für sämtliche Auslastungen verwenden und besitzt einen einzigen Freiheitsgrad.

Dafür erkauft man sich bei niedriger Auslastung überproportional grosse Schaltverluste. Diese rühren davon, dass man zu den Schaltzeitpunkten bei $t = t_0$ und $t = t_2$ grosse Ströme über die Transistoren führt, welche beim Trennen einen hohen Verlust verursachen. Dies ist vor allem auf der Niederspannungsseite ein Problem. Auf der Hochspannungsseite sollte man Zero Voltage Switching implementieren, um zu verhindern, dass ein Transistor unter Spannung eingeschaltet wird. Das Kapitel 4.1.2 beinhaltet eine genauere Analyse der Schaltverluste.

Durch geschickte Kombination verschiedener Schaltverfahren bei verschiedenen Auslastungen, kann die Effizienz weiter erhöht werden. Es ist möglich, für geringe Leistungen Zero Current Switching und für höhere Zero Voltage Switching zu verwenden. [19]

Da in unserem Fall die Schaltfrequenz mit 20kHz relativ klein ist, halten sich die Schaltverluste trotz Phase-shift in Grenzen. Wenn mit bis zu zehn mal so hoher Frequenz geschaltet wird, steigen auch die Verluste auf das zehnfache, was spezielle Massnahmen erforderlich macht.

2.3 Die neuartige Integration

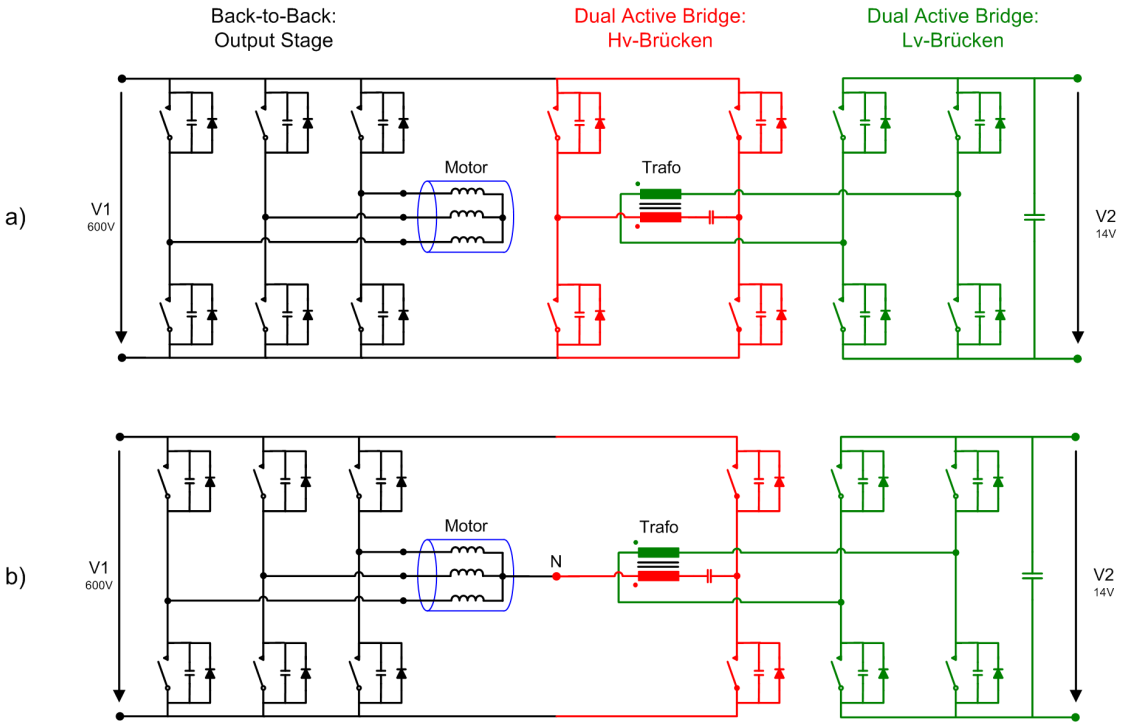


Abbildung 2.7: a) zeigt die herkömmliche Implementation einer DAB und b) die neuartige Integration

Die Abbildung 2.7 stellt die herkömmliche Implementation einer Dual Active Bridge (a) der neuartigen Integration (b) gegenüber. In letzterer dient der Laststernpunkt N des Motors als hochspannungsseitiger Eingang der Dual Active Bridge, womit eine Halbbrücke eingespart werden kann.

2.3.1 Ersatzschaltbild

Der Motor wurde in Abbildung 2.8 durch sein Ersatzschaltbild ersetzt. Es besteht aus der Nullspannung u_M des unbelasteten Motors, dem Motornullwiderstand R_M und der Motornullinduktivität L_M.

Bei der Auswahl des Motors sollte darauf geachtet werden, dass der Motorwiderstand nicht zu gross ist. Andernfalls wird die Spannung in N im belasteten Betrieb zu klein, um die Dual Active Bridge zu betreiben. Für eine prinzipielle Analyse der Schaltung werden wir R_M vernachlässigen.

Die Spannung in N wird durch den im Unterschwingungsverfahren betriebenen Motor definiert. Dies führt zu der in der Abbildung 2.9 dargestellten Spannung u_M. Die Spannung ist periodisch über die dreifache elektrische Periode des Motors.

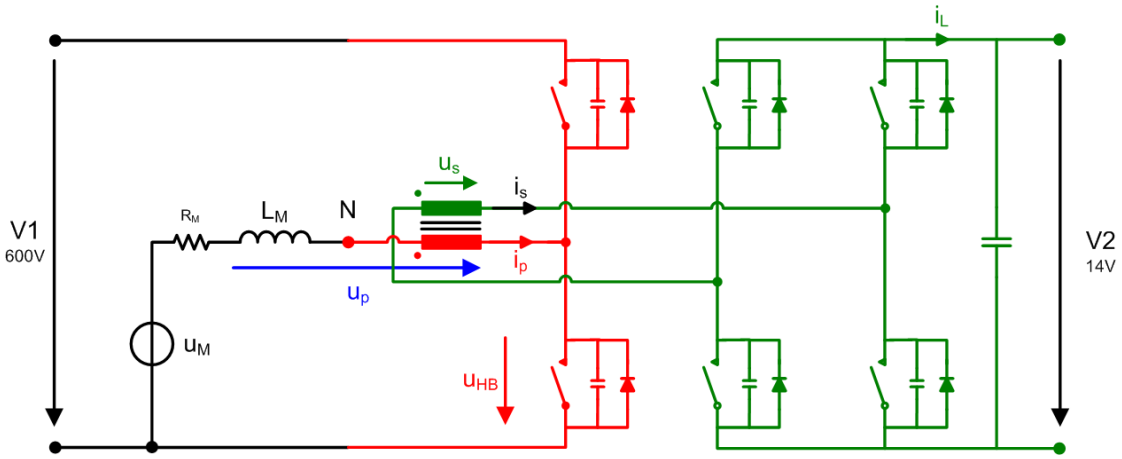


Abbildung 2.8: Ersatzschaltbild der neuartigen Implementation

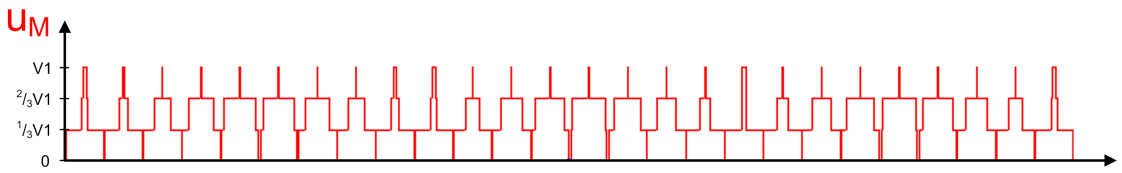


Abbildung 2.9: Nullspannung u_M der Maschine

2.3.2 Hochspannungsbrückenzweig

Neu liegt an der Primärseite des Transformators und der Motorinduktivität L_M die Spannung $u_p = u_M - u_{HB}$ an. u_M ist durch die Regelung des Motors gegeben.

Damit der Transformator nicht sättigt, muss u_p mittelwertsfrei sein: $\overline{u}_p = 0$. Es muss also $\overline{u}_p = \overline{u}_M - \overline{u}_{HB} = 0$ gelten. Deshalb muss die mittlere Spannung über der Hochspannungsbrücke der Laststernpunktspannung entsprechen.

2.3.3 Strom- und Spannungsverläufe

Abbildung 2.10 zeigt die wichtigsten Verläufe des Systems über einige Schaltperioden. Vom Prinzip her verhält sich das System gleich, wie das im Kapitel 2.2 Beschriebene.

u_{HB} wurde so gewählt, dass $\overline{u}_{HB} = \overline{u}_M$ gilt. u_p resultiert dann als deren Differenz: $u_p = u_M - u_{HB}$.

Das Delay d von u_s wurde so gewählt, dass ein Leistungstransport zur Niederspannungsseite resultiert.

Die stromtreibenden Induktivitäten L_σ und L_M erfahren die Spannung $u_{LM} + u_\sigma = u_p - u_s$. Diese bestimmt die Steigung des Stromes i_p, welcher nun mittelwertsfrei ist.

Unter Vernachlässigung des Magnetisierungsstroms ist i_s gleich i_p dividiert durch das Wicklungsverhältnis des Transformators. Wie bei der klassischen Dual Active Bridge ist der Aus-

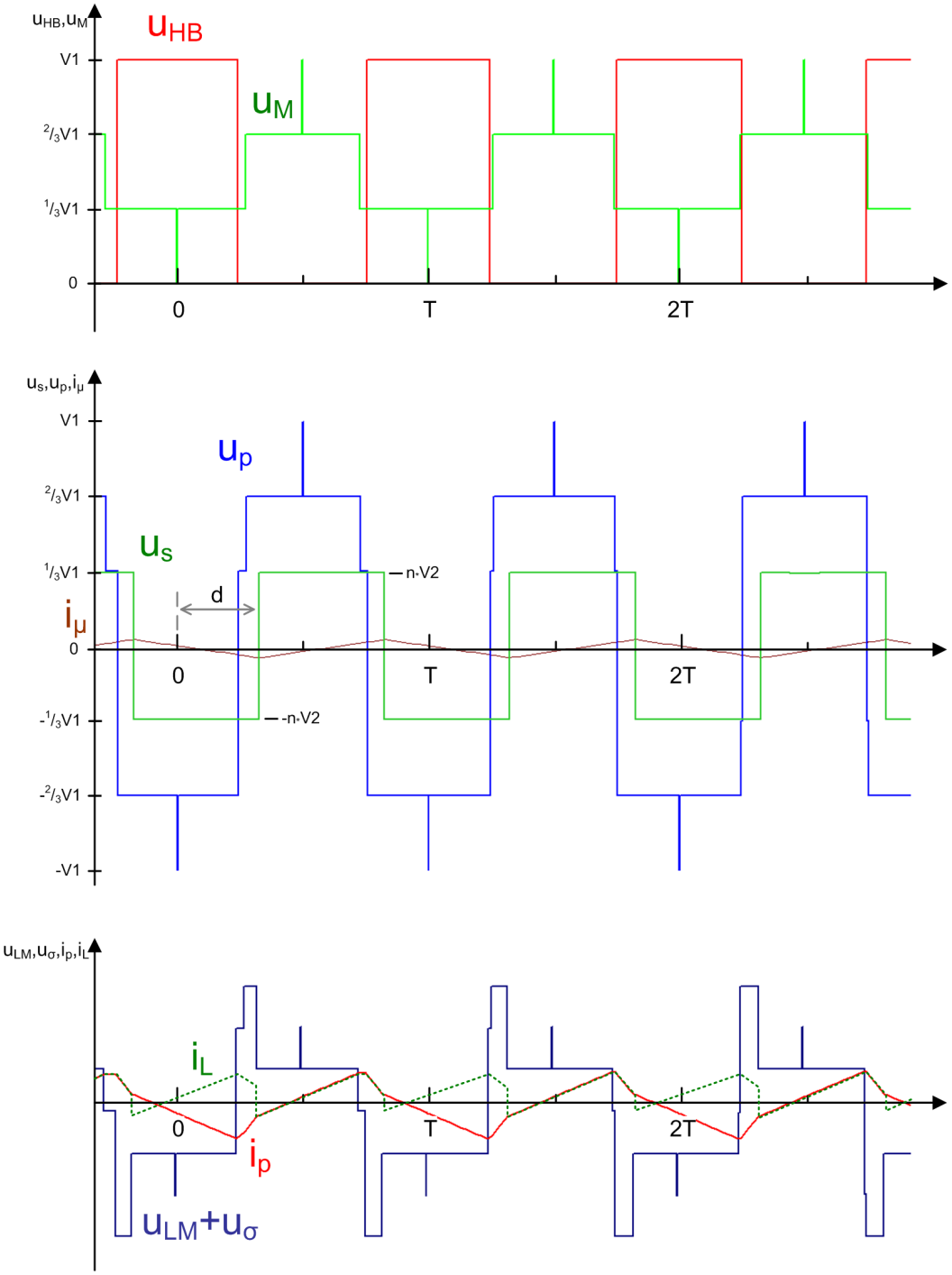


Abbildung 2.10: Strom- und Spannungsverläufe des Gesamtsystem

gangsstrom i_L des Konverters in seinem Betrag durch den sekundärseitigen Transformatorstrom definiert und das Vorzeichen wird durch das Schalten des aktiven Gleichrichters bestimmt.

Man sieht, dass in diesem Beispiel der Mittelwert von i_L über eine Periode positiv ist. Die Wahl dieses Phase-shifts resultiert also in einem Leistungstransport zur Sekundärseite.

2.4 Nichtidealitäten

In diesem Kapitel wollen wir einige, bei der bisherigen Beschreibung der Schaltung vernachlässigte, Nichtidealitäten besprechen.

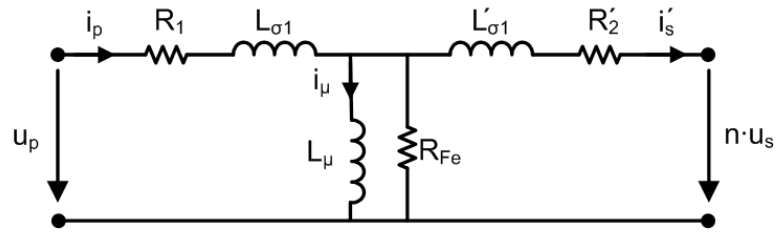


Abbildung 2.11: Erweitertes Ersatzschaltbild eines Transformators

Bis jetzt haben wir immer das Ersatzschaltbild eines idealen Transformators angenommen. Damit haben wir zum Einen Hysterese- und Wirbelstromverluste vernachlässigt. Diese sind im erweiterten Ersatzschaltbild mit R_{Fe} modelliert (Abb. 2.11). Verluste im Eisen führen zu einem kleineren Strom i_L.

Zum Anderen haben wir den sogar grösseren Effekt der Verluste im Kupfer der Wicklungen vernachlässigt. An diesen Widerständen fallen bei fliessenden Strömen i_p und i'_s Spannungen ab (im Ersatzschaltbild R_1 und R'_2). Je grösser der Strom ist, desto kleiner wird die Spannung über den stromerzeugenden Induktivitäten. Dies wirkt sich negativ auf den Laststrom i_L aus.

Der Motorwiderstand hat den gleichen Effekt wie die Verluste im Kupfer. Die Abbildung 2.12 zeigt rot den Verlauf des Stromes durch die Primärwicklung des Trafos. Man sieht deutlich die mit steigendem Strom sinkende Spannung über den stromerzeugenden Induktivitäten und die dadurch verursachte Verlangsamung der Zunahme von i_p.

Nicht berücksichtigt wurden zudem die Leitungsinduktivitäten. Besonders beim Schalten der grossen Ströme auf der Niederspannungsseite machen sich diese in Form von Spannungsüberschwingern bemerkbar.

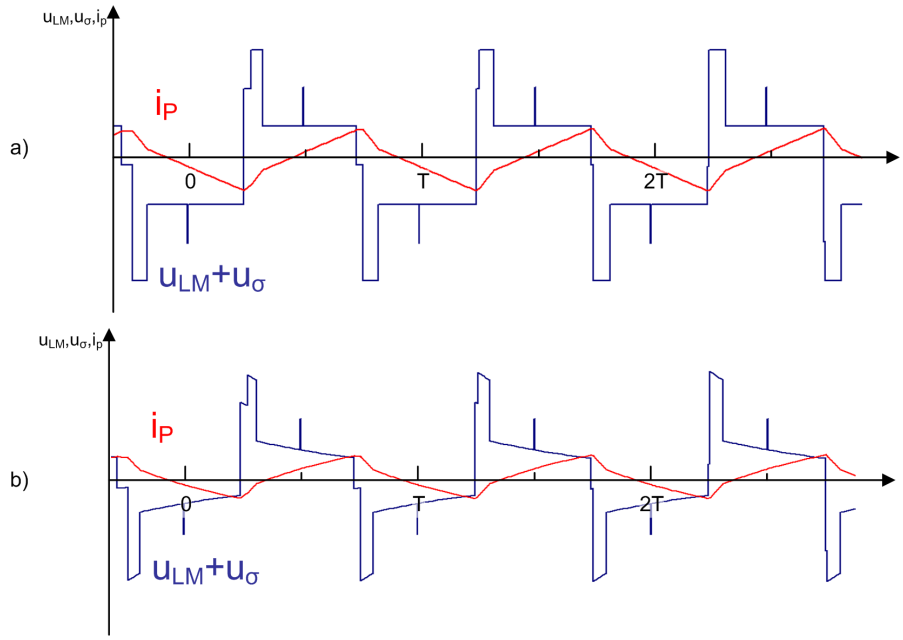


Abbildung 2.12: a) idealisierte Verläufe, b) Verläufe mit Verlusten im Kupfer und am Motorwiderstand

Kapitel 3

Software

Die implementierte Software zum Betrieb und Regelung ist aufgeteilt auf ein FPGA, welches die Schalter ansteuert und ein, für die Regelung verantwortlicher, DSP. Von einem PC aus kann man das System steuern und überwachen.

Der für die Dual Active Bridge verantwortliche Teil ist als Modul in den Code der anderen Komponenten eingebettet. Diese umfassen zusätzlich den Back-to-Back-Konverter und den Motor.

Ein Überblick über die verschiedenen Komponenten und ihre Verdrahtungen bietet die Abbildung 3.4.

Gestartet bin ich vom Quellcode für den Back-to-Back Konverter [12]. Dessen C-Code konnte nach ein paar kleinen Änderungen übernommen und erweitert werden. Neu beinhaltet er die Messung zusätzlicher Spannungen und die Regelung der Dual Active Bridge. Dafür musste das Protokoll zur Kommunikation mit dem FPGA neu entworfen werden.

Der VHDL-Code wurde für eine andere Architektur, bestehend aus zwei älteren FPGAs, geschrieben. Die Portierung wurde dadurch erschwert, dass das neue einzelne FPGA zu langsam für den alten Code war. Deshalb mussten grosse Teile performanter implementiert werden. Natürlich musste, neben der Brücken der Dual Active Bridge, auch die neue Schnittstelle zum DSP implementiert werden.

Dieses Kapitel beinhaltet zuerst eine Übersicht über die Architektur der Software und einige Erklärungen zur verwendeten Terminologie. Anschliessend kommen die Dokumentationen des DSP- und FPGA-Codes in zwei Unterkapiteln. Dort werden die wichtigsten Sachverhalte erläutert, die zum Verständnis der Software nötig sind.

