Diplomarbeit

Thomas Sparr

Programmierung eines digitalen Pulssteuersatzes für Thyristorbrücken mit Ausregelung von Spannungsunsymmetrien

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Thomas Sparr

Programmierung eines digitalen Pulssteuersatzes für Thyristorbrücken mit Ausregelung von Spannungsunsymmetrien

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Informations- und Elektrotechnik
Studienrichtung Automatisierungstechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Gustav Vaupel
Zweitgutachter : Prof. Dr. Thomas Holzhüter

Abgegeben am 20. April 2010

Thomas Sparr

Thema der Diplomarbeit

Programmierung eines digitalen Pulssteuersatzes für Thyristorbrücken mit Ausregelung von Spannungsunsymmetrien

Stichworte

Thyristorbrücke, Welligkeit, Pulssteuersatz, Altera Quartus II, Unsymmetrien, FPGA, Pulsweitenmodulation und Phasenregelkreis (PLL)

Kurzzusammenfassung

Diese Arbeit umfasst die Entwicklung eines Pulssteuersatzes zur Ansteuerung von Thyristorbrücken. Die entwickelte Software wurde mit dem Altera Quartus II Programm geschrieben. Die Software sorgt für eine Welligkeitsminimierung auf der Ausgangsspannung der Thyristorbrücke. Dies geschieht durch Verschieben der Zündzeitpunkte der Thyristoren. Durch die Veränderung der Zündzeitpunkte wirkt sich die Netzunsymmetrie weniger auf die Ausgangsspannung aus. Zum Testen der Software wurde eine Sechspulsbrückenschaltung aufgebaut. Durch einen Widerstand in der Einspeisung der Thyristorbrücke wurde eine Netzunsymmetrie erzeugt. Diese Unsymmetrie wird selbstständig von der Regelung ausgeglichen.

Thomas Sparr

Title of the paper

Programming of a digital pulse control set for thyristor bridge with voltage unbalance deviation control

Keywords

Thyristor bridge, Ripple, pulse logic, Altera Quartus II, unbalance, FPGA, Pulse Width Modulation and Phase-locked-Loops

Abstract

This paper involves the development of a pulse control set for steering thyristor bridges. The developed software was written by means of the Altera Quartus II program. The software ensures a Ripple minimization on the output voltage of the thyristor bridge. This is achieved by shifting the ignition timing of the thyristor bridge. The change in ignition timing leaves the output voltage less affected by the net unbalance. A six pulse bridge rectifier was built in order to test the software. By introducing a resistor into the thyristor bridge, a net unbalance was created. This unbalance is adjusted autonomously by the control.

Inhaltsverzeichnis

I. **Abbildungsverzeichnis**

II. Abkürzungsverzeichnis

Abkürzung	Erklärung
LWL	Lichtwellenleiter
PLL	Phase-locked Loop
DAC	Digital Analog Converter
ADC	Analog Digital Converter
IP	Internet Protokoll
DESY	Deutsche Elektronen-Synchrotron
DCCT	Direct Current Current Transformer
PWM	Pulsweitenmodulation
DFF	Delay Flipflop
JKFF	Jump-/Kill- Flipflop
TCP/IP	Transmission Control Protocol / Internet Protocol

III. Begrifferklärung

Bezeichnung	Erklärung
VCC	Kollektorversorgungsspannung 5V
GND	Massepotenzial in der Elektronik
low	Bit=0 mit einer Spannung von 5V
high	Bit=1 mit einer Spannung von 0V
clk (Clock)	Taktung des Programms

IV. Formelverzeichnis

$$Phasenanschnittswinkel\_in\_Bit = 2000Bit \cdot \frac{Phasenanschnittswinkel\_in\_Grad}{360°} \quad (0.1)$$

$$Phasenanschnittswinkel\_in\_Grad = Phasenanschnittswinkel\_in\_ms \cdot \frac{20ms}{360°} \quad (0.2)$$

1 Einleitung

Diese Diplomarbeit wurde am „Deutsche Elektronen-Synchrotron“ durchgeführt. DESY ist ein Mitglied der Helmholtz-Gemeinschaft und eines der führenden Beschleunigerzentren zur Erforschung der Struktur der Materie. DESY entwickelt und baut große Teilchenbeschleuniger und forscht in den Bereichen Forschung mit Protonen und Teilchenphysik. Dies ist eine einmalige Kombination in Europa.

DESY ist ein mit öffentlichen Mitteln finanziertes nationales Forschungszentrum. Es sind etwa 2000 Mitarbeiter beschäftigt, davon arbeiten etwa 650 Wissenschaftler in den Bereichen Beschleunigerbetrieb, Forschung und Entwicklung. Dazu kommen noch jährlich über 3000 Gastforscher aus über 40 Nationen[1].

Diese Diplomarbeit wurde für die Gruppe Maschine Kraft Kühlung 6 erstellt und fällt in den Bereich der Beschleunigertechnik.

Bei DESY gibt es mehrere Speicherringe, in denen Teilchen auf einer Kreisbahn beschleunigt werden. Die Teilchen werden mit Hilfe der Lorentzkraft auf der Kreisbahn gehalten. Die Lorentzkraft tritt auf, wenn geladene Teilchen durch ein Magnetfeld fliegen. Aus diesem Grund befinden sich in den Speicherringen in regelmäßigen Abständen Elektromagnete. Zur Versorgung dieser Magnete werden Netzgeräte eingesetzt. Die Netzgeräte sind in verschiedenen technischen Gleichrichtungsverfahren umgesetzt. Eine Möglichkeit zur Gleichrichtung der Wechselspannung stellen Brückenschaltungen dar. Mit dieser Technik können hohe Leistungen erzeugt werden. Außerdem ist die Einstellbarkeit der Spannung bzw. des Stromes sehr genau möglich.

Die Thyristoren in den Sechspuls-Brückenschaltungen werden bisher mit einem analogen Steuersatz angesteuert. Bei den analogen Steuersätzen, werden Unsymmetrien die eine Welligkeit von 50 oder 100Hz bei der Ausgangsspannung hervorrufen, manuell ausgeglichen. Hierfür lassen sich die sechs Zündzeitpunkte der Thyristoren verschieben. Die Verschiebung wird durch einen Offset realisiert, der mit Potentiometern im Arbeitpunkt eingestellt wird.

Um diese analoge Steuerung jetzt digital umzusetzen, werden eine digitale Regelungselektronik und ein digitaler Steuersatz benötigt. Die digitale Regelungselektronik wurde schon entwickelt. Der digitale Steuersatz zur Erzeugung der Zündpulse wird als Software implementiert. Durch die Programmierung des Steuersatzes kann die Verschiebung der Zündzeitpunkte von der Software durchgeführt werden. Ein Algorithmus erkennt anhand des Spannungsverlaufes, ob eine Welligkeit in der Ausgangsspannung auftritt und eine mögliche Welligkeit wird über einen PI-Regler ausgeregelt. Die Welligkeit in der Ausgangsspannung wird im Verlaufe der Arbeit als Ripple bezeichnet.

Im Vorfeld dieser Diplomarbeit wurde im Rahmen einer Studienarbeit eine Simulation zur Umsetzbarkeit durchgeführt. Die Studienarbeit „Simulation eines digitalen Steuersatzes mit eigenständiger Ausregelung von auftretender Welligkeit“ beinhaltet eine Simulation mit dem Ansoft Simulationsprogramm Simplorer.

2 Aufgabenstellung

Entwurf und Programmierung einer digitalen Ansteuerung von Thyristorbrücken mit Regelung der Ausgangswelligkeit

Für die Netzgeräte des PETRA III Beschleunigers bei DESY wurde eine digitale Regelungselektronik entwickelt. Als zentrale Recheneinheit werden ALTERA FPGAs eingesetzt. Die Programmierung erfolgt mit Hilfe der Entwicklungsumgebung Quartus.

Derzeit existieren Steuersätze für bipolare Schaltnetzgeräte mit H-Brücke sowie eine Ankopplung an Tiefsetzsteller. In der Software sind Schnittstellen zur Ausgabe der jeweiligen Pulsmuster implementiert. Die Software wird auf den FPGA der sogenannten Interfacekarte gespeichert. Diese Karte wird in die jeweiligen Leistungsteile eingebaut und steuert diesen.

Für die Interfacekarte soll in dieser Diplomarbeit ein Programm erstellt werden, mit dessen Hilfe Sechspulsbrücken angesteuert werden können. Die Zündpulse sollen mit einer Regelung in soweit verschiebbar sein, dass Unsymmetrien, die sich auf die Ausgangswelligkeit der Thyristorbrücke auswirken, reduziert werden. Die Arbeit umfasst die folgenden Punkte:

- Erzeugung der Zündpulse für B6 bzw. B12 Brückenschaltungen
- Einsatz bzw. Modifikation eines vorhandenen Programmteiles einer PLL
- Einbindung einer bereits implementierten PLL zur Erzeugung eines stabilen netzsynchronen Triggers für die Synchronisation der Zündpulse
- Aufbau und Inbetriebnahme einer Thyristorbrücke mit niedriger Spannung und niedrigem Strom
- Ansteuerung dieser Thyristorbrücke mit der Interfacekarte
- Überprüfung der zuverlässigen und stabilen Ausgabe der Zündpulse über längere Zeiträume
- Entwurf einer Regelung zur Verkleinerung der Ausgangswelligkeit der Sechspulsbrücken
- Test der Regelung mit Fehlerszenarien (Spannungsfälle in der Versorgungsspannung und Sollwertsprünge)

3 Entwicklungsumgebung und Hardwarekomponenten

Zur technischen Umsetzung der Aufgabenstellung wurden zur Softwareimplementierung eine Entwicklungsumgebung und bereits existierende Hardware verwendet. Die Aufgabe und die Funktion dieser Bestandteile wird in diesem Kapitel erklärt.

3.1 Entwicklungsumgebung

MAX+plus II und Quartus II

Zur Entwicklung und Programmierung des Steuersatzes wurden zwei Entwicklungsumgebungen verwendet. Beide Programme sind von der Firma Altera und heißen MAX+plus II und Quartus II.
Es handelt sich bei beiden Programmen um eine integrierte Entwicklungsumgebung für den funktionellen Entwurf digitaler Systeme

- in einer Hardwarebeschreibungssprache
- in Form von graphischen Funktionsplänen
- in Mischformen[2]

Das Programm Quartus II ist die Überarbeitung von MAX+plus II. Deswegen sind die Grundfunktionen der beiden Programme gleich. Quartus II wurde wegen der steigenden Komplexität von Softwarelösungen erweitert.

Die MAX+plus II Software wird von Altera nicht mehr vertrieben und somit gibt es keine Updates mehr. Die Firma Altera stellt zur Programmierung in der Entwicklungsumgebung die Sprachen VHDL, AHDL und Verilog zur Verfügung.

Die gesamte Software zur Pulserzeugung wurde in AHDL geschrieben[3]. Diese Programmiersprache ist VHDL sehr ähnlich. Der Vorteil ist die sehr übersichtliche Darstellung durch graphische Anteile. Die einzelnen Teilfunktionen werden in eigenen Blöcken geschrieben. Diese Blöcke haben Ein- und Ausgänge mit denen sie untereinander verbunden werden. Durch diese Aufteilung des Programms in verschiedene Blöcke werden die Programmcodes in den einzelnen Blöcken kurz gehalten. Die kleinen Teilfunktionen können getrennt voneinander getestet werden. Die Funktionstests sind mit MAX+plus II durchgeführt worden.

In MAX+plus II können in einem Diagramm die Entwicklung der Eingangssignale vorgegeben werden. Das Programm zeigt dann wie sich die Ausgänge bei diesen Eingangskonstellationen verhalten.

Z.B. kann bei einem Flipflop der Signalverlauf des Setz- und Rücksetzeingangs vorgegeben werden. Durch ein High-Signal am Setzeingang würde die Simulation den Ausgang des Flipflops dann auf high setzen. Ein anschließendes High-Signal am Rücksetzeingang zieht den Ausgang auf low. Nach Beendigung der Simulation werden die verwendeten Variablen graphisch dargestellt.

3.2 Hardwarekomponenten

Die Software, die entwickelt werden soll, setzt verschiedene Hardwarekomponenten voraus. Dieses Kapitel soll die Komponenten und deren Zusammenspiel veranschaulichen.

Die Abbildung 3.1 zeigt die Komponenten und wie sie untereinander verbunden sind.

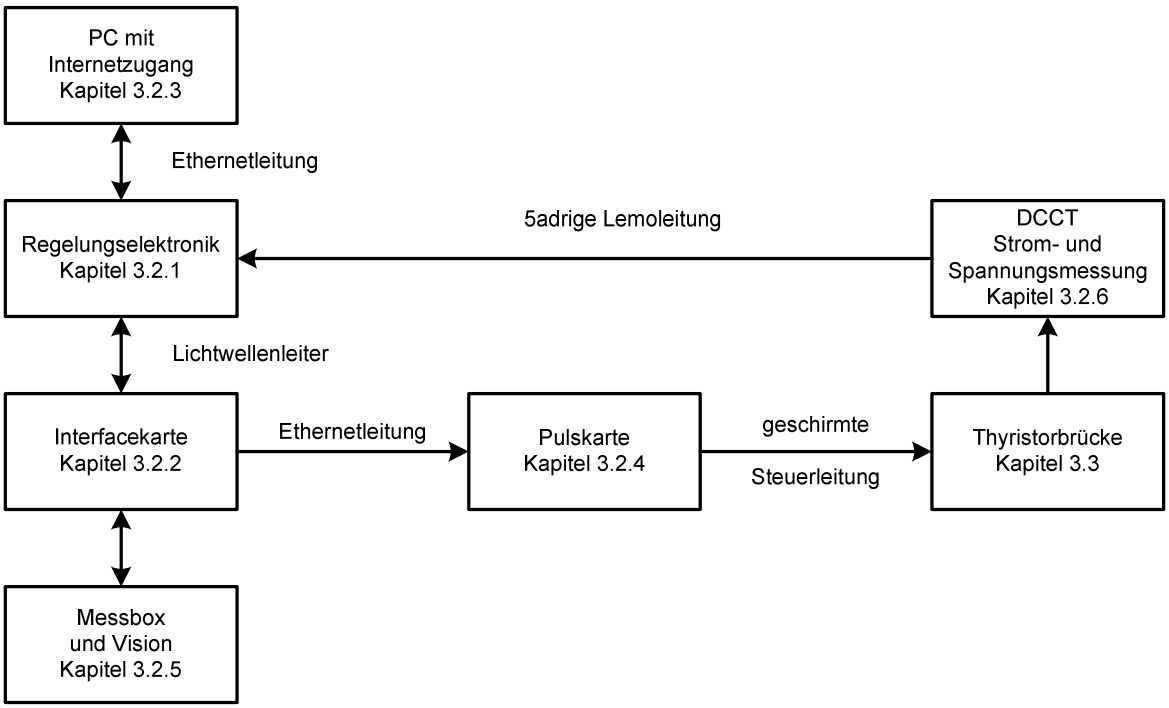


Abbildung 3.1: Übersicht der Hardwarekomponenten

Welche Funktion die einzelnen Bauteile haben, soll in der Folge erläutert werden.

3.2.1 Komponente „Regelungselektronik“

Die digitale Regelungselektronik ist die zentrale Komponente in der Anlage. Die Software auf dem FPGA übernimmt daher übergeordnete Aufgaben. Die Regelungselektronik steuert die Leistungsteile, hat eine hoch präzise Strommessung und enthält den äußeren Regelkreis einer Kaskadenregelung, die den Sollwert des Magnetstromes sicherstellt. Die übergeordneten Aufgaben sind bei dem jetzigen Stand der Ansteuerung von Thyristorbrücken noch nicht relevant. Deswegen beschränkt sich die Funktion bei Thyristorbrücken derzeit auf das Umsetzen von Daten.

Die Regelungselektronik hat eine eigene IP und über diese IP kann mit Hilfe eines Browsers auf die Internetseite der Regelungselektronik zugegriffen werden. Die Kommunikation zwischen der Internetseite und der Interfacekarte geschieht über die Regelungselektronik. Die Datenpakete werden auf der Regelungselektronik umformatiert, damit eine Übertragung über zwei verschiedene Schnittstellen möglich ist. Die Daten von der Internetseite kommen im TCP/IP Protokoll und werden so verändert, dass eine Übertragung über den LWL möglich ist[4].

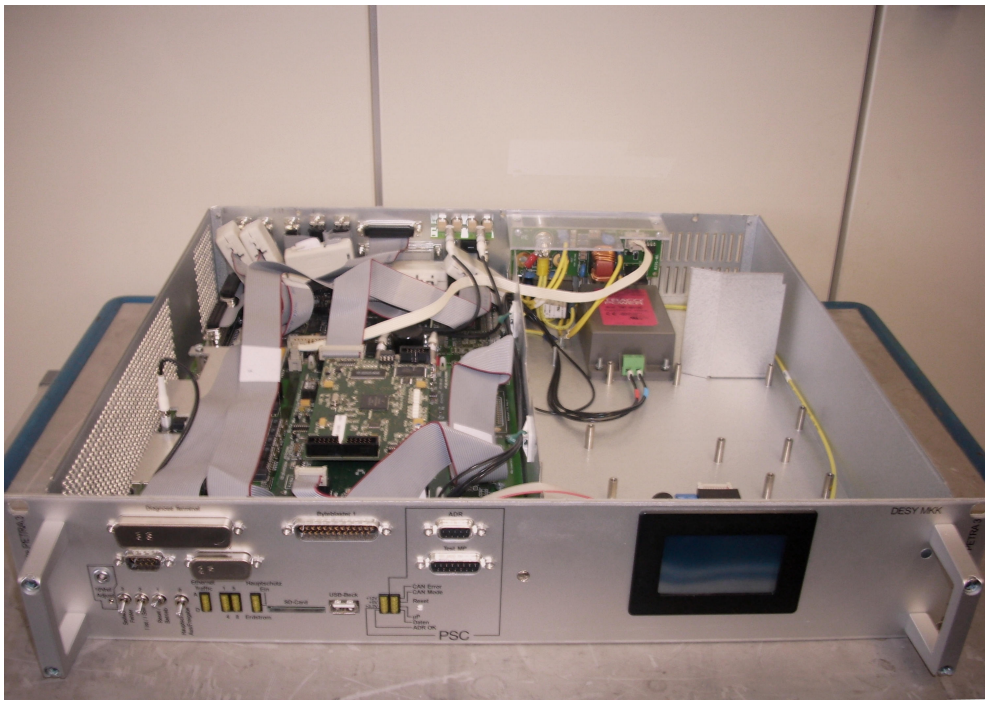


Abbildung 3.2: Bild der Regelungselektronik

In Abbildung 3.2 sind die Front und die internen Komponenten der Regelungselektronik zu sehen.

3.2.2 Komponenten „Interfacekarte“ und „Adapterkarte“

Bei der Entwicklung der Hardware ist für die Interfacekarte eine Adapterkarte entwickelt worden. Diese Adapterkarte dient zur Anpassung der Standardkarte mit dem Leistungsteil. Sie wird fest im Leistungsteil eines Gerätes verbaut. Durch eine Einschuböffnung wird die Interfacekarte auf die Adapterkarte aufgesteckt.

Die Interfacekarte wird für verschiedene technische Gleichrichtungsverfahren verwendet. Statt für jede dieser Anwendungen eine separate Software zu programmieren, wird bei DESY ein gemeinsames Programm verwendet. Diese Lösung macht es leichter, das Programm zu pflegen und lässt keine Verwechselungen bei der Programmauswahl zu. Die Ansteuerhardware soll in Zukunft für alle Gleichrichtungsverfahren gleich sein. Somit ist jede Hardwarekomponente variabel einsetzbar und wird immer mit der gleichen Software programmiert.

Die Interfacekarte erzeugt die Zündpulse, führt die Pulsweitenmodulation und enthält den inneren Spannungsregelkreis der Kaskadenregelung, der zur Sicherstellung des Sollwertes des Magnetstromes verwendet wird. Die Zündpulse werden über Ethernetkabel an die Pulskarte und weiter an die Thyristoren übertragen. Zusätzlich besteht die Möglichkeit, über Lemokabel oder Flachbandkabel Daten auszugeben.

Außerdem kann die Interfacekarte analoge Werte verarbeiten. Die analogen Werte werden mit ADCs digitalisiert. Das wird z.B. mit der Ausgangsspannung der Thyristorbrücke gemacht.

Die Abbildung 3.3 zeigt die für den Versuchsaufbau verwendete Kombination von Interfacekarte und Adapterkarte. Die Adapterkarte ist fest auf einen Träger montiert und wird über einen Transformator mit Spannung versorgt. Die Interfacekarte ist über eine Schiene aufsteckbar und kann mit Schrauben gesichert werden.

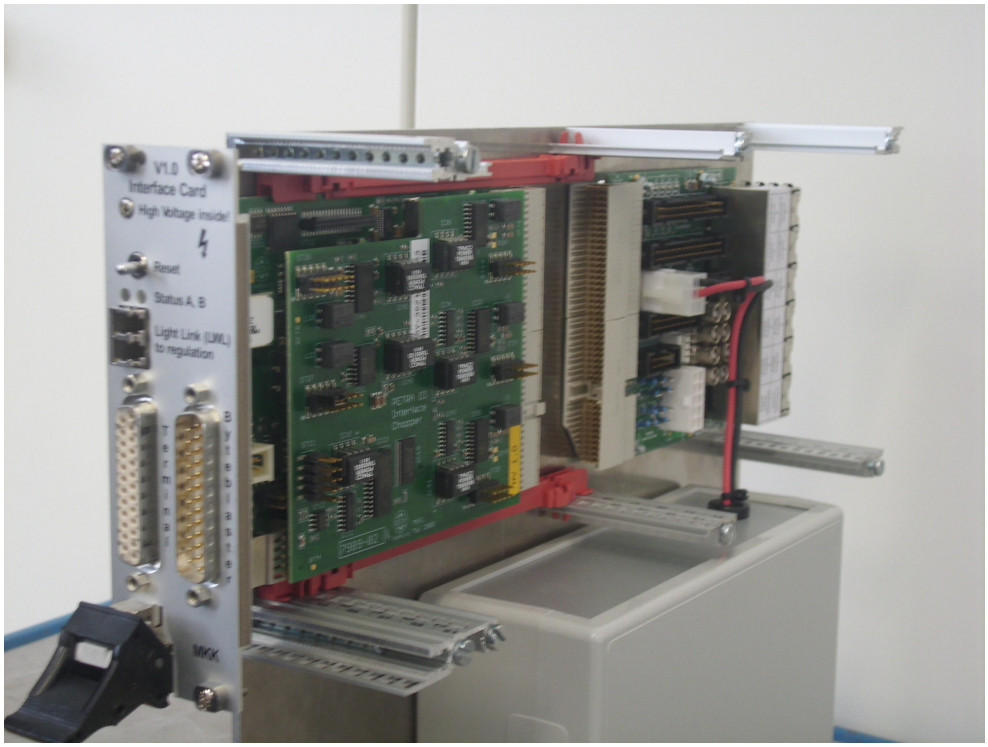


Abbildung 3.3: Bild der Interfacekarte mit Adapterkarte

3.2.3 PC mit Internetzugang

Die Internetseite (Abb. 3.4) von der „Regelungselektronik“ dient zur externen Bedienung und zum Auslesen von Daten.
In der Praxis können so von einem Kontrollraum alle Regelungselektroniken überwacht werden. Die Momentanwerte von z.B. Ausgangsspannung, Temperatur oder Freigabemeldungen können jederzeit abgerufen werden und Fehlermeldungen werden angezeigt.

Die nachfolgende Graphik hebt durch die Nummerierung die relevanten Parameter und Messwerte hervor.

