Fachhochschule
Dortmund
University of Applied Sciences and Arts

wilo

Phasenstrommessung in der Antriebstechnik mittels Sigma-Delta Analog Digital Wandlung

Bachelorarbeit

am Fachbereich Elektrotechnik

Studiengang Elektrotechnik

vorgelegt von:	Julian Noss
Erstprüfer:	Prof. Dr. Michael Karagounis
Zweitprüfer:	Oliver Konopka; Wilo SE
Abgabetermin:	12.03.2024

Selbstständigkeitserklärung

Hiermit versichere ich, dass die von mir vorgelegte Prüfungsleistung selbständig und ohne unzulässige Hilfe erstellt worden ist. Alle verwendeten Quellen sind in meiner Arbeit als wörtliche und sinngemäße Übernahmen aus anderen Werken kenntlich gemacht, damit Art und Umfang der Verwendung nachvollziehbar sind.

Dortmund den 12.03.2024 Julian Noss _____

Abstract

In dieser Arbeit wird die Entwicklung einer Filterstruktur in VHDL zur Auswertung eines Sigma-Delta gewandelten Signals dokumentiert. Dafür werden Funktionsweise, Aufbau und Verwendung des Modulators und des Filters dargestellt. Zur Überprüfung wird der Filter sowohl simuliert als auch auf einem Arty Z7 FPGA Board ausgeführt und der Ausgang über einen DAC mit einem Oszilloskop gemessen.

English Translation:

In this report, the development of a filter structure in VHDL for the evaluation of a Sigma-Delta convertet signal is documented. It presents the functionality, structure, and application of the modulator and filter. To verify the filter, it is both simulated, and implemented on an Arty Z7 FPGA Board, and its output is measured using an oscilloscope through a DAC.

Inhalt

1 Einleitung

Die Umwandlung von analogen in digitale Signale und umgekehrt ist ein grundlegender Prozess in der modernen Elektronik. Diese Technologie wird in einer Vielzahl von Anwendungen eingesetzt, von der Audio-Technik bis zur Messdatenerfassung. Auch für die Messung der Phasenströme eines Elektromotors ist sie von Bedeutung.

Der Sigma-Delta-Wandler spielt in diesem Zusammenhang eine zentrale Rolle. Er ermöglicht eine hochauflösende Digitalisierung analoger Signale mittels Oversampling und Noise-Shaping. Da dieser jedoch nur ein einzelnes Bit zur Darstellung verwendet, wird eine Nachbearbeitung benötigt. Dieser Schritt ist entscheidend für Signalqualität und die Reduktion der Datenrate auf ein praktikables Maß. Dies geschieht durch die Anwendung eines Tiefpassfilters und einer Dezimierung.

Die Arbeit beginnt mit einer Darstellung der Aufgabenstellung und geht dann auf die Grundlagen ein, in denen alle Funktionen und Prozesse, der in dieser Arbeit verwendeten Komponenten erklärt werden. Besonderes Augenmerk wird dabei auf den Aufbau und die Funktionsweise des Sigma-Delta-Wandlers gelegt. Es wird untersucht, welchen Einfluss die einzelnen Komponenten auf das Quantisierungsrauschen haben und welche Vorteile dadurch entstehen.

Anschließend wird eine Filterstruktur vorbereitet und unter Verwendung der optimalen Dezimierungsraten in der „Very High Speed Integrated Circuit Hardware Description Language“ (VHDL) umgesetzt. Um die Filterergebnisse einfacher zu präsentieren, wird direkt im Anschluss das VHDL-Modul für die „Serial Peripheral Interface“ (SPI)-Kommunikation zum Digital-Analog-Wandler (DAC) erstellt, welcher das verarbeitete Signal an seinen Ausgängen repräsentiert.

Zur Verifikation dienen entsprechende Testbenches, deren Ergebnisse kurz darauf präsentiert werden.

Die finale Phase der Arbeit besteht aus einem Versuchsaufbau, der dazu dient, Schwachstellen des Designs aufzuzeigen. Außerdem wird ein Ausblick auf die Verwendung des Designs in weiteren Schaltungen gegeben.

2 Aufgabenstellung

Im Rahmen dieser Arbeit wird die Entwicklung eines Systems zur Messung des Phasenstroms in Motorsteuerungsanwendungen untersucht. Hierbei wird der Sigma-Delta-Analog-Digital-Wandler (ADC) AD7403 auf der „Field Programmable Gate Array" (FPGA)-Plattform XC7Z020-CLG400C verwendet. Das System wird auf der Entwicklungsplattform Arty Z7 implementiert, die auf dem All-Programmable SoC Zynq-7000 Z7-20 basiert. Abbildung 1 zeigt den grundlegenden Aufbau und die drei Hauptbereiche der Arbeit. Links, als ersten Teil den Sigma-Delta-Modulator, dann die Verarbeitung der Daten in der Mitte und schlussendlich die Ausgabe der Daten über DACs.