3.1 Software Architektur

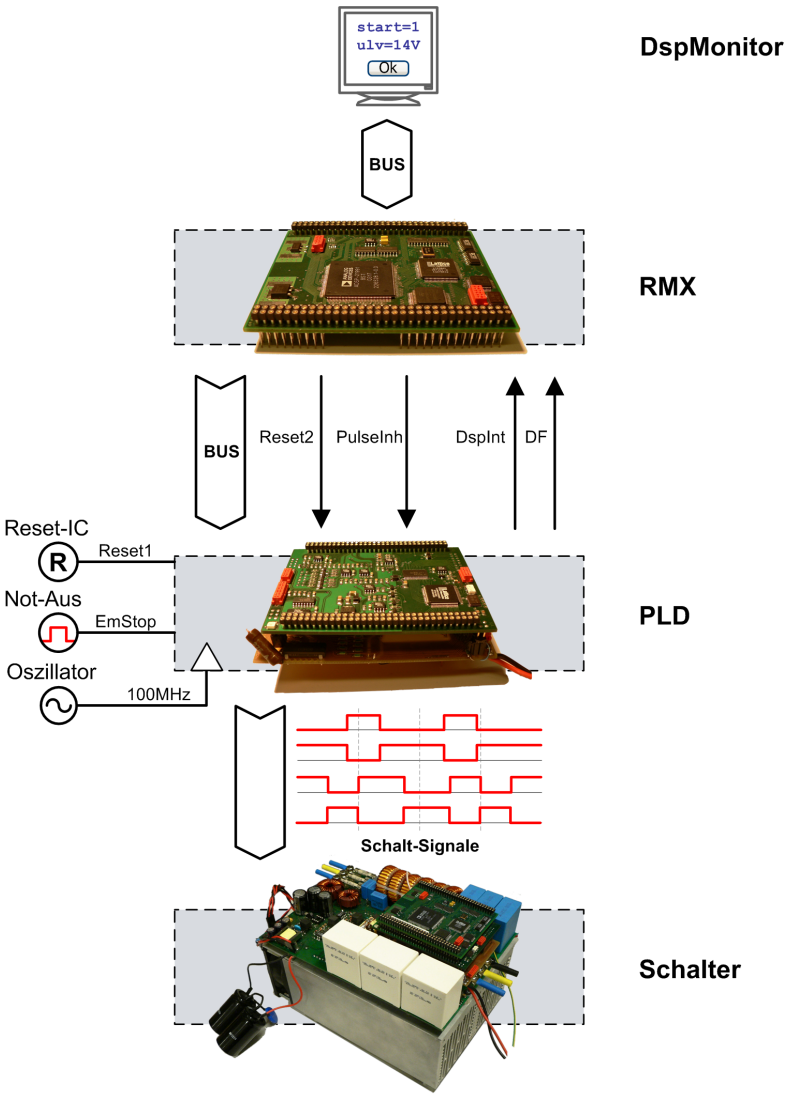


Abbildung 3.1: Übersicht über die Software-Architektur

In Abbildung 3.1 ist die Einbettung des FPGAs und DSPs ins Gesamtsystem dargestellt.

Gestartet und gesteuert kann das System über den Hardware Monitor werden. Vom PC aus kann man direkt Variablen im DSP auslesen und setzen und so dessen Zustand ändern.

Die Regelung und Berechnung der Schaltzeiten ist im DSP implementiert. Über einen Bus schreibt dieser die neuen Daten ins FPGA. Dieser Vorgang wird über die Interrupt-Leitung vom FPGA regelmässig angestossen. Dort werden die Schaltsignale generiert.

Neben dem Bus besitzt das FPGA zwei Reset-Eingänge. Der eine kommt vom RMX, der andere vom Reset IC, welcher eine Spannungsüberwachung beinhaltet. Bemerkt der IC während des Betriebs einen Fehler, so wird via `PulseInh`-Eingang die Pulssperre ausgelöst, und damit alle Schalter ausgeschaltet. `EmStop` ist das Signal des Not-Aus-Schalters und hat den gleichen Effekt.

3.1.1 Implementation der Niederspannungsbrücken

Die Niederspannungsbrücken der DAB werden zur Zeit mit dem Phase-shift Verfahren geschaltet. Sie sind aber so designt, dass später eine beliebige Strategie angewandt werden kann. Dazu müssen keine Änderungen am VHDL-Code vorgenommen, sondern nur eine neue Regelung im DSP implementiert werden.

DSP

Der DSP berechnet den Delay des Phase-shift und leitet daraus die Schalterstellungen und -Zeiten ab. Diese heissen in der verwendeten Terminologie Vektoren (`Vect`) und τ (`Tau`).

Die Vektoren bezeichnen den Zustand des oberen Transistors einer Halb-Brücke. Dabei bedeutet 1 zu und 0 offen. Der untere Transistor schaltet jeweils gegengleich. Das zugehörige τ gibt die Länge des entsprechenden Schaltzustands an.

FPGA

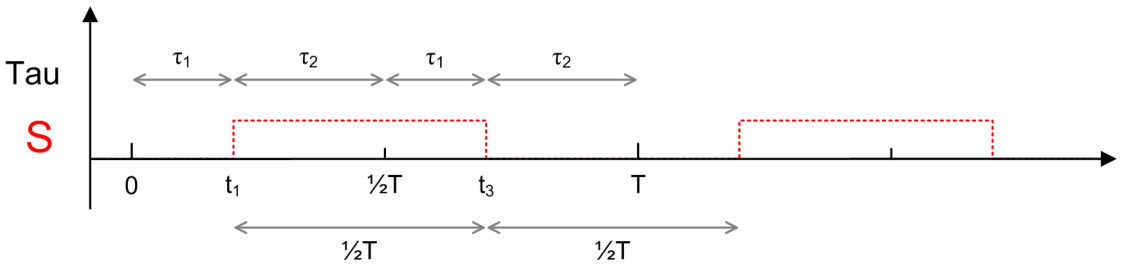


Abbildung 3.2: Schaltmuster einer Niederspannungsbrücke

Das FPGA bekommt vom DSP über den Bus für jede Brücke den ersten Vektor (`Vect1`) und dessen Länge (`Tau1`). Der zugehörige `Vect2` ergibt sich dann zu `1-Vect1` und `Tau2=` $\frac{1}{2}$`T-Tau1`.

Daraus generiert es die Schaltsignale der Niederspannungsbrücken. In Abbildung 3.2 sind die Signale einer Brücke für den übermittelten Vektor `Vect1=0` und `Tau1=`t_1 angegeben.

So sind die Schalter einer Brücke über eine Periode betrachtet immer zu gleichen Teilen ein- und ausgeschaltet.

3.1.2 Implementation der Hochspannungsbrücke

Die Hochspannungsbrücke generiert eine nicht mittelwertsfreie Spannung und unterscheidet sich dadurch von den Niederspannungsbrücken.

DSP

Anhand der Zeiten des BBC berechnet der DSP auch für die Hochspannungsbrücke ein `Vect1` und ein `Tau1`. Diese haben die gleichen Bedeutungen wie die entsprechenden Werte der Niederspannungsbrücke.

FPGA

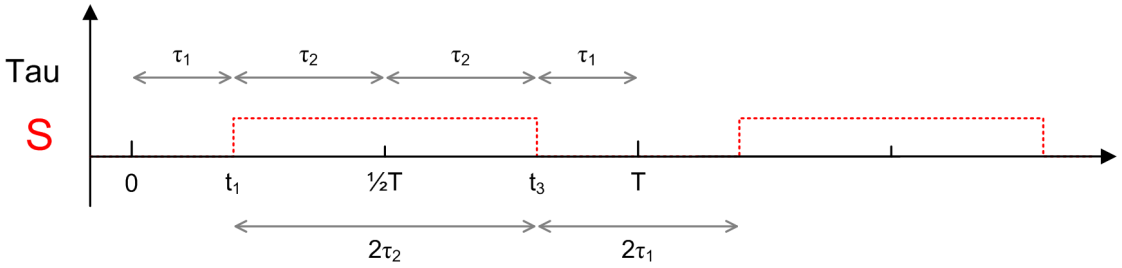


Abbildung 3.3: Schaltmuster einer Hochspannungsbrücke

Das FPGA generiert aus den Daten ein leicht anderes Schaltmuster. Der Unterschied liegt in der Reihenfolge der τ.

Der Mittelwert der Spannung u_{HB} beträgt also:

$$\overline{u}_{HB} = \begin{cases} V_1 \frac{2\tau_1}{T}, & \text{wenn } \texttt{Vect1} = 1 \\ V_1 \frac{2\tau_2}{T}, & \text{sonst} \end{cases}$$

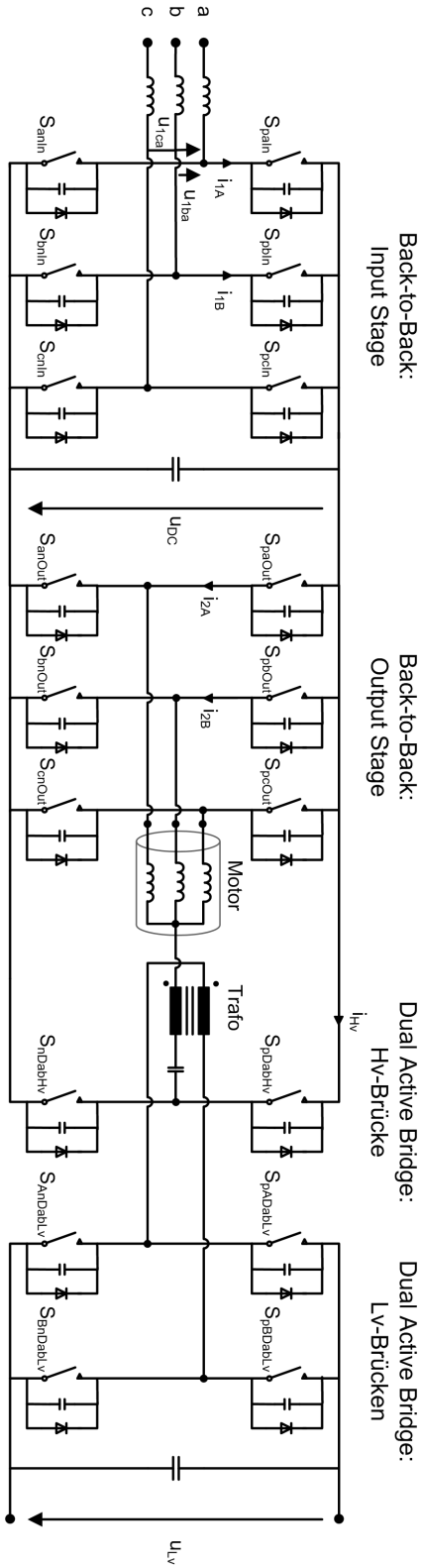


Abbildung 3.4: Übersicht über das Gesamtsystem

3.2 DSP

Beim DSP handelt es sich um ein `ADSP-21991` von Analog Devices, welches mit 80MHz getaktet wird.

Im Folgenden werde ich die wichtigsten Bestandteile des C-Codes vorstellen. Dazu gehören die Funktionen, Variablen und Konfigurationsoptionen.

3.2.1 Funktionen in `main.c`

`main()`

Hier werden verschiedene Initialisierungsroutinen aufgerufen, welche das System in einen definierten Zustand versetzen.

Ablauf:

1. In `Setup_Interrupt_Controller()` werden Interrupts definiert und die zugehörigen Prioritäten zugewiesen. Dabei ist `USR0` die höchste und `USR11` die tiefste Priorität.
2. Mit `Init_MCM()` werden erstmalig Daten ans FPGA gesandt und damit das System initialisiert.
3. In der unendlichen Schleife werden verschiedene Variablen vorzu aktualisiert.

Alle weiteren Funktionen werden über die registrierten Interrupts aufgerufen.

`ISR_Control_Loop()`

Diese Funktion wird vom Interrupt aufgerufen, welcher durch das FPGA ausgelöst wird. Sie ist verantwortlich für die Berechnung und Regelung der Schaltzeiten.

Die Ausführung dieser Funktion dauert etwa 15us, was die maximale Frequenz der Interrupts limitiert.

Ablauf:

1. `ADC_CONV_START()` startet eine AD-Wandlung für die aktuellen Strom- und Spannungswerte. Nach Beendigung von vier Messungen wird `adc_ready` auf `TRUE` gesetzt.
2. In der Zwischenzeit werden in `start_stop_bbc()` und `start_stop_dab()` die Zustände des Back-to-Back-Konverters und der Dual Active Bridge kontrolliert und die Variablen `status` respektive `dab_status` entsprechend `usr_start` gesetzt und die Zero Gate Signale gesetzt.

   Diese Funktionen kontrolliert auch das saubere Hochfahren der Schaltung.
3. Sobald die Messwerte verfügbar sind, kontrolliert `fault_detection()` sie auf Abnormalitäten. Im Fehlerfall deaktiviert die Funktion die Brücken und kommuniziert die Art des Fehlers über die globale Variable `flt_code` (siehe Kapitel 3.2.3).

4. Dann werden die für die Regelung benötigten Berechnungen gemacht und schliesslich die neuen Stellgrössen bestimmt. Dies sind `phi1_ref` und `phi2_ref` für die Input-, respektive Output-Brücken des BBC und `delay` für die DAB.
5. `GetInputSectorSpecificData()` und `GetOutputSectorSpecificData()` finden den aktuellen Sektor und leiten daraus die Schalterstellungen ab.
6. `CalcBBCTaus()` berechnet die Schaltzeiten `TauArray[0-3]` des BBC.
7. `CheckBBCTaus()` kontrolliert die in Schritt 6 berechneten Schaltzeiten des BBC auf Gültigkeit, damit diese weiterverwendet werden dürfen.
8. `CalcDabHvTau()` und `CalcDabLvTaus()` berechnen die Schaltzeiten der DAB und speichert sie in `TauArray[4-6]`.
9. `CheckDabTaus()` kontrolliert die Schaltzeiten der DAB auf Gültigkeit.
10. `GenerateMCM_Data()` schreibt die neuen Daten ins Array `DATA_MCM`, so dass sie ans FPGA geschickt werden können. Das Format dieses Arrays ist in Kapitel 3.2.5 erläutert.
11. `Write_MCM()` schickt elementweise `DATA_MCM` ans FPGA.

`CalcBBCTaus()`
Berechnet die Schaltzeiten `TauArray[0-3]` des BBC.

`CalcDabHvTau()`
Berechnet `TauArray[6]` und `dabhvvect` aus den Einschaltzeiten und Vektoren der Ausgangsseite des BBC.

`TauArray[6]` wird gleich der durchschnittlichen Einschaltzeit der drei Brücken der Output-Stage gesetzt.

`CalcDabLvTaus()`
Berechnet `TauArray[4], TauArray[5]` und die entsprechenden Vektoren `dablvvect_a` und `dablvvectb` aus `delay` (Phase-shift).

`CheckBBCTaus()` und `CheckDabTaus()`
Machen Taus korrekt:

- nicht negativ
- 0 oder grösser als 2
- keine Summe grösser als `Tpulsehalf`

3.2.2 PI-Regler

Datenstrukturen für PI-Regler

im File `'Include/DSP.H'`.

PIV: Zustände des Reglers

```
typedef struct piv_s {
   LWORD yk_1;     // Stellgrösse
   WORD  uk_1;     // letzter Fehler
} PIV;
```

Dieser Regler speichert nicht den aktuellen Fehler und Integralteil, sondern die Stellgrösse und den letzten Fehler.

PICOEF: Koeffizienten des Reglers

```
typedef struct picoef_s{
   WORD b0;     // Gewichtung des letzten Fehlers
   WORD b1;     // Gewichtung des neuen   Fehlers
} PICOEF;
```

PIRAW: Koeffizienten eines herkömmlichen PI-Reglers

```
typedef struct piraw_s{
   WORD kp;      // P-Verstärkung
   WORD wit;     // I-Verstärkung
} PIRAW;
```

Funktionen der PI-Regler

im File `'Control Func/Dsp.c'`.

`S16 satpiv(PIV *state_ptr, PICOEF *coef_ptr, S16 uk, S16 ll, S16 ul)`

`state_ptr`: Zustand des PI-Reglers

`coef_ptr`: Koeffizienten des PI-Reglers

`uk`: Fehler-Input

`ll`, `ul`: Stellgrösse sättigt zwischen `ll` und `ul`

Die Funktion aktualisiert den Zustand und gibt die neue Stellgrösse zurück.
Führt zum Zeitpunkt `k` zur Stellgrösse `yk`:
`yk[k] = b1*uk[k] + b0*uk[k-1] + yk[k-1]`

Dieser Regler sättigt nur einmal und zwar die Stellgrösse. Wenn man den Integralteil und den Fehler speichern würde, müsste man den Integralteil und die Stellgrösse separat sättigen.

PICOEF chgpi(PIRAW *prmts_ptr)

Rechnet die Koeffizienten eines herkömmlichen PI-Kontrollers in diejenigen des in diesem Projekt verwendeten Kontrollers um.

Ein Herkömmlicher würde zum Zeitpunkt `k` mit dem Fehler `uk` zu folgender Stellgrösse `yk` führen:

```
I[k] = uk[k] + I[k-1];
yk[k] = kp*(uk[k] + wit*I[k]);
```

Da wir aber nicht den Integralteil, sondern die Stellgrösse speichern, müssen wir den letzten Fehler (also `uk_1`) von `b0` wieder abziehen:

```
yk[k] = b1*uk[k] + b0*uk[k-1] + yk[k-1]
yk[k] = kp*uk[k] + (kp*wit-kp)*uk[k-1] + yk[k-1]
yk[k] = kp*uk[k] + kp*wit*uk[k-1]-kp*uk[k-1] + yk[k-1]
yk[k] = kp*uk[k] + kp*wit*uk[k-1]-kp*uk[k-1] + kp*uk[k-1]
   + kp*wit*uk[k-2]-kp*uk[k-2] + yk[k-2]
yk[k] = kp*uk[k] + kp*wit*uk[k-1] + kp*wit*uk[k-2]-kp*uk[k-2]
   + yk[k-2]
```

⋮

3.2.3 Variablen

Allgemeine Variablen

In der Tabelle 3.1 sind die wichtigsten globalen Variablen mit ihrer Beschreibung aufgeführt.

Variable	Beschreibung
STATE_ENUM status	Aktueller Zustand Mögliche Werte: 0: OFF, 1: FAULT, 2: RUN
FLTCODE_ENUM flt_code	Fehlercode Mögliche Werte: 0: NO_FAULT, 1: OC_TRIP_A1, 2: OC_TRIP_B1, 3: OC_TRIP_A2, 4: OC_TRIP_B2, 5: OV_TRIP_DC_H, 6: OV_TRIP_DC_L, 7: OPMODE_FAIL, 8: OV_TRIP_LV_H, 9: OV_TRIP_LV_L
OPMODE_ENUM opmode	Betriebsmodus Mögliche Werte: 0: NORMAL, 1: INV_ONLY 2: AFE_ONLY
USRCTRL_ENUM usr_start	Startet und stoppt die Switching Aktivität. Kontroll-Signal vom HWMonitor. Mögliche Werte: 0: USR_STOP, 1: USR_START, 2: USR_RESET

Variable	Beschreibung
int Tpulsehalf	Länge der halben Pulsperiode gemessen in Clock Cycles à 20ns (50MHz). Ist für eine Schaltfrequenz von 20KHz gleich 1250.
int Data_MCM[9]	Daten, welche aufs FPGA geschrieben werden. Wird in `GenerateMCM_Data()` erstellt.

int TauArray[7]	Enthält verschiedene Tau-Werte gemessen in Clock-Cycles à 20ns:
	`0...Tpulsehalf ⇒ 0...25us`
	`⇒ 0x0000...0x4e2`
	`TauArray[0] = TauIn1 = Tau_mue`
	`TauArray[1] = TauIn2 = Tau_nue`
	`TauArray[2] = TauOut1 = Tau_alf`
	`TauArray[3] = TauOut2 = Tau_bet`
	`TauArray[4] = DabLvTau1A =`
	`TauArray[5] = DabLvTau1B =`
	`TauArray[6] = DabHvTau1 =`

int Tau_xi	Freewheeling Intervall (TauIn3) der Input Stage. `Tau_xi = Tpulsehalf - TauArray[0] - TauArray[1];`
int Tau_gam	Freewheeling Intervall (TauOut3) der Output Stage `Tau_gam = Tpulsehalf - TauArray[2] - TauArray[3];`

Tabelle 3.1: Allgemeine Variablen

DAB-spezifische Variablen

In der Tabelle 3.2 sind die wichtigsten globalen, DAB-spezifischen Variablen mit ihrer Beschreibung aufgeführt.

Variable	Beschreibung

DABSTATE_ENUM dab_status	Aktueller Zustand der Dual Active Bridge.
	Mögliche Werte:
	`0: DAB_OFF,`
	`1: DAB_FAULT,`
	`2: DAB_INI,`
	`3: DAB_RUN`

Variable	Beschreibung
int DabLvZG **int DabHvZG**	Zero Gate Signale
int DeadTimeDabLvMd **int DeadTimeDabHvMd**	Dead Time Mode der entsprechenden Brücken. Mögliche Werte: `0=1.27us, 1=1.2us, 2=800ns, 3=400ns, 4=200ns, 5=100ns, 6=50ns, 7=0ns`
int dablvvect_a **int dablvvect_b**	Aktuelle Vektoren der Low Voltage Brücken A und B
int dabhvvect	Aktueller Vektor der High Voltage Brücke
int delay	Delay des Phase-shift, siehe Kapitel 2.2 - *Bedeutung*: Gibt die Position des 1-0 Übergangs relativ zum `DAB_ZER_OFFS_DELAY` an. - *Wertebereich*: als pu Bruch von `Tpulshalf` also: `[-1...1] = [0xe000...0x2000]` - *Leistungsfluss*: 0 für `delay` = 0 zur Lv-Seite für `delay` > 0 zur Hv-Seite `delay` < 0
int ulv	Gemessene Spannung der Niederspannungsseite
int ihv	Gemessener Strom durch die Hochspannungsbrücke

Tabelle 3.2: DAB spezifische Variablen

3.2.4 Konfiguration der DAB

Das Verhalten der DAB kann über folgende Variablen und Konstanten beeinflusst werden. Die Tabelle 3.3 ist nach den Files, in denen die Konstanten definiert sind, sortiert.