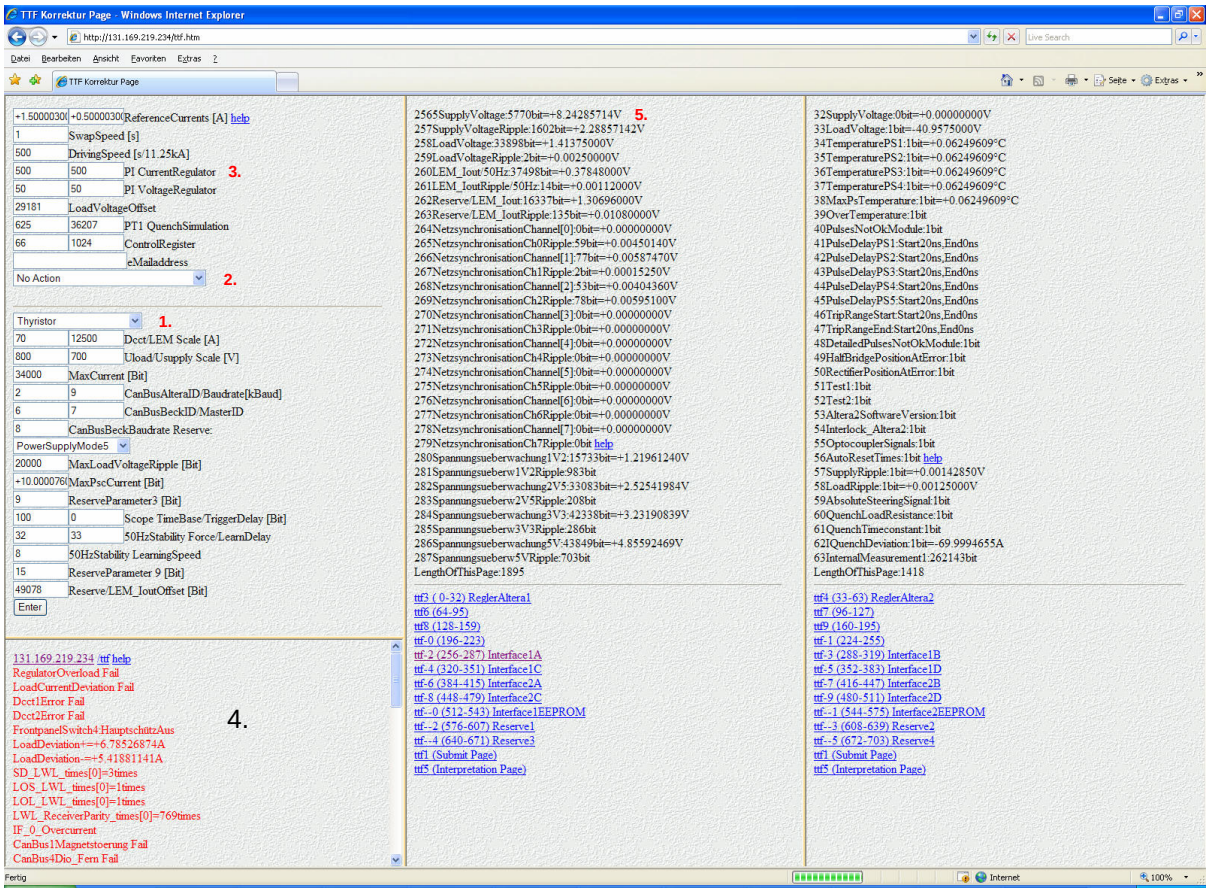
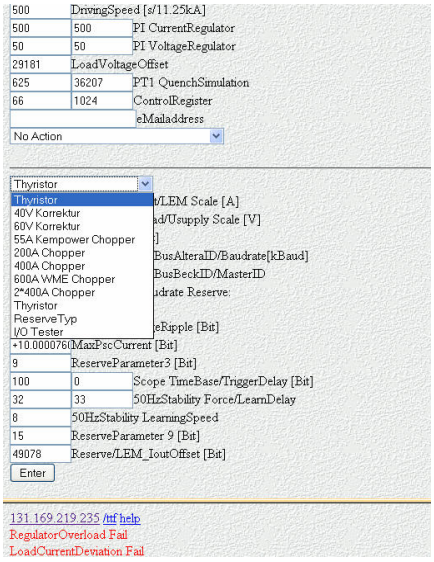


Abbildung 3.4: Screenshot der Internetseite

3.2.3.1 Einstellungsmöglichkeiten auf der Internetseite

Im linken Drittel der Internetseite können Parameter eingetragen werden und Fehlermeldungen werden unten in Rot angezeigt. Wenn ein Thyristorgerät betrieben werden soll, müssen verschiedene Parameter eingestellt werden.



In Abbildung 3.4 muss bei 1. aus einem Menü die Auswahl des Gerätes erfolgen. Dieses Menü ist in Abbildung 3.5 detailliert dargestellt. In dieser Liste wird „Thyristor“ gewählt, um in der Software die richtigen Programmteile zu aktivieren.
Die verschiedenen Gleichrichtungsverfahren haben unterschiedliche Abkürzungen, die eingestellt werden können. Für eine Sechspulsbrücke wird „Thyristor“ gewählt. Die Abkürzung für Brückenschaltungen war bereits in der Entwicklung des Internetseitenlayouts eingebunden.

Abbildung 3.5: Detailansicht des Auswahlmenüs auf der Internetseite

Die Einstellungen können durch einen Menüpunkt bei 2. gespeichert werden. Dann sind nach einem eventuellen Ausschalten der Regelungselektronik die veränderten Einstellungen von Beginn an eingestellt. In der Testphase kann es durch Variation der Parameter zu veränderten Einstellungen kommen. Dann können die gespeicherten Einstellungen manuell unter 2. geladen werden.

Durch die Möglichkeit die Reglerparameter bei 3. zu verändern, muss die Software nicht nach jeder Änderung der Parameter neu kompiliert und eingespielt werden. Dadurch wird ein produktiveres Arbeiten ermöglicht.

Im linken unteren Bereich werden bei 4. die Fehlermeldungen angezeigt. Dieser Bereich ist beim jetzigen Stand der Programmierung noch nicht relevant. Später muss hier z.B. angezeigt werden, ob alle Thyristoren zünden oder ob die Eingangsspannung anliegt.

In den anderen zwei Dritteln werden Variablen der Software angezeigt. Hinter den blauen Links im unteren Bereich können jeweils 30 Werte angezeigt werden. Es werden so über 700 Werte ausgegeben, die jederzeit einsehbar sind. In der Abbildung 3.4 wird unter 5. die Ausgangsspannung der Thyristorbrücke in Volt und in Bit angezeigt.

3.2.4 Komponente „Pulskarte“

Die Pulskarte dient zur Leistungsumsetzung und Potenzialtrennung von sechs Signalen. Die Zündpulse von der Interfacekarte haben zu wenig Leistung, um die Thyristoren der Thyristorbrücke direkt zu zünden. Die Pulskarte setzt die Zündpulse der Interfacekarte mit einer höheren Leistung um. Die Potenzialtrennung bewirkt eine Trennung von Steuer- und Lastteil. Für diese Leistungserhöhung wird eine 230V Versorgungsspannung verwendet.

Abbildung 3.6: Bild der Pulskarte im externen Gehäuse

Die Abbildung 3.6 zeigt die Pulskarte in einem externen Gehäuse. Von oben kommen die sechs Ethernetleitungen von der Interfacekarte. Das leistungsstärkere Signal geht über sechs geschirmte Pulskabel zur Thyristorbrücke. Die Pulskabel kommen von unten in das Gehäuse und die Schirmungen sind mit dem Gehäuse verbunden. Die Schirmung der Pulskarte wird dann zu einer zentralen Erdung am Thyristorbrücken-Nachbau geführt. Hier werden die Erdungen sternförmig zusammengeführt.

3.2.5 Komponente „Messbox“

Um den zeitlichen Verlauf der digitalen Werte anzusehen, kann man ausgewählte Variable mit der Messbox analogisieren und auf einem Oszilloskop abbilden.

Die Messbox kann an einer D-Sub Schnittstelle, an der Interfacekarte und der Regelungselektronik angeschlossen werden. In der Software sind beliebige Daten auf die Schnittstelle gelegt und sie können durch Programmänderungen angepasst werden. Die Messbox setzt diese Daten um.

Die Messbox hat die Möglichkeit zeitgleich sechs Bitwerte und sechs Analogwerte auf Lemobuchsen auszugeben. Die Analogwerte kommen jeweils aus einem DAC, der einen Digitalwert von 16 Bit Breite umsetzen kann.

Diese 12 Ausgänge sind durch ein Menü zwischen 14 Ebenen umschaltbar. Das bedeutet, dass man 168 Variablen ohne Programmänderung ständig abfragen kann.

Die Werte können dann auf einem Oszilloskop angezeigt werden. Hier sind die Variablen der Software im Gegensatz zur Internetseite nicht nur als diskrete Werte zu sehen, sondern auch ihr zeitliche Signalverlauf.

Es ist außerdem möglich, über einen Potentiometer an der Messbox eine Variable an die Interfacekarte zu übergeben. Die analogen Werte können einer entsprechenden Variablen zugeordnet werden. Im gesteuerten Betrieb kann so der Phasenanschnittswinkel mit Hilfe des Potentiometers verändert werden.

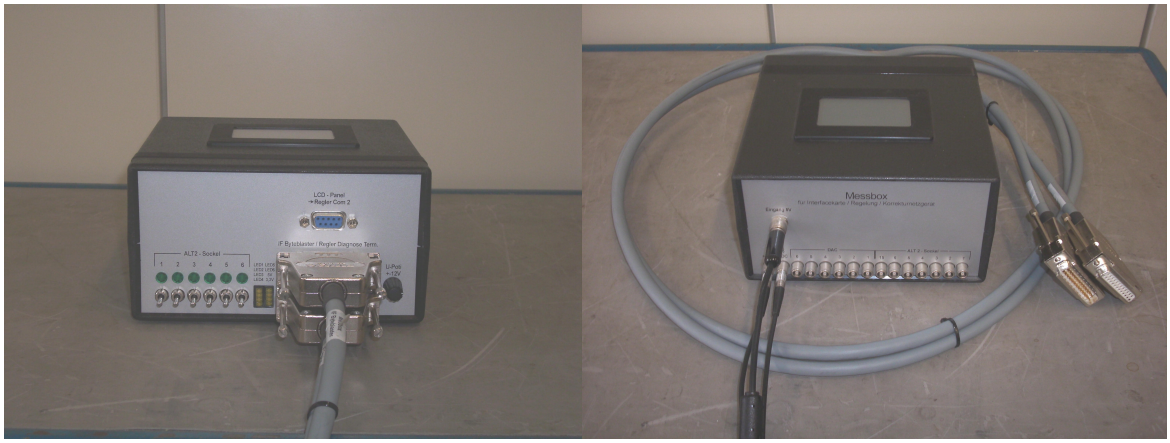


Abbildung 3.7: Vorder- und Rückansicht der Messbox

Bei der Darstellung von zeitlichen Entwicklungen von Messwerten ist ein digitales Oszilloskop nur bedingt einsetzbar. Aus diesem Grund wurden die Funktionstest mit dem digitalen Messdaten-Erfassungssystem Vision aufgezeichnet. Das Vision ist ein mobiles Datenaufzeichnungsgerät das acht analoge Signalverläufe aufzeichnen kann. Die Daten können auf einem integrierten 10 Zoll Bildschirm angezeigt und ausgewertet werden oder über einen USB-Stick auf den PC übertragen werden. Die Software Data Viewer 3 ermöglicht das Abrufen und Bearbeiten von Aufzeichnungen auf dem PC. Es ist weiterhin möglich die Kanaleinstellungen zu verändern und die Signalverläufe zu analysieren.

3.2.6 Direct Current Current Transformer DCCT

Um präzise Gleichströme zu messen werden, DCCTs eingesetzt. Die Messung erfolgt über das Prinzip der Kompensation von zwei entgegengesetzten magnetischen Flüssen.
Am Blockdiagramm (Abb. 3.8) kann man das Prinzip der Messung erkennen. Der Fluxdetector gibt ein Rechtecksignal mit der Amplitude U_p=-U_p auf die Spule S. Bei einem Strom I_p=0 ist die gleichgerichtete Spannung null und somit y und x auch. Wenn I_p ansteigt, verschiebt sich durch die Magnetisierung des Eisenkerns die Spannung über S. Es gilt Up>-Up. Die daraus resultierende Gleichspannung führt zu einem Fehlersignal auf y. Der Wert von y gibt die, durch die Gleichspannung verursachte, Nullpunktverschiebung bei der Rechteckspannung an. Der Verstärker A regelt den Strom I_C solange bis die Magnetisierung in der Spule C ausreicht, die Magnetisierung durch den Gleichstrom auszugleichen. Dadurch wird die Rechteckspannung wieder symmetrisch und y wird null. I_c fließt durch den Widerstand R_B und dieser Spannungsabfall ist dann proportional zum Strom. Dieser analoge Wert wird an die digitale Regelungselektronik übertragen. Hier wird der Wert mit einem hochpräzisen ADC digitalisiert.

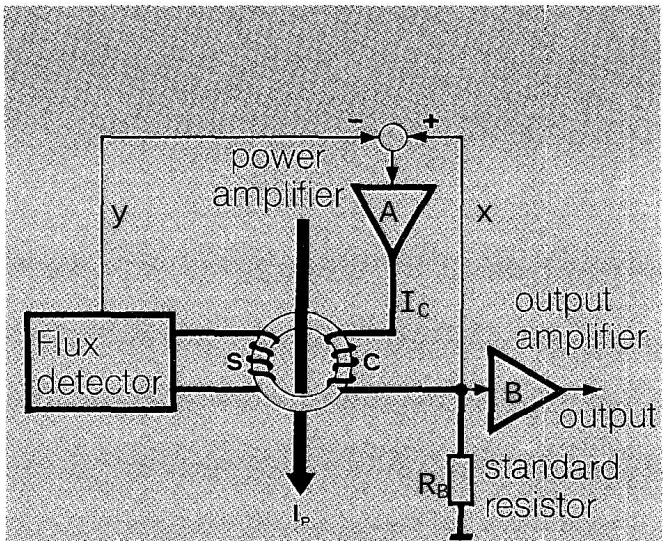


Abbildung 3.8: Prinzipschaltbild des DCCTs

3.3 Thyristorbrücke

Bei DESY werden Thyristorgeräte in den Beschleunigeranlagen mit hohen Leistungen betrieben. Um in der Entwicklungsphase den Steuersatz testen zu können, sind so hohe Leistungen nicht geeignet. Durch eine falsche Zündreihenfolge könnte es zu Leistungserhöhungen bei einzelnen Pulsen kommen und dies könnte zu erheblichen Schäden der Anlage führen. Zudem müssen zwei Personen während des Betriebes anwesend sein.

Aus diesem Grunde sollte für Tests eine Thyristorbrücke mit den ungefähren Ausgangskenndaten von 20-30V und 1-2A aufgebaut werden. Der in Abbildung 3.9 gezeigte Aufbau erfolgte ausnahmslos mit vorhandenen Bauteilen, deswegen sind z.B. die Thyristoren überdimensioniert.

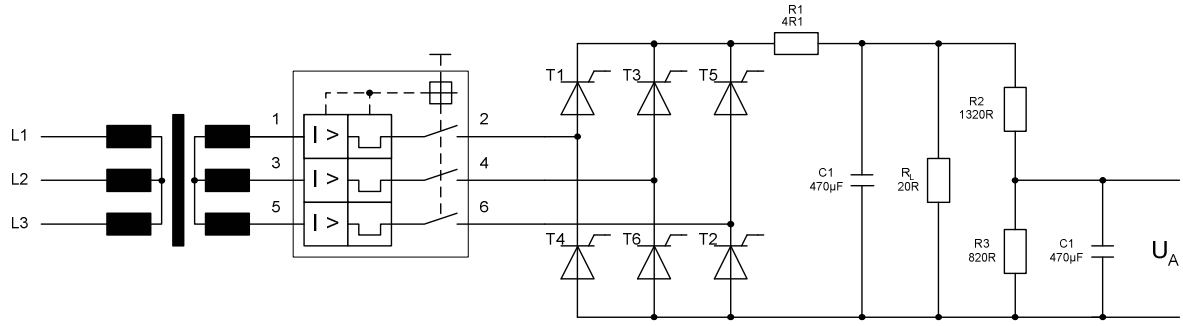


Abbildung 3.9: Schaltplan der Thyristorbrücke

Kenndaten:

Transformator:	Ausgangsspannung	3x14,5V
	Ausgangsstrom	2,5A
Thyristorbrücke:	Semikon SKKT 42/16 E G6	
	Datenblatt (siehe Anhang A3)	
Motorschutz:	Typ: GV2-M08	
	Auslösestrom:2,5A-4A	

Die Thyristorbrücke hat bei Vollaussteuerung 20V und 1A. Bei diesen Ausgangsgrößen ist bei Fehlfunktionen keine Gefährdung gegeben. Außerdem kann die Versuchsanlage ohne zweite anwesende Person betrieben werden.

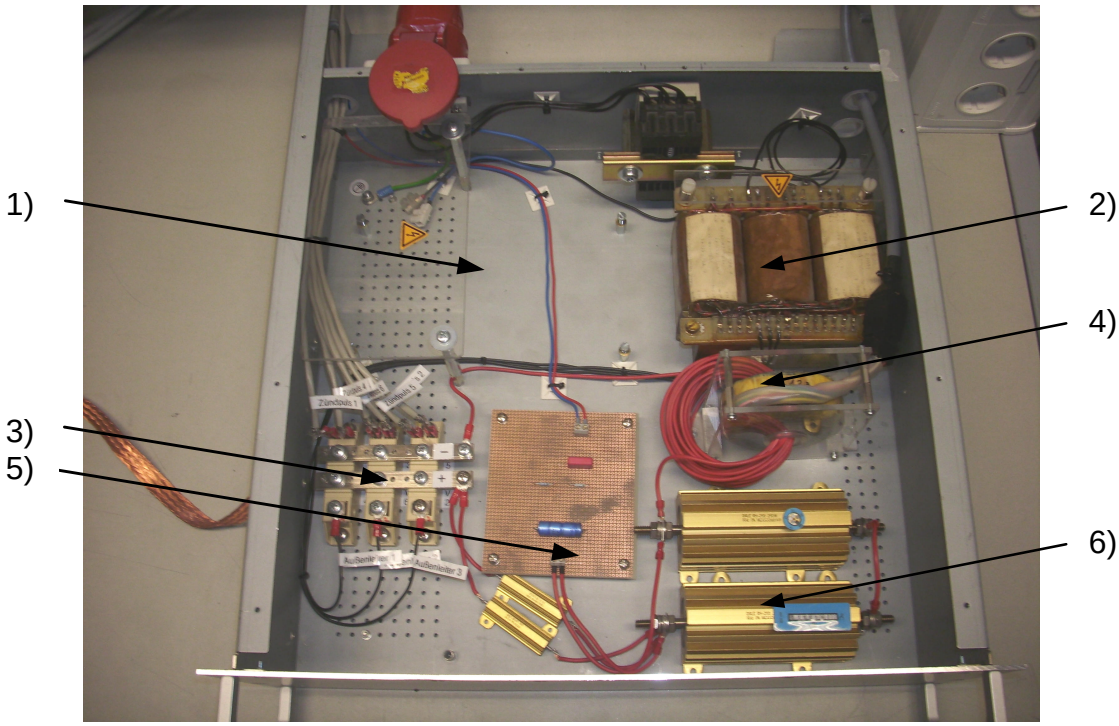


Abbildung 3.10: Bild der Thyristorbrücke

Die Abbildung 3.10 zeigt die zum Testen aufgebaute Thyristorbrücke.
Erklärung des Aufbaus:

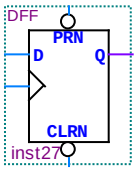
1) Ehemaliger Platz für die Pulskarte. Nach Auftreten von EMV- Effekten wurde ein separates Gehäuse für die Pulskarte verwendet.
2) Transformator zur Erzeugung der Versorgungsspannung für die Thyristorbrücke.
3) Thyristorbrücke mit sechs Thyristoren. Es sind immer zwei Thyristoren in einem Modul verbaut.
4) DCCT-Kern mit 20 Windungen. Die 20 Windungen durch den Kern sind wegen des geringen Ausgangsstromes nötig. Der DCCT- Kern setzt 60A Ausgangsstrom in 10V Messspannung um. Der Wertebereich wird durch die 20 Windungen durch den Kern effektiver genutzt. Somit ist bei Vollaussteuerung eine Stromstärke von 20A bezogen auf die Messung gegeben. Mehr Windungen sind wegen des geringen Durchmessers der Öffnung im DCCT-Kern nicht möglich.
5) Auf der Platine ist der Spannungsteiler zur Erzeugung einer Ausgangsspannung von 10V und der Tiefpassfilter verlötet.
6) Die Brückenschaltung wird mit zwei 10Ω Widerständen belastet. Die Widerstände sind in Reihe geschaltet und setzen die Ausgangsleistung der Thyristorbrücke in Wärme um.

4 Zündpulserzeugung

Das Konzept zur Erzeugung der Zündpulse beruht auf Countern. Die Programmierung auf der Grundlage von Countern wurde wegen der Programmiererfahrung der Entwicklungsingenieure bei DESY gewählt. Bei der Programmierung einer Pulsweitenmodulation ergeben sich durch die Counter gewisse Vorteile. Die Umsetzung von Winkeln in Zeit ist durch Counterwerte leicht zu ermitteln. Die zeitlich präzise Zündpulsausgabe ist mit einer Genauigkeit von 80ns möglich.

4.1 Programmierung eines Counters in AHDL

Die Counter werden in AHDL mit Delay Flipflop Bausteinen realisiert[5]. Ein DFF ist ein flankengesteuertes Verzögerungsflipflop, das erst beim Taktsignal mit dem Ausgang auf das Eingangsignal reagiert.



Ein DFF Baustein hat einen Eingang D, einen Ausgang Q, einen Löscheingang CLRN und einen Clock-Eingang, der mit einem Dreieck gekennzeichnet ist oder clk heißt. Bei Programmstart sind Q und D mit Null initialisiert. Bei jedem Clock wird vom Baustein der Wert D auf Q geschrieben. Indem jetzt der Eingang D bei jedem Zyklus den um eins erhöhten Wert von Q erhält, wird der Ausgangswert Q bei jedem Zyklus um eins größer. Somit stellt der Ausgang Q vom DFF den Counterwert dar. Ein 10 Bit breiter DFF kann so von 0 bis 1023 zählen.

4.2 Aufbau der Blöcke in den Altera Entwicklungsumgebungen

Der Simulationsaufbau in MAX+plus II ist in Abbildung 4.1 zu sehen.

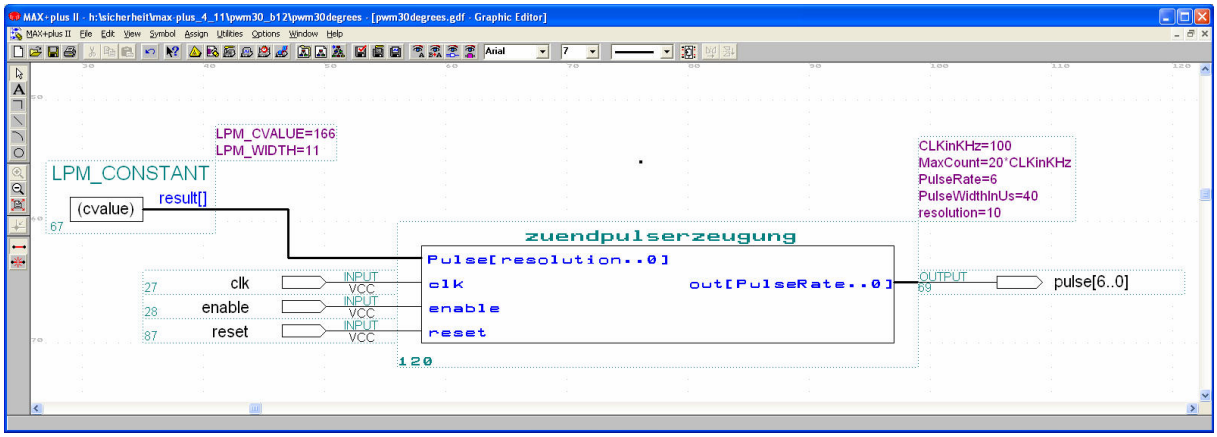


Abbildung 4.1: Screenshot der graphischen Darstellung der Zündpulserzeugung

Das Programm besteht aus zwei sogenannten Blöcken. Der Block „LPM_CONSTANT“ ist in der Programmbibliothek der Entwicklungsumgebung vorhanden und dient zur Vorgabe eines Phasenanschnittswinkels. Er gibt einen konstanten Bitwert aus, der in Breite und Wert verändert werden kann. Diese Werte können in der Parameterliste, die in der oberen rechten Ecke des Blocks angehängt ist, verändert werden. In der Parameterliste führt man die Parameter auf, die in einem bestimmten Rahmen verändert werden können. Dies erleichtert Änderungen und es ist schnell zu sehen welche Einstellungen gerade gewählt wurden. Sobald in einem Block Parameter definiert werden, wird diese Liste automatisch mit erstellt.