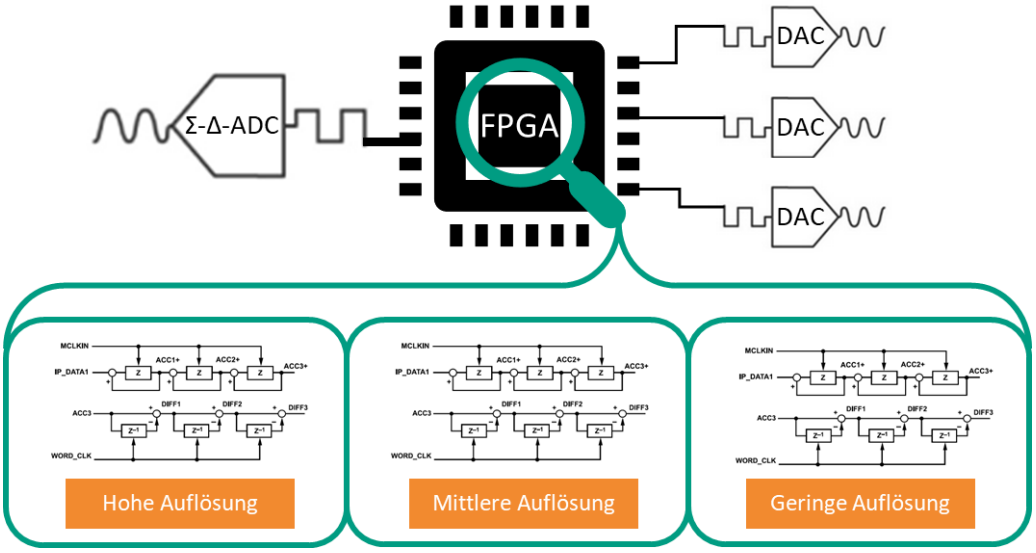

Abbildung 1: Grundlegender Aufbau

2.1 Sigma-Delta-ADC

Zunächst wird das Prinzip des Sigma-Delta-ADC erläutert. Hierbei handelt es sich um eine Technik, die sich durch ihre Fähigkeit auszeichnet, mit nur einem Bit hochauflösende digitale Signale aus analogen Eingängen zu generieren. Damit ist diese besonders in der digitalen Signalverarbeitung und bei präzisen Messanwendungen von Bedeutung. Die Bearbeitung des Themas beleuchtet die Funktionsweise, Vorteile und Herausforderungen dieser Technologie im Kontext der Phasenstrommessung.

2.2 Entwicklung einer Auswertelogik für die Phasenstrommessung

Der Fokus der Arbeit liegt insbesondere auf der Entwicklung einer Logik zur Auswertung des Signals, das vom Sigma-Delta-ADC erfasst wird, um die Phasenströme als digitale Werte zur Verfügung zu stellen. Es werden drei Auflösungsstufen verwendet:

- **Hochauflösend** für die präzise „Field Oriented Control" (FOC)-Motorregelung
- **Mittlere Auflösung** für die Überstromüberwachung
- **Geringe Auflösung** für die Erkennung von Kurzschluss- bzw. Erdschlussfällen

Durch die Anpassung der Auflösung können Messdaten entsprechend der Anforderungen des jeweiligen Anwendungsfalls besser verarbeitet werden.

Zur Verifizierung der entwickelten Auswertlogik wird eine Testbench verwendet, die eine systematische Überprüfung der Funktionalität und Präzision des Systems unter simulierten Bedingungen ermöglicht.

2.3 Ausgabe des ausgewerteten Signals mittels DAC

Um die Filterfunktion zu überprüfen, wird ein DAC verwendet, der die Filterwerte als analoge Werte repräsentiert. Die verarbeiteten Daten müssen in einer Form vorliegen, die eine Auswertung und Ausgabe durch den DAC ermöglicht. Um eine Übertragung über den SPI-Bus zur Zielhardware zu ermöglichen, ist eine spezifische Anpassung der Daten hinsichtlich ihrer Größe erforderlich.

3 Grundlagen

Dieses Kapitel beschäftigt sich mit dem theoretischen Fundament der Arbeit. Zunächst werden die relevanten Begrifflichkeiten definiert und ihre Bedeutung im Kontext der Arbeit erläutert. Es dient als Ausgangspunkt für eine tiefergehende Auseinandersetzung mit dem Thema.