Variable	Beschreibung
	'User C Files/main.c':
#define BBCINP	Wenn BBCINP definiert ist, wird der Code für die Input Stage des BBC kompiliert. Der Code ist dann nicht mehr konsistent, da die DAB die AD-Wandler der Input Stage verwendet.
#define DELCONST	Wenn definiert, wird der PI-Regler der DAB ignoriert und das konstante Delay `CONSTDELAY` verwendet.
	'Include/GLBDEF.H':
DABLV_DEADTIME, DABHV_DEADTIME	Auswahl der Totzeit für High und Low Voltage Brücken.
	'Include/main.h':
LV_KP, LV_WIT	Koeffizienten des PI-Regler, siehe Kapitel 3.2.1 und 3.2.6.
DAB_MIN_DELAY = -0.25, DAB_MAX_DELAY = 0.25	Sättigungswerte für `delay`
DAB_ZER_OFFS_DELAY = 0.25	Offset des `delay`, welcher einem Leistungstransfer von 0 entspricht.
VLV_BASE = 14V	Basis der pu Notation, entspricht 1pu und wird durch den Integer 2^{13} (8192) repräsentiert. Somit können Werte von -4 bis 4pu (56V) dargestellt werden.
ulv_ref = VLV_SETPT = 14V	Referenzspannung für Regler der DAB-Niederspannungsseite. Ist in pu Notation gleich `0x2000` für `VLV_BASE=14V` und `VLV_SETP=14V`.
int sensv_ulv, offs_ulv	Korrekturen für den AD-Wandler der gemessenen Spannung `ulv` der Niederspannungsseite
int sensv_ihv, offs_ihv	Korrekturen für den AD-Wandler des gemessenen Stroms `ihv` der Hochspannungsbrücke
V_LV_TRIP_H = 20V	Schwellwerte für Fehlerbedingungen `OV_TRIP_LV_H`, wird in `fault_detection()` verwendet (siehe Kapitel 3.2.1). Angegeben in pu mit der oben gewählten Basis. Zum Beispiel entspricht `V_LV_TRIP_H=1.4` einer Überspannungsschwelle von `1.4*14V=20V`

Variable	Beschreibung
V_LV_TRIP_L	Wird zur Zeit ignoriert.
START_DAB_TH = 110V	DAB läuft nur, solange VDC über dieser Spannung liegt.
CONSTDELAY	Konstantes `delay` für deaktivierten PI-Regler für Testzwecke

Tabelle 3.3: Konfiguration der DAB

Addr	15 13	12	11 0
0x0	IntDivMode	DblUpdRate	FWin
0x1	VecIn1	IZG	TauIn1
0x2	VecIn2	---	TauIn2
0x3	VecOut1	OZG	TauOut1
0x4	VecOut2	---	TauOut2
0x5	DabLvVectA	DabLvZG	DabLvTau1A
0x6	DabLvVectB	DFlag	DabLvTau1B
0x7	DabHvVect	DabHvZG	DabHvTau1
0x8	DTimeModeIn	---	DTimeModeOut, DTimeModeDabLv, DTimeModeDabHv

Tabelle 3.4: Memory-Mapping des FPGA

3.2.5 Kommunikation zum FPGA

Nachdem der DSP die neuen Pulsmuster-Daten berechnet hat, werden sie über den Bus ans FPGA geschickt. Der Bus ist 16Bit breit und wird in den Adressbereich des DSP gemappt. Von dort aus kann er mit dem Befehl `dm()` angesprochen werden [9, S. 174 ff]. Dabei ist wichtig, dass die Daten nach aufsteigender Adresse sortiert geschrieben werden.

Das Format des Memory Mappings kann der Tabelle 3.4 entnommen werden. Es ist darauf ausgelegt, möglichst kompakt zu sein. Die Bedeutung der Variablen sind in der Tabelle 3.5 aufgeführt. Viele dieser Variablen werden direkt aus den globalen Variablen des DSP entnommen (Kapitel 3.2.3, Tabellen 3.1 und 3.2).

3.2.6 Regelung

Obwohl eine Regelung implementiert ist, wird die DAB zur Zeit im gesteuerten Betrieb verwendet. Die Auslegung der Regelung war nicht Teil dieser Arbeit. Im nächsten Semester wird diese in Form einer Semesterarbeit ergänzt.

Details zur Implementation der PI-Regler stehen im Kapitel 3.2.1 auf Seite 19 und die dazugehörigen Konfigurationsoptionen sind im Kapitel 3.2.4 auf Seite 26 aufgeführt.

Variable	Beschreibung	Bits
IntDivMode	Modus des IntDividers steuert nach wievielen Schaltperioden das FPGA einen Interrupt ans DSP sendet. `INTR_DIV_ARG = /INTR_DIV: 0 =/1, 1=/2, 2=/4, 3=/8`	3
DblUpdRate	Wird momentan ignoriert, würde Updates vom DSP jede Halbperiode erlauben.	1
FWin	Länge einer Halbperiode in Clock Cycles à 20ns. Entspricht `Tpulsehalf = 1250`.	12
TauIn1, TauOut1	Dauer der Schalterstellung `VecIn1` respektive `VecOut1`	12
TauIn2, TauOut2	Dauer der Schalterstellung `VecIn2` respektive `VecOut2`	12
VecIn1, VecIn2, VecOut1, VecOut2	Vektoren für Schalterstellungen	3
DabLvTau1A, DabLvTau1B, DabHvTau1	Längen des ersten Segments der Halbperiode in Clock-Cycles à 20ns	12
DabLvVectA, DabLvVectB, DabHvVect	Schalterstellungen der DAB-Brücken	1
IZG, OZG, DabLvZG, DabHvZG	Zero Gate Signale	1
DTimeModeIn, DTimeModeOut, DTimeModeDabLv, DTimeModeDabHv	gewünschte Dead Times	3
DFlag	Debug Flag, wird ignoriert	1

Tabelle 3.5: Variablen

3.3 FPGA

Auf dem Board befindet sich ein FPGA vom Typ `LCMX02280C` der Familie MachXO von Lattice Semiconductor. [17]

Der VHDL-Code baut auf dem des Matrix Converter Moduls auf, welches für die Ansteuerung des BBC verantwortlich ist. Weitere Dokumentation zu diesem Code kann in [12] gefunden werden.

3.3.1 Signale

In Tabelle 3.6 sind die externen Ports des FPGAs, nach ihrer Herkunft sortiert, aufgeführt.

Weitere Details zum Timing und Werten der vom DSP kommenden Signalen sind in den Manuals des Bausteins [8] und [9] dokumentiert.

Port	Beschreibung
Signale vom DSP:	
DspMS2	Memory Select vom DSP (`0` für FPGA)
DspAddr	Adresse bezeichnet den Index in `MCM_Data`
DspData	16bit Daten vom DSP
DspWr	Write Strobe vom DSP. Ist aktiv, wenn gültige Daten anliegen. Für Details des Timings siehe Kapitel “External Port Write Cycle Timing” in [8, S. 26].
Control-Signale:	
Clk100	100MHz Clock.
PulseInhibit	Fehlersignal vom RMX[1,2]
EmStop	Signal vom Not-Aus-Schalter[1,2]
Reset1	Reset[2] vom RMX.
Reset2	Reset[2] der Spannungsüberwachung vom Reset IC
Output:	
DspIntDiv	Interrupt, welcher den Control-Loop des DSPs triggert.
S*	Schaltersignale zu den Brücken
DFlag*	Debug Flags

[1] Führt zur Beendigung aller Schaltaktivität.
[2] Ist active-low.

Tabelle 3.6: Ports des FPGAs

Die internen Signale und ihre Bedeutungen sind in [12] aufgeführt.

Interne Signale verschiedener Unterblöcke haben jeweils die gleiche Bedeutung, wenn sie sich nur durch ihr Suffix unterscheiden. Der BBC verwenden die Codes `In` und `Out` für die Input- respektive Output-Stage. Komponenten der Dual-Active-Bridge haben `DabHv` und `DabLv` für Hoch- und Niederspannung. Die Niederspannungsseite spaltet sich noch auf zwei Brücken `DabLvA` und `DabLvB` auf.

3.3.2 Blöcke

MCM

Das MCM ist die Top-Level-Entity. Sie kann in BBC-, DAB und gemeinsame Komponenten unterteilt werden. Eine Übersicht der Blöcke und ihre Schnittstellen bietet die Abbildung 3.5.

`MemoryI`
: `MemoryI` empfängt und speichert die Daten der Pulsmuster der nächsten Perioden. Sie gelangen über den Bus vom DSP ins FPGA. Das verwendete Format ist im Kapitel 3.2.5 beschrieben. Getaktet wird das ganze durch `DspWr`, also mit etwa 30MHz ([8, S. 26]). Diese Daten werden so lange gespeichert und verwendet, bis neue vom DSP eintreffen.

  Unter anderem berechnet `MemoryI` auch die `Tau3`s des BBC respektive `Tau2`s der DAB. Sie entsprechen der Ergänzung der anderen Taus der gleichen Brücke zur halben Pulsperiode `FWin`.

`Sequencer`
: Der `Sequencer` stellt die `Tau`s und Vektoren des nächsten Abschnitts des Pulsmusters bereit. Er lädt alle Perioden neue Daten vom `MemoryI` und sendet den DSP-Interrupt an den `IntDivider`. [12]

`DabBridge`
: Steuert die Schalter einer Brücke an. Die Pulsmusterdaten werden jede Periode vom `MemoryI` geladen.

`Timer`
: Der Countdown im `Timer` zählt die Zeit bis zum nächsten Schaltvorgang. Sobald der Zähler auf 1 steht, wird ein neuer Countdown mit dem nächsten `Tau`-Wert vom `Sequencer` gestartet.

  Bei den Zählerständen 1 und 3 werden Synchronisationssignale `TimerOne` und `TimerThree` gesendet. [12]

`DelRisingEdge`
: Dieser Block generiert die eigentlichen Schaltersignale. Er verzögert die steigenden Flanken der Ausgangssignale. So können die durch die Schaltverzögerungen t_{on} und t_{off} der Schalter verursachten Querströme über die Brücken vermieden werden. [12]

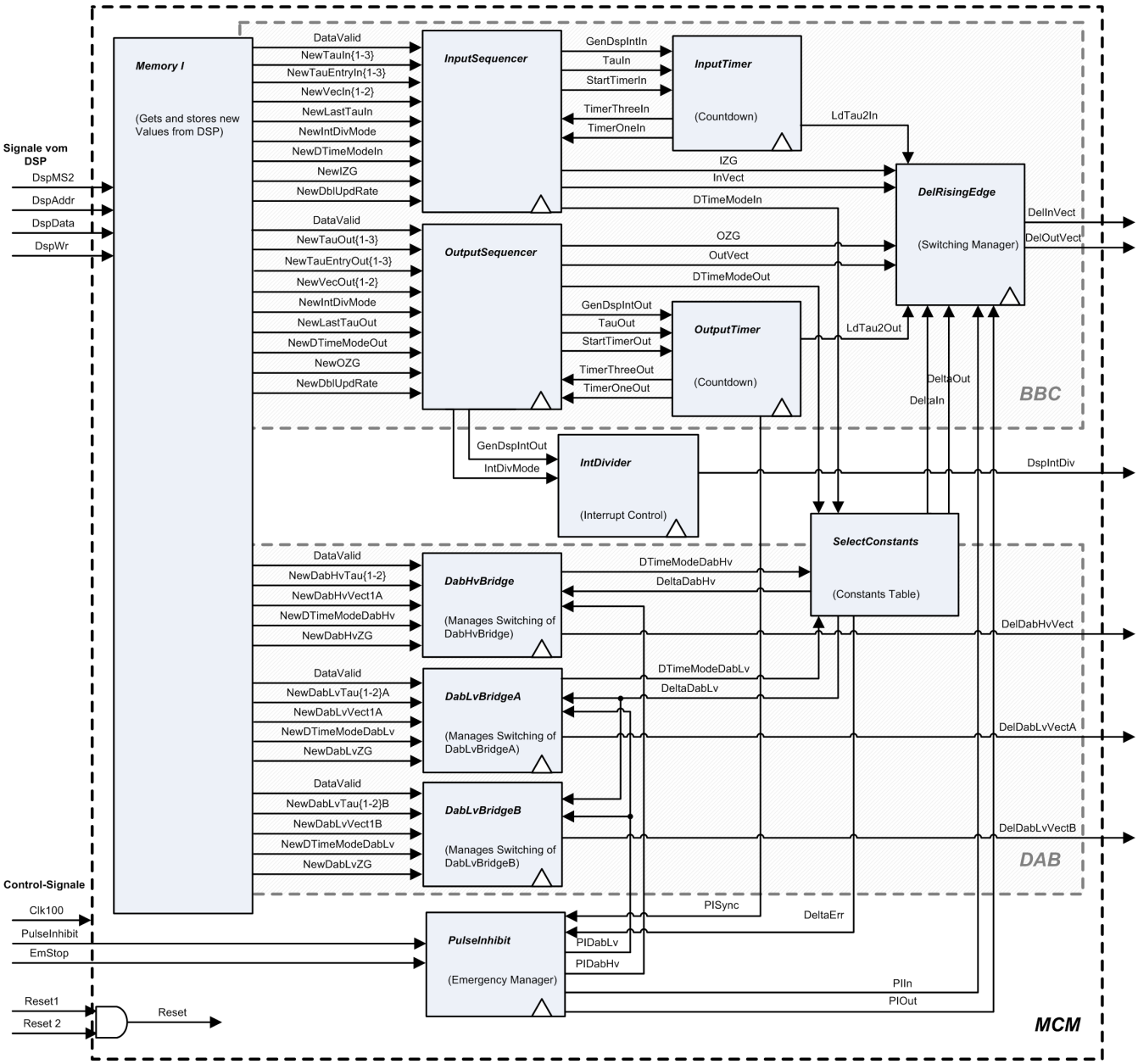


Abbildung 3.5: Blöcke des MCM

`IntDivider`

Der DSP kann über die Variable `IntDivMode` steuern, nach wie vielen Perioden er jeweils ein Interrupt bekommt. `IntDivider` reicht Interrupts vom `Sequencer` nur in der gewünschten Frequenz weiter.

Per Default wird jede zweite Periode ein Interrupt ausgelöst.

`PulseInhibit`

Dient der Schnellabschaltung im Fehlerfall. Ursache dafür ist entweder `PulseInhibit` von RMX, wenn ein Messwert ausserhalb des "grünen Bereichs" liegt, oder ein `EmStop` vom Not-Aus-Schalter. Dies veranlasst die Entity `PulseInhibit` die Ausgangsstufe via Signale an die `DelRisingEdge`-Entities in den Freilaufzustand zu bringen und mit der Zeit die Brücken abzuschalten. [12]

`SelectConstants`

Beinhaltet eine Tabelle der Zeitkonstanten `DeadTime`s und `DeltaErr`, welche mittels der `DTimeMode` Eingänge abgefragt werden können.

`ClockDivider`

Teilt den eingehenden Clock von 100MHz auf 50MHz. Damit alle Timing Constraints erfüllt werden können.

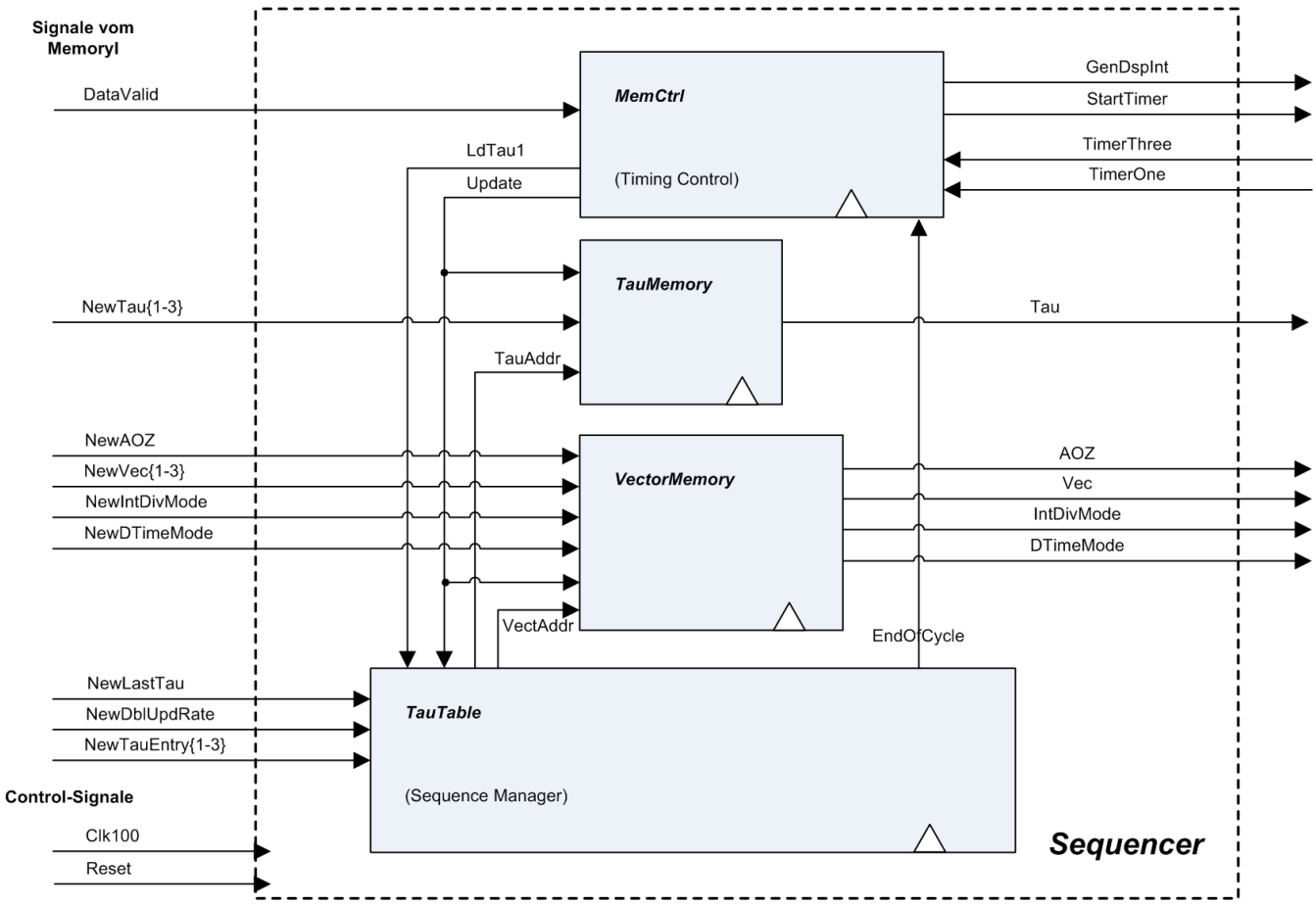


Abbildung 3.6: Blöcke des `Sequencer` des BBCs

BBC

Die Blöcke des `Sequencers` des Back-to-Back-Konverters sind in [12] beschrieben. Die Abbildung 3.6 zeigt den internen Aufbau.

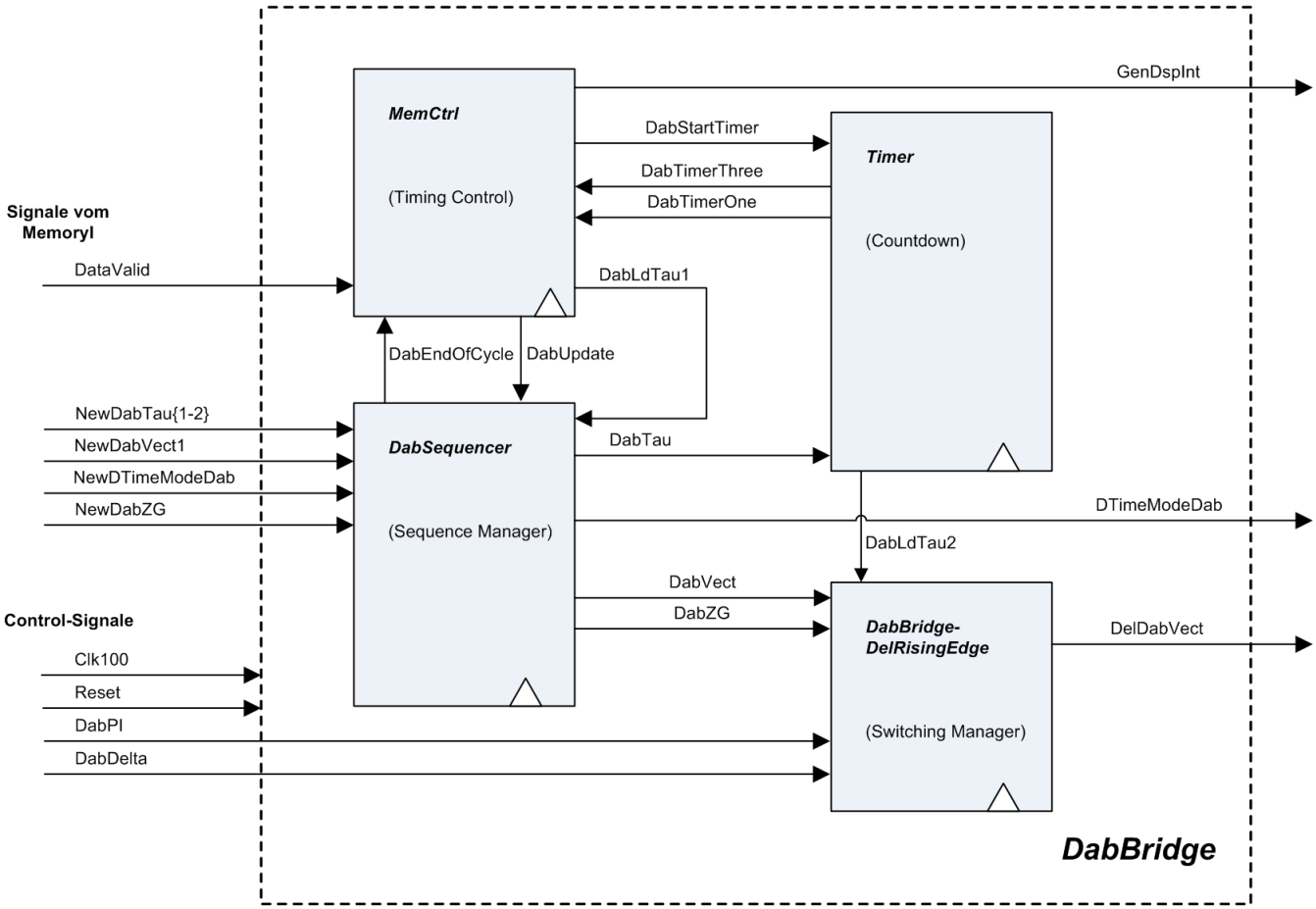


Abbildung 3.7: Blöcke der `DabLvBridge` und `DabHvBridge`.