Bei den Blöcken befinden sind auf der linken Seite immer die Eingänge und rechts die Ausgänge. Dadurch können sie untereinander verbunden werden, wobei die betreffenden Ein- und Ausgänge die gleiche Bitbreite haben müssen. Der Block „LPM_CONSTANT“ übergibt ein elf Bit breites Signal an die „zuendpulserzeugung“. Die anderen Eingänge bekommen „INPUT“ Variablen zugeordnet. Hierüber können diese Eingänge in der Simulation Signale bekommen.

An die Ausgänge werden „OUTPUT“ Variablen angeschlossen, damit die Werte in der Simulation dargestellt werden können.

4.3 Erklärung der Simulation

In der Simulation ist es sinnvoll, eine möglichst kurze Zykluszeit zu wählen, damit die Simulation nicht mit einem zu großen Zeitaufwand verbunden ist. Mit Hilfe der Parameterliste können die Blöcke variabel programmiert werden. Eine bestimmte Taktung der Software benötigt eine Mindestbitbreite der Variablen. Das bedeutet, dass ein Counter, der bis 2000 zählen muss, nur die dafür nötige Bitbreite benötigt. Diesen Sachverhalt soll das folgende Beispiel erläutern.

Zur ersten Erprobung wurde eine Taktung von 100KHz gewählt. Diese Taktung muss über den „clk“ Eingang an die Blöcke übergeben werden. Das führt zu 2000 Programmzyklen pro 20ms. Die Periodenlänge in der alle sechs Thyristoren gezündet werden, ist wie in der Realität auf eine Frequenz von 50Hz definiert. Der Counter, der die gesamte Periode durchläuft, muss z.B. bei 2000 Programmzyklen in 20ms nur bis 2000 zählen können. Deswegen wurde für „resolution“ eine Zehn eingesetzt. Der Counter ist durch den Parameter „resolution“ auf [10..0] definiert. Von null bis zehn sind elf Bit, und das führt zu einem Maximalwert von 2048Bit.

Die folgende Formel belegt das:

$$2^{11} Bit = 2048 Bit$$

Durch den Parameter „resolution“ kann die Bitbreite der Variablen an die Zykluszeit angepasst werden.

Durch die Flexibilität der Taktfrequenz wird der Wert von „MaxCount“ mit der Formel 20*CLKinKHz aus der Taktfrequenz errechnet. Aus der 100KHz Taktfrequenz mal 20 ergibt sich ein „MaxCount“ Wert von 2000.

4.4 Erklärung der Ein- und Ausgänge

Der Block „zuendpulserzeugung“ (Abbildung 4.1) wurde für diese Arbeit entwickelt und dient der Zündpulserzeugung. Dieser Block hat vier Eingänge und einen Ausgang.

Der Eingang „Pulse[resolution..0]“ ist mit dem Wert „resolution“ aus der Parameterliste auf elf Bit definiert. Dieser Eingang bekommt vom Block „LPM_CONSTANT“ den Wert für den Phasenanschnittswinkel. Nach diesem Wert richtet sich dann der Zeitpunkt, an dem die Zündpulse an die Thyristoren ausgegeben werden.

Den Programmtakt bekommt der Block über den „clk“ Eingang. Die Taktfrequenz gibt die Geschwindigkeit an, mit der die Counter hochzählen.

Bei den Blöcken gibt es Eingänge, die im Fehlerfall ein Eingreifen in den Ablauf gestatten. Hierfür sind ein „reset“ Eingang und ein „enable“ Eingang vorgesehen. Der „reset“ Eingang setzt alle wichtigen Variablen auf null. Der Eingang ist Low aktiv. Der „enable“ Eingang ist High aktiv und sperrt die wichtigen Variablen. Als wichtige Variablen gelten z.B. die Counter und die Ausgänge. Wenn im Programm ein Fehler auftritt, wird „enable“ auf Low gesetzt und durch die Sperrung der Counter bleibt das Programm stehen. Ausgangsvariablen können zusätzlich gesperrt werden.

Der Ausgang „out[Pulserate..0]“ gibt die Zündpulse aus. In der Parameterliste wird eingestellt, ob man eine Sechs- oder Zwölfpulsbrücke betreiben will. Bei einer Zwölfpulsbrücke sind die Ausgänge von zwei Sechspulsbrücken zusammengeschaltet. Dadurch ist die Ausgangsspannung durch die Zwölf- statt Sechspulse deutlich glatter. Die jeweils sechs Pulse der beiden Brücken sind zueinander um 15° verschoben, damit die erzeugte Gleichspannung nur aus den Scheitelpunkten der zwölf Pulse besteht. Der Block „zuendpulserzeugung“ kann beide Sechspulsbrücken mit den 15° Verschiebung zwischen den beiden Sechspulsbrücken ansteuern Der Ausgang ist dann 7 oder 13 Bit breit. Das nullte Bit ist als Zusatzbit festgelegt. Mit diesem Bit kann zum Debuggen ein beliebiges Signal aus dem Block ausgegeben werden. Es ist manchmal sinnvoll, sich z.B. ausgeben zu lassen, wann ein Counter wirklich gelöscht wird, um mögliche Fehler beheben zu können.

4.5 Erklärung des Blockes „zuendpulserzeugung“

Die Funktionen der einzelnen Blöcke soll in der Folge erklärt werden. Zur detaillierten Beschreibung befindet sich im Anhang und auf der beigelegten DVD der ausführlich kommentierte Sourcecode.

Wie zu Beginn des Kapitels erklärt, bilden Counter die Basis der Zündpulserzeugung. Im Programm sind 13 Counter in einem zweidimensionalen Feld definiert. In dem Counterfeld ist für jeden Thyristor ein eigener Counter vorhanden. Die Counter eins bis sechs sind für eine Sechspulsbrücke und sieben bis zwölf für die zweite. Die Erzeugung der Zündpulse für die Erweiterung von einer Sechs- auf eine Zwölfpulsbrücke ist für dies Arbeit noch nicht relevant. Es wird nur mit einer Sechspulsbrücke gearbeitet, aber die Programmierung der Zündpulserzeugung ist bereits für Zwölfpulsbrücken ausgelegt.

Als Beispiel „CounterDFF[2][10..0]“ ist für die Zündpulserzeugung von Thyristor 2 der ersten Sechspulsbrücke zuständig. Der Counter „CounterDFF[0][10..0]“ ist der Maincounter. Dieser Maincounter dient zur Synchronisation der Perioden. Zur ersten Erprobung wurde noch kein PLL Signal verwendet. In der Simulation war eine feste Frequenz von 50Hz gegeben und deswegen wird der Maincounter genau nach 20ms auf null gesetzt. Der Parameter, der den Counter zurücksetzt, ist in der Parameterliste mit „MaxCount“ aufgeführt.

Die Abbildung 4.2 zeigt den „Waveform Editor“ von MAX+plus II. Damit dieser arbeiten kann, muss für die „INPUT“ Variablen „clk“ ein Bitmuster vorgegeben werden. Für die Simulation sind „reset“ und „enable“ nicht relevant und sind auch noch nicht im Sourcecode eingebunden. Diese Variablen sind für die spätere Übernahme in Quartus II von Anfang an implementiert worden. Das Taktsignal muss aber vorgegeben werden. Ohne dieses Signal kann die Simulation kein Ergebnis liefern. Hierfür wird ein Rechtecksignal mit der Periodenzeit von 10µs generiert. Nun haben alle Eingänge Werte und die Simulation kann die entstehenden Ausgangsvariablen und internen Variablen berechnen.

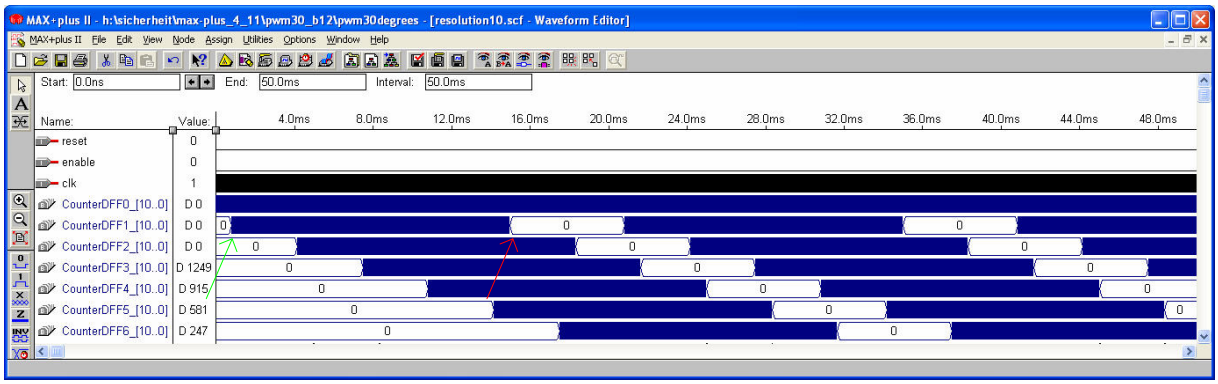


Abbildung 4.2: Übersicht der Counter zur Zündpulserzeugung mit 100KHz Taktung

Der Maincounter läuft jetzt als Referenz die gesamte Periode durch. Um aber die einzelnen Zündpulse zu erzeugen, sind weitere Counter nötig. Deswegen werden zu diesem Maincounter für jeden der sechs Zündpulse eigene Counter zeitversetzt gestartet.

In Abbildung 4.2 ist durch den grünen Pfeil der Start von Counter 1 hervorgehoben. Beim roten Pfeil ist der Maximalwert des Counters erreicht und er wird gelöscht. Der Zeitraum, in dem der Counter zählt, ist der zulässige Bereich, in dem sich der Phasenanschnittswinkel befinden darf. In der Simulation laufen die Counter 13,33ms. Damit sind die geforderten Phasenanschnittswinkel von 20° bis 120° möglich und es ist zusätzlich noch eine Toleranz für die Ausregelung von Unsymmetrien gegeben.

Die Counter 1 bis 6 laufen um 3,333ms zueinander verzögert los. Diese Zeit entspricht 60° und ist der Abstand zwischen zwei Zündpulsen. D urch diese Verschiebung kann der Phasenanschnittswinkel bei allen Pulsen gleich berücksichtigt werden. Der Phasenanschnittswinkel wird über den Eingang „Pulse[resolution..0]“ an den Block übergeben.

Zur Berechnung welcher Dezimalwert welchem Phasenanschnittswinkel entspricht, wird der „MaxCount“ Wert aus der Parameterliste von der Zündpulserzeugung verwendet. Dieser Wert gibt den Maximalwert des Maincounters an.

Die Umrechnungsformel lautet:

$$Phasenanschnittswinkel\_in\_Bit = 2000Bit \cdot \frac{Phasenanschnittswinkel\_in\_Grad}{360°} \quad (0.1)$$

Der Block gibt dann den entsprechenden Bitwert aus. Die Abbildung 4.3 enthält zusätzlich zu den Countern auch die Zündpulse. Die Zündpulse sind bei einem Phasenanschnittswinkel von 30° erzeugt worden. Zur Gegenüberstellung ist in Abbildung 4.4 die theoretische Lage der Zündzeitpunkte dargestellt.

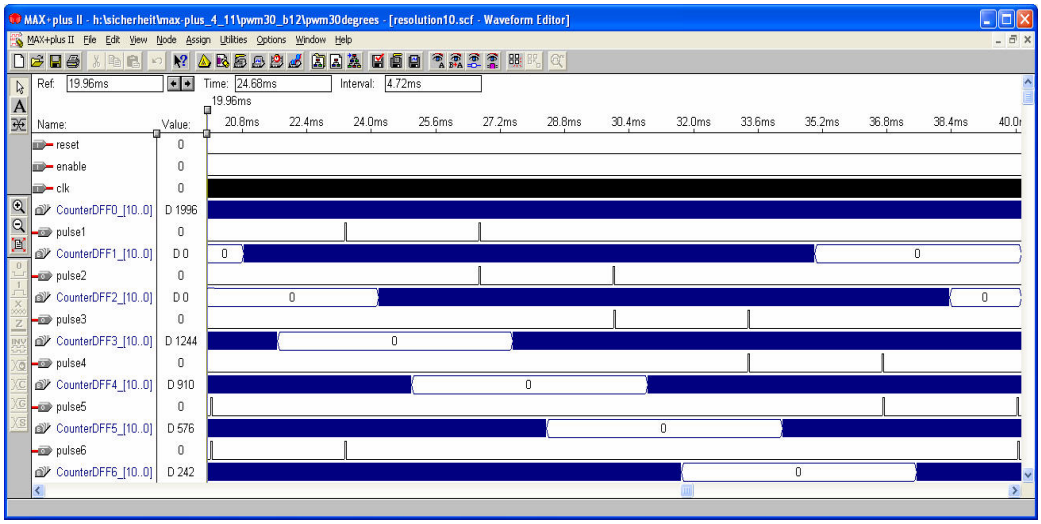


Abbildung 4.3: Übersicht der simulierter Zündpulse mit Alpha=30° mi t 100KHz Taktung

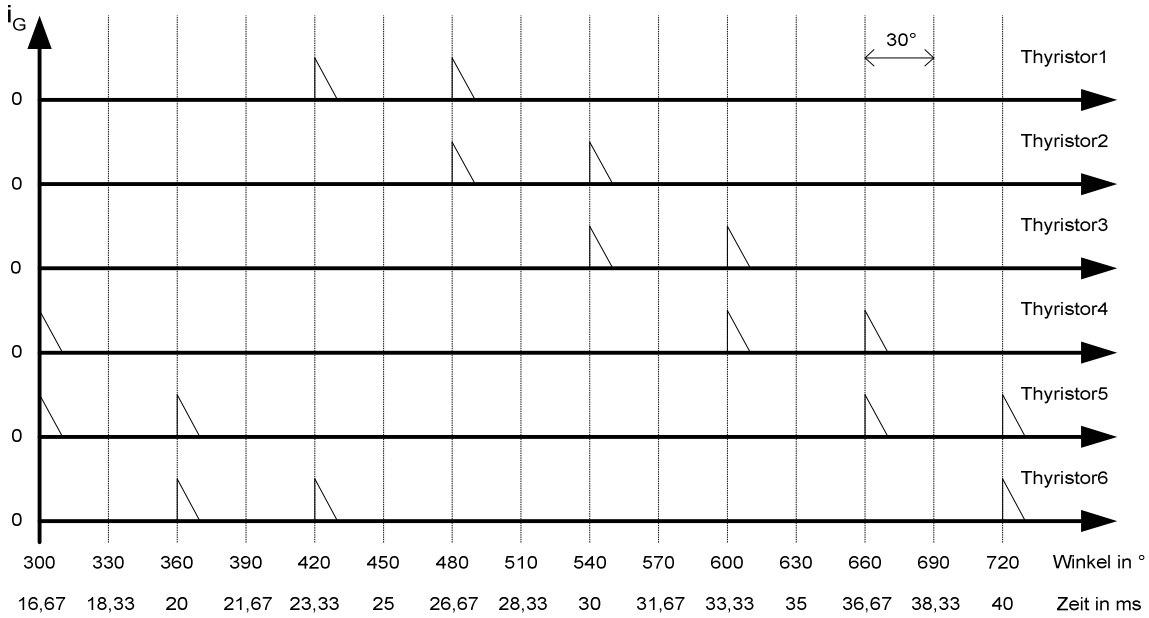


Abbildung 4.4: Übersicht der theoretischen Zündpulse mit Alpha=30°

Die Länge der Zündpulse ist in der Parameterliste auf 40µs eingestellt.
Die Überprüfung der Zündabstände ergab einen Abstand von 3,33ms. Die simulierten Werte stimmen mit den theoretischen Werten überein.

In der Simulation ist es nicht möglich analoge Signale darzustellen. In den folgenden Kapiteln wird deshalb auf die Simulation nicht mehr eingegangen, sondern nur noch mit Quartus II gearbeitet. Dies schließt reale Messungen ein, und es ist die Lage der Zündpulse zu den Außenleitern zu sehen.

5 PLL Umsetzung und Frequenzflexibilität

Bei diesem System mit getrennten Steuer- und Lastteil ist eine Synchronisation erforderlich. Die Thyristoren müssen je nach Phasenanschnittswinkel zu bestimmten Spannungen der Außenleiter zünden. Dafür muss der Nulldurchgang der Außenleiterspannung detektiert werden. Es gibt verschiedene Umsetzungen wie dies geschehen kann.

Für diese Arbeit wurde eine in der Software implementierte PLL gewählt. Eine PLL ist entsprechend [7] „ein Regelsystem, dessen Aufgabe darin besteht, einen Oszillator in Frequenz und Phase mit einem Eingangssignal zu synchronisieren. Im synchronisierten Zustand des PLL ist die Phasenverschiebung zwischen Eingangssignal und Oszillatorsignal null oder wenigstens ein Minimum; sobald aber zwischen beiden Signalen eine Phasenverschiebung auftritt, wird der Oszillator so lange nachgeregelt, bis die Phasenverschiebung wieder null oder minimal wird.“

Eine PLL ist im Vergleich zu anderen Synchronisationsmöglichkeiten besser für verrauschte Eingangssignale geeignet. Dies ist durch die Regelung begründet, die das Ausgangssignal der PLL nur langsam den Frequenzänderungen nachführt. Dadurch wird auf einen durch Rauschen verschobenen Nulldurchgang nur leicht reagiert und die Zündzeitpunkte werden nicht gleich zu stark verschoben.

5.1 Erzeugung des Synchronisationssignals

Zur Synchronisation wird ein Referenzsignal der Netzspannung benötigt. Durch die Eigenentwicklung der Regelungselektronik bei DESY wurde dafür schon eine Möglichkeit vorgesehen. In der Regelungselektronik wird mit einem beschalteten Optokoppler das Sinussignal der Netzspannung in ein Rechtecksignal umsetzt. Der Ausgang des Optokopplers schaltet zwischen 5V und GND im Netztakt hin und her. Dieses Signal dient als Synchronisationssignal für die PLL.

Von der Regelungselektronik wird mit einer Flachbandleitung das Signal auf ein 5V Eingangspin der Interfacekarte gegeben. Auf der Interfacekarte wird das Signal „Mains_Phase_Signal“ genannt.

5.2 Bausteine zur Frequenzsynchronisation

Zur Frequenzsynchronisation werden mehrere Blöcke verwendet. Die eigentliche PLL war schon implementiert und wurde nur übernommen. Die anderen beiden Blöcke verarbeiten das Ausgangssignal der PLL weiter.

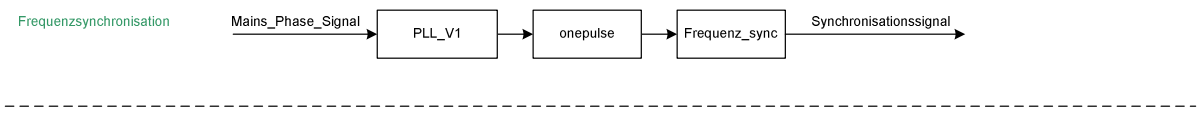


Abbildung 5.1: Übersicht der Blöcke zur Synchronisation

Baustein: „PLL_V1“

Aufgabe

Die PLL soll für eine phasen- und frequenzrichtige Synchronisation sorgen.

Ablauf

Anhand eines Synchronisationssignals wird von der PLL ein Rechtecksignal generiert. Die Frequenz des Rechtecksignals wird dem Synchronisationssignal angepasst. Die PLL wartet auf den phasenrichtigen Nulldurchgang des Synchronisationssignals und rastet (locked) dann ein. Der Nulldurchgang mit einer positiven Steigung wird detektiert und die PLL kann nun das Rechtecksignal an die Netzfrequenz anpassen. Das geschieht mit einem Regelkreis. Durch das langsame Ausregeln des Ausgangs hält die PLL die Welligkeit auf der Ausgangsspannung möglichst klein.

Baustein: „onepulse“

Dieser Block formt aus dem Rechtecksignal einen Puls von einem Programmtakt Länge. Diese Pulse sind 80ns lang und dienen zur Flankendetektierung.

Baustein: „frequenz_sync“

Die Netzfrequenz schwankt ständig leicht um die Nennfrequenz von 50Hz. Diese Schwankungen müssen in der Ausgabe der Zündpulse berücksichtigt werden. Dieser Baustein ermöglicht einen zuverlässigen Betrieb, auch bei Frequenzabweichungen von einigen Hertz, bezogen auf die Netzfrequenz.

Die Zündpulserzeugung benötigt zur Erzeugung der Zündpulse die Zeitwerte für ein Sechstel und ein Zwölftel der Periode. Das sind dann die Werte für 60° und 30° einer Periode. Diese beiden Werte verwendet die Zündpulserzeugung, um die einzelnen Zündpulse in der folgenden Periode an die Frequenz anzupassen.

Die Werteaufnahme für einen Sechstel und einen Zwölftel der Periodenzeit ist mit Countern realisiert, weil Quartus II keine einfache Division in einem Block zulässt. Somit stellte die Ermittlung mit Countern die sinnvollste Alternative dar.

Aufgabe

Der Block muss einer Abweichung der Netzfrequenz von 50Hz erkennen und messen wie groß sich diese Abweichung auswirkt. Wenn die Frequenz kleiner als 50Hz wird, und somit die Periodendauer größer als 20ms wird, reicht es nicht nur die gesamte Periode zu strecken. Es muss eine Normierung der Zündzeitpunkte auf die variable Periodendauer vorgenommen werden. Die einzelnen Zündzeitpunkte müssen ebenfalls verschoben werden. Wenn ein Phasenwinkel einen Zündpuls bei der Hälfte jeder Periode bewirken würde, müsste dies bei 50Hz nach 10ms jeder Periode sein. Wenn die Frequenz aber 49,5Hz ist, muss der Zündpuls bei 10,1ms erzeugt werden. Um dies zu gewährleisten, wird über Counter ermittelt wie lang eine Periode ist, und welche Werte für 30° und 60° verwendet werden müssen. Diese Werte werden an den Block „zuendpulserzeugung“ übertragen und dort werden die Zündzeitpunkte verschoben.

Ablauf

Der Maincounter wird mit 12,5 MHz Clock betrieben. Um einen Wert für 30° und 60° zu erhalten, müssen zwei Counter nur jeden sechsten bzw. zwölften Takt hochgezählt werden. Diese Werte werden dann in der nächsten Periode benutzt. Somit reagiert das Programm immer mit einer Periode Verzögerung auf Fehler.

6 Reale Zündpulserzeugung mit Quartus II

In diesem Kapitel wird dargestellt, wie die Weiterentwicklung von der Simulation bis zum lauffähigen Thyristorgerät erfolgt.

6.1 Erstellen der Blöcke in Quartus II

Die Übernahme der in MAX+plus II entwickelten Programmteile ist inhaltlich ohne Änderungen in Quartus II möglich. In Quartus II müssen nur neue Blöcke erstellt werden und der Sourcecode der Blöcke von MAX+plus II hineinkopiert werden. Nach dem Kompilieren funktionieren die Blöcke wie in MAX+plus II.

6.2 Bausteine zur Zündpulserzeugung

Der Block „zuendpulserzeugung“ erzeugt die Zündpulse. Für einen sicheren Betrieb werden zwei weitere Blöcke benötigt, deren Funktion und Aufgabe in der Folge erläutert werden.

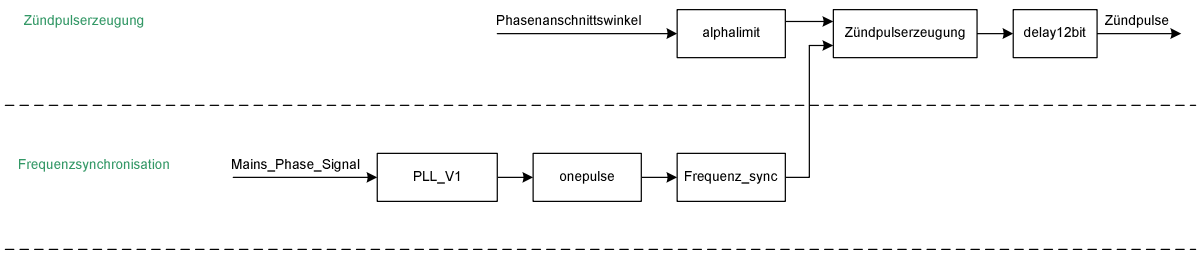


Abbildung 6.1: Übersicht der Blöcke in Quartus II zur Zündpulserzeugung

Baustein: „alphalimit“

Aufgabe

Die Zündpulserzeugung bekommt als Sollwert den Phasenanschnittswinkel. Mit diesem Baustein wird der Wertebereich des Phasenanschnittswinkels begrenzt. Der Wertebereich ist bei ohmscher Last auf 20° bis 120° begrenzt. Hierbei ist bei 120° die Ausgangsspannung gleich null und bei 20° ist die Brücke fast voll ausgesteuert. Die 20° Differenz zur Vollaussteuerung ist als mögliche Toleranz für die Regelung eingeplant. Bei Unsymmetrien wird von der Regelung ein zusätzlicher Winkelwert erzeugt. Für diesen zusätzlichen Winkel ist die Toleranz eingeplant, weil es sonst keine Möglichkeit gäbe, die durch die Unsymmetrie entstehende Regelabweichung auszugleichen.