3.1 Dreiphasenmotor

Ein Dreiphasenmotor ist ein Elektromotor, der mit Drehstrom betrieben wird. Der Motor setzt sich aus zwei Hauptkomponenten zusammen: dem Stator (auch Ständer genannt), an dem sich die Anschlüsse der drei Phasen befinden (siehe Abbildung 2), und dem Rotor (auch Läufer genannt), der sich mittig befindet und als Permanentmagnet dargestellt wird.

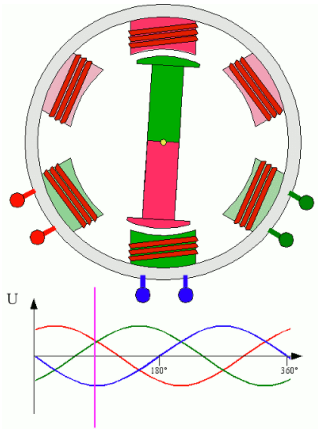

Abbildung 2: Vereinfachter Aufbau eines Dreiphasen-Synchron-Motors [1]

Der Stator eines Dreiphasen-Synchronmotors besteht aus einem zylindrischen Eisenkern mit mehreren Aussparungen, in denen sich die dreiphasigen Wicklungen befinden. Die Wicklungen werden von einem Drehstrom durchflossen, was zur Erzeugung eines zeitlich veränderlichen und räumlich rotierenden Magnetfeldes im Inneren des Motors führt. Die Anordnung der Wicklungen in einem Dreiphasensystem ermöglicht es, ein gleichmäßiges und kontinuierlich rotierendes Magnetfeld zu erzeugen, das für die Drehbewegung des Rotors essenziell ist. Die Auslegung dieser Wicklungen und ihre spezifische Anordnung sind kritische Faktoren, welche die Effizienz des Motors und die Qualität des erzeugten Magnetfeldes beeinflussen. Der Stator besteht in der Regel aus lamelliertem Stahl, um die Wirbelstrom-

verluste zu minimieren und die magnetische Leitfähigkeit zu maximieren. Diese Konstruktion trägt zur Effizienz des Motors bei, indem sie sicherstellt, dass das Magnetfeld mit minimalen Energieverlusten erzeugt wird. [2]

Der Rotor befindet sich im Zentrum des Motors und kann sich frei drehen. Je nach synchronem oder asynchronem Betrieb kann der Aufbau des Rotors variieren. Es ist daher wichtig, den Rotortyp entsprechend der Anwendung und Betriebsweise auszuwählen. Da Synchronmotoren aufgrund ihrer anpassbaren Drehzahl und damit verbundenen Effizienz durch präzise Steuerung besonders für den Einsatz in Pumpen geeignet sind, befasst sich dieser Abschnitt hauptsächlich mit ihrer Konstruktion. Der Rotor eines Synchronmotors kann auf zwei Arten konstruiert werden:

- Permanentmagneten: Bei dieser befinden sich Permanentmagnete entweder auf der Oberfläche oder eingebettet im Rotor. Diese Konfiguration erzeugt ein konstantes Magnetfeld, ohne dass eine externe Stromquelle benötigt wird. Die Permanentmagnete folgen dem rotierenden Statorfeld synchron.
- Mit Elektromagneten (Erregungswicklung): Bei dieser Bauart besitzt der Rotor Wicklungen, die über Schleifringe und Bürsten oder einer bürstenlosen Erregermaschine mit Gleichstrom versorgt werden. Durch diese Erregung entsteht ein Magnetfeld, dessen Stärke durch das Verändern des Erregerstromes angepasst werden kann. Das ermöglicht eine präzise Steuerung der Motorleistung. [2]

3.2 Strommessung

Der Stromfluss durch die Statorwicklungen ist eine relevante Regelgröße für den Betrieb von Dreiphasenmotoren, die in vektor- oder feldorientierten Steuerungsverfahren benötigt wird (siehe Abbildung 3).