DAB

Die `DabBridge` besteht aus den Entities auf der Abbildung 3.7. Ihr Aufbau ist ähnlich dem des BBCs, nur das jede Brücke einzeln angesteuert wird. Die `DabLvBridge` und die `DabHvBridge` unterscheiden sich durch verschiedene `DabSequencer`.

`DabSequencer`

Der `DabSequencer` ist der Koordinator einer Brücke. Er lädt die Pulsmusterdaten von `MemoryI` herunter und führt sie in der richtigen Reinenfolge aus. Am Ausgang liegen *immer* die nächsten Daten bereit.

Mit `EndOfCycle` wird an `MemCtrl` signalisiert, dass das nächste vom Timer gelesene `Tau` das letzte der Periode ist.

`DabLvSequencer`: Erzeugt eine mittelwertsfreie Pulsform für den Phase-shift (siehe Kapitel 3.1.1).
`DabHvSequencer`: Erzeugt eine Pulsform mit dem gewünschten Mittelwert (siehe Kapitel 3.1.2).

Für genauere Informationen zu den Pulsmustern siehe Kapitel 2.2 und 2.3.2.

`MemCtrl`

Koordiniert das Laden neuer Werte. Dazu gehört der Nachladebefehl `Update`, wenn eine Periode abgelaufen ist und die neuen Daten von `MemoryI` geladen werden können.

Beim Ablaufen des Timers auf `TimerOne` aktiviert `LdTau1` das Aktualisieren auf die übernächsten Werte, während der `Timer` und `DabBridgeDelRisingEdge` die aktuellen Werte am Ausgang für den nächsten Abschnitt lesen.

`Timer`

Der `Timer` ist im oberen Unterkapitel beschrieben.

`DabBridgeDelRisingEdge`

Dieser Block ist im oberen Absatz beschrieben. Der einzige Unterschied zum `DelRisingEdge` ist, dass `DabBridgeDelRisingEdge` nur zwei und das Gegenstück des BBC drei Schalter ansteuert.

Funktionen des MachXO

Von der Funktionalität des `MachXO`-FPGAs wird zur Zeit nur der `GSR` verwendet. Mit dem `PLL` könnte der Systemtakt noch auf 75MHz gesteigert werden.

Global Set Reset (`GSR`)

Der Global Set Reset ist eine spezielle Leitung, welche mit dem Set- und Reset-Eingang jedes Flip-Flops des FPGAs verbunden ist.

Durch Definition der `GSR`-Pseudoentity wird diese vorgeroutete Leitung verwendet und mit einem bestimmten Signal verknüpft. Dadurch können Leitungen und Platz gespart werden, was mehr Performance bringt.

Das Reset-Signal `Reset` wird von der Software automatisch erkannt und für den `GSR` verwendet. [16, S. 114]

Phase Locked Loop (`PLL`)

Das FPGA besitzt zwei sysCLOCK PLLs, mit denen die Frequenz eines eingehenden Clock-Signals herunter geteilt und multipliziert werden kann. Es könnte auch ein Delay eingeschaltet werden.

Das PLL kann über die Entity `PllClkDiv` benutzt werden. Diese wurde vom Programm IPexpress der ispLEVER Design Software konfiguriert und erzeugt. So kann der Input-Clock `Clk100` von 100MHz auf eine beinahe beliebige Frequenz herunter geteilt oder multipliziert werden. [18], [17, S. 18, 157ff]

Die Ports des PLL sind:

CLK ist der zu teilende Input-Clock (100MHz)
RESET initialisiert alle internen Zustände und ist active high. Mit Vorteil wird hier ein anderes Signal als der Reset der restlichen Logik verwendet.
CLKOP der generierte Output Clock der konfigurierten Frequenz.
LOCK ist 1, wenn der gewünschte Output Clock erreicht und gelockt ist. Sobald im Betrieb der Input-Clock oder Feedback-Signale ungültig werden, verliert das PLL den LOCK und es muss ein Reset durchgeführt werden.

Das PLL wird momentan aus Stabilitätsgründen nicht verwendet. Um einen fehlerfreien Clock zu garantieren, müsste eine Logik implementiert werden, die das PLL bei Verlust des Locks wieder zurücksetzt ([17, S. 159]). Sie dürfte sich nicht irritieren lassen, von kurzzeitigen Schwankungen des LOCK Signals.

3.3.3 Zeitverhalten

Das Zeitverhalten des BBC- und des DAB-Teils sind sich recht ähnlich.

Zeitverhalten einer BBC-Stage

Initialisierung

Takt	Entity	Signal	Wirkung
1	**MemoryI**	DataValid	Daten sind gültig
2	**MemCtrl**	Update	Veranlasse TauMemory, VectorMemory und TauTable neue Werte von MemoryI zu lesen.
3	**TauMemory** **VectorMemory** **TauTable**	---	Laden neue Daten von MemoryI, diese liegen im nächsten Takt bereit.
4	**MemCtrl**	StartTimer LdTau1	Timer soll Tau von TauTable laden TauTable soll übernächstes Tau berechnen
5	**TauTable** **TauMemory** **VectorMemory**	TauAddr Tau Vec	Adresse des übernächsten Taus Wert des übernächsten Taus Übernächster Vektor

Running, während eines Zyklus

Takt	Entity	Signal	Wirkung
1	**Timer**	TimerOne	Timer abgelaufen und lädt nächstes Tau
	MemCtrl	LdTau1	TauTable soll übernächstes Tau berechnen
2	**TauTable**	TauAddr	Adresse des übernächsten Taus
	TauMemory	Tau	Wert des übernächsten Taus
	VectorMemory	Vec	Übernächster Vektor

Running, Ende eines Zyklus

Takt	Entity	Signal	Wirkung
x	**MemoryI**	DataValid	Daten vom DSP sind gültig
y	**TauTable**	EndOfCycle	Das jetzt anliegende, also das nächste vom Timer verwendete Tau, ist das letzte.
1	**Timer**	TimerThree	Counter ist drei
	MemCtrl	GenDspInt	Sende Interrupt-Signal an IntDivider
		Update	Veranlasse TauMemory, VectorMemory und TauTable neue Werte aus MemoryI zu lesen.
2	**TauMemory**	---	Laden Daten von MemoryI
	VectorMemory		
	TauTable		
3	**Timer**	TimerOne	Timer abgelaufen und lädt nächstes Tau
	MemCtrl	LdTau1	TauTable soll übernächstes Tau berechnen
4	**TauTable**	TauAddr	Adresse des übernächsten Taus
	TauMemory	Tau	Wert des übernächsten Taus
	VectorMemory	Vec	Übernächster Vektor

Zeitverhalten einer DAB-Brücke

Initialisierung

Takt	Entity	Signal	Wirkung
1	**MemoryI**	DataValid	Daten sind gültig
2	**MemCtrl**	DabUpdate	Veranlasse DabSequencer neue Werte von MemoryI zu lesen.
3	**DabSequencer**	---	Lädt Daten von MemoryI
4	**MemCtrl**	DabStartTimer	Starte Timer
		DabLdTau1	DabSequencer soll übernächstes Tau berechnen
	DabTimer	DabLdTau2	DabDelRisingEdge soll nächste Vektoren laden
			DabTimer lädt nächstes Tau
5	**DabSequencer**	DabTau	Übernächstes Tau
		DabVect	Übernächster Vektor

Running, während eines Zyklus

Takt	Entity	Signal	Wirkung
1	**DabTimer**	DabTimerOne	Timer abgelaufen und lädt nächstes Tau
		DabLdTau2	DabDelRisingEdge lädt nächste Werte
	MemCtrl	DabLdTau1	TauTable soll übernächstes Tau berechnen
2	**DabSequencer**	DabTau	Übernächstes Tau
		DabVect	Übernächster Vektor

Running, Ende eines Zyklus

Takt	Entity	Signal	Wirkung
x	**MemoryI**	DataValid	Daten vom DSP sind gültig
y	**DabSequencer**	EndOfCycle	Das jetzt anliegende, also das nächste vom DabTimer verwendete Tau, ist das letzte
1	**DabTimer**	DabTimerThree	Counter steht auf drei
	MemCtrl	DabUpdate	DabSequencer soll neue Werte aus MemoryI lesen.
2	**DabSequencer**	---	Lädt Daten von MemoryI
3	**DabTimer**	DabTimerOne	Timer abgelaufen und lädt nächstes Tau
		DabLdTau2	DabDelRisingEdge lädt nächste Werte
	MemCtrl	DabLdTau1	TauTable soll übernächstes Tau berechnen
4	**DabSequencer**	DabTau	Übernächstes Tau
		DabVect	Übernächster Vektor

3.3.4 Optimierungen

Der VHDL-Code lief ursprünglich verteilt auf zwei PLDs. So wurde eine Art Pipelining über die Bausteine realisiert. Auf dem neuen Board ist aber nur noch ein FPGA eingebaut. Unglücklicherweise hat das Neue das langsamste Speedgrade von `-3` und damit können die Berechnungen nicht mehr in der geforderten Taktfrequenz von 100MHz ausgeführt werden.

Deshalb wird der eingehende Clock mit dem `ClockDivider` auf 50MHz herunter geteilt (siehe Kapitel 3.3.2). In einer späteren Ausbaustufe kann mit dem schnelleren Speedgrade `-5` der Takt problemlos auf 100MHz erhöht werden (siehe Kapitel 3.3.4).

Dieses Kapitel beschreibt einige Optimierungen, wie VHDL-Code performanter gemacht werden kann. Zudem werden einige Änderungen am ursprünglichen Code beschrieben, welche im Zuge der Anpassung auf das neue FPGA vorgenommen wurden. Einige dieser und weite Techniken sind in [17, S. 197ff] und [20] beschrieben.

Beim Optimieren sollte man immer zuerst den langsamsten Pfad finden. Wenn man weiss, welches die limitierenden Faktoren sind, kann man eine der unten aufgeführten Methoden anwenden. Es ist wichtig, nach jedem Schritt eine Erfolgskontrolle durchzuführen um bei allfälligen Regressionen nochmals über die Bücher zu gehen.

Oft verwirren schon kleine Änderungen den Compiler derart, dass er weniger gute Optimierungen durchführen kann und in langsameres Design erzeugt.

Überflüssige Bits abspecken

Jedes Bit kostet Zeit. Deshalb ist es wichtig, dass man zum Beispiel unbenötigte Resultate nicht berechnet. Wenn man im Hinblick auf zukünftige Erweiterung zusätzliche, zur Zeit brachliegende, Funktionalitäten programmiert, muss man sie aber nicht unbedingt löschen. Oft reicht

es, sie nicht nach aussen zu führen, damit das Synthese-Programm die Logik weg rationalisiert. Kandidaten für diese Optimierungen waren `DblUpdRate` und der `resetsync`. Beide werden momentan nicht verwendet, könnten aber einfach wieder aktiviert werden. Die `DFlag`s werden nur zu Debugzwecken verwendet, später können diese Ouput-Ports des MCMs einfach mit konstanten Werten versorgt oder ganz weggelassen werden.

Signale sollten nur so lang wie nötig sein. Besonders offensichtlich gilt dies für die Counter der Timer, da muss der Übertrag sequenziell durch alle Bits rippeln. Aus diesem Grund wurde der `delTimer` von 8 auf 7 Bit reduziert und im `IntDivider` nur noch 3 anstatt 5 Bits verwendet.

Timer aufspalten

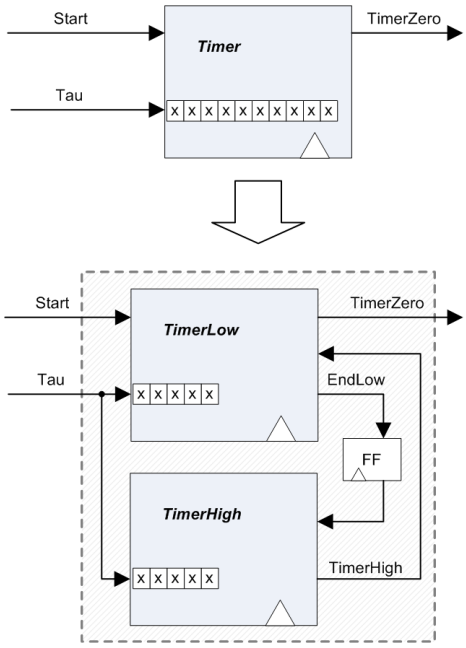


Abbildung 3.8: Timer aufspalten

Beim dekrementieren des Counters muss der Übertrag durch die ganze Breite des Registers rippeln. Dies kostet einige Zeit, besonders bei den 12 Bit Countern in den `Timer`n aber auch in den `DelTimer`n, welche 7 Bit Counter benutzen.

Dieser Effekt wird drastisch reduziert, wenn man den Counter, wie in der Abbildung 3.8 ersichtlich, auf zwei oder mehrere aufteilt. Der `CounterLow` mit den niederwertigen Bits informiert dabei jeweils den nächst Höheren über ein Signal `EndLow`, dass er im nächsten Takt Null wird. Im nächsten Takt dekrementiert der `CounterHigh`. Es hat also jeder Counter eine Taktperiode Zeit für die Berechnung, welche neu aber nur halb so viele Bits umfasst.

So kann eine Art Pipelining realisiert werden. Der Geschwindigkeitszuwachs beim Aufspalten der Berechnung auf beliebig viele Counter geht auf Kosten zusätzlicher Signale und Logik

für die Terminierungserkennung. Die Balance zu finden ist nicht ganz einfach und kann eigentlich nur durch Ausprobieren getroffen werden.

Konkret hat in unserem Projekt die Aufspaltung des `Timers` auf drei 4 Bit Counter und die des `DelTimers` auf einen 4 und 3 Bit Counter einen Geschwindigkeitszuwachs im zweistelligen Prozentbereich gebracht.

Pipelining

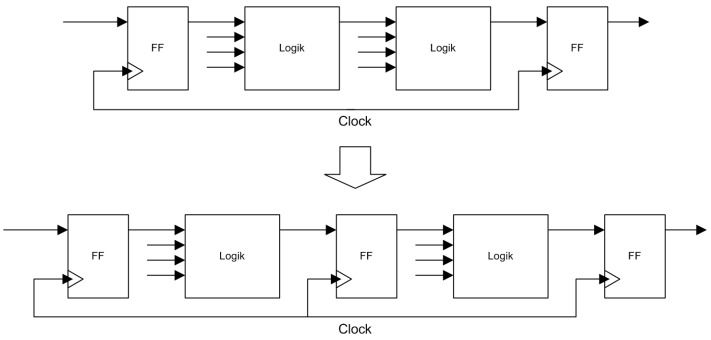


Abbildung 3.9: Pipelining

Wenn es um Optimierungen von VHDL-Code geht, denkt man zuerst an Pipelining. Dies ist auch hier nahe liegend, zumal noch über die Hälfte der SLICEs frei sind. Leider sind die Möglichkeiten bei gegebener Architektur ziemlich beschränkt.

Gute Möglichkeiten für Pipelining finden sich im `MemoryI`. Dort kann zum Beispiel die Berechnung von `Tau3` auf zwei Takte aufgespalten werden. In den Sequence Managers `TauTable` und `DabSequencer` wird vor zu der erste Zustand des Zustandsautomaten (FSM) berechnet.

FSM optimieren

Im FPGA verrichten über 10 verschiedene endliche Zustandsautomaten ihren Dienst. Sie werden oft in Kernbestandteilen, wie der Ablauflogik, und in Controllern verwendet.

Es hat sich gezeigt, dass es sich lohnt möglichst viele Zustände zu verwenden. Eine FSM wird schneller, wenn man mehr Zustände und dafür weniger Logik in den einzelnen Zuständen verwendet. Dies hängt mit dem Unterschied zwischen `case`- und `if`-Verzweigungen zusammen. Darauf wird im Abschnitt ‘Logik vereinfachen’ näher eingegangen.

Ein wichtiger Faktor bei FSMs ist die Zustandskodierung, also deren binäre Repräsentation. Synplify® erkennt Zustandsautomaten und versucht, eine geeignete Kodierung zu wählen [20, S. 45]. Über das Attribut `syn_encoding` kann man dies beeinflussen [20, S. 220]. Wenn genügend Fläche auf dem FPGA zur Verfügung steht, ist `onehot` die beste Wahl. Dabei wird jeder Zustand in einem eigenen Bit kodiert, bei Vergleichen muss nur dieses eine Bit berücksichtigt werden. Es gibt auch die Möglichkeit, eine eigene Kodierung zu verwenden (`original`), oder den FSM Compiler für eine FSM zu deaktivieren (`syn_state_machine`).

Der `DabSequencer` verwendet die Direktive `onehot`, was auch dem Default entspricht.

Die `TauTable` ist etwas spezieller. Erstens wurden so viele Zustände eingeführt, bis der Output ohne weitere Logik vom Zustand abhängt. Dieser liegt so schneller an. Zweitens sind die Ausgänge direkt in die Zustandsrepräsentation kodiert. Und es kann weitere Logik eingespart werden. Dies verbesserte die Performance stark, was den grossen Aufwand rechtfertigt. Leider ist dies nicht generell der Fall, sondern funktioniert nur für sehr komplizierte FSMs.

Logik vereinfachen

Die ideale Ausnutzung der Ressourcen ist erreicht, wenn alle Pfade gleich lang sind. Wenn also nicht eine lange Kette von Berechnungen die Kurzen ausbremst.

Es ist wünschenswert möglichst wenige Logikebenen zu haben. Vielfach bringt diese Reduktion aber nur wenig oder gar keine Gewinne. Dies liegt an der Arbeitsweise des Compilers. Wenn er jeden Ausgang als Funktion der Eingänge darstellt und vereinfacht, erkennt und entfernt er viele Redundanzen. Dennoch lohnen sich manuelle Optimierungen.

Zum Vergleich der verschiedenen Verzweigungsanweisung dient das folgende Codestück (aus [17]) als Illustration.

```
if s1 = '1' then
   O1 <= x;
elsif s2 = '1' then
   O2 <= y;
elsif s3 = '1' then
   O3 <= z;
end if;
```

```
if s1 = '1' then
   O1 <= x;
end if;
if s2 = '1' then
   O2 <= y;
end if;
if s3 <= '1' then
   O3 <= z;
end if;
```

Der Compiler übersetzt den Output `O3` im linken Fall zu:

```
O3 <= z and (s3) and (not (s1 and s2));
```

und im rechten Fall:

```
O3 <= z and s3;
```

Dementsprechend sollte man auf `elsif` verzichten, wenn man die Priorisierung nicht unbedingt braucht. Im gleichen Sinne sollte man wenn möglich `case` Anweisungen benützen. Bei diesen kommt es, ungleich den `elsif`, nicht darauf an, wie viele Fälle man unterscheidet.

Eine kleinere, allgemeine Optimierung ist noch erwähnenswert. Vergleiche mit Null sind am effizientesten und sollten wenn möglich verwendet werden.

Synplify® Optimierungen

Synplify® ist ein mächtiges Synthese Werkzeug, welches viele Optimierungen beherrscht. Es ist ausserordentlich vielseitig konfigurierbar. Hier werden einige seiner Optionen mit ihren Wirkungen und Gefahren beschrieben. Die meisten der Konfigurationsanweisungen kann man entweder im Constraint File in Synplify® angeben, oder im 'Design Planner' des 'ispLEVER Project Navigator' konfigurieren. Einige Optionen findet man dort auch in den Properties zu 'Place & Route Design' (Rechts-klick).