Ablauf

Es wird der Eingangswert mit einem Maximal- und Minimalwert verglichen. Sollte der Eingangswert den Wertebereich verlassen, wird statt dem Eingangswert der größt- oder kleinstmögliche Wert ausgegeben.

Baustein: „zuendpulserzeugung“

Der Baustein erhält, zusätzlich zu den in Kapitel 4 gezeigten Eingängen, auch das Synchronisationssignal der PLL und die Werte für 30° und 60° vom Baustein „frequenz_sync“. Mit diesen Werten werden die Zündpulse jetzt auch bei Frequenzschwankungen zu den richtigen Zeitpunkten ausgegeben.

Baustein: „delay12bit“

Aufgabe

Um einen Thyristor zu zünden, muss ein bestimmter Zündstrom im Gate überschritten werden[8]. Der Zündpuls muss nicht besonders lang sein um einen Thyristor zu zünden. Um ein sicheres Zünden zu garantieren wird ein Doppelpuls verwendet. Als Doppelpuls werden zwei kurze Pulse bezeichnet, die nacheinander auf das Gate des Thyristors gegeben werden. Beide Pulse sind einzeln in der Lage den Thyristor zu zünden. Falls der erste Puls den Thyristor nicht zündet, kommt der zweite noch zur Sicherheit.

Bei den SKKT Thyristormodulen muss der Zündpuls >3µs lang sein und die Pulskarte benötigt eine Abschaltzeit von <20µs. Was zu einer Zeit für den Doppelpuls von ca. 26µs führt.

Ablauf

Der Baustein zur Zündpulserzeugung ist so geschrieben, dass durch Parameter die Zündpulslänge flexibel eingestellt werden kann. Aus diesem Grund muss auch der Block „delay12bit“ flexibel sein. Wenn ein Zündpuls erzeugt wird, wird die Länge über einen Counter gemessen und als Referenz für den Sicherheitspuls gespeichert. Ein zweiter Counter läuft nach Ende des Zündpulses solange, bis die in der Parameterliste eingestellte Wartezeit zwischen den Pulsen erreicht ist. Danach wird der zweite Zündpuls mit der identischen Länge des ersten ausgegeben. Durch die veränderbare Zündpulsdauer kann die Länge der Zündpulse eingestellt werden ohne das Programm zu verändern.

Es ist aber darauf zu achten, dass die Pause zwischen den Pulsen mindestens so lang ist, wie die Abklingzeit der Pulskarte. Sollte dies nicht der Fall sein, ist der Doppelpuls nur ein längerer Einzelpuls. Dies ist auch möglich, ist aber unsicherer als das Zünden mit einem Doppelpuls.

6.3 Inbetriebnahme der Thyristorbrücke

Bei der Inbetriebnahme der Thyristorbrücke müssen zwei wichtige Punkte beachtet werden:

- Die richtige Zuordnung der Zündpulse zu den Thyristoren muss stimmen.
- Die Zuordnung von den Außenleitern und den Thyristoren muss mit dem Zündmuster der Zündpulserzeugung übereinstimmen.

Mit Hilfe des Oszilloskopes kann, bevor die Netzspannung auf den Transformator gegeben wird, schon eine Kontrolle der erzeugten Zündpulse erfolgen.
Durch die Überprüfung des Zündpulsmusters kann auch die Funktion der Interfacekarte kontrolliert werden. Wenn das richtige Zündpulsmuster ausgegeben wird, können die Zündpulse an der Pulskarte angeschlossen werden. Die Zuordnung der Pulse zu den Außenleiterspannungen ist in Abbildung 6.2 gezeigt.

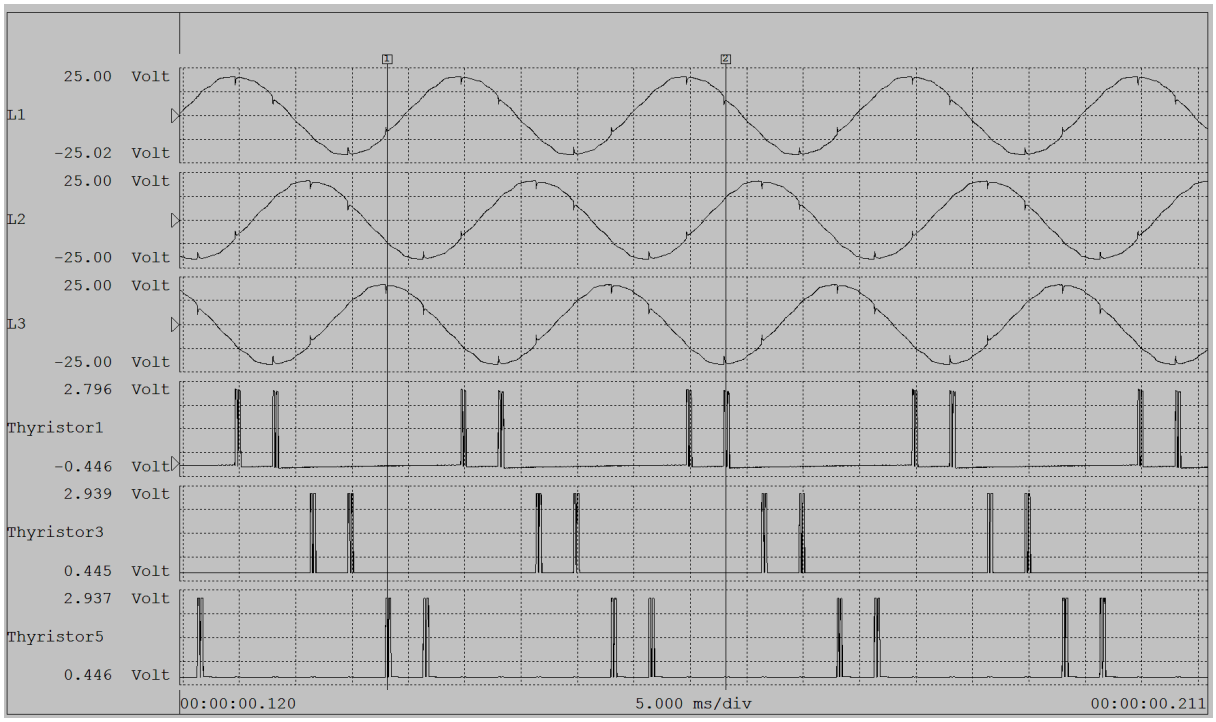


Abbildung 6.2: Zuordnung von Zündpulsen zu den Außenleitern

Die Abbildung 6.2 zeigt die drei Ausgangsspannungen des Transformators und die Zündpulse 1, 3 und 5.

Wie in Abbildung 4.4 (Seite 30) zu sehen ist, müssen immer zwei Thyristoren gleichzeitig gezündet werden. Thyristor 1 wird im 60° Abstand zweimal gezündet. Dies geschieht beim ersten Mal zeitgleich mit Thyristor 6 und beim zweiten Mal mit Thyristor 2.

Auf den Spannungen der Außenleitern sieht man an den Spannungsspitzen, wann die Thyristoren zünden. Auf allen Außenleiterspannungen sieht man vier Spannungsspitzen pro Periode. Die Thyristoren, die die positiven Halbschwingungen zünden, sind dargestellt und die Zündzeitpunkte der anderen drei Thyristoren sieht man an den Spannungsspitzen in den negativen Halbschwingungen.

Zur Überprüfung der Zündpulsabstände ist der Abstand von 30ms zwischen den Coursern zu entnehmen. Die beiden Courser sind genau auf den Beginn von zwei Zündpulsen gesetzt. Somit ergibt sich ein regelmäßiger Abstand von 3,333ms zwischen zwei Zündpulsen.

Durch die Rechnung lässt sich das belegen:

In den 30ms werden 9 Zündpulse erzeugt.

$$30ms / 9Zündpulse = 3,\overline{3}ms / Zündpuls$$

Weiterhin ist:

$$3,\overline{3}ms \cdot \frac{360°}{20ms} = 60°$$

Damit ist ein Abstand zwischen den Pulsen von 60° eingehalten.

7 System mit Filter

Eine Sechspulsbrückenschaltung hat einen 300Hz Ripple auf der Ausgangsspannung. Um diesen Ripple zu verringern, muss im Betrieb ein Tiefpassfilter zur Glättung verwendet werden[9].

Dieser Tiefpass hat zwei Effekte:

1. Die pulsierende Wechselspannung wird geglättet.
2. Bei geringer Aussteuerung der Brücke tritt kein Lückbetrieb auf.

Bei der geringen Leistung der Testanlage wurde ein RC-Tiefpass verwendet. Bei höheren Leistungen würde über dem Widerstand eine zu große Verlustleistung auftreten, deswegen wird dann ein LC-Tiefpass verwendet.

Die Abbildung 7.1 zeigt den Unterschied zwischen der ungefilterten (links) Ausgangsspannung und der gefilterten (rechts) im lückenden Betrieb. Die Ausgangsspannung (blau) zeigt die sechs Pulse der sechs Thyristoren. Dazu sind noch die beiden Zündpulse von Thyristor 1 abgebildet.

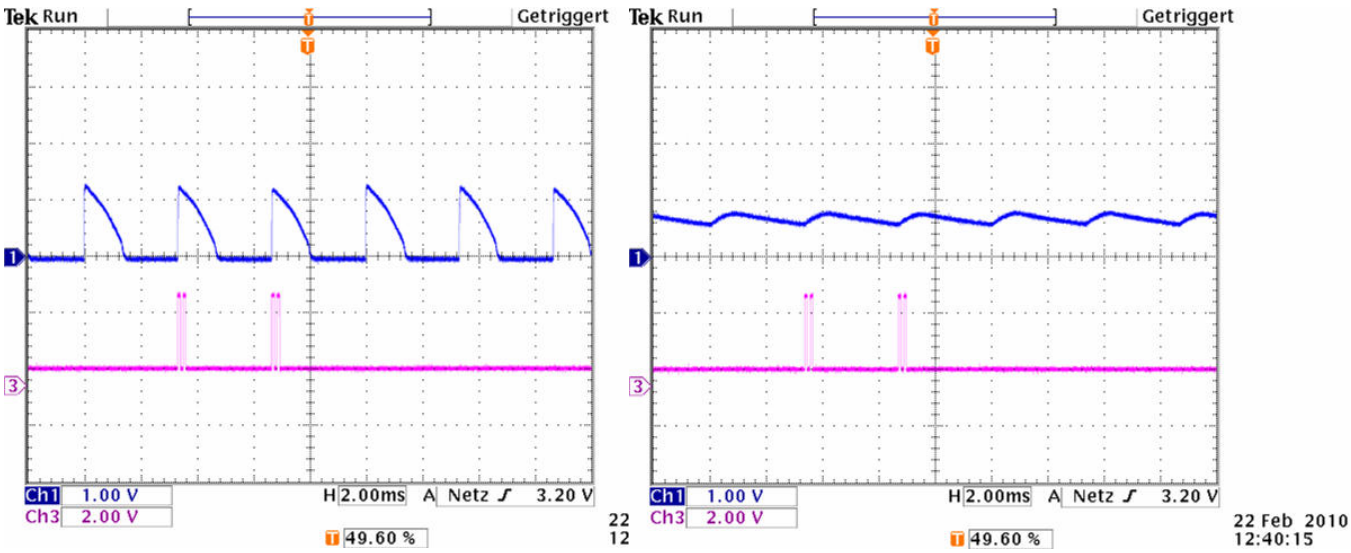


Abbildung 7.1: Vergleich gefilterte und ungefilterte Ausgangsspannung

8 EMV- Probleme

Bei der Erprobung des Steuersatzes traten verschiedene EMV Effekte auf. Die Ausgangsspannung der Thyristorbrücke war mit einem 50KHz Rauschen überlagert. Im hochohmigen Spannungsteiler, der die Ausgangsspannung auf 10V teilt, wirkte sich das hochfrequente Rauschen für eine genaue Messwerterfassung zu stark aus. Das Rauschen lies bei einer nur wenig ausgesteuerten Brücke keine Istwertaufnahme zu.

Durch gezieltes Verändern des Versuchaufbaus stellte sich die Pulskarte als Störquelle für das 50KHz Rauschen heraus. Ein einzelnes Gehäuse, in dem die Pulskarte eingebaut wurde, und eine Überarbeitung der Erdung verkleinerte die Amplitude, des 50KHz Rauschens sehr. Zusätzlich wurden geschirmte Pulskabel verwendet[10].

Noch weiter verkleinern lies sich das Rauschen durch den Einbau von Tiefpassfiltern. Ein analoger Tiefpass im Spannungsteiler und einem digitalen auf der Interfacekarte verringerten die Amplitude des Rauschens auf ein Minimum. Der digitale Filter befindet sich direkt nach dem Eingang der Ausgangsspannung.

Die Amplitude des hochfrequenten Rauschens auf der Netzspannung ist für einen Betrieb bei einer geringen Aussteuerung der Thyristorbrücke noch immer zu groß. Bei einer Aussteuerung von unter ¼ der Vollaussteuerung der Thyristorbrücke ist noch keine verlässliche Istwertaufnahme möglich. Die Amplitude des Rauschens ist im Verhältnis zur Ausgangsspannung zu groß und beeinflusst deshalb den Wert zu stark. Die Messwerte variieren, für die Regelung zum Ausgleich von Unsymmetrien, zu sehr.

Das Rauschen wirkt sich beim Teststand nur wegen der geringen Spannungshöhe so stark aus. Die Thyristorgeräte, die in den Beschleunigern verwendet werden, arbeiten mit einer Spannung >100V. Diese Spannung liegt um den Faktor 5 höher im Vergleich zu der Spannung der Testanlage. Daher wird sich das Rauschen dort auch nicht im diesem Maße auswirken.

Eine Kontrolle an einer anderen Pulskarte vom gleichen Typ ergab die gleichen EMV-Effekte. So konnte eine Fehlfunktion eines Bauteils ausgeschlossen werden.

Die Abbildung 8.1 zeigt die starke Verrauschung der Ausgangsspannung vor den Maßnahmen zur Verkleinerung der EMV-Effekte. Die Abbildungen 8.1 und 8.2 zeigen nur den Wechselspannungsanteil auf der Gleichspannung, weil so das Rauschen detaillierter angezeigt werden kann.

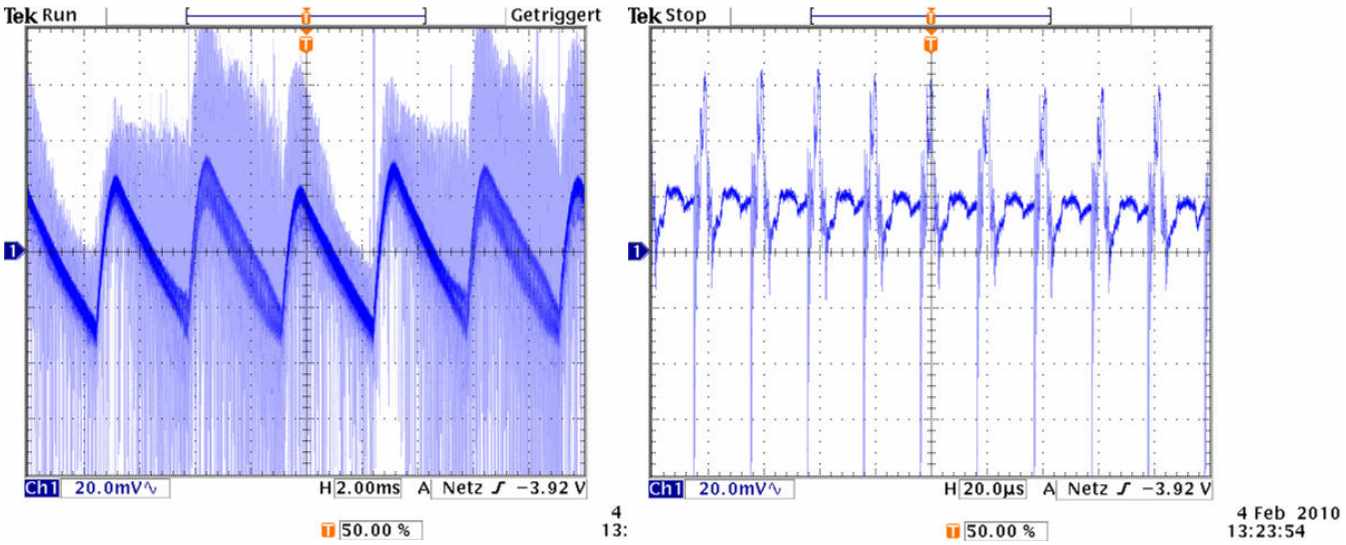


Abbildung 8.1 Auswirkung der EMV-Effekte auf die Ausgangsspannung

Abbildung 8.2 zeigt die durch gezielte Gegenmaßnahmen sehr viel sauberere Ausgangsspannung.

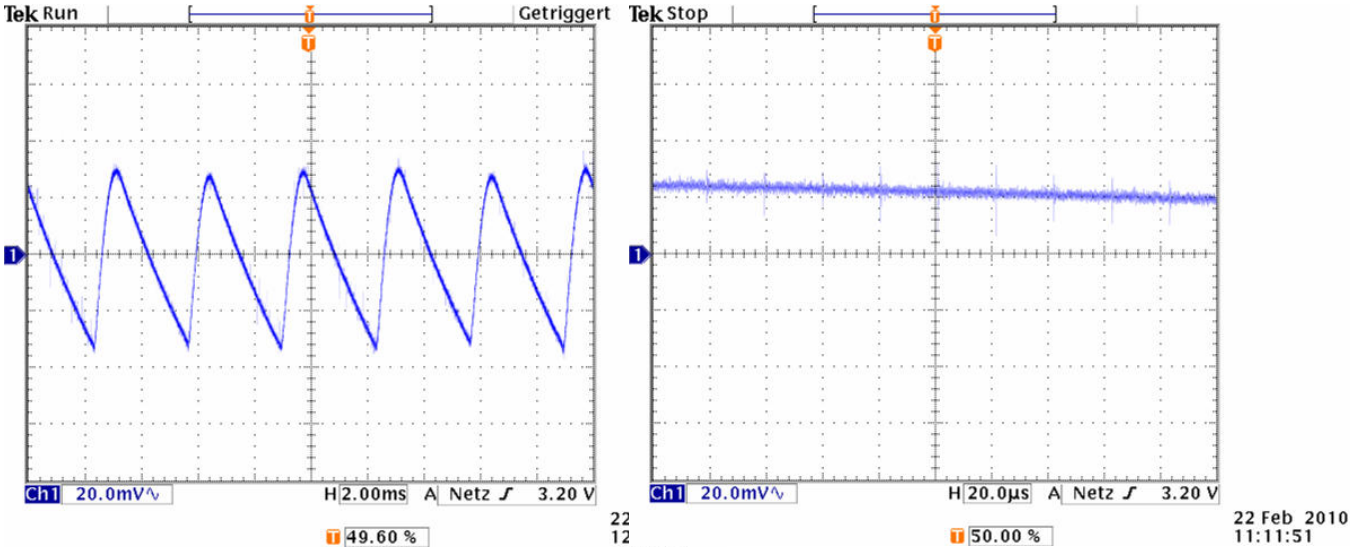


Abbildung 8.2: Die Ausgangsspannung nach Eindämmung der EMV-Effekte

9 Regelung

9.1 Anforderung an die Regelung

Die Regelung soll eine Verkleinerung der Ripple durch Ausgleich der Unsymmetrien durchführen. Es kann durch unterschiedliche Einflüsse zu Unsymmetrien kommen:

- netzseitige Spannungsunterschiede in den 3 Außenleitern
- Unterschiede im Zündverhalten der Thyristoren
- durch Transformator-Unsymmetrien

Eine detaillierte Erklärung der drei Probleme ist in der Studienarbeit [11] nachzulesen.

Als Beispiel ist in Abbildung 9.1 die gefilterte Ausgangsspannung zu sehen. Die Messung zeigt nur den Wechselspannungsanteil auf der Gleichspannung, um eine detaillierte Darstellung der Unsymmetrien zu bekommen.

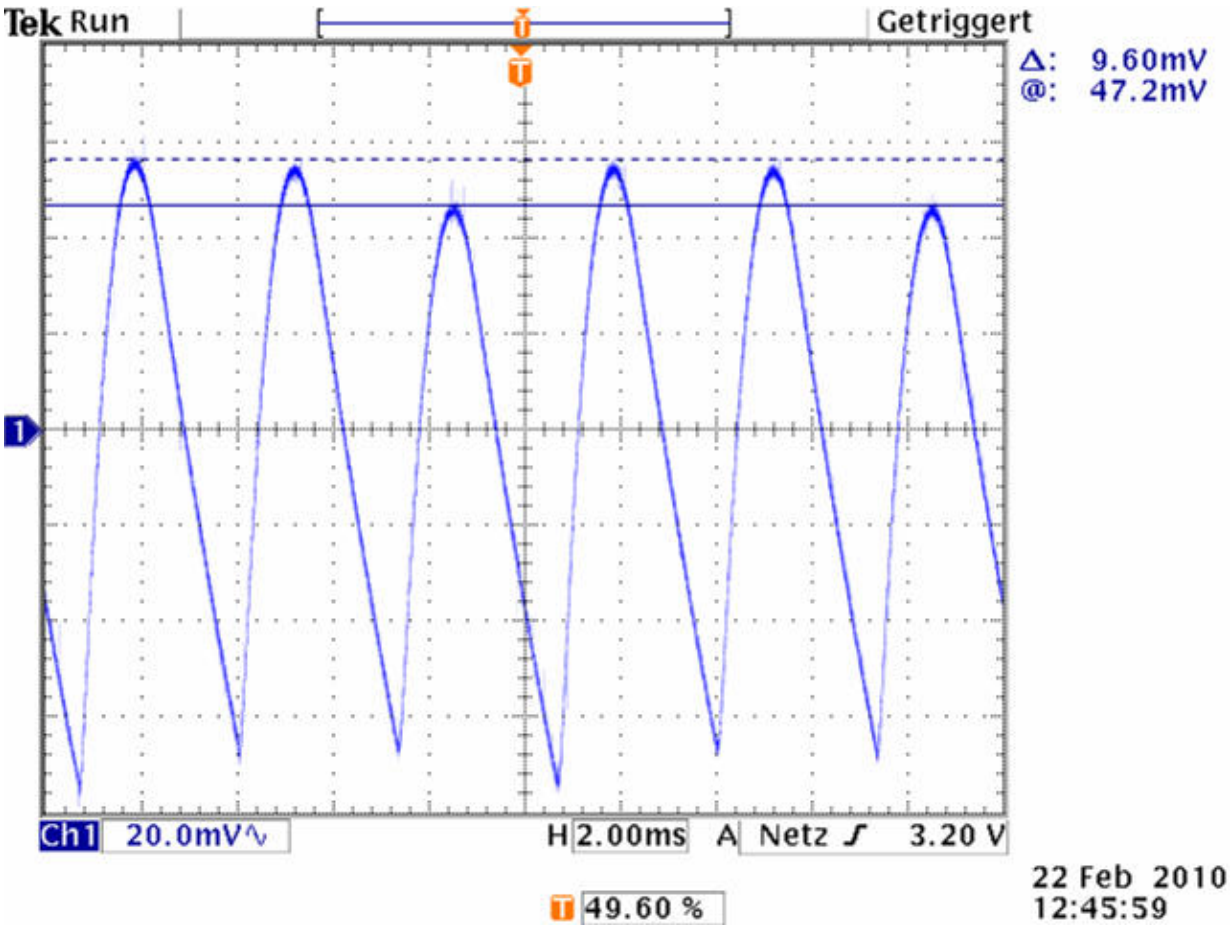


Abbildung 9.1: Wechselspannungsanteil der nicht geregelten Ausgangsspannung

Es ist deutlich die Abweichung in der Amplitude zwischen den einzelnen Pulsen zu erkennen. Diese Ungenauigkeiten sind zu jeder Zeit vorhanden und können sich noch wesentlich stärker auswirken.

Die Spannungs-Zeit-Fläche der sechs Pulse ist ohne Unsymmetrien gleich. Bei Abbildung 9.1 unterscheiden sich die Spannungs-Zeit-Flächen. Die Regelung soll dies anhand des Scheitelpunktes der sechs Pulse erkennen.