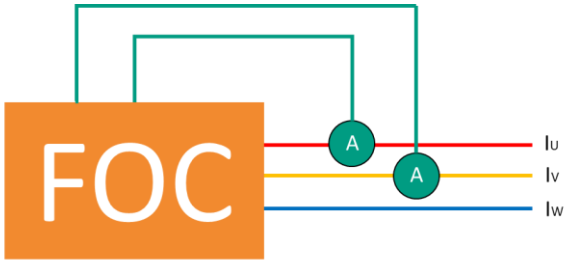

Abbildung 3: Phasenstrom als Parameter einer FOC

Solche Steuerungstechniken nutzen die gewonnenen Strommesswerte, um das Magnetfeld, innerhalb des Motors, anzupassen. Dadurch wird eine ideale Leistungsabgabe des Motors unter verschiedenen Lastbedingungen ermöglicht. Durch die Analyse der Strommessdaten können ineffiziente Betriebszustände identifiziert werden, so dass z.B. die FOC, entsprechende Korrekturen vornehmen kann. Dafür benötigt die FOC eine hochauflösende Repräsentation der Phasenströme, um selbst kleinste Änderungen erkennen zu können. [3]

Die kontinuierliche Überwachung des Stromflusses dient zudem der frühzeitigen Erkennung potenzieller Fehler und Gefahren. Dadurch können präventive Maßnahmen zur Vermeidung von Betriebsstörungen implementiert werden. Für die Umsetzung der Fehlerkennung wird allerdings eine deutlich schnellere Verarbeitung der Messdaten benötigt, um Schäden zu vermeiden. Eine Verringerung der Auflösung ist dabei irrelevant, da sich Fehlerfälle besonders durch große Änderungen zeigen.

3.3 Sigma-Delta-Modulation

Das Kapitel beschäftigt sich mit der Sigma-Delta-Modulation, einer Technik der digitalen Signalverarbeitung, die bei der Umwandlung von analogen in digitale Signale verwendet wird. Diese Technik ermöglicht eine hohe Auflösung und Genauigkeit über einen weiten Dynamikbereich, indem sie das Prinzip des Oversamplings und des Noise Shapings nutzt. Besonders zeichnet sich der Modulator dadurch aus, dass er dafür nur ein Bit benötigt. Das macht ihn ressourcen- und platzsparend, insbesondere bei einer galvanischen Trennung.

Abbildung 4 zeigt den Aufbau eines solchen Modulators. Auf der linken Seite befindet sich das eingehende Signal, das mit einer Frequenz abgetastet wird, die deutlich höher ist als die Nyquist-Frequenz des Eingangssignals.

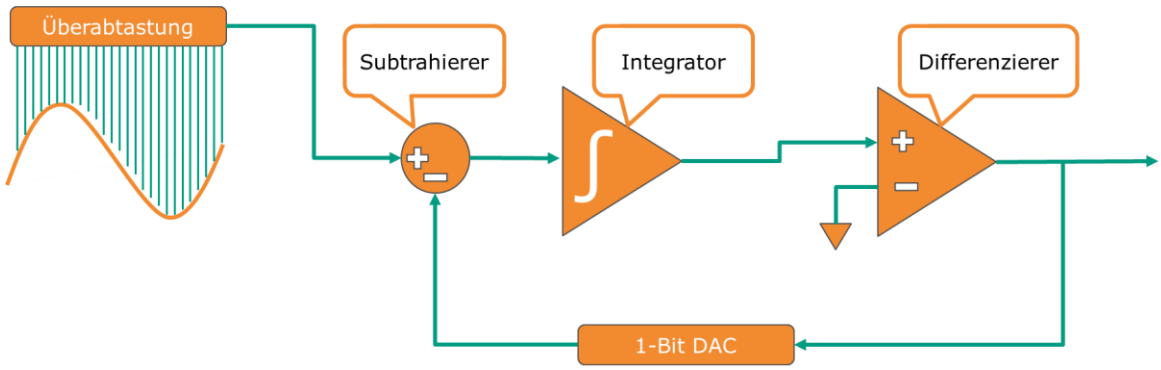

Abbildung 4: Blockschaltbild eines Sigma-Delta-Modulators erster Ordnung

3.3.1 Nyquist-Frequenz

Die Nyquist-Frequenz ist ein Begriff aus der Informationstheorie und Signalverarbeitung. Sie ist entscheidend, um ein Signal mit möglichst geringem Informationsverlust digital rekonstruieren zu können. Gemäß dem Nyquist-Shannon-Abtasttheorem muss ein kontinuierliches Signal mindestens mit der doppelten Höchstfrequenz abgetastet werden, die im Signal enthalten ist. Folglich ist die höchste sinnvoll darstellbare Frequenz die Hälfte der Nyquist-Frequenz (siehe Abbildung 5). [4]