Die wichtigsten Angaben sind jeweils die gewünschten Frequenzen der verschiedenen Clock-Nets. So priorisiert man die kritischen Netze und erlaubt dem Compiler, langsamere weniger vorteilhaft anzuordnen. Anderen kritischen Netze kann man auch direkt Prioritäten zuweisen. Dabei ist 3 der Standardwert und alle höheren entsprechen wichtigeren Netzen. Es dürfen aber auf keinen Fall solche priorisiert werden, die ihre Timing Constraints sowieso erfüllen. Es bringt nichts, die noch weiter auf Kosten der zu Langsamen zu optimieren.

Ein mächtiges Attribut, welches man in VHDL-Code jeder Entity spezifizieren kann, ist `syn_hier`. Damit kann kontrolliert werden, wie viel der ursprünglichen Hierarchie des Codes erhalten bleiben soll. Nach der Auflösung der Hierarchie kann das Programm oftmals weitere Signale und Logik über hierarchische Grenzen hinweg vereinfachen. [20, S. 243]

3.4 Arbeitsweise und Organisation des Quellcodes

In diesen Kapitel will ich Tools vorstellen, die ich verwendet habe. Ich will kurz beschreiben, wie die Verzeichnisse organisiert sind und wo was abgelegt ist. Dies sollte das Nachvollziehen und Weiterführen dieser Arbeit erleichtern.

3.4.1 Verzeichnisse

Auf der CD befindet sich das Verzeichnis `Software`. Darin sind sämtliche Quelltexte und zum Betrieb und Test nötigen Dateien abgelegt.

VHDL-Code des FPGAs

Das Verzeichnis `MCM` beinhaltet alle mit dem VHDL-Code verwandte Files in folgenden Unterverzeichnissen. VHDL-Files sind nach ihren Entities benannt und besitzen meist zusätzlich ein Suffix. Dieses setzt sich aus einem Versionsbuchstaben und eventuell `'_tb'` für Testbenches zusammen.

MCM/
Das Toplevel-Directory enthält das ispLEVER Projektfile `mcm.syn` und andere, von diesem Programm benötigte, Files.

BBC/
Die vom Back-to-back Konverter verwendeten Entities.

DAB/
Entities der Dual Active Bridge

common/
Gemeinsam verwendete Entities und Testbenches

lattice/
Die Lattice- und FPGA-spezifischen Files, siehe auch unter `modelsim`.

modelsim/
Da in Modelsim einige FPGA-spezifischen Entities wie der `GSR` oder das `PLL` nicht verfügbar sind, befindet sich hier eine leicht angepasste Version der Toplevel-Entity `mcm`.

old/
Ist ein Sammelverzeichnis für alte oder nicht verwendete Versionen verschiedener Entities. Nicht alle sind lauffähig und einige enthalten Bugs. Sie werden zu Dokumentationszwecken aufbewahrt.

Hier befinden sich auch für andere Clockfrequenzen angepasste `SelectConstants`.

Tools/
Dies ist ein Sammelbehälter für diverse Files.

Dazu gehören von Modelsim verwendete Files wie Waveforms (`*.do`) und Scripts (siehe auch Kapitel 3.4.2).

C-Code des DSPs

Im Verzeichnis `'BBC u DAB gesteuert'` befindet sich das VisualDSP++ Projekt mit dem entsprechenden C- und Assembler-Quellcode.

Unter `Tools` ist das File `swMonitor.smc`. Dies beinhaltet die Einstellungen des swMonitors. Zudem befindet sich dort `calc-offs-a-sensv.xls`, welches aus Messwerten den Offset und die Skalierung für den AD-Wandler berechnet.

3.4.2 Verwendete Programme und Scripte

Grundsätzlich habe ich den Code in Eclipse entwickelt. Dies bietet die Vorteile einer sehr professionellen IDE mit vielen interessanten Features wie Syntax Highliting, Tab-Completion, integriertes CVS und viele nützliche Editor-Funktionen.

FPGA-Entwicklung

Für das eigentliche Kompilieren des Codes ist man an die Programme des Herstellers gebunden. Wichtig ist, dass man mindestens Version 6 der ispLEVER Design Software installiert. Erst ab dieser Version wird der verwendete Chip vollständig unterstützt.

Getestet habe ich die Entities mit ModelSim. Dieses Programm ist auf dem Tardis-Cluster installiert und lässt sich mit dem Kommando `vsim` starten. `vsim` startet man mit Vorteil im Verzeichnis `MCM`, welches die Entities enthalten sollte.

Zuerst muss man also die `*.vhd` Dateien in ein eigenes Verzeichnis auf den Unix-Rechnern kopieren. Um diesen Vorgang zu vereinfachen, liegt im Verzeichnis `MCM/Tools` das Script `copy-tardis.sh`. Um es zu verwenden, müssen darin noch einige Variablen mit den richtigen Verzeichnissen gesetzt werden. Wenn man nicht jedes mal sein Passwort eingeben will, kann man ein RSA-Keypair mit `ssh-keygen` erzeugen und installieren.

Kompilieren kann man den Quellcode mit dem Script `vsim-compile.sh`, welches auch ins Verzeichnis auf dem entfernten Rechner kopiert werden muss. Dieses Script kommentiert alle Lattice-spezifischen Attribute aus, welche von Modelsim nicht verstanden werden und kompiliert die `*.vhd` Dateien.

In ModelSim muss man dann im Workspace die Simulation der Testbench-Entity starten. Wenn man dies schon getan hat, aber neu kompilierte Entities testen will, muss man in der Konsole von Modelsim `'restart -f'` ausführen.

Dann kann man die Wave-View (Menu: View ⇒Wave) starten und Signale per drag-and-drop hinzufügen oder ein geeignetes `*.do`-File (aus `MCM/Tools`) laden.

Mit dem Kommando `'run n'` wird die Testbench über `n` Nanosekunden simuliert und die erzeugten Signale angezeigt.

DSP-Entwicklung

Für die Entwicklung des DSP-Codes wurden nur Eclipse mit dem CDT Plugin und VisualDSP++ von Analog Devices verwendet.

3.5 Weiterführende Arbeiten

Die beschriebene Software Architektur erlaubt vielseitige Erweiterungen, da sie im Hinblick auf weitergehende Integration und Features designt wurde. Folgende Bereiche können noch erweitert oder optimiert werden.

Die Geschwindigkeitsprobleme des VHDL-Code lösen sich einfach durch ersetzen des FPGAs mit einem des schnellsten Speedgrades. Der Code kann unverändert übernommen werden und muss nur neu kompiliert werden.

Am PES laufen bereits die Vorbereitungen für die Umstellung auf einen neuen DSP. Dieser wird genug AD-Wandler haben, um auch die Input-Stage des Back-to-Back-Konverters zu betreiben. Nachdem die Motorsteuerung in den DSP-Code integriert wird, kann das System mit dem Synchronmotor betrieben und getestet werden.

Ein noch nicht behandeltes Thema ist die Regelung der Dual Active Bridge. Es muss die Abhängigkeit des Leistungsflusses vom Delay des Phase-shifts untersucht und die Regelung entsprechend ausgelegt werden.

Als weiteres Feature könnte man noch optimierte Switchingstrategien untersuchen.

Kapitel 4

Hardware

4.1 Spezifikationen

Im Hardwareteil dieser Arbeit wurde die Gleichrichterschaltung auf der Niederspannungsseite der Dual Active Bridge entworfen und realisiert. Bei erneutem Betrachten der Gesamtschaltung in Abbildung 4.1, ist die angesprochene Gleichrichterschaltung grün eingefärbt. Die anderen Teile des Konverters wurden in anderen Arbeiten gestaltet.

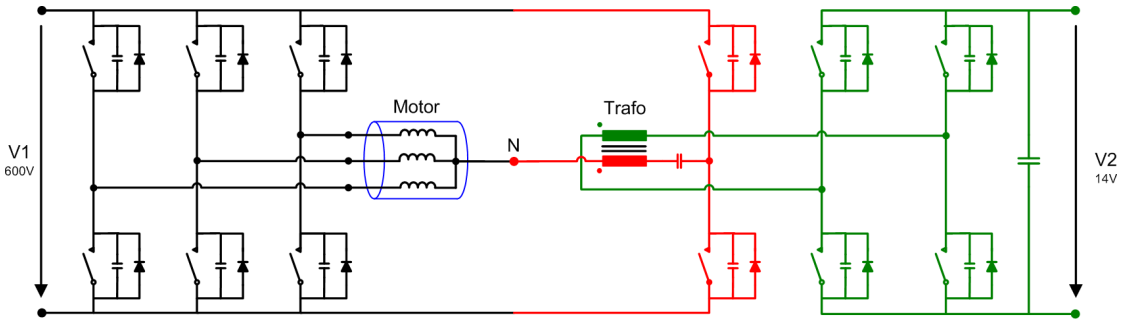


Abbildung 4.1: Gesamtschaltung mit grün eingefärbter Dreiphasenbrückenschaltung

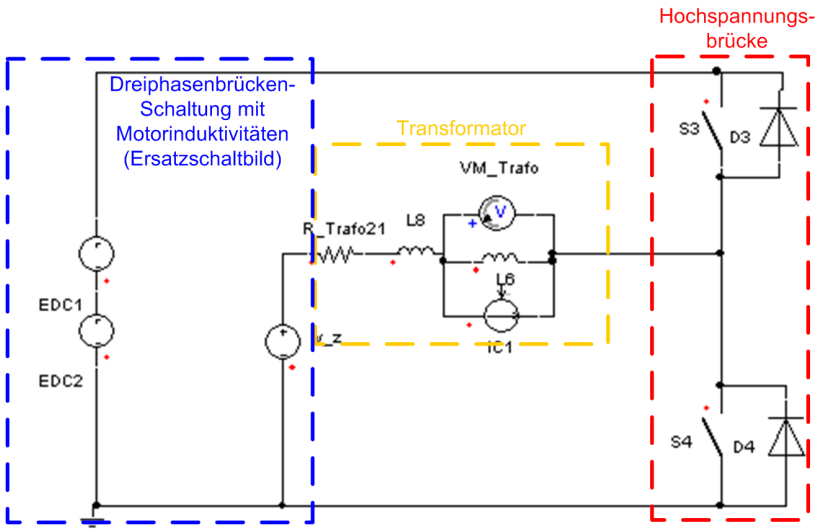
Wie schon Eingangs dieser Arbeit erwähnt, sollte die Schaltung für eine Eingangsspannung von $V_1 = 600V$ und eine Ausgangsspannung von $V_2 = 14V$ ausgelegt werden. Dabei soll die Ausgangsleistung $P_2 = 1kW$ betragen. Die Schaltfrequenz war auf $f_S = 20kHz$ festgelegt.

Eine schon bestehende Simulation erleichterte das Dimensionieren, Auslegen und das Verständnis für den DC/DC-Konverter

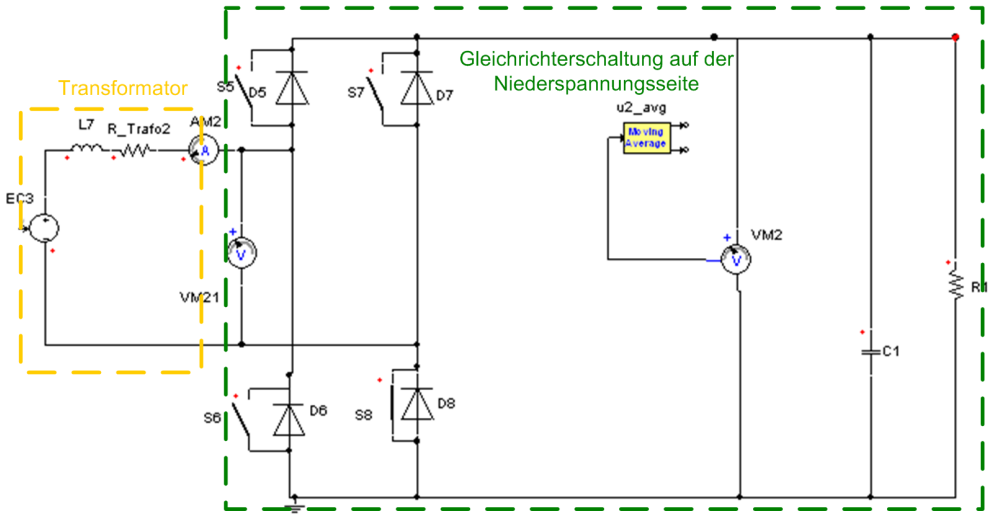
Da die Gleichrichterschaltung schon einmal für eine Dual Active Bridge realisiert wurde, konnten die Schemas, sowie einige Ideen für Anordnung der Bauelemente und die Führung der Leiterbahnen übernommen werden.

4.1.1 Simulationen

Wie erwähnt, wurde uns eine Simulation der Gesamtschaltung zur Verfügung gestellt. Dabei wurden die Dreiphasenbrückenschaltung, der Motor und dessen Motorinduktivität durch den blau umrahmten Teil in der Simulation ersetzt (Abb. 4.2). Die Spannungserzeugung der Dreiphasenbrückenschaltung, sowie der Schaltsignale erfolgte über verschiedene Zustandsautomaten.



Simulation der Dreiphasenbrückenschaltung(Ersatzschaltbild) und der Hochspannungsbrücke mit SIMPLORER 7.0



Simulation der Gleichrichterschaltung auf der Niederspannungsseite mit SIMPLORER 7.0.

Abbildung 4.2: Simulationsmodell der Schaltung mit dem Ersatzschaltbild der Dreiphasenbrückenschaltung

Durch Variation der Phasenmodulationsamplitude M der Dreiphasenbrückenschaltung resultierten stark verschiedene Spannungs- und Stromverläufe. Die Phasenmodulationsamplitude M beträgt maximal $\frac{2}{\sqrt{3}} \approx 1.15$ und minimal 0. Je kleiner M gewählt wird, desto stärker bilden sich Stromextremwerte aus (Abb. 4.3). Es ist zu erwähnen, dass die Simulation für das Phase-Shift-Modulationsverfahren ausgelegt ist. Es gibt ebenfalls andere Modulationsverfahren, mit welchen die stark ausgebildeten Stromspitzen unterbunden werden können. Diese waren aber nicht Teil dieser Semesterarbeit.

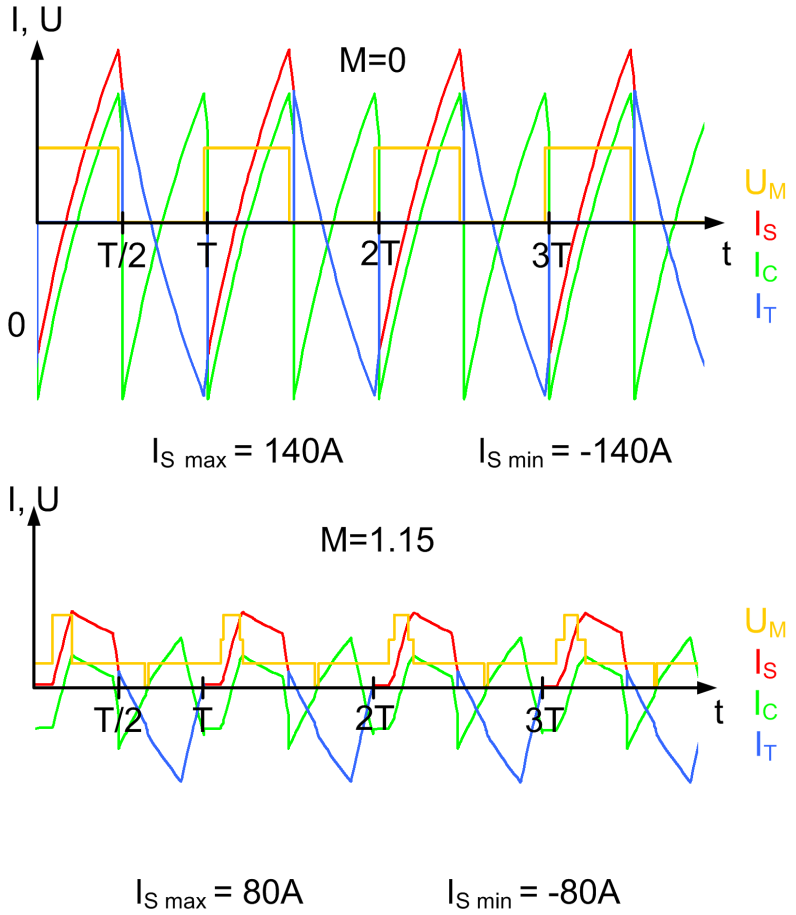


Abbildung 4.3: Vergleich der Spannungs- und Stromverläufe zwischen M=0 und M=1.15

Um geeignete Bauelemente zu finden, wurde von einem $M = 0$ als schlimmste anzunehmende Situation ausgegangen. Es ist darauf Acht zu geben, dass in der Simulation, infolge des grossen Motorwiderstandes, bei einer Ausgangsspannung von $V_2 = 14V$ nur eine Ausgangsleistung von $P_2 = 500W$ übertragen wird, statt der verlangten Ausgangsleistung von $1kW$. Um trotzdem eine grobe Abschätzung für die Gleichrichterschaltung vornehmen zu können, wurden die Stromwerte aus Tabelle 4.1 mit dem Faktor 2 multipliziert .

M	$I_{S\,max}$ [A]	$I_{S\,rms}$ [A]	$I_{C\,max}$ [A]	$I_{C,rms}$ [A]	$I_{T\,max}$ [A]	$I_{T\,rms}$ [A]
1.15	81.7	44.2	57.9	26.3	81.7	31.2
1.00	90.2	52.4	64.6	33.8	90.2	37.0
0.75	104.4	65.2	76.1	45.9	104.4	46.2
0.50	118.7	75.7	90.8	55.5	118.7	53.47
0.25	133.4	82.9	94.6	61.7	133.4	58.47
0.00	148.4	85.6	93.72	64.3	148.3	60.8

Tabelle 4.1: Charakteristische Strom- und Spannungswerte für verschiedene Phasenmodulationsamplituden M, müssen noch mit dem Faktor 2 multipliziert werden

4.1.2 Leistungsschalter

In der Gleichtrichterschaltung werden MOSFETs als Leistungsschalter eingesetzt. Sie besitzen einen geringen Durchlasswiderstand $r_{DS(on)}$ und unterstützen hohe Frequenzen f_S.

Da Spannungsspitzen auftreten können, muss der MOSFET eine Nennspannung von mindestens 25V aufweisen. Ebenfalls muss er die entsprechenden Ströme und Stromimpulse aushalten können.

Ein weiteres Kriterium um einen geeigneten MOSFET zu finden, ist die Verlustleistung P_V, welche durch den MOSFET erzeugt wird.

Die Verlustleistung setzt sich aus den Schalt- und den Leitverlusten zusammen.

$$P_V = P_S + P_L$$

[14]

Schaltverluste P_S

Auf der Niederspannungsseite einer Konverterschaltung treten, wie der Name bereits sagt, tiefe Spannungen, dafür aber hohe Ströme auf. Die hohen Ströme haben beträchtlichen Einfluss auf die Schaltverluste und die Wahl der entsprechenden Schaltstrategie.

Durch die hohen Ströme ist unbedingt auf die auftretenden parasitären Induktivitäten Rücksicht zu nehmen. Im Folgenden wird darauf eingegangen, warum Hardswitching auf der Niederspannungsseite vorzuziehen ist. Um Abschätzungen vorzunehmen wird vom schlimmsten Fall der Simulation (Phasenmodulationsamplitude M=0) ausgegangen.

Softswitching auf der Niederspannungsseite: Die Energie in den Induktivitäten L_1 und L_2 (Abb. 4.4) verhält sich proportional zu I^2 und ist auf der Niederspannungsseite ungleich grösser als die Energie, welche in den parasitären Kapazitäten der Schalter gespeichert werden kann. Denn deren Energie verhält sich sich proportional zu U^2.

Der Schaltvorgang läuft nun wie folgt ab (Abb. 4.4):

1. Schalter T_2 ist leitend und wird dann ausgeschaltet.
2. Die Kapazitäten von T_1 und T_2 laden sich um.
3. In guter Nährung kann ein Zero Voltage Switching von T_1 angenommen werden. Das heisst, es gibt praktisch keine Einschaltverluste über T_1.
4. Die Induktivität L_1 muss nun zuerst magnetisiert werden, während die gespeicherte Energie in L_2 über dem Transistor abgebaut werden muss, was zu grossen Ausschaltverlusten über T_2 führt.
5. der Strom wird vollständig über T_1 geführt.