Die zuvor aufgezeigten Auswirkungen von Unsymmetrien und die hohen Genauigkeitsansprüche von DESY stellen folgende Ansprüche an die Regelung:

1. Es muss im laufenden Betrieb der Geräte ständig eine Überwachung der einzelnen Pulshöhen erfolgen.
2. Wenn eine Abweichung festgestellt wird, muss sie mit geringen Überschwingern ausgeregelt werden.
3. Das Ausregeln muss bei Sollwertsprüngen schnellst möglich durchgeführt werden.
4. Die Regelung muss die gesamte Bandbreite der möglichen Sollwerte abdecken.
5. Bei zu starkem Absinken einer Phase muss dies erkannt und mit einer Fehlermeldung angezeigt werden.

Zur Einstellung des Reglers können die Regelparameter auf der Internetseite variiert werden.

9.2 Technische Umsetzung der Regelung

Dieses Kapitel soll einen Überblick über das Prinzip der verwendeten Regelung geben.

Die unterschiedlich hohen Pulse werden durch Unsymmetrien hervorgerufen. Um diese Abweichungen auszugleichen, müssen die Zündzeitpunkte der Thyristoren verschoben werden. Zur Feststellung wie stark jeder Zündzeitpunkt verschoben werden muss, werden die sechs Spannungshöhen der Pulse aufgenommen. Die Regelung benötigt zur Verschiebung der Zündpulse eine Regeldifferenz. Die Regeldifferenz wird durch Subtraktion von zwei Spannungen gebildet. Als Sollwert für die Pulshöhe der Pulse wurde bei der ersten Umsetzung die Spannungshöhe von Puls 1 verwendet. Dies bedeutet, der Zündzeitpunkt von Thyristor 1 wird nicht verändert. Die anderen fünf Zündzeitpunkte werden von der Regelung solange verschoben bis die maximale Pulshöhe gleich der von Puls 1 ist. Zur Regelung werden also alle sechs maximalen Pulshöhen benötigt, um einen Ist- und einen Sollwert zur Verfügung zu haben.

Nach der Aufnahme der sechs Werte kann für jeden Puls eine Regelabweichung errechnet werden. Der Zündzeitpunkt für Thyristor 1 wird nicht verändert, deswegen wird die Regelabweichung für Puls 1 auf null gesetzt.
Die Regelabweichung für Puls 2 errechnet sich wie folgt:

Regelabweichung 2 = Sollwert(Puls 1) – Istwert(Puls 2)

Die fünf Regelabweichung wird auf einen PI-Regler gegeben und der Ausgangswert des Reglers wird als zusätzlicher Winkel (Korrekturwinkel) bei der Zündpulserzeugung berücksichtigt. Wenn sich der Korrekturwinkel durch die Regelung langsam verändert, wird die Regeldifferenz zu null und ein stationärer Zustand wird erreicht. Dieser Vorgang läuft separat, aber zeitgleich bei den fünf Pulsen ab.

Dieses Vorgehen führt dazu, dass die Spannungsunsymmetrien ausgeglichen werden, aber nicht auf die Spannungshöhe geachtet wird. Die durch die Ausregelung abgesunkene oder erhöhte Spannung wird dann aber durch die übergeordnete Spannungsregelung angeglichen. Nach der Ausregelung der Unsymmetrien wird mit der aktuellen Netzspannung die minimale Welligkeit der Ausgangsspannung erreicht.

9.3 Problem beim Umsetzen der Regelung

Die fünf Subtraktionen zur Berechnung der Regeldifferenzen können einen positiven oder negativen Wert ergeben. Ein negatives Ergebnis würde hier aber als positiver Bitwert berücksichtigt werden. Die Regelung in der Software arbeitet mit 2^{18}Bit. Dies bedeutet der maximale Bitwert ist 262143. Um jetzt die großen Bitwerte als negative Zahlen zu berücksichtigen wird der Arbeitspunkt verschoben.

Das bedeutet, dass alle Bitwerte <=131071 als positiver Wert und alle >131071 als negativer Wert berücksichtigt werden.

Ein kurzes Beispiel soll dies veranschaulichen:

Bitbreite: 2^{11} = 2047 Bit

Pulshöhe Puls 1 = 500 Bit
Pulshöhe Puls 2 = 550 Bit

Regeldifferenz = 500Bit – 550Bit = 1997Bit

Dieser Bitwert von 1997Bit wird von der Regelung als -50Bit berücksichtigt.

9.4 Bausteine zur Regelung

Die Abbildung zeigt alle Blöcke, die zum Betrieb einer Thyristorbrücke benötigt werden. Die vier Blöcke der Regelung werden im Anschluss erklärt.

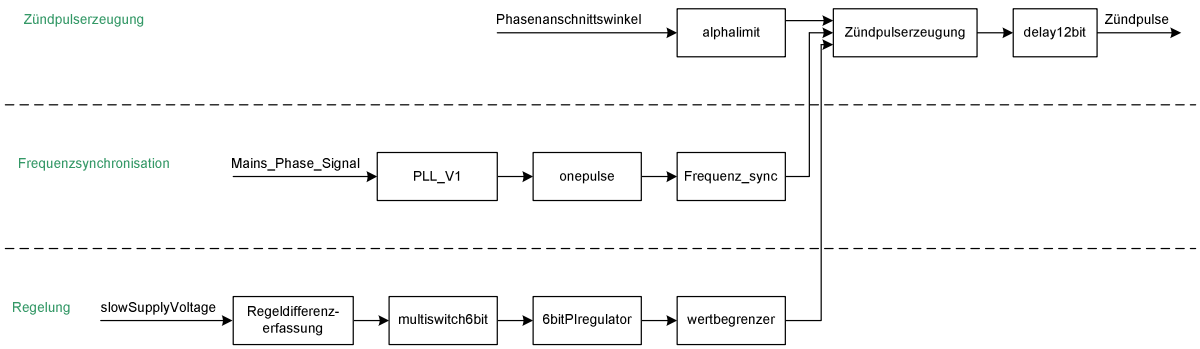


Abbildung 9.2: Gesamtübersicht der Programmbausteine

Baustein: „multiswitch6bit“

Aufgabe

Der multiswitch6bit schaltet die Regelung ein und aus. Durch einen externen Schalter ist es möglich den Reglereingang auf Null zu schalten. Das bedeutet, dass eine mögliche Regeldifferenz für Netzwerke vom Reglereingang weggeschaltet wird und stattdessen der Eingang auf GND gelegt wird.

Ablauf

Der Block hat zwei 18Bit breite Eingänge. Der erste Eingang bekommt die Regeldifferenz und der andere Eingang ist mit GND verbunden. Durch das Schaltsignal des externen Schalters wird nun entweder die Regeldifferenz oder eine Null an die Regelung weitergegeben.

Dadurch ist es in der Testphase möglich, erst den Fehler zu erzeugen und dann die Regelung einzuschalten. Beim Ausschalten der Regelung behalten die Integratoren ihre Werte bei.

Baustein: „Regeldifferenzerfassung“

Aufgabe

Zur Ausregelung des Ripples werden die Maximalwerte der sechs Pulshöhen aufgenommen, dann kann nach der Speicherung der Werte durch Subtraktion die Regeldifferenz ermittelt werden.

Ablauf

Die Ermittlung der Pulshöhen kann auf unterschiedliche Weisen erfolgen. Es wurden zwei Möglichkeiten getestet.

Ermittlung des Maximalwertes mit drei Werten

Das Programm startet nach dem Zünden jedes Thyristors einen Counter. Durch diesen Counter werden im Abstand von 0,1ms Werte der Ausgangsspannung aufgenommen. Immer drei Werte werden gespeichert. Wenn von diesen drei Werten der mittlere Wert am Größten ist, ist annähernd die höchste Spannung detektiert.
Dieser Wert wird dann als maximale Spannungshöhe zur Regeldifferenzberechnung verwendet. Eine Verringerung der Abtastungszeit würde eine höhere Genauigkeit ergeben, aber dadurch kann es aufgrund des Rauschens zu Fehlern kommen. Es wird z.B. ein falscher Wert oder gar keiner aufgenommen.
Ein falscher Wert kann durch die Aufnahme des mittleren Wertes auf einer Amplitudenspitze des Rauschens erfolgen. Dann wäre der mittlere Wert größer als der folgende Wert, obwohl die Ausgangsspannung noch ansteigt.

Ermittlung der Pulshöhe mit neun Werten

Die Umsetzung mit neun Werten beruht auf dem gleichen Prinzip. Hier werden wiederum drei Werte im Abstand von 0,1ms aufgenommen. Außerdem wird noch je ein Wert 0,4µs vor und nach den drei Werten aufgenommen. Jetzt kann auf die gleiche Weise erkannt werden, ob der höchste Wert erreicht ist. Durch die drei Werte, die auf der Spitze aufgenommen wurden, kann durch die Auswahl des mittleren Wertes eine Lage auf einer Amplitudenspitze des Rauschens ausgeschlossen werden.

Baustein: „6bitPIregulator“

Es wurde ein bestehender Programmteil genommen und angepasst. Die Grundlage war ein PI-Regler mit einem Eingang. Dieser konnte nur eine Regeldifferenz ausregeln. Dieser Block hätte sechsmal in die Software eingebunden werden müssen, um die sechs Regeldifferenzen auszuregeln. Dadurch hätte sich die Übersichtlichkeit verschlechtert. Aus diesem Grund wurde der Block so erweitert, dass die sechs Regeldifferenzen von diesem Block verarbeitet werden können. Dies macht das Programm übersichtlicher.

Aufgabe

Dieser Baustein ist ein digitaler PI-Regler. Als Eingangswert bekommt er die sechs Regeldifferenzen. Der Reglerausgang wird zu dem Phasenanschnittswinkel addiert und so werden die fünf Zündzeitpunkte unabhängig voneinander verschoben.

Ablauf

Der Aufbau entspricht einem analogen Regler. Die Regeldifferenz wird auf den P-Teil und I-Teil gegeben. Die Teile arbeiten parallel und die Regelwerte werden addiert. Der Regelwert dient zur zeitlichen Verschiebung der Zündzeitpunkte.

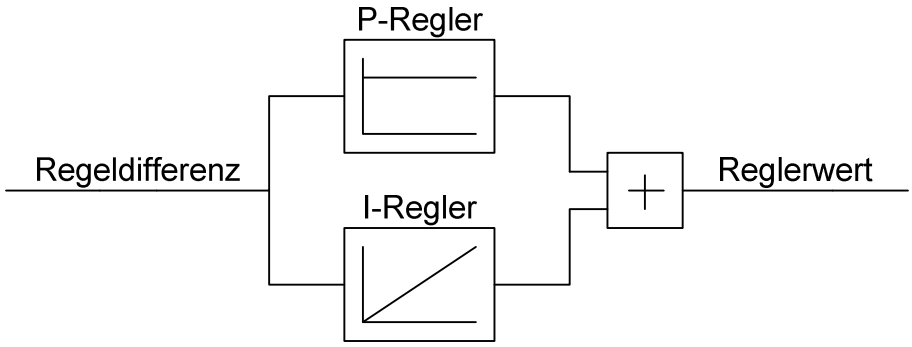


Abbildung 9.3: Aufbau des PI-Reglers

Baustein: „wertbegrenzer“

Aufgabe

Bei einer Software kann es immer dazu kommen, dass es durch Programmänderungen zu unerwarteten Effekten kommt. Deswegen sollte es an verschiedenen Stellen Sicherungen geben, die das Verlassen eines sicheren oder vertretbaren Wertebereichs abfangen.

Ein Problem stellt der I-Anteil des Reglers dar. Der Wert des Integrators kann schnell sehr hohe Werte erreichen. Dies kann beim Verschieben der Zündzeitpunkte schnell zu einer starken Welligkeit der Ausgangsspannung führen. Gerade bei hohen Leistungen kann das großen Schaden anrichten. Zur Begrenzung des Reglerausganges wurde deshalb der Block „wertbegrenzer“ entwickelt.

Ablauf

Der „wertbegrenzer“ hat einen einstellbaren Parameter, der angibt bis zu welchem Wert der Reglerausgang zulässig ist. Sollte dieser Wert überschritten werden, gibt der Baustein stattdessen den Grenzwert aus.

Durch ein Overflow Signal kann dies auf der Internetseite als Fehlermeldung angezeigt werden.

9.5 Geregeltes System

Zur Verdeutlichung, wie sich der Korrekturwinkel auswirkt, ist die Messung wie bei Abbildung 9.2 mit eingeschalteter Regelung wiederholt worden. Es ist im Vergleich zum nicht geregelten System eine sehr viel kleinere Abweichung der Pulshöhen zu sehen. Die Abweichung bei den Minimalpunkten ist ein wenig größer, aber im Verhältnis zu vorher ist der gesamte Ripple kleiner geworden.

Dies ist ein systematischer Fehler. Eine Überarbeitung des Algorithmus sollte die Istwerterfassung so modifizieren, dass eine Regelung auf eine möglichst kleine Abweichung bei den Maximal- und Minimalwerten erfolgt. Dadurch wird das Ripple minimiert.

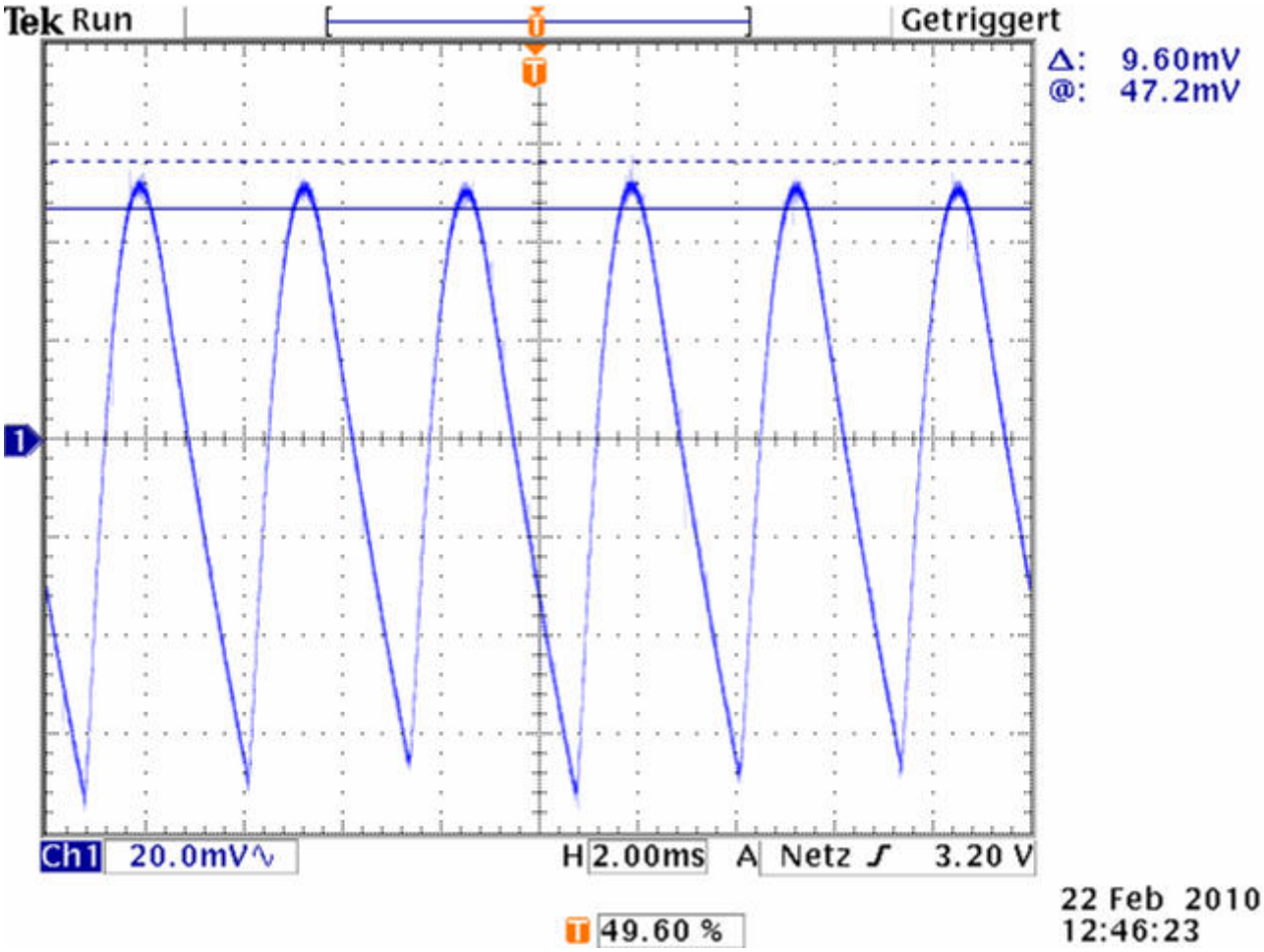


Abbildung 9.4: Wechselspannungsanteil der ausgeregelten Ausgangsspannung

9.6 Test anhand von Fehlerszenarien

In diesem Unterkapitel wird auf das Ergebnis der Simulation im Rahmen der Studienarbeit eingegangen. Zudem soll ein Vergleich zwischen Simulationsergebnis und realem Ergebnis erfolgen.

9.6.1 Ergebnis der Simulation

In der Simulation wurden zwei Arten von Tests durchgeführt. Dies waren Sollwertsprünge und Spannungsfälle in den Außenleitern. Die Reaktionen der Regelung in der Simulation sind in Abbildung 9.5 und 9.6 dargestellt.

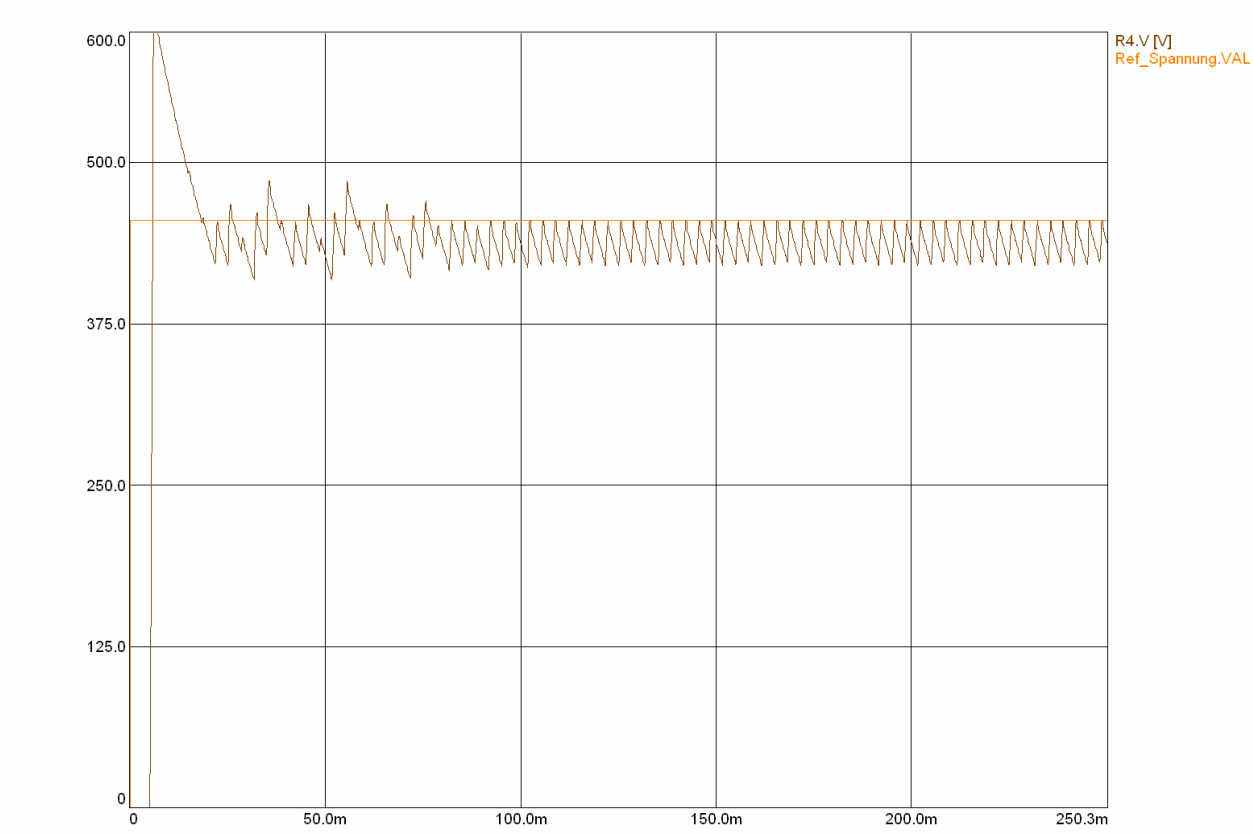


Abbildung 9.5: Regelung auf Referenzspannung in Simplorer

Die Abbildung 9.5 zeigt wie in Simplorer alle sechs Pulse auf eine Referenzspannung geregelt werden. Zu Simulationsbeginn ist durch Veränderung der Spannungen der drei Phasen eine Netzunsymmetrie vorhanden. Außerdem wurde mit einem zusätzlichen Winkel jeder der sechs Zündzeitpunkte leicht verschoben. Dadurch sind alle sechs Pulse unterschiedlich hoch. Nach 75ms wurde die Regelung aktiviert. Nach zwei Perioden ist der Ripple fast vollständig verschwunden.

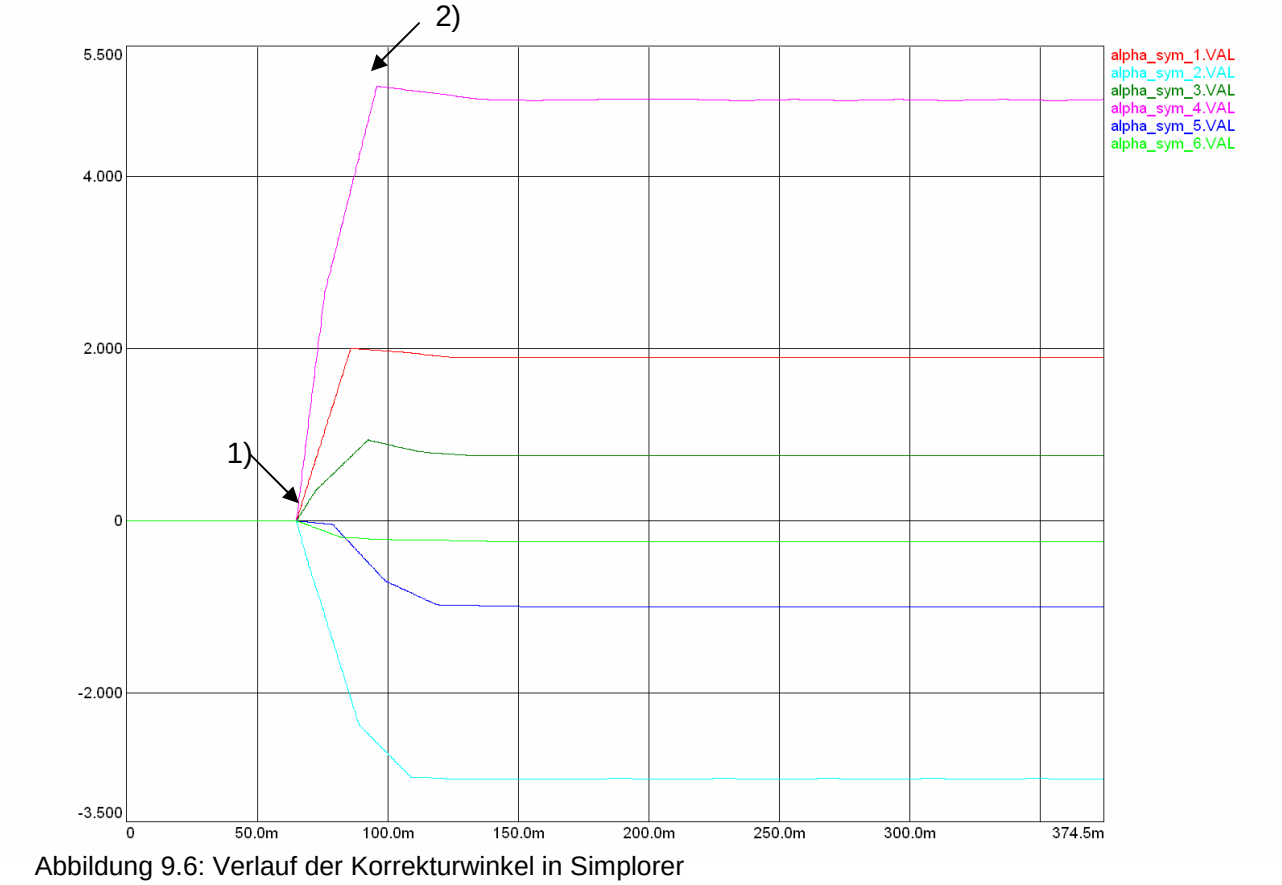


Abbildung 9.6: Verlauf der Korrekturwinkel in Simplorer

An den Korrekturwinkeln der sechs Zündzeitpunkte ist noch besser der Einschaltmoment der Regelung unter 1) zu sehen und wann die Korrekturwinkel bei 2) einen stationären Wert erreichen.