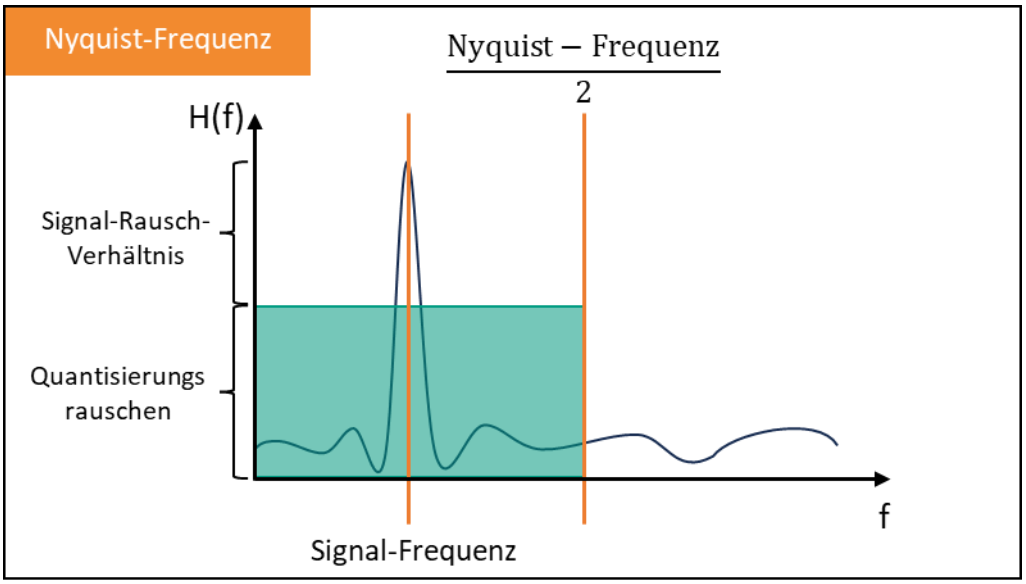

Abbildung 5: Darstellung des Quantisierungsrauschens bei der Nutzung der Nyquist-Frequenz

Wenn die Abtastfrequenz kleiner als das Doppelte der höchsten Signal-Frequenz ist, kann es zu Aliasing kommen. Dadurch werden hohe Frequenzen im digitalen Signal als niedrigere Frequenzen fehlinterpretiert, was zu Informationsverlust führt (Siehe Abbildung 6). [4]

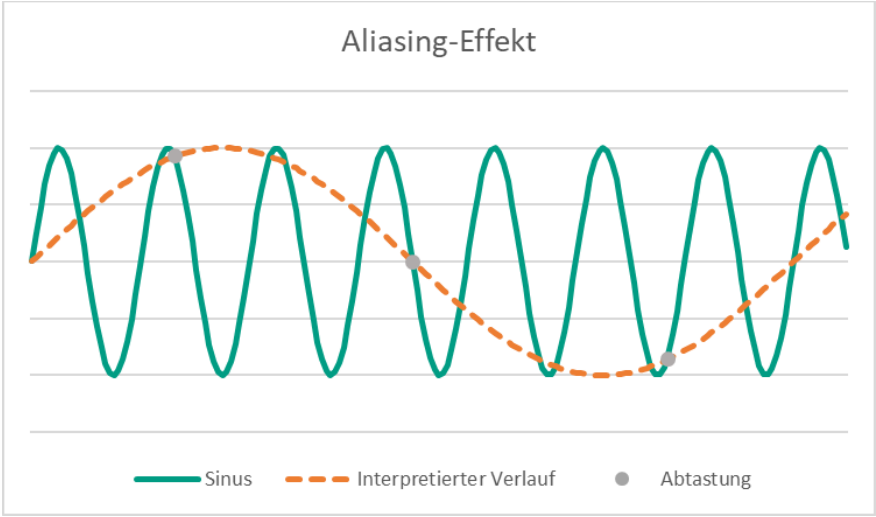

Abbildung 6: Beispiel einer Fehlinterpretation durch eine zu geringe Abtastrate

Abbildung 7 gibt einen genaueren Einblick in das Frequenzspektrum des abzutastenden Signals und die Auswirkung des Aliasing-Effekt auf dieses. In der oberen Hälfte der Abbildung ist die Abtastung unter Berücksichtigung des Nyquist-Shannon-Abtasttheorem zu sehen. Die dort abgebildeten Spiegelfrequenzen liegen am Rand des Signal-Frequenzbereiches an und können durch die Verwendung eines Tiefpassfilters mit möglichst steiler Filterflanke entfernt werden. [4]