Die Schaltverlustenergie kann über die Formel

$$E_{off} = \frac{1}{2}\left(L_1 + L_2\right) I_{off}^2 \frac{V_{(BR)DSS}}{V_{(BR)DSS} - V_1}$$

angenähert werden [11]. Die Formel berechnet die Verlustenergie pro Schaltvorgang und Halbbrücke für Ströme mit einem RMS-Wert, der grösser ist als $100A$. Der Term $\frac{V_{(BR)DSS}}{V_{(BR)DSS}-V_1}$ wird in die Formel einbezogen, da die Spannung über dem Transistor nur bis zu einem gewissen Wert, der Break Down Spannung $V_{(BR)DSS}$, steigen kann. [11], [15], [14]

Hardswitching auf der Niederspannungsseite: Für das Hardswitching verringern sich auf der Niederspannungsseite die Verluste. Der Schaltvorgang läuft folgendermassen ab (Abb. 4.5):

1. T_1 ist leitend und wird dann ausgeschaltet.
2. Der Strom wird durch die Diode übernommen.
3. T_2 wird eingeschaltet. Der Strom über T_2 fliesst infolge der Induktivität nicht sofort.
4. Die Spannung fällt nun über den Induktivitäten und nicht über den Transistoren ab. Das hat zur Folge, dass die Schaltverluste über den Transistoren gering gehalten werden können. Der Strom durch L_2 nimmt im gleichen Masse ab, wie der Strom durch L_1 zunimmt. Die Energie in den parasitären Induktivitäten muss nicht im Transistor abgebaut werden.
5. Sobald der Strom negativ wird, lädt sich die Kapazität C_2 auf, was zu einer Spannungsabnahme über L_1 und L_2 führt, bis die volle Spannung über dem offenen Schalter T_2 anliegt und der Strom stabil durch die Induktivitäten fliesst.

Es treten somit viel geringere Schaltverluste für Hardswitching auf. [15], [14]

Um die Schaltverluste zu eruieren, wurden die Softswitching-Schaltverluste durch Abschätzungsformeln angenähert.

Neben der parasitären Induktivität der Schalter kommt zusätzlich noch eine parasitäre Leitungsinduktivitäten dazu. Als Leitungsinduktivität wurde $L_L = 12nH$ angenommen.

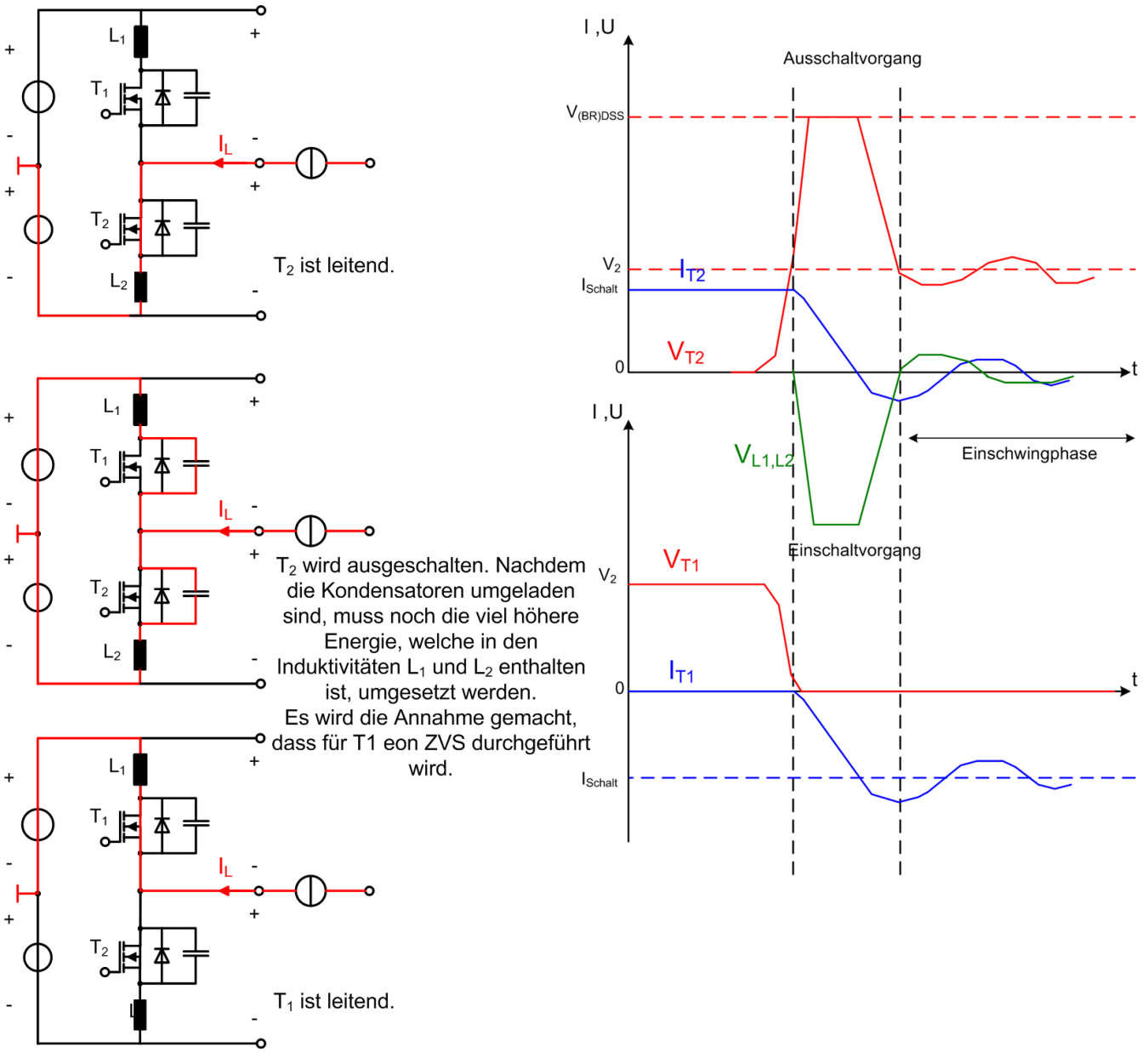


Abbildung 4.4: Softswitching auf der Niederspannungsseite. Ersatzschaltbilder; Spannungs- und Stomverläufe (qualitativ)

Die Gesamtinduktivität $L_1 + L_2$ für die Schaltverluste über die vier Schalter berechnet sich somit nach der Formel:

$$2 \cdot \frac{L_S}{4} + L_L = L_1 + L_2$$

Der Ausschaltstrom kann aus der Simulation entnommen werden und beträgt für die Gleich-

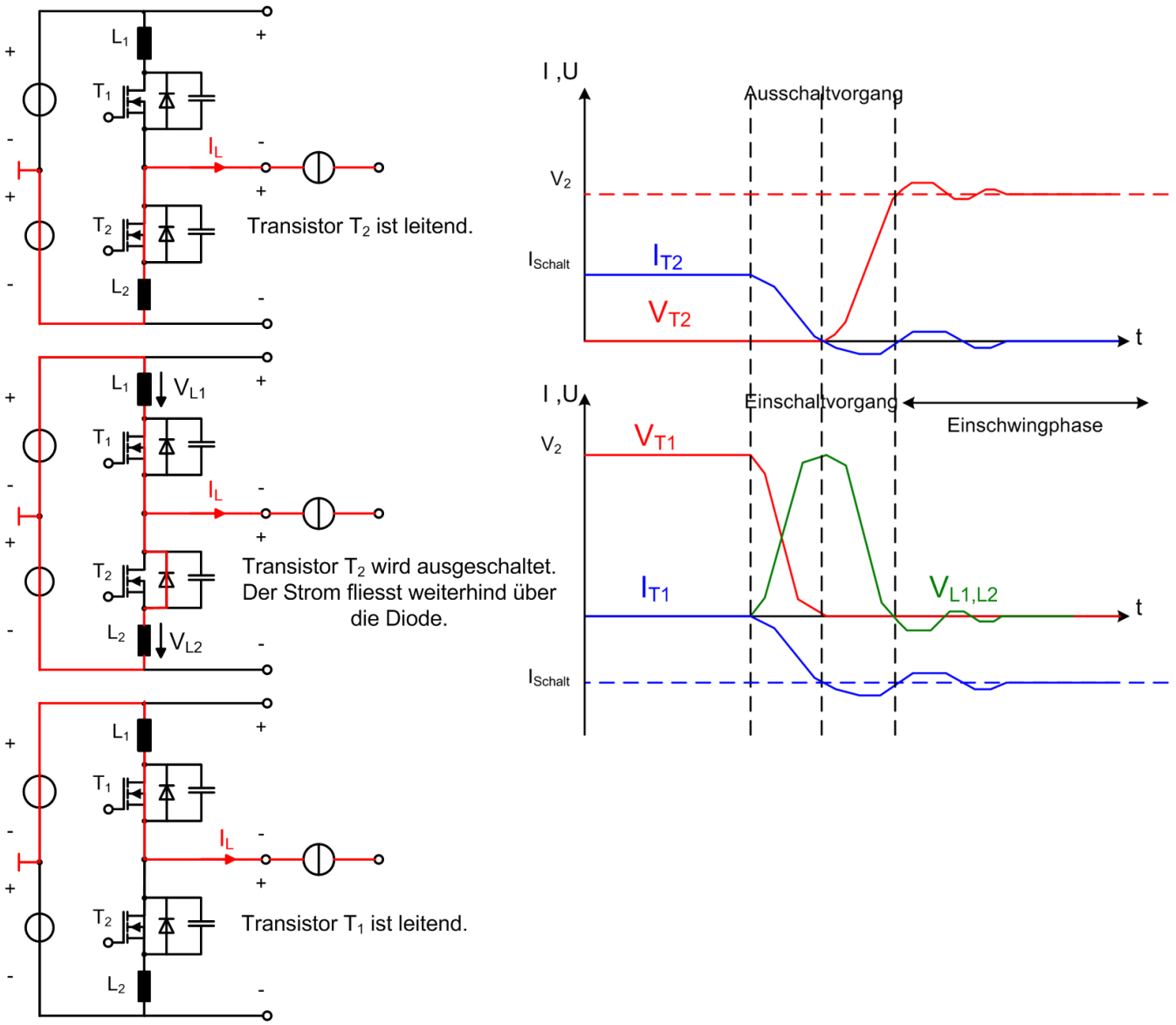


Abbildung 4.5: Hardswitching auf der Niederspannungsseite. Ersatzschaltbilder; Spannungs- und Stromverläufe (qualitativ)

richterschaltung $I_{off} = 120A$.

Da es sich bei der Gleichrichterschaltung um eine Vollbrücke handelt, fällt pro Schaltvorgang die doppelte Verlustenergie ab.

Die mittlere Verlustleistung lässt sich somit wie folgt abschätzen:

$$P_S = 2 \cdot f_S \left(\frac{1}{2} (L_1 + L_2) I_{off}^2 \frac{V_{(BR)DSS}}{V_{(BR)DSS} - V_1} \right)$$

Typ	$V_{(BR)DSS}$	$L_1 + L_2$	Schaltverlustenegie pro Schaltvorgang	Schaltverlustleistung
IRF2804	$40V$	$15nH$	$3.32 \cdot 10^{-4}J$	$6.64W$
IRL1404	$40V$	$15nH$	$3.32 \cdot 10^{-4}J$	$6.64W$
IRF2204	$40V$	$15nH$	$3.32 \cdot 10^{-4}J$	$6.64W$
IRL3713	$30V$	keine Angaben; Annahme $16.5nH$	$4.45 \cdot 10^{-4}J$	$8.91W$
IRF1503	$30V$	$16.5nH$	$4.45 \cdot 10^{-4}J$	$8.91W$

Tabelle 4.2: Schaltverluste über der Vollbrücke

Typ	Leitwiderstand	Leitverlustleistung
IRF2804	$2.3m\Omega$	$34.2W$
IRL1404	$3.1m\Omega$	$46.1W$
IRF1503	$3.3m\Omega$	$49.1W$
IRF2204	$3.6m\Omega$	$53.6W$
IRL3713	$4m\Omega$	$59.5W$

Tabelle 4.3: Leitverlustleistung für Gleichrichterschaltung und verschiedenen Typen von Schaltern

Daraus resultieren für die betrachteten Schalter die Schaltverluste, welche in Tabelle 4.2 aufgeführt sind.

Leitverluste

Die Leitverluste können über die Formel

$$P_L = r_{DS\,on}\,(I_{on})^2\,\frac{t_{on}}{T} \approx (I_{T\,rms})^2 \cdot r_{DS(on)}$$

abgeschätzt werden. [14]

Aus der Simulation kann ein Effektivwert für den Strom von $2 \cdot I_{T\,rms} \approx 122A$ entnommen werden.

Die Leitverluste berechnen sich zu den Werten, welche in Tabelle 4.3 angegeben sind.

Es ist zu erkennen, dass bei einer niedrigen Frequenz von 20kHz die Leitverluste die tragenden Verluste über den Transistoren bilden. Deshalb ist es nötig, einen Transistor mit möglichst geringem Leitwiderstand ($r_{DS(on)}$) zu wählen.

Wenn die Auswahl nach Tabelle 4.3 getroffen werden würde, wäre der Transistor IRF 2804, mit der geringsten Verlustleistung, der geeignetste Transistor. Da der Transistor IRF 2804 aber

Typ	IRL1404
Hersteller	International Rectifier
Nennspannung	$40V$
Dauerstrombelastung	$75A$
Stromimpulsbelastung	$790A$
Leitwiderstand	$3.1m\Omega$ ($V_{GS} = 10V$)
Parasitäre Induktivität L_D/L_S	$4.5nH/7.5nH$
Gesamtverluste durch Schalter	$52.8W$

Tabelle 4.4: Spezifikation des IRL 1404

zum Zeitpunkt der Bestellung nicht mehr lieferbar war, fiel die Entscheidung auf den *IRL 1404*. Die wichtigsten Eigenschaften dieses Transistors sind in der Tabelle 4.4 aufgeführt.

Das Datenblatt des Schalters befindet sich auf der beiliegenden CD.

4.1.3 Ausgangskapazität

Die Ausgangskapazität sollte so dimensioniert werden, dass die Ausgangsspannung um maximal 1% schwankt.

$$\Delta U_{2\ max} = 0.01 \cdot U_2 = 0.14V$$

Der Wechselanteil des Ausgangsstroms fällt über der Kapazität ab. Die Ladung, welche in die Kapazität hineinfliesst, kann über die Simulation abgeschätzt werden (Abb. 4.6). Anstelle der Kapazität (C_1 Abb. 4.2) wurde in der Simulation eine Spannungsquelle mit der gewünschten Ausgangspannung von $14V$ eingeführt.

Der Kapazitätswert kann über folgende Beziehung berechnet werden:

$$C\frac{\partial U_2}{\partial t} = I_c(t) \qquad \Rightarrow \qquad C > \frac{\Delta Q}{\Delta U_2} \qquad \text{wobei} \qquad \Delta Q = \int_t I_c(t)\partial t$$

[4], [13]

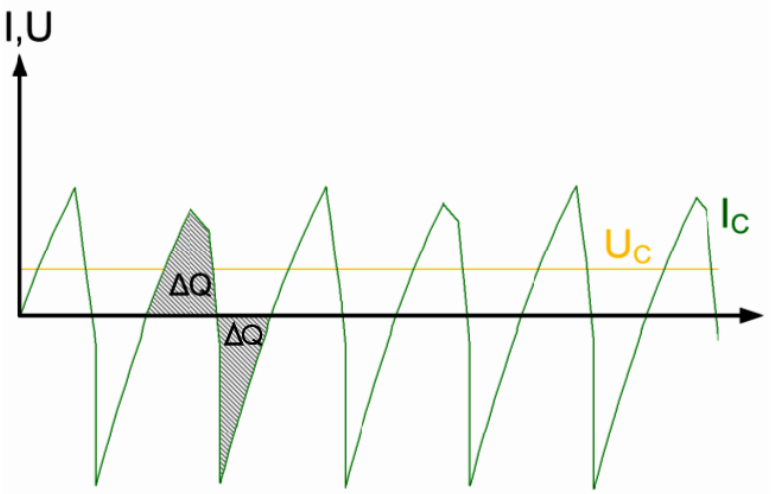


Abbildung 4.6: Abschätzung von ΔQ in der Simulation

Der zu unterdrückende Spannungsrippel ist am grössten für ein $M = 0$. Damit ist auch der Kapazitätswert für ein $M = 0$ am grössten (Tabelle 4.5).

Die Kapazität beträgt, anhand der oben genannten Gleichungen, $14.76mF$. Um diese Kapazität zu realisieren wurde eine Parallelschaltung der Kondensatoren ausgelegt. Neben dem Kapazitätswert muss ebenfalls die Strombelastbarkeit der Kondensatoren genügend hoch sein.

Um diese Parallelschaltung zu realisieren, boten sich grundsätzlich zwei Arten von Kondensatoren an.

M	ΔQ	I_{crms} [A]	C [mF]
1.15	335μC	52.6	4.79
1.00	480μC	67.6	6.86
0.75	700μC	90.2	10.00
0.50	870μC	111	12.43
0.25	990μC	123.4	14.14
0.00	1033μC	128.6	14.76

Tabelle 4.5: Ladungs- und Kapazitätswerte für verschiedene Phasenmodulationsamplituden nach der Gleichung 4.1.3

Elektrolytkondensatoren

Die Vorteile eines Elektrolytkondensators sind:

- Grosse Kapazität

Die Nachteile sind:

- Chemische Prozesse laufen ab, obwohl der Kondensator unbenutzt ist. Diese lassen ihn auch in unbenutztem Zustand altern. Sie können auch ein Auslaufen der Kondensatoren zur Folge haben.
- Besitzt eine nicht so gute Temperaturfestigkeit wie Keramikkondensatoren.
- Bei einigen Elektrolytkondensatoren ist auf die Polarität zu achten.

[2]

Keramikkondensatoren

Vorteile eines Keramikkondensators sind:

- Kompakte Bauweise mit geringem Volumen
- Hohe Stromfestigkeit
- Geringe Alterung im Vergleich zum Elektrolytkondensator
- Keine Polarität

Die Nachteile sind:

- Keramikkondensatoren besitzen viel kleinere Kapazitätswerte als Elektrolytkondensatoren.

[3]

Im Bezug auf Keramikkondensatoren ist zu erwähnen, dass die dielektrische Konstante je nach Keramiksorte stark variieren kann. Dabei sind Materialien mit grosser dielektrischer Konstante temperatur-, frequenz- und spannungsabhängig.

Einige Klassifizierungen von Keramiken sind X7R, Y5V, X5R. Mithilfe eines Web-Tools , welches von Murata aus [5] zur Verfügung gestellt wird, können die Unterschiede zwischen den verschiedenen Keramiken gut dargestellt werden (Abb. 4.7).

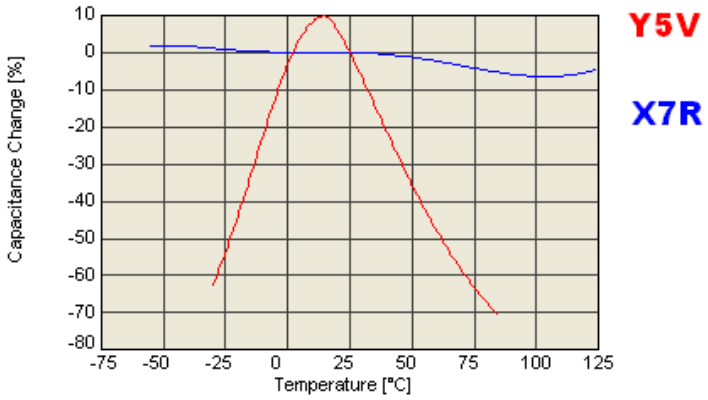


Abbildung 4.7: Temperaturverhalten der Keramik vom Typ Y5V und X7R

Aus Abbildung 4.7 ist ersichtlich, dass Kondensatoren mit der Keramik Y5V nicht geeignet sind, da deren Kapazitätswert schon bei geringem Temperaturanstieg stark sinkt. Bei hohen Temperaturen kann bei einem Kondensator mit dem dritten Buchstaben V in der Klassifizierung der Keramik der Kapazitätswert um bis zu 80% des angegebenen Wertes fallen. Bei Kondensatoren mit einem R als Endbuchstabe variiert der Wert der Kapazität hingegen nur um etwa $\pm$15%.

X5R unterscheidet sich von X7R durch einen kleineren Temperaturbereich, in welchem er betrieben werden kann.

Anhand dieser Eigenschaften kann die Suche nach Keramikkondensatoren schon stark eingeschränkt werden. [3]

Kondensatorauswahl

Für die Stromfestigkeit muss

$$I_{C,rms} = 128.6A$$

durch die Parallelschaltung erreicht werden können.

Wie in Tabelle 4.5 erwähnt, muss gleichzeitig ein Kapazitätswert von $14.76mF$ erreicht werden.