Dem gegenüber soll jetzt die reale Umsetzung gestellt werden. In der Simulation ergaben sich die in den vorangegangenen Kapiteln erklärten Probleme nicht. Bei der realen Umsetzung ist das Ergebnis deswegen nicht ganz so genau wie die der Simulation.

9.6.2 Ergebnis der realen Regelung

Um gezielt Netzunsymmetrien zu erzeugen, um die Regelung zu untersuchen, wurde ein Vorwiderstand in die Brückenschaltung eingebaut. Dieser Vorwiderstand ist zwischen Transformator und Thyristorbrücke im zweiten Außenleiter eingesetzt. Der Widerstand lässt sich mit Hilfe eines Schalters überbrücken, damit ein Spannungssprung simuliert werden kann. Der Widerstand hat 2Ω und es fallen ungefähr 10% der Spannung bei Vollaussteuerung der Brücke darüber ab.

Bei der Abbildung 9.7 wurde mehrmals der Widerstand in den Stromkreis zu- und abgeschaltet. Das obere Signal zeigt diesen Spannungsfall (Spgfall).

Da auf die Spannungshöhe von Puls 1 geregelt wird, fällt die Ausgangsspannung (AusgSpg) durch den Spannungsfall um 2,75V ab. Dies resultiert daraus, dass bei Puls 1 der Stromfluss von Außenleiter 1 zu Außenleiter 2 geht. Wenn nun Außenleiter 2 ein niedrigeres Potenzial hat, ist auch die Höhe des Pulses kleiner.
Da nun alle anderen Pulse auf diesen Puls geregelt werden, muss die Spannung insgesamt kleiner werden.

Die Ausgangsspannung würde durch einen Spannungsfall in Außenleiter 3 nicht abnehmen. Bei einer niedrigeren Spannung in Außenleiter 3 würde die erste Pulshöhe nicht betroffen sein und die fünf anderen Pulse würden auf die höhere Spannung geregelt werden.

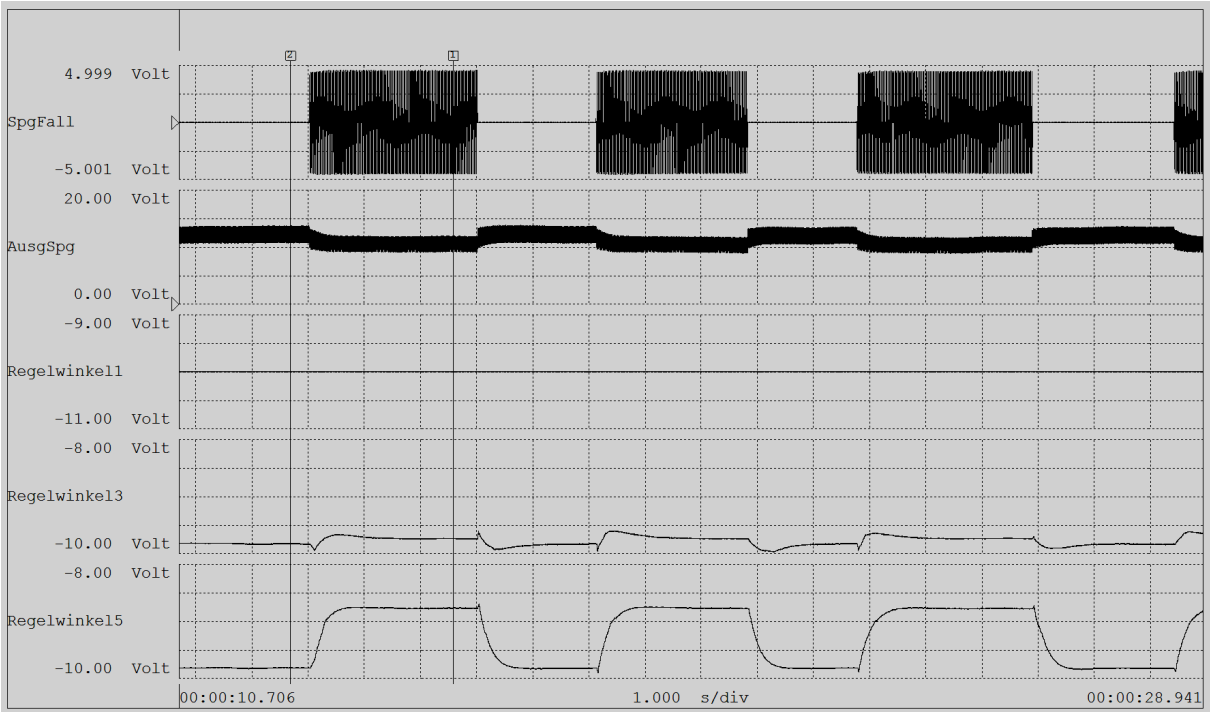


Abbildung 9.7: Ausregelung eines Spannungsfalls in einem Außenleiter

Die in Abbildung 9.7 gezeigte Entwicklung der Regelwinkel (Korrekturwinkel) ist ähnlich wie die in der Simulation. Der Unterschied ist der Wertebereich. Die Regelwinkel haben einen Offset von 10V. Dies ist auf die Verschiebung des Arbeitspunktes des Reglers zurückzuführen.

Der Fehler wird erzeugt und die Regelwinkel verschieben die Zündzeitpunkte soweit, dass die Pulse gleich hoch sind. Der Regler ist in dieser Messung langsam eingestellt gewesen, damit die Entwicklung deutlich zu sehen ist.

In den nächsten drei Abbildungen wird durch eine höhere Auflösung detailliert gezeigt wie sich die Regelung auswirkt. Abbildung 9.8 zeigt die Gesamtübersicht des simulierten Fehlers.

Beim linken Courser wird der Widerstand zugeschaltet. Die Regelung ist erst einmal ausgeschaltet. Dass zeigt das low Signal bei „RegelungEin“. Die Ausgangsspannung bekommt durch den Spannungsfall eine sehr große Unsymmetrie, die in der Vergrößerung in Abbildung 9.9 gezeigt wird.

Beim rechten Courser in Abbildung 9.8 wird jetzt die Regelung eingeschaltet. Der PI-Regler bekommt die Regelabweichung und beginnt diese auszugleichen. Nachdem ein stationärer Zustand erreicht wurde, wird der Widerstand überbrückt und die Regelung verkleinert die Regelwinkel. Die Regelwerte müssen jetzt nur die wirkliche Netzunsymmetrie ausgleichen.

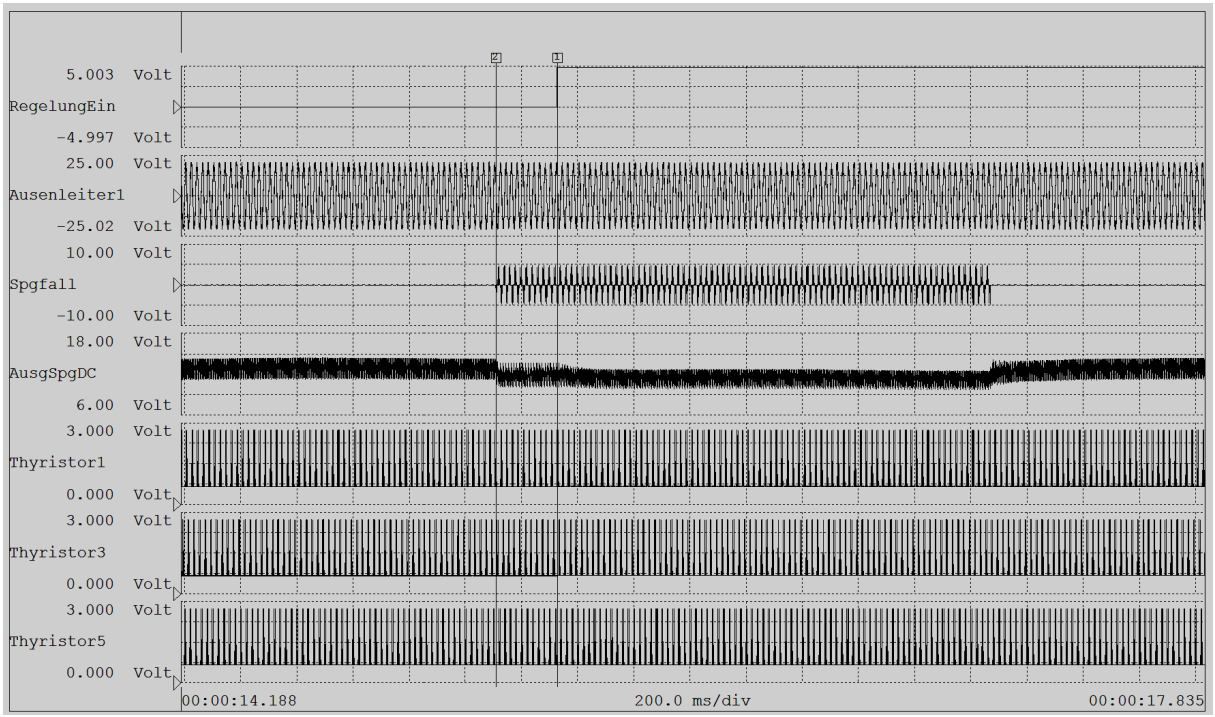


Abbildung 9.8: Übersicht zur Spannungsfallaufschaltung

In Abbildung 9.9 ist vergrößert dargestellt wie sich der Ripple auf der Spannung verkleinert.

Beim linken Courser wird der Widerstand zugeschaltet. Dadurch wird die Ausgangsspannung unsymmetrisch. Die Regelung wird beim rechten Courser eingeschaltet und das Ripple wird von Periode zu Periode kleiner. Ungefähr nach vier Perioden ist der nicht unerhebliche Spannungsunterschied ausgeglichen. Die einzelnen Zündpulse werden bis zu 4,5°verschoben, um diese A bweichung auszugleichen.

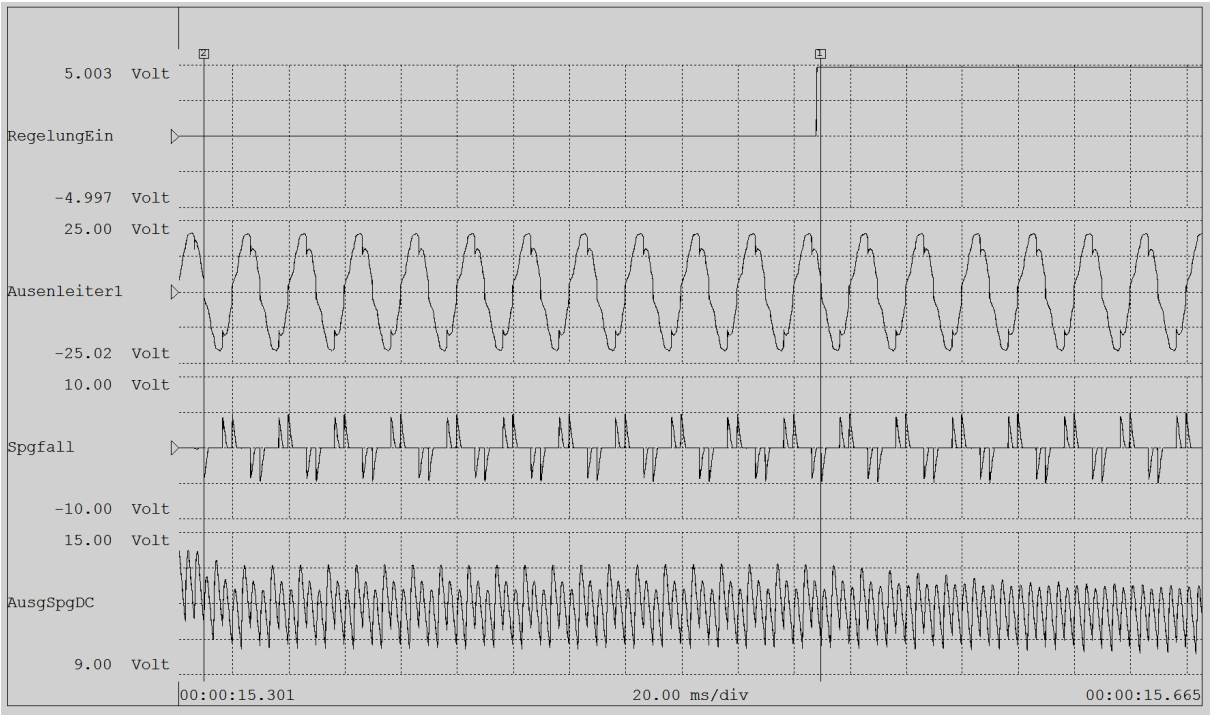


Abbildung 9.9: Einschalten des Spannungsfalls

Bei einer optimal eingestellten Regelung würde die Regeldifferenz in zwei Perioden vollständig ausgeregelt sein.

In Abbildung 9.10 ist das Kurzschließen des Widerstandes zu sehen. Der Spannungsfall verschwindet und die Regelung verkleinert die Regelwinkel bis wieder ein symmetrischer Spannungsverlauf vorliegt.

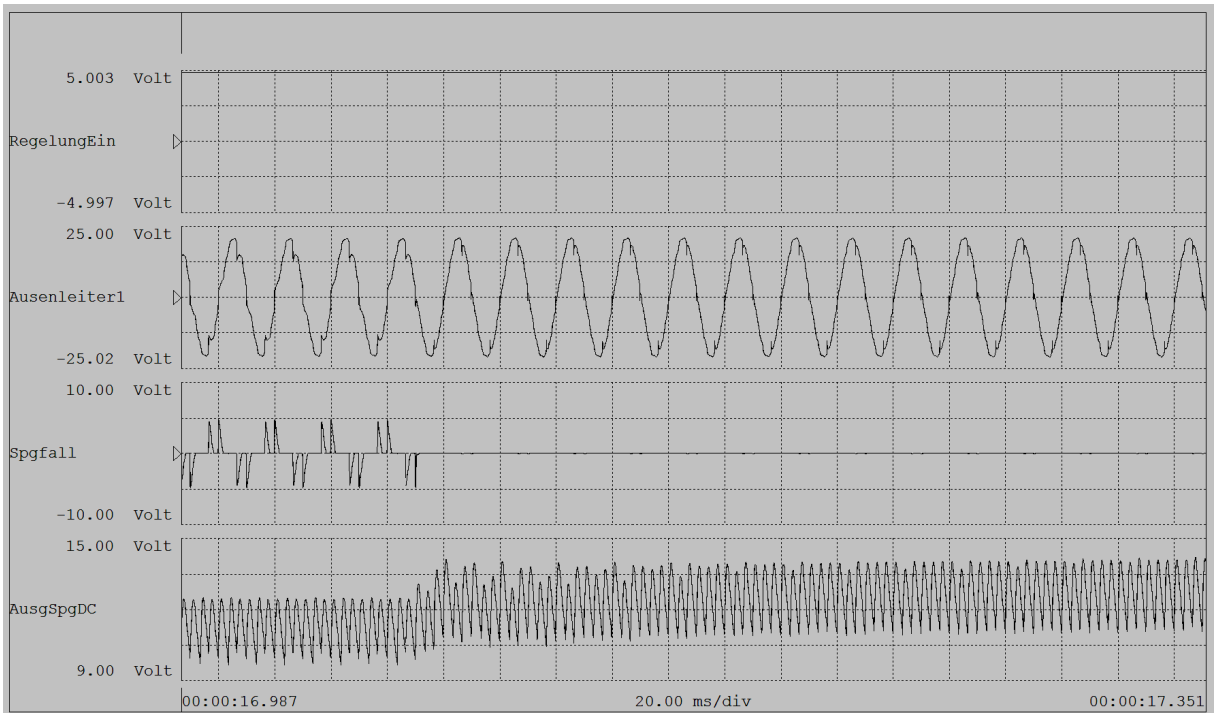


Abbildung 9.10: Ausschalten des Spannungsfalls

9.7 Sollwertsprung bei Unsymmetrien

Diese Änderungen sind ohne Unsymmetrien für die Zündpulserzeugung kein Problem. Wenn jetzt aber eine starke Unsymmetrie der Phasen vorliegt, wirkt diese sich bei unterschiedlichen Phasenanschnittswinkeln auch unterschiedlich stark aus. Dies kann nur durch eine laufende Anpassung durch die Regelung klein gehalten werden.

Um den Extremfall zu testen, wurde keine kontinuierliche Änderung verwendet, sondern ein wirklicher Sprung von 35° vorgegeben. Diese entspricht einem Spannungssprung von 25%.

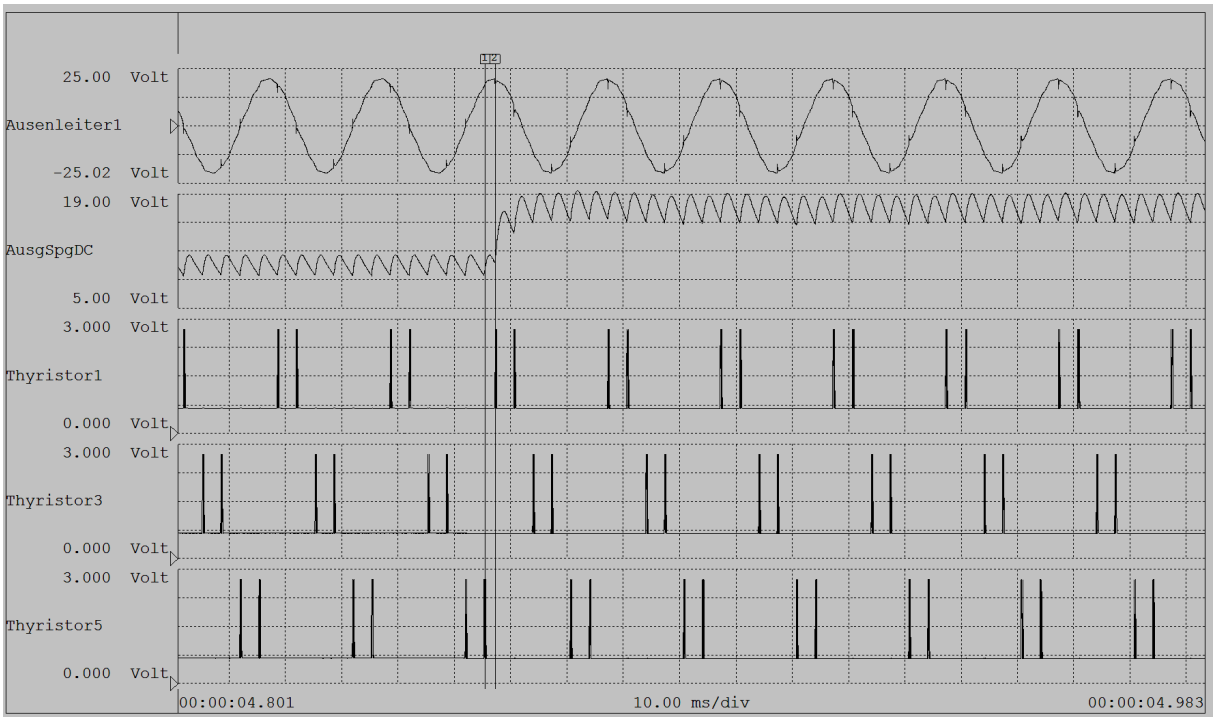


Abbildung 9.11: Aufnahme des Vision vom Sollwertsprung

Wie Abbildung 9.11 zeigt, wirkt sich bei einem kleineren Phasenanschnittswinkel die Unsymmetrie stärker aus als bei einem größeren. Die Regelung gleicht dieses Ripple aber in wenigen Perioden aus und somit führen selbst Sollwertsprünge zu keinem starken Ripple.

Die beiden Courser markieren den Zündpuls vor und nach dem Sollwertsprung. An der Zeit zwischen den Zündpulsen kann genau die Höhe des Sprunges festgestellt werden.

$$Phasenanschnittswinkel\_in\_Grad = Phasenanschnittswinkel\_in\_ms \cdot \frac{20ms}{360°} \quad (0.2)$$

Die 1,94ms entsprechen einer Phasenanschnittswinkeländerung von 34,92°.

9.8 Auswertung der Ergebnisse

Als Grundlage für die Diplomarbeit dienten die Ergebnisse der Simulation im Rahmen der Studienarbeit „Simulation eines digitalen Steuersatzes mit eigenständiger Ausregelung von auftretender Welligkeit“.

Bei der Simulation stellten sich die Phasen- und Netzunsymmetrien als die Fehler heraus, die die stärkste Welligkeit bei der Ausgangsspannung hervorrufen. Diese Fehler bewirken einen 50 bzw. 100Hz Frequenzanteil bei der Ausgangsspannung. Die Fehler wurden analysiert und es wurde eine mögliche Lösung entwickelt.

Es wurde eine Thyristorbrücke zum Testen der entwickelten Software aufgebaut.
Für die vorhandene Hardware musste die benötigte Software zur Erzeugung der Zündpulse entwickelt werden. Die Software wurde mit Quartus II geschrieben und auf dem FPGA der Interfacekarte implementiert.

Der Ablauf des digitalen Steuersatzes ist nach dem gleichen Prinzip wie in der Simulation umgesetzt worden. Die Zündpulse für die sechs Thyristoren werden erzeugt und an den Testaufbau weiter gegeben.

Die Regelung soll anhand des Spannungsverlaufes der Ausgangsspannung erkennen, ob eine 50 bzw. 100Hz Welligkeit vorliegt und diese Unsymmetrien ausgleichen. Es wurde im Vergleich zur Simulation nicht auf eine Referenzspannung, sondern auf eine einheitliche Pulshöhe geregelt.

Die Regelung verschiebt fünf Phasenanschnittswinkel solange, bis die fünf Pulshöhen mit der Höhe des ersten Pulses übereinstimmen. Es wird aus der Differenz von den Pulshöhen(Istwert) und der Höhe des ersten Pulses(Sollwert) die Regelabweichung ermittelt. Die Regelabweichung wird nun auf einen PI-Regler gegeben und dieser verschiebt die Zündzeitpunkte mit einem Korrekturwinkel solange bis die Regelabweichung zu null wird.

Die entwickelte Regelung kann in einem realistischen Fehlerbereich alle Fehler, die sich auf die Symmetrie der Ausgangsspannung auswirken, ausregeln. Dies führt zu einer Verkleinerung des Ripples auf der Ausgangsspannung und somit zu einer höheren Genauigkeit des Ausgangsstromes.

10 Fazit und Verbesserungsansätze

Durch die Tests sind noch Verbesserungsansätze offengelegt worden. Wie schon im Verlauf der Arbeit beschrieben, ist die Messwerterfassung noch nicht ganz ausgereift. Das Hauptproblem dabei ist das für den Spannungsbereich der Thyristorbrücke zu starke Rauschen. Wegen dieses Problems muss noch eine Weiterentwicklung der Thyristorbrücke erfolgen.

Weiterhin sollte der Algorithmus zur Istwertaufnahme nicht nur die Spitzenspannung der sechs Pulse angleichen, sondern den gesamten Ripple minimieren. Dieser Schritt sollte aber erst bei einem im Betrieb befindlichen Thyristorgerät erfolgen. Bei diesen Geräten ist ein effektiveres Filter verbaut. Durch diese Filter werden die 300Hz der Thyristorbrücke sehr stark gedämpft und bei dieser geglätteten Spannung muss getestet werden, welche Regelung den Ripple am Besten verkleinert.

Bisher wurde nur der unterlagerte Regelkreis zur Glättung des Ripples genutzt. Um einen geregelten Betrieb mit Sollwertvorgabe für die Thyristorbrücke zu ermöglichen, muss noch die Kaskadenregelung mit Strom- und Spannungsregelung eingebunden werden. Dann ist es möglich die Stromstärke als Sollwert vorzugeben. Die Istwerte von Ausgangsstrom und Ausgangsspannung werden bereits aufgenommen und somit ist die Grundlage für den geregelten Betrieb gelegt. Durch diesen Schritt ist auch die Sollwertvorgabe in Form der Stromstärke über die Internetseite möglich.