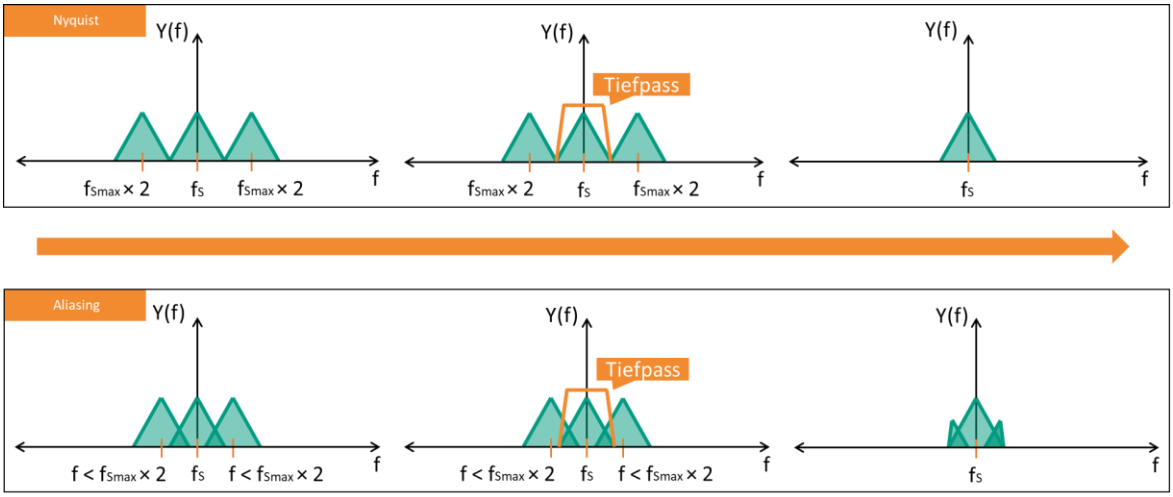

Abbildung 7: Frequenzspektrum des abzutastenden Signals

Wird das Abtasttheorem nicht eingehalten, kommt es zu einer Überlappung der Frequenzen. Die Überlappungen bleiben, selbst nach der Filterung erhalten, was eine präzise Rekonstruktion, in höheren Frequenzbereichen des Signals unmöglich machen kann, da die ursprünglichen Daten an dieser Stelle durch die Überlappung verfälscht werden. [4]

Durch Überabtastung wird nicht nur das Nyquist-Shannon-Abtasttheorem erfüllt, sondern es kann auch ein Teil des Quantisierungsrauschens aus dem Nutzbereich gestreckt werden.

3.3.2 Quantisierungsrauschen

Quantisierungsrauschen entsteht, wenn ein analoges Signal in ein digitales Signal umgewandelt wird. Bei der Quantisierung wird der kontinuierliche Wertebereich des analogen Signals in eine begrenzte Anzahl von diskreten Stufen unterteilt. Da diese Stufen nicht unendlich fein sind, kann das digitale Signal das analoge Original nicht perfekt wiedergeben. Der Unterschied zwischen dem tatsächlichen analogen Wert und dem nächstgelegenen digitalen Wert wird als Quantisierungsfehler bezeichnet. Die zufällige Verteilung dieser Fehler über das Signal zeigt sich als Quantisierungsrauschen. Der Abstand zwischen dem Noise-Floor, der sich durch das Quantisierungsrauchen ergibt, und der maximalen Amplitude des

Signals wird als Signal-Rausch-Verhältnis (SNR) bezeichnet und kann durch die Formel (1) in dB bestimmt werden, wobei n für die Auflösung in Bit steht. [5]

$$SNR = 6{,}02 \times n + 1{,}76dB \quad (1)\ \textbf{[6]}$$

Eine Erhöhung des SNR verbessert die Effektive Auflösung der digitalen Darstellung, durch die Erhöhung der Anzahl an Effektiven Bits (ENOB). ENOB beschreibt, wie viele der Bits eines ADCs oder DACs, unter der Berücksichtigung von Rauschen und Verzerrung, tatsächlich nützlich sind, um das Signal präzise zu digitalisieren oder zu rekonstruieren. [7]