Um die Parallelschaltung zu dimensionieren, gelten die folgenden Formeln:

$$C_{tot} = \sum_{1}^{n} C_{kond} \qquad \text{wobei} \qquad n : \text{Anzahl Kondensatoren}$$

Kondensator-Typ	Kapazität $[\mu F]$	Stromfestigkeit (I_{SF}) $[mA]$	Anzahl	Gesamtkapazität C_{tot} $[mF]$	Gesamtvolumen $[cm^3]$
Elektrolytkondensatoren					
Rubycon					
ZA	100	640	201	20.1	68.92
	220	1230	105	23.1	103.08
	330	1650	78	25.74	98.01
ZL	220	760	170	37.40	98.26
	330	1030	125	41.25	122.71
	470	1430	90	42.3	113.09
Keramikkondensatoren					
Murata					
GRM31CR61E106KA12	10	ca. 1A	1476	14.76	12.09
GRM32DR71E106KA12	10	ca. 1A	1476	14.76	23.616
GRM55ER71E156KA01	15	ca. 1A	984	14.76	70.11
GRM32ER61E226KE15	22	ca. 1A	671	14.76	13.42

Tabelle 4.6: Vergleich zwischen einigen ausgewählten Kondensatoren

Für die Stombelastbarkeit pro Kondensator gilt:

$$I_{C,rms/kond} = \frac{I_{C,rms}}{n}$$

Daraus bieten sich die in Tabelle 4.6 zusammengefassten Varianten an.

Die Anzahl Elektrolytkondensatoren wird aufgrund ihrer Stombelastbarkeit gross und die Kapazität ist dementsprechend überdimensioniert.

Aufgrund dieser Tatsachen und den oben genannten Vorteilen von Keramikkondensatoren, fiel die Wahl auf den Keramikkondensatoren GRM32CR71E106KA12, welcher ebenfalls für die Bootstrap-Schaltung gebraucht wurde. Weitere Spezifikationen finden sich in Tabelle 4.7 sowie in der beiliegenden CD.

Produzent	Murata
Typ	GRM32CR71E106KA12
Footprint	1210
Kapazität	$10\mu F$
Keramik	X5R
Anzahl	1476

Tabelle 4.7: Einige Angaben zum GRM32DR71E106KA12

4.1.4 Leiterbahnen und Durchkontaktierungen

Leiterbahnen

Wie bereits erwähnt, treten auf der Niederspannungsseite nicht sehr hohe Spannungen auf, dafür umso grössere Ströme. Aus der Simulation resultiert ein maximaler Sekundärstrom von $I_{S\,rms} = 171.2A$.

Daraus folgt, dass die Leiterbahnen nicht weit auseinander liegen, dafür aber breit dimensioniert werden müssen.

Die Leiterbahnbreite bestimmt sich grundsätzlich über eine maximale zulässige Temperatur, welche etwa 110°C beträgt. Für höhere Temperaturen könnten sich eventuell Leiterbahnen aus der Leiterplatte lösen, oder die Leiterplatte kann sich verformen.

Es gibt sehr viele verschiedene Artikel zu der Stromfestigkeit von Leiterbahnen. Diese Arbeit orientiert sich an einem Artikel, welcher im Internet unter [7] zu finden ist.

Anhand des Artikels kann die Temperaturerhöhung durch die Formel

$$\Delta T = B_{LP} b^{-1.43} \frac{35\mu m}{d} I^2$$

berechnet werden. Die Kupferschichtdicke d ist in μm und die Breite der Leiterbahn b in mm zu messen. B_{LP} ist eine Proportionalitätskonstante, welche vom Aufbau der Leiterplatte abhängig ist. [7]

Das Kupfer, welches für die Leiterbahnen verwendet wird, ist $35\mu m$ dick. Die Leiterplatte ist vierlagig aufgebaut. Die Proportionalitätskonstante beträgt damit $B_{LP} = 1.6$. Die Leiterbahnen dürfen sich, bei einer angenommenen Raumtemperatur von 25°C, um maximal 85°C erwärmen.

In Tabelle 4.8 sind einige Leiterbreiten für verschiedene Strombelastungen zusammengestellt, dass sich die Leiter um maximal 85°C erhöhen.

Daraus konnte eine Leiterbahnbreite von $82mm$ abgeschätzt werden. Im schlimmsten Fall, dass heisst für eine Modulationsamplitude von $M = 0$, reicht die Breite gerade noch aus, dass die kritische Temperatur nicht erreicht wird.

Die Stromfestigkeit der Leiterbahnen kann noch durch geeignete Kühlung (Ventilator etc.) und Verwendung dünner Substrate erhöht werden. In der Automobilindustrie werden allgemein dickere Kupferschichten ($70\mu m$) verwendet, um die Baugrösse zu minimieren.

Strombelastung $I_{S\,rms}[A]$	Leiterbreite[mm]
88	33
104	41
130	56
150	69
164	78
170	81

Tabelle 4.8: Leiterbreite bei gegebener Strombelastung und einem maximalen ΔT von 80°C

Spannung zwischen den Leiterplatten [V]	Mindestabstand ohne Lötstopplack [mm]	Mindestabstand mit Lötstopplack [mm]
0...30	0.64	0.25
31..50	0.64	0.38
51..100	0.64	0.50
101..150	1.27	0.50

Tabelle 4.9: Leiterbahnabstände im Vergleich zu den auftretenden Spannungen

Um die Spannungsbelastung und der damit zusammenhängende Leiterbahnabstand abzuschätzen, wurde die Tabelle 4.9 benutzt. [6]

Die maximale Spannung beträgt um die 12V für Leiterbahnen im Ansteuerungsbereich und um die maximal 25V für Leiterbahnen im Leistungsteil der Gleichrichterschaltung.

Aus der Tabelle 4.9 kann ein Mindestabstand von $0.25mm$ entnommen werden, welcher mit Lötstopplack ausreichend ist. Damit ist der Abstand zwischen den Leiterbahnen im Ansteuerungsbereich mit $0.254mm$ ausreichend. Im Leistungsbereich der Gleichrichterschaltung ist der Abstand mit $0.6mm$ ebenfalls genügend gross dimensioniert.

Bohrdurchmesser [mm]	zulässiger Strom [A]
0.2	1
0.3	2
0.4	2.5
0.5	3.0
0.6	3.5
0.7	4
0.8	5

Tabelle 4.10: Strombelastbarkeit von Durchkontaktierungen

Durchkontaktierungen

Durchkontaktierungen, auch Via genannt, bieten die Möglichkeit Leiterbahnen, welche sich auf verschiedenen Lagen der Leiterplatten befinden, zu verbinden.

Je nach Durchmesser einer Durchkontaktierung kann sie mehr oder weniger Strom führen. In Tabelle 4.10 sind einige gebräuchliche Werte aufgeführt. [6]

Damit die Forderung der Strombelastbarkeit erfüllt werden kann, können auch mehrere Durchkontaktierungen parallel geschaltet werden.

Um den Durchmesser der grossen Vias genügend zu dimensionieren, wurde die Tabelle 4.10 linear fortgesetzt.

4.1.5 Weitere Elemente der Schaltung

Die Elemente des Ansteuerungsteils mussten nicht dimensioniert, aber zum Teil nochmals überprüft werden. Nachfolgend werden diese Elemente kurz erläutert.

Stecker

Als Stecker wird ein 10-poliger Schalter von Micromatch verwendet. Über ihn werden die Steuersignale für die MOSFETs, eine Versorgungsspannung von 5V sowie der Ground auf die Leiterplatte geführt.

Common Mode Choke

Common Mode Noise sind elektrische Störungen, welche durch eine Leitung kommen und auf einem Ground zum Verursacher zurückkehren. Auf ihrem Weg durch die Leitungen können sie das Verhalten einer Schaltung stark beeinflussen, indem sie zum Beispiel Transistoransteuerungssignale verfälschen. Mit einem Common Mode Choke können solche Störungen unterdrückt werden.

In dieser Schaltung wird eine vierfache Drossel SMD von EPCOS (Datenblätter im Anhang) verwendet. [1]

Optocoupler

Die wesentliche Aufgabe eines Optocouplers ist die optische Kopplung, bzw. die Potentialtrennung zwischen dem Kontroll- und dem Ansteuerungsteil einer Schaltung. Der Eingang des Optocouplers ist direkt mit dem Kontrollteil verbunden, während der Ausgang direkt an den Ansteuerungsteil einer Schaltung gekoppelt ist. Die Signalübertragung zwischen Ein- und Ausgang erfolgt über eine LED (Abb.4.8).

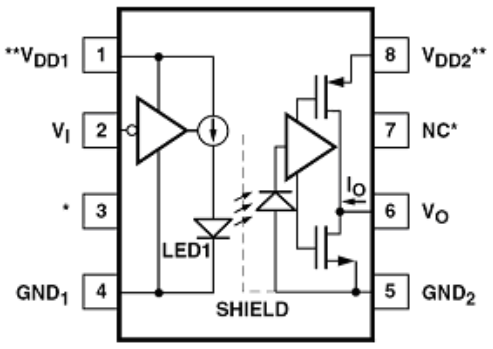


Abbildung 4.8: Schematische Darstellung eines Optocouplers

Hersteller	IXYS
Typ	IXDI414YI
Gehäusetyp	5-Pin TO-263
Ausgangsstrom I_{DC}	4A
Versorgungsstrom V_{cc}	4.V ... 18V
Versorgungsstrom	0 ... 3mA

Tabelle 4.11: Einige Spezifikationen des Gatedrivers

Diese Potentialtrennung ist nötig, wenn das sekundärseitige Sourcepotential schwankt. In der behandelten Gleichrichterschaltung betrifft das vorallem die oberen Schalter aus der schematischen Darstellung in Abb. 4.15.

In dieser Schaltung wird ein Optocoupler vom Typ Agilent HCPL-0723 (Datenblatt im Anhang) verwendet. [14]

Gatedriver

Der Gatedriver ist sozusagen die Grenze zwischen der Ansteuerungselektronik und der Leistungselektronik.

Als Gatedriver wird ein IXDI414YI verwendet.

Der IXDI414YI ist ein invertierender Gatedriver, welcher aus einem high Eingangsignal ein low Ausgangssignal erzeugt und umgekehrt.

Es musste überprüft werden, wie viel Zeit dieser Gatedriver braucht um die Kapazität des Transistors aufzuladen. Diese Zeit muss kleiner sein als die Schaltperiode $\frac{1}{f_S} = 50\mu s$.

Die Gate-Ladung, welche durch den Gatedriver zur Verfügung gestellt werden muss, beträgt für den IRL1404 $75nC$. Aus Tabelle 4.11 kann ein Ausgangsstrom von 4A abgelesen werden. Dieser Strom muss reichen um vier Transistoren aufzuladen.

Über die Beziehung

$$\Delta t = \Delta Q / I = \frac{75nC}{1A} = 75ns << 50\mu s$$

kann die Ladungszeit auf $75ns$ abgeschätzt werden, was viel kleiner ist als die Schaltperiode. Somit kann der Gatedriver als geeignet betrachtet werden. [14]

ICs für Versorgung

Es stehen ebenfalls zwei ICs zur Verfügung, welche eine konstante Versorgungsspannung für die entsprechenden Gatedriver und Optocoupler zur Verfügung stellen.

In dieser Anwendung werden die zwei ICs LF120CDT und L4941BDT von ST Microelectronics benutzt.

Bootstrap Schaltung

In der Gleichrichterschaltung gibt es Optocoupler und Gatedriver, an welchen, auf Grund ihrem nicht konstanten Sourcepotential, anfangs keine und später eine schwankende Versorgungsspannung anliegt. In dieser Schaltung betrifft dies die oberen Schalter in der schematische Abbildung 4.15. Wenn die unteren Schalter ausgeschalten sind, liegt das Sourcepotential auf 14V, was zu einer Vorsorgungspannung von 0V für die entsprechenden Gatedriver und Optocoupler führt.

Die Bootstrap-Schaltung (Abb.4.9) dient dazu, eine konstante Spannungsversorgung für diese Gatedriver und Optocoupler, zu Verfügung zu stellen. Wenn die unteren Schalter durchgeschalten sind, ist $U_A = 12$V. Zum Einschaltzeitpunkt ist die Eingangsdiode leitend und die Kapazitäten C_1 und C_2 der Bootstrap-Schaltung werden aufgeladen. Eine Spannung von $U_{B1} = 12$V am Ausgang der Bootstrap-Schaltung baut sich auf. Diese dient zur Versorgung des Gatedrivers.

Durch einen Spannungsteiler wird ebenfalls eine Versorgungsspannung von $U_{B2} = 5$V aufgebaut, welche für die sekundärseitige Versorgung der dazugehörigen Optocoupler zuständig ist. Die Zenerdiode dient dazu, dass über der 5V-Versorgung keine Überspannungen auftreten können.

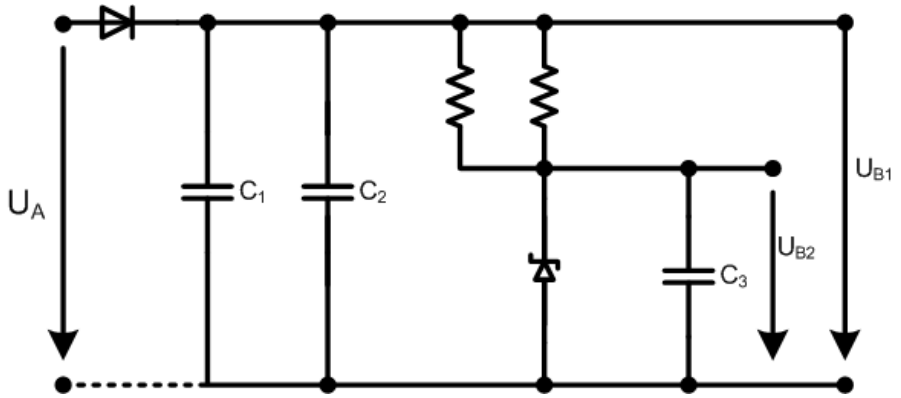


Abbildung 4.9: Bootstrap Schaltung

4.2 Schemas und Layout

4.2.1 Schemas

In den Schemas sind die Verbindungen zwischen den verschiedenen Bauelementen eingezeichnet. Sie stellen das eigentliche Gerüst und den Plan zum Designen dar.

Die Schemas bestanden bereits und mussten nur noch wenig bearbeitet oder ergänzt werden.

Die Schemas und das Layout wurde mit Programm ALTIUM DESIGNER bearbeitet.

Aus der Tabelle 4.2.1 können die Ergänzungen zu den entsprechenden Schemas entnommen werden.

altes File	neues File	Inhalt und Änderungen
Overview.SCHDOC	Übersicht.SCHDOC	Zeigt, wie die verschiedenen Schaltschemas zusammengehören. Alle nicht zum Niederspannungsteil der Dual Active Bridge gehörenden Schemas wurden gelöscht.
LVDriver.SCHDOC	Ansteuerung.SCHDOC	Beinhaltet die Ansteuerung der Schaltelemente. Im Ansteuerungsteil wurden mit den Common Mode Chokes ergänzt. Zusätzlich wurden die Gatedriver durch invertierende Gatedriver ersetzt und die Optocoupler dementsprechend angepasst.
LVPower.SCHDOC	Schaltung.SCHDOC	Dieses File beschreibt den Leistungsteil der Gleichrichterschaltung und ist im wesentlichen unverändert. Die Kondensatoren wurden in ein externes File exportiert.
Connectors.SCHDOC	Anschluss.SCHDOC	Definiert die Anschlüsse für das Board. Es mussten neue Transformator- und Lastanschlüsse definiert werden. Der Stecker für die Signalübertragung vom Kontrollteil zum Ansteuerungsteil der Schaltung konnte beibehalten werden.
	Kondenstor.SCHDOC	Ein neues File, in welchem die vielen Kondensatoren vermerkt sind.

Tabelle 4.12: Ergänzungen zu den Schemas

Die Schemas befinden sich im Anhang dieser Arbeit und ebenfalls auf der beiliegenden CD.

4.2.2 Neue Footprints

Da die Gleichrichterschaltung schon einmal ausgelegt, gelayoutet und gebaut wurde, waren die meisten Footprints schon vorhanden.

Trotzdem mussten noch zwei neue Footprints erstellt werden.

Für den Transformator- und den Lastanschluss wurde der in Abbildung 4.10 dargestellte Footprint erstellt.

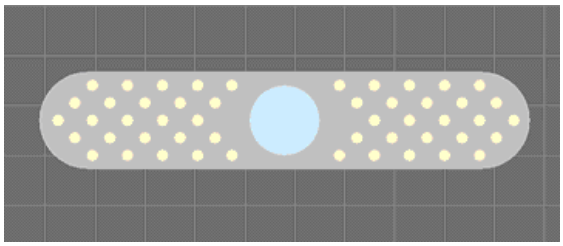


Abbildung 4.10: Neuer Footprint für Transformator- und Lastanschluss

Damit die Transistoren auf der Rückseite der Leiterplatte angebracht, abgeklappt und gleichzeitig noch am Kühlkörper fixiert werden konnten, mussten Bohrlöcher in der Leiterplatte angebracht werden.

Damit die Bohrlöcher am richtigen Ort sind, wurde ein neuer Footprint für den abgeklappten Transistor entworfen.

Im erstellten Prototypen sind die Transistoren so angeordnet, dass sie auf die falsche Seite abgeklappt werden müssten (Abb.4.11). Nachträglich wurde dieser Fehler im Footprint und im Layout korrigiert.

4.2.3 Layout

Eine Arbeit, welche einen grossen Teil der Semesterarbeit in Anspruch nahm, war das erstellen eines Layouts.

Obwohl schon, wie erwähnt, ein Layout bestand, musste ein neues Layout gemacht werden. Es konnten jedoch Ideen des schon bestehenden Layouts zum Teil erneut umgesetzt werden. Die Breite der Leiterplatte war auf $16cm$ festgelegt. Die Leiterplatte wurde schlussendlich $12cm$ lang. Es galt möglichst parasitäre Induktivitäten, Kapazitäten und Widerstände zu vermeiden.

Um parasitäre Widerstände klein zu halten, sind die Leiterbahnen möglichst kurz zu wählen. Um parasitäre Induktivitäten zu verhindern gilt es Stromschleifen, welche eine grosse Fläche einschliessen, zu vermeiden. Die Leiterbahnen sind nahe beieinander anzuordnen. [14]

Was nicht in dieser Arbeit beachtet wurde ist, dass die Leiterbahnen, welche vom Transformator zu den MOSFETs führen, durch den Transformator sowieso schon eine Induktivität besitzen. Es sollten vielmehr die parasitären Induktivitäten bei den Leiterbahnen von den MOSFETs zur Last vermieden werden, da sonst grosse Schaltverluste, wie in Kapitel 4.1.2 erwähnt, auftreten können.

In Bild 4.12 ist dargestellt, wie die Leiterbahnen durch die Leiterplatte angeordnet wurden.

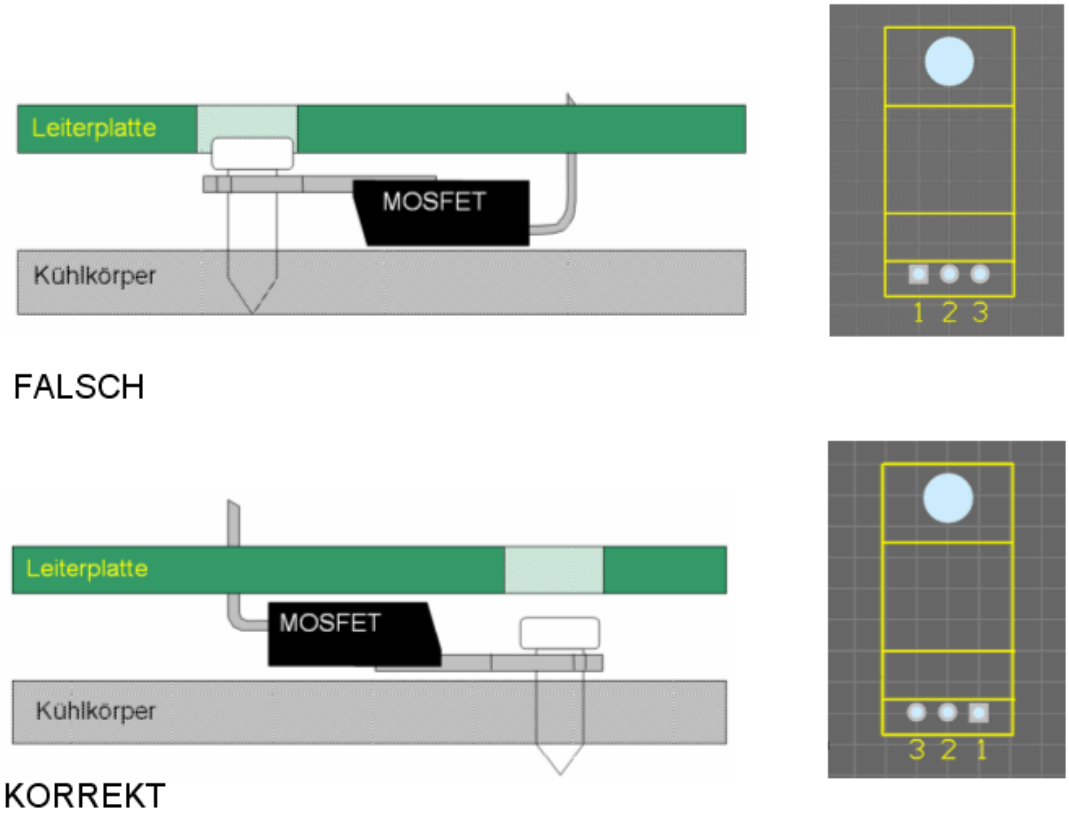


Abbildung 4.11: Falscher und korrekter Footprint für den Transistor

In dieser Leiterplatte gibt es eine Lage zu wenig um dieses Konzept konsequent durchzuziehen. Die Leiterbahnen, welche I_L führen, sind deshalb nebeneinander angeordnet, was, wie schon erwähnt, nicht eine optimale Lösung ist.