Schließlich muss die ohmsche Last durch eine induktive Last ersetzt werden. Die Thyristorbrücken bei DESY versorgen Magnete, die eine induktive Last sind. Bei einer induktiven Last muss zusätzlich eine Erprobung der gesamten Software erfolgen.

Danksagung

Mein besonderer Dank gilt den Prüfern Herrn Prof. Dr. Gustav Vaupel und
Herrn Prof. Dr. Thomas Holzhüter für die Betreuung dieser Diplomarbeit.

Weiterhin gilt mein besonderer Dank den Betreuern und Angestellten beim Forschungsinstitut DESY, die mich bei der Ausführung der Arbeit durch Ihre kooperative Zusammenarbeit unterstützt haben.
Dieser Dank gebührt in besonderer Weise Herrn Hans-Jörg Eckoldt, Herrn Robert Hanneken und Herrn Niels Heidbrook, sowie den namentlich nicht genannten Mitarbeitern der Gruppe MKK6.

V. Literaturverzeichnis

[1] Vgl.: http://pr.desy.de/e113/index_ger.html

[2] Vgl.: http://www.et.fh-jena.de/wagner/ProgLog_1-Dateien/Entwicklungsablauf_mit_Quartus_II.pdf

[3] Vgl.: http://www.et.fh-jena.de/wagner/ProgLog_1-Dateien/Hardwarebeschreibungssprache_AHDL.pdf

[4] Vgl.: Brechmann, Dzieia, Hörnemann, Hübscher, Jagla, Klaue, Wickert: Elektrotechnik Tabellen Energieelektronik, 5. Auflage, Westermann Verlag, Braunschweig 2002 Seite 313

[5] Vgl.: Mitschke, Fedor.: Glasfasern: Physik und Technologie, 1. Auflage, Spektrum Akademischer Verlag, 2005

[6] Vgl.: Vaupel, Gustav.: Leistungselektronik Skript, HAW Hamburg

[7] Vgl.: Best, Roland: Theorie und Anwendungen des Phase-locked Loops, 4. überarbeitete Auflage, ATVerlag Aarau, Stuttgart 1987 Seite 11

[8] Vgl.: Heumann, K.: Grundlagen der Leistungselektronik, 2. Auflage, Teubner Studienbücher, Stuttgart 1978 Seite 30

[9] Vgl.: Tietze, Schenk.: Halbleiter-Schaltungstechnik, 10. Auflage, Springer-Verlag, München 1993 Seite 9

[10] Vgl.: Frohberg, Kolloschie, Löffler: Taschenbuch der Nachrichtentechnik, Carl Hanser Verlag, München 2008 Seite 106

[11] Vgl.: Sparr, Thomas.: Studienarbeit: „Simulation eines digitalen Steuersatzes mit eigenständiger Ausregelung von auftretender Welligkeit“, Hamburg 2009

VI. Anhang

Anhang A1: Sourcecode

Im Anhang A1 befinden sich die zehn geschriebenen oder abgeänderten Blöcke aus Altera Quartus II. Die Blöcke sind unter folgenden Dateipfaden im Gesamtprogramm zu finden:

Altera_Quartus_II_Programm\Petra3Regelung1Korrektur\

- Petra3_Interface\alphalimit
- \Petra3_Interface\Zuendpulserzeugung
- \Petra3_Interface\delay12bit
- \Petra3_Interface\PLL_V1
- \finished_logic\onepulse
- \Petra3_Interface\frequenz_sync
- \Petra3_Interface\Regeldifferenzerfassung
- \Petra3_Interface\finished_logic\multiswitch6bit
- \Petra3_Interface\finished_part\6BitPIregulator
- \Petra3_Interface\wertbegrenzer

Anhang A2: Studienarbeit

Studienarbeit: „Simulation eines digitalen Steuersatzes mit eigenständiger Ausregelung von auftretender Welligkeit“

Anhang A3: Datenblatt vom Semikon Thyristor Modul

Die Anhänge A1, A2 und A3 sind in elektronischer Form auf einer CD abgelegt und beim Prüfer Prof. Dr. Gustav Vaupel einzusehen.