3.3.3 Überabtastung

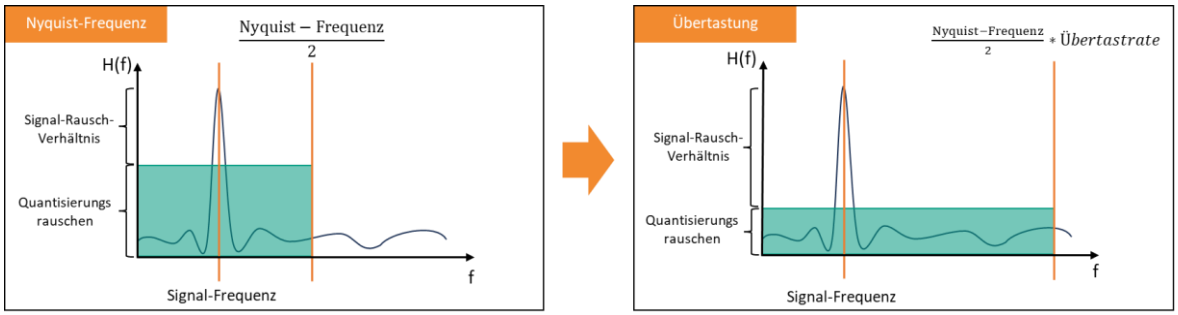

Abbildung 8: Effekt der Überabtastung auf das Quantisierungsrauschen

Abbildung 8 zeigt, wie der Sigma-Delta-Modulator durch die Überabtastung mit einer Frequenz, weit oberhalb der Nyquist-Frequenz, das Quantisierungsrauschens auf ein breiteres Frequenzspektrum streckt und somit den Signal-Rausch-Abstand erhöht. Zu dem Effekt kommt es, da die Erhöhung der Abtastrate eines digitalen Signals keinen Einfluss auf die Gesamtenergie des Quantisierungsrauschens hat. Durch die Erhöhung der Abtastfrequenz muss dieselbe Menge an Energie des Quantisierungsrauschens auf den gesamten neuen Frequenzbereich, der durch die Abtastrate bestimmt wird, gestreckt werden. Folglich kommt es zu einer Abnahme der Energiedichte des Rauschens in jedem Frequenzband, einschließlich des zu betrachtenden Frequenzbands. Durch diesen Effekt wird ein verbessertes Signal-Rausch-Verhältnis erzielt, welches durch die Formel (2) bestimmt werden kann. [8]

$$SNR = 6{,}02 \times n + 1{,}76dB + 10 \times log_{10}(\ddot{U}berabtastungsrate)dB \quad (2)\ \textbf{[6]}$$

Die Überabtastung beeinflusst nicht nur das Quantisierungsrauschen. Auch im Bereich des Aliasings kommt es zu Verbesserungen. Die Überabtastung erfüllt nicht nur das Nyquist-Shannon-Abtasttheorem, sondern schiebt das Auftreten der Spiegelfrequenz weiter vom ursprünglichen Frequenzbereich des Signals weg (Siehe Abbildung 9). Das ermöglicht die

Verwendung eines simpleren Tiefpasses, da dieser nun keine steile Filterflanke mehr benötigt. Zudem wirken sich Abweichungen in den Filtereigenschaften weniger auf das Endergebnis aus.

Abbildung 9: Auswirkung des Überabtastens auf das Frequenzspektrum des Signals

3.3.4 Subtrahierer

Das überabgetastete Signal durchläuft als erstes den, in Abbildung 4 zu sehenden, Subtrahierer. Hier wird die Abweichung zwischen dem eingehenden Signal und einem rückgeführten Wert ermittelt. Der zurückgeführte Wert entspricht dabei dem Ausgangssignal des Modulators, das durch die Nutzung eines 1-Bit-Digital-Analog-Wandlers in eine analoge Form umgewandelt wurde. Die resultierende Differenz beschreibt den Fehler zwischen dem aktuellen Eingangssignalwert und der zuletzt generierten Ausgabe des Modulators. Die Fehlerdifferenz wird anschließend an den Integrator weitergeleitet.

3.3.5 Integrator

Bei dem Integrator handelt es sich um die Sigma-Komponente des Sigma-Delta-Modulators. In der Mathematik steht das Symbol „Σ" (Groß-Sigma) für Summation. Dieser Prozess nutzt die Summierung um die Differenz zwischen dem Eingangssignal und dem zurückgeführten Signal über die Zeit zu Integrieren. Durch die kontinuierliche Summierung des Fehlers und seine Rückführung in den Modulationsprozess wird das Quantisierungsrauschen bei niedrigeren Frequenzen reduziert, wo sich üblicherweise das Nutzsignal befindet. Dafür nimmt jedoch das Rauschen bei höheren Frequenzen zu. [7]

Durch Formel (3), welche die Übertagungsfunktion eines Modulators erster Ordnung beschreibt, wird deutlich wie dieses Verhalten zustande kommt.

$$y(z) = 1 \times x(z) + (1 - z^{-1}) \times e(z) \qquad (3)\ \textbf{[6]}$$

Das Eingangssignal (x) ist in der Lage, das System unbeeinflusst zu durchlaufen, während das Rauschen (e) einen Hochpassfilter erster Ordnung passieren muss. [6]

Dadurch wird das Rauschen aus dem Nutzsignalbereich entfernt, was eine höhere Auflösung und bessere Signalqualität im relevanten Frequenzbereich ermöglicht (vgl. Abbildung 10).