Ein weiterer Lösungsansatz, welcher in der früher erstellten Schaltung realisiert wurde, wäre einen externen Leiter anzubringen oder zusätzliche Lagen zu benützen. In diesem Layout wurde auf einen externen Leiter verzichtet.

Die Leiterbahnbreiten sind relativ knapp bemessen, was eine genügende Kühlung unabdingbar macht. Um die Kühlung zu vereinfachen, wurden die zu kühlenden Elemente (die Transistoren) auf dem Bottom Layer angebracht. Damit sich Strom und Wärme gut über die ganze Leiterplatte verteilen, wurden die Transistoren, welche gleichzeitig ein und ausgeschaltet werden, möglichst verteilt angeordnet (Abb. 4.15).

Der ganze Ansteuerungsteil wurde auf dem Top Layer realisiert (Abb. 4.14). Es wurde versucht, die Signalleiterbahnen möglichst kurz zu halten, was sich durch die Anordnung der Transistoren jedoch stark erschwerte.

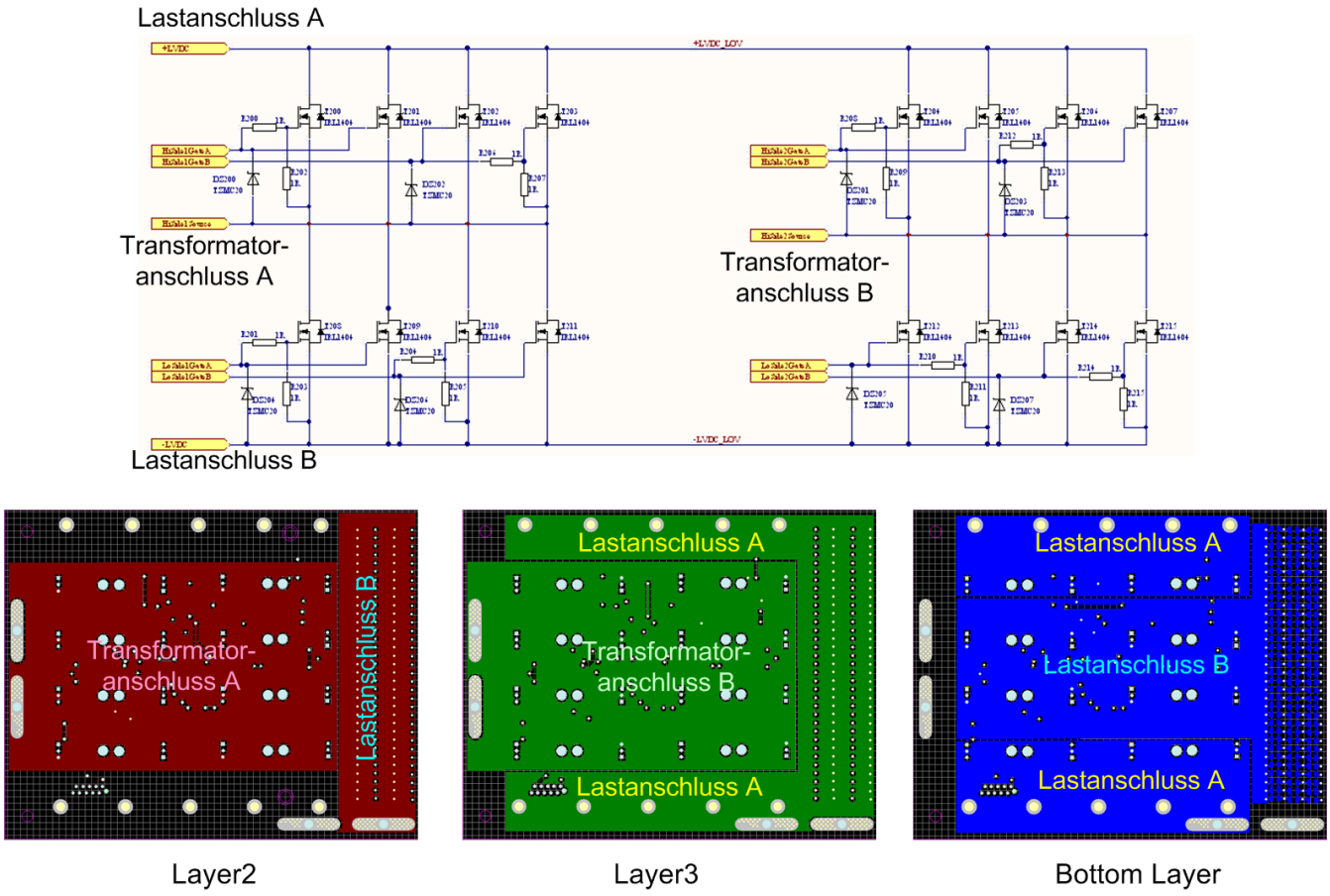


Abbildung 4.12: Die Verschiedenen Leiterbahnen in der Leiterplatte

4.3 Realisierung und Tests

Nachdem die Bauteile evaluiert und das Layout erstellt waren, musste die Schaltung noch realisiert werden. In den nachfolgenden Bildern (Abb. 4.14 und Abb. 4.15) ist zu sehen, wo welche Bauteile angebracht wurden.

Nachdem die Leiterplatte fertig bestückt war, konnten einige Tests durchgeführt werden. In diesen Tests konnte die Schaltung auf ihre Funktionsfähigkeit unter kleiner Last überprüft werden.

Es ist zu erwähnen, dass vorerst nur eine Ausgangskapazität von 2mF realisiert wurde.

Es zeigte sich, dass die Versorgung der Ansteuerungselemente, sowie das Verhalten der Gleichrichterschaltung unter kleiner Last einwandfrei funktionieren.

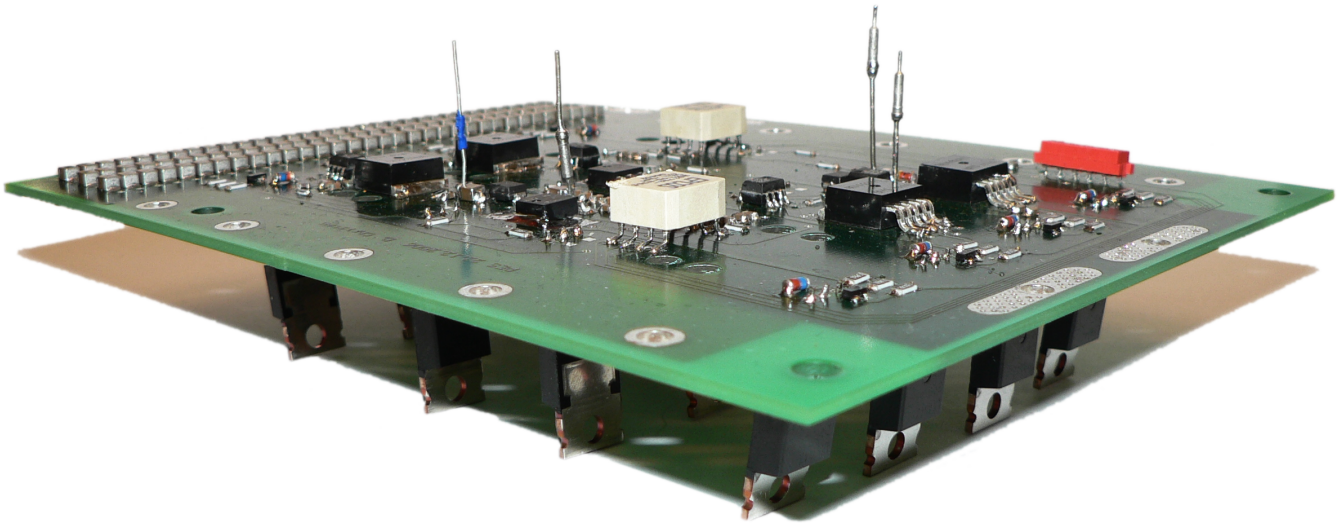


Abbildung 4.13: Fertiges PCB

4.4 Probleme

Im Folgenden wird auf die Probleme und Fehler die aufgetreten sind eingegangen.

- Die Leiterbahnen von den MOSFETs zur Last sind möglichst übereinander anzuordnen
- Ausgangskapazität ist nur $2mF$ gross. Unter Berücksichtigung, dass der Spannungsrippel auch grösser als 1% werden darf und einer idealen Phasenmodulationsamplitude von M=1.15, kann jedoch die Ausgangskapazität redimensioniert werden.
- Der Footprint der Transistoren ist auf die verkehrte Seite abgeklappt, was aber nachträglich in den Schemas und im Layout geändert wurde.
- Es ist unbedingt darauf zu achten, dass beim Testen der Optocoupler kein Kurzschluss mit dem 3. Anschluss passiert, sonst kann das Bauteil zerstört werden.

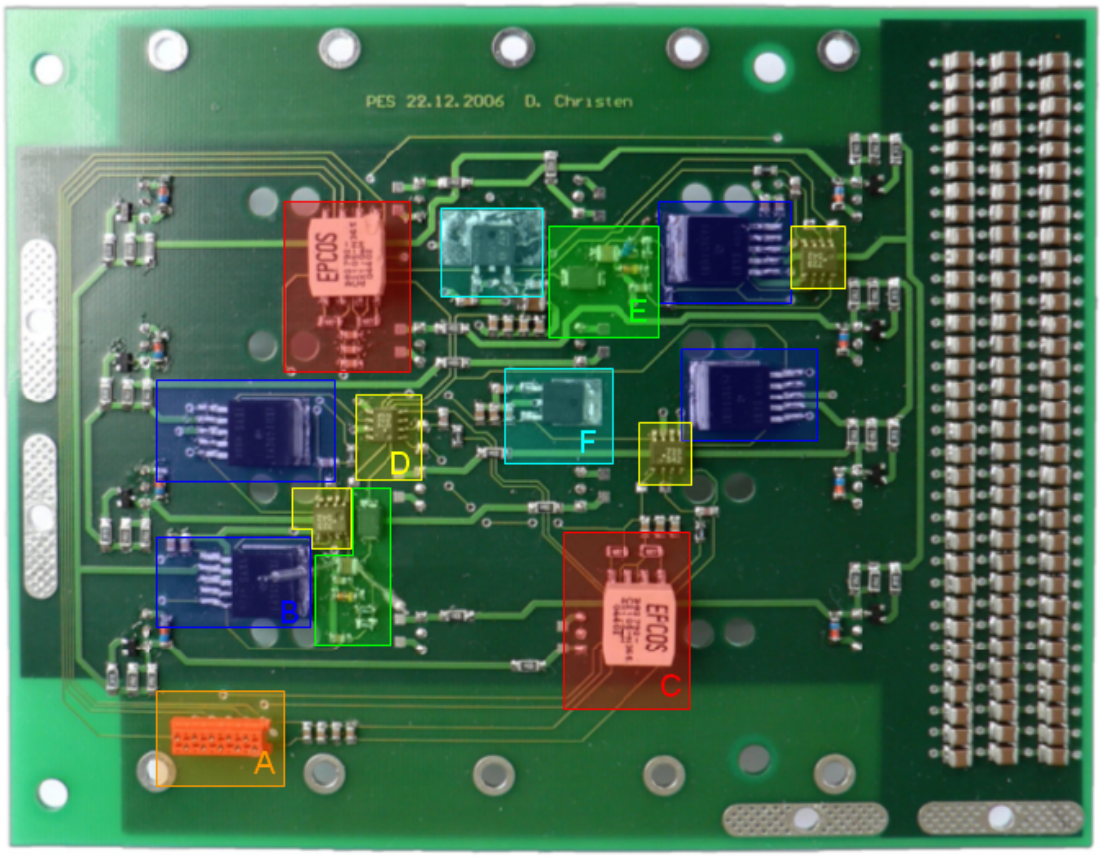
4.5 Weiterführende Arbeiten

Die Hochspannungsbrücke, sowie der Transformator müssen noch spezifiziert, ausgelegt und realisiert werden.

Tests und Analysen der Gleichrichterschaltung unter voller Last müssen noch durchgeführt werden.

Es ist ebenfalls wichtig, einen geeigneten Kühlkörper für die Leiterplatte zu wählen und eventuell eine Temperaturanalyse für die Gleichrichterschaltung durchzuführen.

Eventuell sind alternative Lösungen für die Realisation der Ausgangskapazität zu finden.



A: Stecker

B: Gatedriver

C: Common Mode Choke

D: Optocoupler

E: Bootstrap Schaltung

F: IC für Versorgungsspannung

Abbildung 4.14: Der Top Layer des fertigen PCBs

Um die ganze Schaltung kompakter bauen zu können, müssten die Kupferlagen im PCB dicker gewählt und externe Leiterplatten angeordnet werden. Damit könnte auch das Problem von parasitären Induktivitäten teilweise gelöst werden.

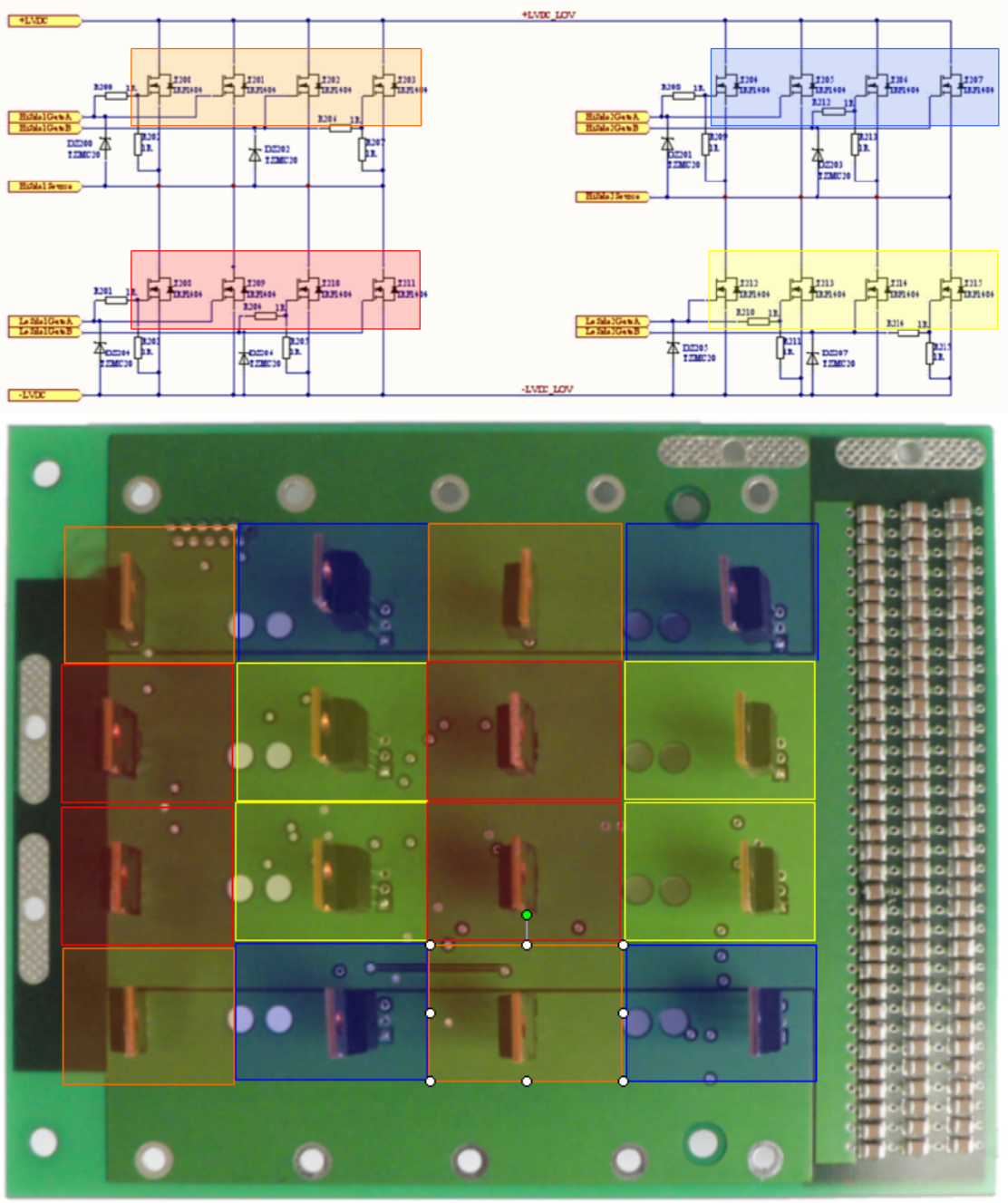


Abbildung 4.15: Anordnung der Bauelemente auf dem Bottomlayer

Literaturverzeichnis

[1] Differential and Common Mode Noise.
http://www.murata.com/emc/knowhow/pdfs/te04ea-1/26to28e.pdf.

[2] Elektrolytkondensator.
http://de.wikipedia.org/wiki/Elektrolytkondensator.

[3] Keramikkondensator.
http://de.wikipedia.org/wiki/Keramikkondensator.

[4] Kondensator (Elektrotechnik).
http://de.wikipedia.org/wiki/Kondensator_(Elektrotechnik).

[5] Murata Chip Capacitor Characteristics Data Library.
http://www.murata.com/designlib/mccdl/index.html.

[6] Tips für Entwickler.
http://www.fs-leiterplatten.de/ebemess.htm.

[7] Dr. Johannes Adam. Neues von der Strombelastbarkeit von Leiterbahnen.
http://www.flomerics.de/Produkte/Flotherm/Fachartikel/Strombelastbarkeit_von_Leiterbahnen.pdf.

[8] ANALOG DEVICES. *ADSP-21991 Mixed Signal DSP Controller*, 2003. Revision 0
CD: 'ADSP-219x compiler and library manual.pdf'.

[9] ANALOG DEVICES. *ADSP-219x DSP Instruction Set Reference*, Dezember 2005. Revision 2,
CD: 'ADSP-2199x-Instruction Set Reference.pdf'.

[10] Kheraluwala et al. Performance Characterization of a High-Power Dual Active Bridge dc-to-dc Converter. *IEEE transactions on industry applications, vol. 28, no. 6*, 1992.
CD: '06_Kheraluwala1992.pdf'.

[11] J. W. Kolar F. Krismer, S. Round. Performance Optimization of a High Current Dual Active Bridge with a Wide Operationg Voltage Range.

[12] Institut für Leistungselektronik, ETH. *PLD_MCM*.
CD: 'pld_mcm.pdf'.

[13] Prof. Dr. J. W. Kolar. Leistungselektronik, 2005.

[14] Prof. Dr. J. W. Kollar. Leistungselektronische Systeme I, 2006.

[15] F. Krismer. Optimization of bi-directional DC to DC converters for battery applications.

[16] Lattice Semiconductor Corp. *FPGA Design Guide*, April 2006.
CD: 'fpga_design_guide.pdf'.

[17] Lattice Semiconductor Corp. *MachXO Family Handbook*, Dezember 2006. Version 01.8
CD: 'MachXO Handbook.pdf'.

[18] Lattice Semiconductor Corp. *MachXO sysCLOCK Design and Usage Guide*, September 2006.
CD: 'MachXO sysCLOCK.pdf'.

[19] YU JIN SONG. *ANALYSIS AND DESIGN OF HIGH FREQUENCY LINK POWER CONVERSION SYSTEMS FOR FUEL CELL POWER CONDITIONING*. PhD thesis, Texas A&M University, August 2004.
CD: 'FULL-BRIDGE DC-DC CONVERTER.pdf'.

[20] Synplicity Inc. *Synplicity FPGA Synthesis, Synplify for Lattice Reference Manual*, September 2005.
CD: 'Synplicity - Synplify Reference Manual.pdf'.

Anhang A

Anhang

A.1 CD

Auf der beiliegenden CD sind Anhänge und Quellen in folgender Verzeichnisstruktur organisiert:

`Hardware/`
Enthält Schemas der erstellten Schaltung, sowie Datenblätter zu den verwendeten Bauteilen.

`Software/`
Enthält den geschriebene VHDL- und C-Code. Weitere Details zur Organisation und zum Inhalt stehen im Kapitel 3.4.1 auf der Seite 43.

`Literatur/`
Von uns zitierte Quellen sind in diesem Verzeichnis abgelegt, sofern sie digital vorliegen.

`Simulation/`
Beinhaltet die von uns verwendete Simulation des Gesamtsystems.

A.2 Schemas