Anhang A1: Sourcecode

Date: March 12, 2010 Zuendpulserzeugung.tdf Project: ttfkorr2alteras_2

```
1       %/*******************************************************
2       Autor:Thomas Sparr
3       Erstellungsdatum: 12.01.2010
4       Titel:Zuendpulserzeugung
5       1. Programmierung von
6       1 Hauptcounter der mit der PLL auf die 50Hz synchronisiert wird
7       12 Untercounter damit die Pulse nicht im Überlauf gezündet werden
8       Pulslänge wird extern in mikrosec eingestellt und intern umgerechnet
9       der Sicherheitspuls wird erzeugt
10      /%*******************************************************
11      PARAMETERS      (CLKinMHz = MainClkInMHz / 4, MaxCount = ((20000*MainClkInMHz) / 4),
                       PulseRate = 6,PulseWidthInUs = 50, resolution = 17);
12      CONSTANT PulseWidth = (PulseWidthInUs * CLKinMHZ);--Umrechnug von •s in clk-Perioden
13
14      SUBDESIGN Zuendpulserzeugung
15              (Phasenwinkel[resolution..0]:           INPUT = GND;--externer Wert des Phasenanschnittswinkels
16              PllSignal:                              INPUT = GND;--Synchronisationssignal von der PLL
17              30degrees[resolution..0]:               INPUT = GND;--Wert für 30° vom "frequenz_sync" Block
18              60degrees[resolution..0]:               INPUT = GND;--Wert für 60° vom "frequenz_sync" Block
19              Regelwert[6..1][resolution..0]:         INPUT = GND;--Ausgleichswinkel von der Regelung
20              overflow[6..1]:                         INPUT = GND;--vorsorglich definierte Variable
21              clk:                                    INPUT = GND;--Programmtakt des Blocks
22              enable:                                 INPUT = VCC;--Eingang zum Sperren von Variablen
23              reset:                                  INPUT = GND;--Eingang zum Reseten von Variablen
24              out[12..1]:                             OUTPUT;)--Ausgangsvariable zur Ausgabe der Zündpulse
25
26      --intere Variablen
27      VARIABLE        30degreesDFF[6..0][resolution..0],
                        --Übernimmt den externen Wert für die Anzahl von Clk in 30 Grad (variiert bei Frequenzänderungen)
28                      60degreesDFF[6..0][resolution..0],
                        --Übernimmt den externen Wert für die Anzahl von Clk in 60 Grad (variiert bei Frequenzänderungen)
29                      30degreesSafeDFF[6..1][resolution..0],
                        --Speichert den 30 Grad Wert jedes Pulses bis zur nächsten Periode (wird zu Beginn des Counters aktualisiert)
30                      60degreesSafeDFF[6..1][resolution..0],
                        --Speichert den 60 Grad Wert jedes Pulses bis zur nächsten Periode (wird zu Beginn des Counters aktualisiert)
31                      PWMCounterDFF [6..0][resolution..0],         --Counter zum Eingrenzen des Zündbereiches
32                      PhasenwinkelDFF [6..0][resolution..0],       --Übernimmt den externen Wert des Phasenanschnittswinkel
33                      PhasenwinkelSafeDFF[6..1][resolution..0]: DFFE;
                        --Speichert den Zündwinkel jedes Pulses bis zur nächsten Periode (wird zu Beginn des Counters aktualisiert)
34                      outJKFF [PulseRate..1],       --Hilfsflipflop zur Speicherung der Ausgangszustände
35                      PWMCounterClrnJKFF,           --Löscht den MainCounter
36                      EnableJKFF[6..1]: JKFF;       --Freigabe der Unterzähler
37
38
39      BEGIN-- Beginn des Hauptprogramms
```

```
40          -- Übergabe des Taktsignals an die interne Variablen
41          30degreesDFF[][].clk=clk;           60degreesDFF[][].clk=clk;     30degreesSafeDFF[][].clk=clk;
42          60degreesSafeDFF[][].clk=clk;       PWMCounterDFF[][].clk=clk;    PhasenwinkelDFF[][].clk=clk;
43          PhasenwinkelSafeDFF[][].clk=clk;    outJKFF[].clk=clk;            PWMCounterClrnJKFF.clk=clk;          EnableJKFF[].clk=clk;
44
45          --Der PWMCounterDFF[0][] ist der Hauptzähler (Maincounter). Er läuft den vollen Zyklus durch und wird über die PLL synchronisiert
46          --Das PLL-Signal setzt ein Flipflop und setzt somit den MainCounter auf Null.
47          --Der MainCounter wird außerdem über das Reset-Signal genullt. Das FF setzt sich beim nächsten Clk selber zurück.
48          --Bei jedem Clk wird der Counter um eins erhöht.
49          PWMCounterClrnJKFF.j = PllSignal;
50          PWMCounterClrnJKFF.k = PWMCounterClrnJKFF.q == vcc;
51          PWMCounterDFF[0][].ena = vcc;
52          PWMCounterDFF[0][].clrn = PWMCounterClrnJKFF.q == gnd and not reset;
53          PWMCounterDFF[0][].d = PWMCounterDFF[0][].q+1;
54
55          --Übernahme des externes Phasenanschnittswinkels auf eine interne Variable
56          if Phasenwinkel[]<=1 then PhasenwinkelDFF[0][].d = 1;--Phasenwinkel<=1 bewirkt einen Winkel von 1
57                  else PhasenwinkelDFF[0][].d = Phasenwinkel[] ;--Interner Phasenwinkel = externer Phasenwinkel
58          end if;
59
60          --Speicherung der Frequenz bis zum Ende des Zyklus
61          --bewirkt ein konstanten Zünden der Thyristoren über 1 Periode
62          --verhindert, dass durch eine Frequenzänderung ein Thyristor nicht oder doppelt in einer Periode gezündet wird
63          --es wird der Zündzeitpunkt vom 1 und 2 Puls gespeichert
64          30degreesDFF[0][].d=30degrees[];
65          60degreesDFF[0][].d=60degrees[];
66
67          --PhasenwinkelSafeDFF speichert den aktuellen Wert des Phasenwinkels für eine Periode, damit Änderungen
            sich nicht in einer negativen Weise auswirken.
68          --Es wäre sonst möglich, dass ein Puls ausgelassen wird, weil kurz vor dem Zünden der Winkel vergrößert wird.
69          --Für den Wert wird sowohl die 30 Grad Verschiebung, wegen der Counter-Lage berücksichtigt, als auch der
            "Regelwert" der vom Regler aus der Regeldifferenz erzeugt wird.
70          FOR h IN 1 to 6 GENERATE
71                  PhasenwinkelSafeDFF[h][].ena = PWMCounterDFF[h][].q == gnd;
72                  PhasenwinkelDFF[h][].d = PhasenwinkelDFF[0][].q + Regelwert[h][];
73                  PhasenwinkelSafeDFF[h][].d = (PhasenwinkelDFF[h][].q + 30degreesSafeDFF[h][].q);
74          END GENERATE;
75
76          --Die Werte für 30 und 60 Grad werden wie der Phasenwinkel nur zwischen 2 Perioden aktualisiert.
77          --Außerdem wird mit der "if" Bedingung abgesichert, dass kein unzulässiger Wert für 30 bzw. 60 Grad
            übernommen wird.
78          --Falls der Wert zu stark abweicht, wir der alte Wert beibehalten.
79          FOR m IN 1 to 6 GENERATE
80                  30degreesSafeDFF[m][].ena = PWMCounterDFF[m][].q == gnd;
81                  30degreesDFF[m][].d = 30degreesDFF[0][].q;
82                  if 30degrees[]>10000 and 30degrees[]<30000 then 30degreesSafeDFF[m][].d = 30degreesDFF[m][].q;
83                          else 30degreesSafeDFF[m][].d=30degreesSafeDFF[m][].q;
84                  end if;
```

```
85
86                  60degreesSafeDFF[m][].ena = PWMCounterDFF[m][].q == gnd;
87                  60degreesDFF[m][].d = 60degreesDFF[0][].q;
88                  if 60degrees[]>30000 and 60degrees[]<50000 then 60degreesSafeDFF[m][].d = 60degreesDFF[m][].q;
89                        else 60degreesSafeDFF[m][].d=60degreesSafeDFF[m][].q;
90                  end if;
91            END GENERATE;
92
93            --Die sechs zeitversetzten Counter werden mit Hilfe der EnableJKFF und dem Enablebit gesperrt. Die Freigabe
              erfolgt nur zu bestimmten Zeitpunkten.
94            --Der Counter für Zündpuls 1 wird nach ca. 20833 Clk´s (Wert für 30degrees) vom MainCounter gestartet.
95            --Die Counter 2-6 werden immer vom vorigen Counter bei dem Wert von 60degrees gestartet.
96            --Die sechs Counter laufen jeder 300 Grad (208330 Clks) lang.
97            FOR e IN 1 to 6 GENERATE
98                  EnableJKFF[1].j = PWMCounterDFF[0][].q       == 30degreesSafeDFF[e][].q and 30degreesSafeDFF[e][].q != gnd;
99                  EnableJKFF[e].j = PWMCounterDFF[e-1][].q     == 60degreesSafeDFF[e][].q and e!=1 and 30degreesSafeDFF[e][].q != gnd;
100                 EnableJKFF[e].k = PWMCounterDFF[e][].q       >= 60degreesSafeDFF[e][].q+60degreesSafeDFF[e][].q+60degreesSafeDFF[e][].q+
                                                                    60degreesSafeDFF[e][].q+60degreesSafeDFF[e][].q;
101
102                 PWMCounterDFF[e][].ena = vcc;
103                 PWMCounterDFF[e][].d = PWMCounterDFF[e][].q+1;
104                 PWMCounterDFF[e][].clrn = EnableJKFF[e].q == vcc and not reset;--clrn löscht bei false
105           END GENERATE;
106
107
108
109           if PulseRate==6 then out[12..7]=gnd;--Setzt die Ausgänge der 2 Brücke auf GND, wenn nur eine Brücke genutzt wird
110           end if;
111
112           --Das FF "outJKFF" dient zur Ausgabe der Zündpulse für die eingestellte Dauer. Wenn PhasenwinkelSafeDFF
              einen Wert >0 hat und der zugehörige Counter den Wert erreicht
113           --wird das FF gesetzt. Nach Ablauf der Zünddauer wird das FF zurückgesetzt. Die "out" Ports werden von den
              FFs angesteuert. Jeder Thyristor wird pro Periode zweimal gezündet.
114
115           FOR c IN 1 to 6 GENERATE
116                 outJKFF[c].j = PWMCounterDFF[c][].q == PhasenwinkelSafeDFF[c][].q and PhasenwinkelSafeDFF[c][].q!= gnd and not enable;
117                 outJKFF[c].k = PWMCounterDFF[c][].q == PhasenwinkelSafeDFF[c][].q + PulseWidth;
118           END GENERATE;
119
120           out[1]=outJKFF[1].q==vcc or outJKFF[2].q==vcc;--jeder Thyristor wird 2 mal durch den "out[]" Ausgang gezündet
121           out[2]=outJKFF[2].q==vcc or outJKFF[3].q==vcc;
122           out[3]=outJKFF[3].q==vcc or outJKFF[4].q==vcc;
123           out[4]=outJKFF[4].q==vcc or outJKFF[5].q==vcc;
124           out[5]=outJKFF[5].q==vcc or outJKFF[6].q==vcc;
125           out[6]=outJKFF[6].q==vcc or outJKFF[1].q==vcc;
126     END;
```

```
1       %/*************************************************
2       Autor: Niels Heidbrook
3       Erweitert: Thomas Sparr
4       Erstellungsdatum: 06.01.2010
5       Titel: PLL_V1
6
7       Aufgabe:
8       Synchronisierung auf die Netzfrequenz
9
10      Bemerkung:
11      Dieser Block wurde von Niels Heidbrook entwickelt und geschrieben.
12
13      Dieser Sourcecode kann nicht mit kurzen Kommentaren erklärt werden.
14      Zum Nachvollziehen der kompakten Programmierung ist eine gute Programmierkenntnis erforderlich.
15      Die Programmierung hat bei dieser Diplomarbeit nur eine untergeordnete Bedeutung,
16      deswegen wird dieser Baustein nicht weiter erläutert.
17      /%*************************************************
18
19      include "onepulse";
20      PARAMETERS      ( DelayTimeInUs=20000,CLKinMHz=50 / 4,AverageMainsPeriodTimeDFFs=6,AverageOverlapDFFs=4,
21                        MaxOverlapEqualZeroIntegrator=1,OverlapCounterPregulator=4,MaxOverlapToBeLockedInPPM=20000);
22      constant delaycycles=DelayTimeInUs * CLKinMHz;
23      constant SearchingRange=delaycycles / 10;
24      constant OneFourthDelaycycles=delaycycles * 4 / 10;-- / 4
25      constant ThreeFourthDelaycycles=delaycycles - OneFourthDelaycycles;
26      constant maxdff=log2(delaycycles+SearchingRange);
27      constant MaxOverlapToBeLocked=OneFourthDelaycycles / AverageOverlapDFFs * MaxOverlapToBeLockedInPPM / 1000000;
28
29      SUBDESIGN PLL_V1
30      ( D,clk,reset: INPUT = gnd;
31      Q,Locked: OUTPUT;)
32      variable        delaycounter[maxdff..0],MainsPeriodTimeCounter[maxdff..0],MainsPeriodTime[maxdff..0], MaxDelayCounter[maxdff..0]:DFFE;
33                      InDFF,DelayedEndOfInDFF,AverageMainsPeriodTime[AverageMainsPeriodTimeDFFs+maxdff..0],
34                      OverlapCounter[maxdff+OverlapCounterPregulator..0],Overlap[maxdff..0],
35                      OverlapEqualZeroIntegrator[maxdff+MaxOverlapEqualZeroIntegrator..0],
                        AverageOverlap[AverageOverlapDFFs+maxdff..0],LockedDFF:DFFE;
36                      OutJKFF,ResetJKFF:jkff;
37                      EndOfInDFF:onepulse;
38                      AverageMainsPeriodTimeInitNode[AverageMainsPeriodTimeDFFs+maxdff..0],AverageOverlapAddNode[AverageOverlapDFFs+maxdff..0]:node;
39      begin
40              ResetJKFF.(clk,j,k)=(clk,reset,EndOfInDFF.q);
41              OverlapCounter[].d=(OverlapCounter[].q+(1 and (OutJKFF.q xor InDFF.q)
42              and ((delaycounter[].q<OneFourthDelaycycles) or (delaycounter[].q>ThreeFourthDelaycycles)))) and not DelayedEndOfInDFF.q;--
43
```

```
44              OverlapCounter[].clk=clk;
45              Overlap[].(clk,ena,clrn)=(clk,DelayedEndOfInDFF.q,not ResetJKFF.q);
46              AverageOverlapAddNode[maxdff..0]=Overlap[].q- ((( OutJKFF.q) and OverlapCounter[maxdff+
                OverlapCounterPregulator..OverlapCounterPregulator].q) or
47              ((not OutJKFF.q) and -OverlapCounter[maxdff+OverlapCounterPregulator..OverlapCounterPregulator].q));
48
49              AverageOverlapAddNode[AverageOverlapDFFs+maxdff..maxdff+1]=AverageOverlapAddNode[maxdff];
50              AverageOverlap[].(clk,ena,clrn)=(clk,EndOfInDFF.q,not ResetJKFF.q);
51              AverageOverlap[].d=AverageOverlap[].q-AverageOverlapAddNode[];
52              Overlap[].d=AverageOverlap[AverageOverlapDFFs+maxdff..AverageOverlapDFFs].q;
53              OverlapEqualZeroIntegrator[].(clk,ena)=(clk,EndOfInDFF.q);OverlapEqualZeroIntegrator[].d=
                OverlapEqualZeroIntegrator[].q-1+(2 and OutJKFF.q);
54              InDFF.(clk,d)=(clk,D);EndOfInDFF.(clk,d)=(clk,InDFF.q);DelayedEndOfInDFF.(clk,d)=(clk,EndOfInDFF.q);
55              MainsPeriodTimeCounter[].d=(MainsPeriodTimeCounter[].q+1) and not DelayedEndOfInDFF.q;
                MainsPeriodTimeCounter[].clk=clk;
56              MainsPeriodTime[].(clk,ena)=(clk,EndOfInDFF.q);MainsPeriodTime[].d=MainsPeriodTimeCounter[].q;
57              AverageMainsPeriodTimeInitNode[AverageMainsPeriodTimeDFFs+maxdff..AverageMainsPeriodTimeDFFs]=
                MainsPeriodTime[].q;
58              AverageMainsPeriodTimeInitNode[AverageMainsPeriodTimeDFFs-1..0]=gnd;
59              AverageMainsPeriodTime[].(clk,ena)=(clk,DelayedEndOfInDFF.q or ResetJKFF.q);
60              AverageMainsPeriodTime[].d= (
61              ( ((AverageMainsPeriodTime[].q-1) and AverageMainsPeriodTime[
                AverageMainsPeriodTimeDFFs+maxdff..AverageMainsPeriodTimeDFFs].q>MainsPeriodTime[].q)or
62              ((AverageMainsPeriodTime[].q+1) and AverageMainsPeriodTime[
                AverageMainsPeriodTimeDFFs+maxdff..AverageMainsPeriodTimeDFFs].q<=MainsPeriodTime[].q)
63              )and not ResetJKFF.q)or(AverageMainsPeriodTimeInitNode[] and ResetJKFF.q);
64              OutJKFF.(clk,j,k)=(clk,delaycounter[].q==0,
65              delaycounter[maxdff-1..0].q==AverageMainsPeriodTime[AverageMainsPeriodTimeDFFs+
                maxdff..AverageMainsPeriodTimeDFFs+1].q);-- -Pulsewidth
66              delaycounter[].d=(delaycounter[].q+1) and (delaycounter[].q<MaxDelayCounter[].q) and not ResetJKFF.q;
                delaycounter[].clk=clk;
67              MaxDelayCounter[].(clk,ena)=(clk,EndOfInDFF.q or ResetJKFF.q);
68              MaxDelayCounter[].d=(AverageMainsPeriodTime[AverageMainsPeriodTimeDFFs+maxdff..
                AverageMainsPeriodTimeDFFs].q+Overlap[].q)
69              +OverlapEqualZeroIntegrator[maxdff+MaxOverlapEqualZeroIntegrator..
                MaxOverlapEqualZeroIntegrator].q;
70              LockedDFF.d= (( AverageOverlap[AverageOverlapDFFs+maxdff..AverageOverlapDFFs])<
                MaxOverlapToBeLocked)or
71              ((not AverageOverlap[AverageOverlapDFFs+maxdff..AverageOverlapDFFs])<
                MaxOverlapToBeLocked);
72              Locked=LockedDFF.q;LockedDFF.clk=clk;
73              Q=OutJKFF.q;
74      end;
```

```
%/*******************************************************
Autor: Thomas Sparr
Erstellungsdatum: 10.12.2009
Titel: frequenz_sync

Aufgabe:
Runter Brechung der Periodendauer auf 1/6 und 1/12
Speicherung der Werte für ein 1/6 (60°) und 1/12 (30°) von einer Periode
diese Werte erhält die "zuendpulserzeugung" zur exakten Erzeugung der Zündpulse
*******************************************************/%

PARAMETERS (resolution = 17);-- Parameter für die Parameterliste

SUBDESIGN frequenz_sync-- Benennung des Bausteins und Ein- und Ausgangs Definition
        (PllSignal:                 INPUT = gnd;-- Synchronisationssignal von der PLL
        clk:                        INPUT = gnd;-- Programmtakt des Blocks
        enable:                     INPUT = vcc;-- Eingang zum Sperren von Variablen
        reset:                      INPUT = gnd;-- Eingang zum Reseten von Variablen
        30degrees[resolution..0]:   OUTPUT; -- Ausgangsvariable
        60degrees[resolution..0]:   OUTPUT;) -- Ausgangsvariable

--intere Variablen
VARIABLE        HilfsCounterDFF[2..1][resolution..0], -- bewirkt 2 langsamere Taktsignale
                MainCounterDFF[2..1][resolution..0], -- zählt die Durchläufe der HilfsCounter
                degreesPllDFF[2..1][resolution..0]:DFFE;     -- speichert die Werte für 1/6 und 1/12 einer Periode
                                                            für die Dauer einer Periode
BEGIN-- Beginn des Hauptprogramms
        HilfsCounterDFF[][].clk = clk;-- Übergabe des Taktsignals an die interne Variable
        MainCounterDFF[][].clk = clk;-- Übergabe des Taktsignals an die interne Variable
        degreesPllDFF[][].clk = PllSignal;-- diese Variable wird auf die PLL getaktet

        --Die Hilfscounter zählen bis 6 bzw. 12. Wenn die Hilfscounter den MaxCount wert erreicht haben, starten
        --sie neu. Maincounter zählt die Durchläufe der Hilfscounter und enthält am Ende jeder Periode den Wert für
        --30 bzw. 60 Grad. Die PLL gibt die Periodenzeit an.
        MainCounterDFF[2][].ena = HilfsCounterDFF[2][].q==5;--HilfsCounter zur Erzeugung eines 6-fachen clk´s
        MainCounterDFF[1][].ena = HilfsCounterDFF[1][].q==11;--HilfsCounter zur Erzeugung eines 12-fachen clk´s
        MainCounterDFF[][].clrn = not PllSignal;-- löschen vom MainCountr durch das PLL-Signal

        -- Schleife die von 1 bis 2 zählt; die Laufvariable e bekommt die jeweilige Durchlaufzahl
        FOR e IN 1 to 2 GENERATE
                --HilfsCounter und MainCounter werden jede Periode um 1 erhöht
                HilfsCounterDFF[e][].d = HilfsCounterDFF[e][].q+1;
                MainCounterDFF[e][].d = MainCounterDFF[e][].q+1;
        END GENERATE;
```

```
45
46              --Löschen der Hilfscounter mit PLL oder bei erreichen des Endwertes
47              HilfsCounterDFF[2][].clrn = not PllSignal and (HilfsCounterDFF[2][].q!=6);
48              HilfsCounterDFF[1][].clrn = not PllSignal and (HilfsCounterDFF[1][].q!=12);
49
50              --degreesPllDFF bekommt am Periodenanfang den aktuellen Wert für die beiden Winkel und behält diesen bis Perioden-Ende
51              degreesPllDFF[][].d = MainCounterDFF[][].q;
52
53              30degrees[] = degreesPllDFF[1][].q; --Übergabe der Zwischengespeicherten Werte an die Ausgänge
54              60degrees[] = degreesPllDFF[2][].q; --Übergabe der Zwischengespeicherten Werte an die Ausgänge
55      END;
```

```
1       %/*****************************************************
2       Autor:Thomas Sparr
3       Erstellungsdatum: 16.11.2009
4       Titel:alphalimit
5
6       Aufgabe:
7       Eingrenzen vom Phasenanschnittswinkel
8       Der Maximal- und Minimalwert vom Phasenanschnittswinkel können vorgegeben werden
9       Dadurch wird genug Reserve zum Ausregeln von Unsymmetrien bereitgehalten
10      *****************************************************/%
11
12      PARAMETERS ( CLKinMHz = 1, resolution = 7, highestvalue = 1, lowestvalue = 1);-- Parameter der Parameterliste
13
14      SUBDESIGN alphalimit--Benennung des Bausteins und Ein- und Ausgangs Definition
15                    (clk:                         INPUT=gnd;--Programmtakt des Blocks
16                    PulseWidth[resolution..0]:    INPUT=gnd;--Eingangsvariable (Phasenanschnittswinkel)
17                    out[resolution..0]:           OUTPUT;)--Ausgangsvariable
18
19      VARIABLE PulseWidthDFF[resolution..0]:DFFE;--interne Variable zur Zwischenspeicherung
20
21      BEGIN-- Beginn des Hauptprogramms
22
23              PulseWidthDFF[].clk=clk;-- Übergabe des Taktsignals an die interne Variable
24
25              --es wird geprüft ob Alpha in einem bestimmten Bereich liegt,
26              --wenn ja, wird der Wert übergeben und sonst wird der Grenzwert weitergegeben
27              --die interne Variable bekommt den Wert
28              if PulseWidth[]>highestvalue then PulseWidthDFF[].d=highestvalue;
29                      else if PulseWidth[]<lowestvalue then PulseWidthDFF[].d=lowestvalue;
30                              else PulseWidthDFF[].d=PulseWidth[];
31                      end if;
32              end if;
33              out[]=PulseWidthDFF[].q;-- Übergabe der internen Variable an den Ausgang
34      END;
```

```
1       %/*******************************************************
2       Autor: Thomas Sparr
3       Erstellungsdatum: 28.10.2009
4       Titel: delay12bit
5
6       Es wird mit 3 Countern ein zweiter Puls, der um eine Zeit t nach dem ersten kommt, erzeugt.
7       der 1 Counter bestimmt die Länge des Pulses
8       der 2 Counter bekommt durch WaitInUs den Abstand
9       der 3 Counter begrenzt den Puls auf die Länge des Referenzpulses
10      *******************************************************/%
11      PARAMETERS (CLKinMHz = 12.5, WaitInUs = 15 , PulseRate = 6);--Parameter der Parameterliste
12
13      CONSTANT wait = (WaitInUs * CLKinMHZ);--Umrechnung von usec auf clk
14      CONSTANT maxdff=12;--Constant um alle drei Counter in ihrer Bitbreite synchron ändern zu können
15
16      SUBDESIGN delay12bit
17              (pulse[PulseRate..1]:  INPUT = gnd;--Zündpulse am Eingang
18              clk:                   INPUT = gnd;--Programmtakt des Blocks
19              out[PulseRate..1]:     OUTPUT;)--Zündpulse am Ausgang
20
21      VARIABLE        counterscan[PulseRate..1][maxdff..0],--Scant den Puls
22                      counterwait[PulseRate..1][maxdff..0],--Berücksichtigt die Zeit zwischen den beiden Pulsen
23                      counterpulse[PulseRate..1][maxdff..0],--gibt die Länge des zweiten Pulses vor
24                      InDFF[PulseRate..1]:DFFE;--übernimmt den Eingangswert ,ist VCC wenn ein Zündpuls erzeugt wird
25                      OutJKFF[PulseRate..1]:JKFF;--speichert den Ausgabewert
26      BEGIN
27              InDFF[].clk=clk;-- Übergabe des Taktsignals an die interne Variablen
28              OutJKFF[].clk=clk;
29              counterscan[][].clk=clk;
30              counterwait[][].clk=clk;
31              counterpulse[][].clk=clk;
32
33              FOR e IN 1 to PulseRate GENERATE
34                      InDFF[e].d = pulse[e] == vcc;--Detektierung des Zündpulses
35                      counterscan[e][].d = counterscan[e][].q+1;--Hochzählen der nicht durch enable gesperrten Counter
36                      counterwait[e][].d = counterwait[e][].q+1;--Hochzählen der nicht durch enable gesperrten Counter
37                      counterpulse[e][].d = counterpulse[e][].q+1;--Hochzählen der nicht durch enable gesperrten Counter
38              END GENERATE;
39
40              FOR k IN 1 to PulseRate GENERATE
41                      counterscan[k][].ena = InDFF[k].q==vcc;--während des Zündpuls wird Counter 1 entsperrt
42                      counterwait[k][].ena = counterwait[k][].q!=wait and counterscan[k][].q!=gnd and InDFF[k].q==gnd;
                        --nach feststellen der Zündpulslänge wird Counter 2 aktiv
43                      counterpulse[k][].ena = counterwait[k][].q==wait;
                        --Wenn Counter 2 die vorgegebene Wartezeit ereicht hat beginnt Counter 3 zu zählen
44
```

```
45                  if (counterpulse[k][].q == counterscan[k][].q and counterpulse[k][].q != gnd
                                                        and OutJKFF[k].q == gnd) then
46                  --wenn der zweite Puls ausgegeben wurde, werden die Counter genullt
47                         counterscan[k][].clrn = gnd;
48                         counterwait[k][].clrn = gnd;
49                         counterpulse[k][].clrn = gnd;
50                         else
51                                counterscan[k][].clrn = vcc;
52                                counterwait[k][].clrn = vcc;
53                                counterpulse[k][].clrn = vcc;
54                  end if;
55
56                  OutJKFF[k].j = counterwait[k][].q == wait and counterpulse[k][].q < counterscan[k][].q;
                    --nach dem Erreichen der Wartezeit wird der Ausgang auf VCC gesetzt
57                  OutJKFF[k].k = counterpulse[k][].q == counterscan[k][].q;
                    --wenn der Counter 3 den Wert von Counter 1 erreicht hat wird der Zündpuls beendet
58                  out[k] = OutJKFF[k].q or pulse[k];
                    --der Ausgang ist 1, wenn der Sicherheitspuls erzeugt wird und wenn der eigentliche Puls am Eingang anliegt
59            END GENERATE;
60    END;
```

```
1       %/*****************************************************
2       Autor: Thomas Sparr
3       Erstellungsdatum: 06.01.2010
4       Titel: Regeldifferenzerfassung
5
6       Aufgabe:
7       Erfassung der 6 Pulse
8       Auswertung der Pulshoehen (Bildung der Regeldifferenz)
9       *****************************************************/%
10      PARAMETERS (MaxCount = 500 ,PulseRate = 6, resolution = 17);--Parameter der Parameterliste
11
12      SUBDESIGN Regeldifferenzerfassung-- Benennung des Bausteins und Ein- und Ausgangs Definition
13              (PllSignal:                     INPUT = gnd;--Synchronisationsignal von der PLL
14              thyristorpulse[6..1]:           INPUT = gnd;--die von der "zuendpulserzeugung" erzeugten Zündpulse
15              pulshoehe[17..0]:                       INPUT = gnd;--Ausgangsspannung, laufender IST-Wert
16              clk:                                    INPUT = gnd;--Programmtakt des Blocks
17              enable:                         INPUT = vcc;--Eingang zum Sperren von Variablen
18              reset:                          INPUT = gnd;--Eingang zum Reseten von Variablen
19              regeldifferenz[6..1][17..0]:    OUTPUT;--Ausgangsvariable
20              pulshoeheout[6..1][17..0]:      OUTPUT;)--Ausgangsvariable
21
22      --intere Variablen
23      VARIABLE        CounterDFF[6..1][resolution..0], --Counter zur Aufnahme der drei Werte zur Ermittlung der Pulshöhe
24                      regeldifferenzDFF[6..1][17..0], --Zwischenspeicherung der Regeldifferenz für eine Periode
25                      MainCounterDFF[6..1][resolution..0], --grenzt den Bereich von der jeweiligen Pulsspitzenspannung ein
26                      wendepunkta[6..1][17..0], --dritter aufgenommener Momentanwert der Pulshöhe
27                      wendepunktb[6..1][17..0], --zweiter aufgenommener Momentanwert der Pulshöhe
28                      wendepunktc[6..1][17..0], --erster aufgenommener Momentanwert der Pulshöhe
29                      pulshoeheDFF[6..1][17..0]: DFFE; --Zwischenspeicherung der Pulshöhe für eine Periode
30                      EnableJKFF[6..1]: JKFF; --Flipflop zur Freigabe der Pulshöhenaufnahme
31
32      BEGIN-- Beginn des Hauptprogramms
33              CounterDFF[][].clk              = clk;-- Übergabe des Taktsignals an die interne Variablen
34              EnableJKFF[].clk                = clk;
35              regeldifferenzDFF[][].clk       = clk;
36              pulshoeheDFF[][].clk            = clk;
37              MainCounterDFF[][].clk          = clk;
38              wendepunkta[][].clk             = clk;
39              wendepunktb[][].clk             = clk;
40              wendepunktc[][].clk             = clk;
41
```

```
42          --Die Schleife fragt jeden Clk ab ob ein Thyristor gezündet wird. Bei "thyristorpulse=1" wird ein FlipFlop
43          --gesetzt. Das FF startet einen Counter und dieser bewirkt bei Wert X eine Speicherung der aktuellen Höhe der Ausgangsspannung.
44          --Der Wert wird in der Variablen "pulshoeheDFF" gespeichert, bis in der nächsten Periode der Zündpuls
            wieder high ist.
45          --Der Counter zählt bis 100000 weiter, damit der Sicherheitspuls übersprungen wird.
46
47          --Schleife die von 1 bis 2 zählt; die Laufvariable e bekommt die jeweilige Durchlaufzahl
48          FOR e IN 1 to 6 GENERATE
49                 EnableJKFF[e].j = thyristorpulse[e]==vcc;--durch den Zündpuls bei einem Thyristor wird ein Flipflop gesetzt
50                 EnableJKFF[e].k = MainCounterDFF[e][]>=60000;--durch erreichen einesWertes bei MainCounter
                                                                wird das Flipflop zurückgesetz
51                 MainCounterDFF[e][].ena = EnableJKFF[e].q;--das gesetzte Flipflop gentsperrt den MainCounter
52                 MainCounterDFF[e][].clrn= EnableJKFF[e].q;--bei nicht gesetzten Flipflop wird der MainCounter genullt
53                 MainCounterDFF[e][].d = MainCounterDFF[e][].q+1;--wenn der MainCounter freigegeben ist (Enable=1) erhöht jeder clk um eins
54
55                 CounterDFF[e][].ena = EnableJKFF[e].q;--das gesetzte Flipflop entsperrt den Counter
56                 CounterDFF[e][].clrn= CounterDFF[e][].q<=1250 and EnableJKFF[e].q and MainCounterDFF[e][]>=5000;
57                 -- im Abstand von 1250 clk´s wird der Counter genullt
58                 -- dies gilt nur wenn das Flpflop gesetzt ist und der Zündpuls mindesten 5000 Clks zurückliegt
59                 CounterDFF[e][].d = CounterDFF[e][].q+1;
                   --wenn der MainCounter freigegeben ist (Enable=1) erhöht jeder Clk um eins
60
61                 if CounterDFF[e][].q==20 then wendepunktc[e][].ena=vcc;
                   --wenn der Counter den Wert 20 hat wird "wendepunktc" entsperrt
62                        else wendepunktc[e][].ena=gnd;
63                 end if;
64                 wendepunktc[e][].d=pulshoehe[]; --und bekommt den Momentanwert der Ausgangsspannung
65
66                 if wendepunktc[e][].q>=200 and CounterDFF[e][].q==15 then wendepunktb[e][].ena=vcc;
67                 --wenn "wendepunktc" einen Wert hat und Counter den Wert 15 hat wird "wendepunktb" entsperrt
68                        else wendepunktb[e][].ena=gnd;
69                 end if;
70                 wendepunktb[e][].d=wendepunktc[e][].q; --und "wendepunktb" bekommt den Wert von "wendepunktc"
71
72                 if wendepunktb[e][].q>=200 and CounterDFF[e][].q==10 then wendepunkta[e][].ena=vcc;
73                 --wenn "wendepunktb" einen Wert hat und Counter den Wert 10 hat wird "wendepunkta" entsperrt
74                        else wendepunkta[e][].ena=gnd;
75                 end if;
76                 wendepunkta[e][].d=wendepunktb[e][].q; --und "wendepunkta" bekommt den Wert von "wendepunktb"
77
78                 --es wird so jeden clk ein neuer Wert aufgenommen und die Werte jedesmal eine Variable weiter gegeben
79
80
81                 if wendepunktb[e][].q>wendepunktc[e][].q and wendepunktb[e][].q>wendepunkta[e][].q
                                                              and wendepunktc[e][].q > 0 then pulshoeheDFF[e][].ena = vcc;
82                 --wenn jetzt der mittlere Wert, also "wendepunktb" am Größten ist wird "pulshoeheDFF" entsperrt
83                        else pulshoeheDFF[e][].ena = gnd;
84                 end if;
```

```
85
86                  --die drei Werte für die Spitzenspannungsermittlung werden durch reset, durch ein nichtgesetztes
                    Flipflop oder eine zu niedrige "pulshoehe" genullt
87                  wendepunkta[e][].clrn=not reset and EnableJKFF[e].q and pulshoehe[17..0] > 100;
88                  wendepunktb[e][].clrn=not reset and EnableJKFF[e].q and pulshoehe[17..0] > 100;
89                  wendepunktc[e][].clrn=not reset and EnableJKFF[e].q and pulshoehe[17..0] > 100;
90
91                  pulshoeheDFF[e][].d = wendepunktb[e][].q; --wenn "pulshoeheDFF" entsperrt ist bekommt er den
92                  -- Wert von "wendepunktb" dieser Wert ist ungefähr der Wert der Spitzenspannung
93          END GENERATE;
94
95          --In den 6 Clk´s nach der Wertaufnahme der Pulshöhe wird mit dem Wert der Pulshöhe des 1 Pulses die
            Regeldifferenz berechnet.
96          --Mit dem Enableeingang wird dieser Wert bis zur nächsten Periode gespeichert. Die Regeldifferenz wird
            erst berechnet,
97          --wenn die betreffenden Pulshöhen auch Werte haben.
98          FOR d IN 2 to 6 GENERATE
99                  regeldifferenzDFF[d][].ena = (MainCounterDFF[d][].q>50000 and MainCounterDFF[d][].q<=50100);
100                 if pulshoeheDFF[d][].q>=400 and pulshoehe[17..0] > 100 then regeldifferenzDFF[d][].d =
                       pulshoeheDFF[d][].q - pulshoeheDFF[1][].q;
101                 else regeldifferenzDFF[d][].ena = vcc;
102                         regeldifferenzDFF[d][].d = gnd;
103                 end if;
104         END GENERATE;
105
106         pulshoeheout[][] = pulshoeheDFF[][].q;--der zwischengespeicherte Wert der Spannungshöhe wird ausgegeben
107         regeldifferenzDFF[1][].d = 0; --"pulshoheDFF[1][]" wird als Sollwert angesehen, deswegen wird die "regeldifferenz[1][]" auf null
            gesetz
108         -- dies geschieht, weil es zu keiner Regeldifferenz kommen kann
109         regeldifferenz[][] = regeldifferenzDFF[][].q;--die zwischengespeicherte Regeldifferenz wird ausgegeben
110 END;
```

```
1       %/*****************************************************
2       Autor: Niels Heidbrook
3       Erweitert: Thomas Sparr
4       Erstellungsdatum: 26.01.2010
5       Titel: multiswitch6bit
6
7       Aufgabe:
8       Ausschalten der Regelung
9       Statt der realen Regeldifferenz bekommt der Regler
10      eine Regeldifferenz von null
11  *****************************************************/%
12
13      PARAMETERS (max = 15);--Parameter für die Parameterliste
14
15      SUBDESIGN multiswitch6bit-- Benennung des Bausteins und Ein- und Ausgangs Definition
16              (A[6..1][max..0]:      INPUT = gnd;--Eingang der Regeldifferenz
17              B[6..1][max..0]:       INPUT = gnd;--Eingang mit Nullpotenzial
18              s:                     INPUT = gnd;--Steuereingangg
19              Q[6..1][max..0]:       OUTPUT;)--Ausgangsvariable
20
21      BEGIN-- Beginn des Hauptprogramms
22              if s==vcc then Q[][]=A[][];--wenn der Steuerbefehl high ist wird die reale Regeldifferenz auf den Regler gegeben
23                      else Q[][]=B[][];--sonst eine Null
24              end if;
25      END;
```

```
1       %*******************************************************
2       Autor:Niels Heidbrook
3       Erweitert:Thomas Sparr
4       Erstellungsdatum: 06.01.2010
5       Titel:6BitPIregulator
6
7       Aufgabe:
8       Ausgleich der Regeldifferenz
9
10      Bemerkung:
11      Dieser Block wurde von Niels Heidbrook entwickelt und geschrieben.
12      Nur die Schleife und die Variablen Erweiterung von z.B. c[cResolution..0] auf c[6..1][cResolution..0] wurde eingefügt.
13      Dies ermöglicht eine Verarbeitung von 6 Werten.
14
15      Diesen Block zu erklären, bedarf mehr wie ein paar Kommentaren.
16      Es wird mit 4 Include-Dateien gearbeitet, die wiederum auch wieder mit Include Dateien arbeiten.
17      Diese Zusammenhänge setzen eine gute Programmierkenntnis voraus.
18      Die Programmierung hat bei dieser Diplomarbeit nur eine untergeordnete Bedeutung,
19      deswegen wird dieser Baustein nicht weiter erläutert.
20  *******************************************************%
21
22      INCLUDE "LimitedAddition";INCLUDE "MsbRespectiveMulClk";INCLUDE "iregulator";INCLUDE "onepulse";
23      PARAMETERS (cResolution = 15,eResolution = 15,uResolution = 15,POutputBitHigh = 25);
24      SUBDESIGN 6BitPIregulator
25              (c[1..0][cResolution..0]:     INPUT=GND;
26              e[6..1][eResolution..0]:      INPUT=GND;
27              PRegulatorCLK:                INPUT=GND;
28              IRegulatorCLK:                INPUT=GND;
29              Reset:                        INPUT;
30              clk:                          INPUT;
31
32              u[6..1][uResolution..0]:      OUTPUT;
33              P[6..1][uResolution..0]:      OUTPUT;
34              I[6..1][uResolution..0]:      OUTPUT;
35              Poverflow[6..1]:              OUTPUT;
36              Ioverflow[6..1]:              OUTPUT;
37              PIoverflow[6..1]:             OUTPUT;
38              AdderOverflow[6..1]:          OUTPUT;
39              busyn[6..1]:                  OUTPUT;)
40
41      VARIABLE uMulLimitatedAdding[6..1]:LimitedAddition with ( Resolution=uResolution);
42              LimitedAdding[6..1]:LimitedAddition with ( Resolution=uResolution);
43              Pregulator[6..1]:MsbRespectiveMulClk with ( Abit=eResolution,Bbit=cResolution,
                ResultBitHigh=POutputBitHigh, ResultBitLow=POutputBitHigh-uResolution);
44              integrator[6..1]:iregulator with ( eResolution=eResolution,cResolution=
                cResolution,uResolution=uResolution);
```

```
45      BEGIN
46              FOR f IN 1 to 6 GENERATE
47                      Pregulator[f].(clk,Reset,start)=(clk,Reset,PRegulatorCLK);
48                      Pregulator[f].A[]=e[f][];
49                      Pregulator[f].B[]=c[0][];
50
51                      integrator[f].(clk,Reset,RegulatorCLK)=(clk,Reset,IRegulatorCLK);
52                      integrator[f].e[]=e[f][];
53                      integrator[f].c[]=c[1][];
54
55                      LimitedAdding[f].A[]=Pregulator[f].result[];
56                      LimitedAdding[f].B[]=integrator[f].u[];
57
58                      P[f][]=Pregulator[f].result[];
59                      I[f][]=integrator[f].u[];
60                      u[f][]=LimitedAdding[f].Result[];
61
62                      Poverflow[f]=Pregulator[f].overflow;
63                      Ioverflow[f]=integrator[f].overflow;
64                      AdderOverflow[f]=LimitedAdding[f].overflow;
65                      PIoverflow[f]=Pregulator[f].overflow or integrator[f].overflow or LimitedAdding[f].overflow;
66
67                      busyn[f]= Pregulator[f].busyn nor integrator[f].busyn;
68              END GENERATE;
69      END;
```

```
%/*****************************************************
Autor: Thomas Sparr
Erstellungsdatum: 06.01.2009
Titel: wertbegrenzer

Aufgabe:
begrenzt den Wert des Reglerausgangs, damit die Zündpulse zeitlich nicht unzulässig verschoben werden können
es soll ein Weglaufen des Regelwertes durch starkes Ansteigen des Integrators verhindert werden
*****************************************************/%
PARAMETERS (resolution = 17, MaxDifferenz=8000);-- Parameter der Parameterliste

SUBDESIGN wertbegrenzer--Benennung des Bausteins und Ein- und Ausgangs Definition
       (u[6..1][resolution..0]:      INPUT = gnd;--Eingang für die Werte des Reglerausganges
       clk:                          INPUT = gnd;--Programmtakt des Blocks
       out[6..1][resolution..0]:     OUTPUT;--Ausgangsvariable
       overflow:                     OUTPUT;)--Ausgangsvariable

BEGIN-- Beginn des Hauptprogramms

       -- Schleife die von 1 bis 6 zählt; die Laufvariable e bekommt die jeweilige Durchlaufzahl
       FOR e IN 1 to 6 GENERATE
       --Mit dem Parameter MaxDifferenz wird der Regelwert der Regelung kontrolliert.
       --Wenn der Regelwert den zulässigen Bereich verlässt wird dieser begrenzt.
       --Die Zündpulserzeugung erhält dann den minimal- oder maximalzulässigen Wert.
       --Das Verlassen des Wertebereichs wird durch den Ausgang "overflow" angezeigt.
              if u[e][]>MaxDifferenz and u[e][]<120000 then out[e][]=MaxDifferenz;
              overflow=vcc;
                     else if u[e][]<262143-MaxDifferenz and u[e][]>120000 then out[e][]=262143-MaxDifferenz;
                     overflow=vcc;
                            else out[e][]=u[e][];
                     end if;
              end if;
       END GENERATE;
END;
```

Hiermit erkläre ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §25(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommen Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hetlingen, den _____