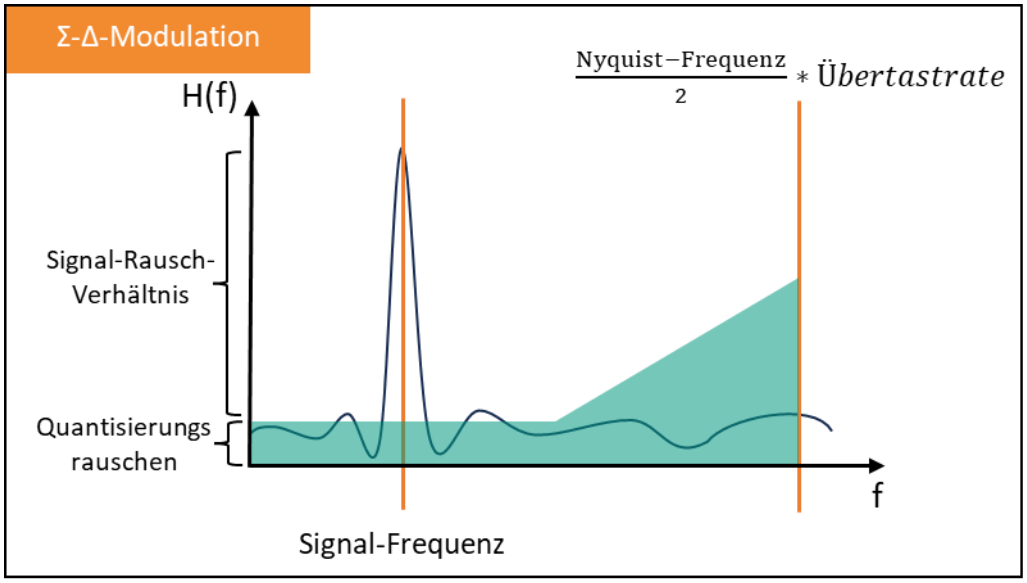

Abbildung 10: Einfluss auf das Quantisierungsrauschen bei der Verwendung von Überabtastung und Σ-Δ-Modulation

Im Rahmen dieser Arbeit wird ein Modulator zweiter Ordnung verwendet. Durch die Erhöhung der Ordnung wird eine weitere Integrator-Stufe ergänzt (vgl. Abbildung 11), was den Noise-Shaping-Effekt verstärkt (siehe Abbildung 12). [9]

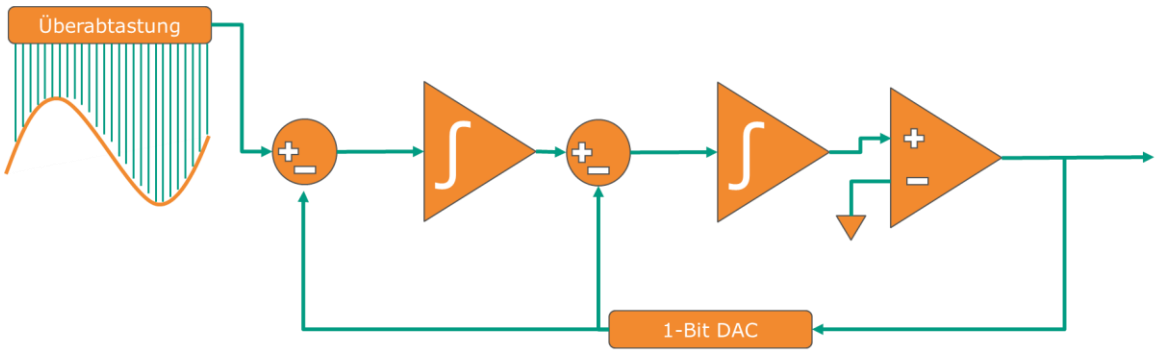

Abbildung 11: Sigma-Delta-Modulator der zweiten Ordnung

Durch das Erweitern um die Integrationsstufe kommt es zwar auf Seiten der Signalübertragung zu einer Verzögerung, doch das Rauschen durchläuft nun einen Hochpass-Differenzenfilter zweiter Ordnung (vgl. Formel (4)), was Grund für die für den verstärkten Effekt ist.

$$y(z) = z^{-1} \times x(z) + (1 - z^{-1})^2 \times e(z) \quad (4)$$ **[9]**

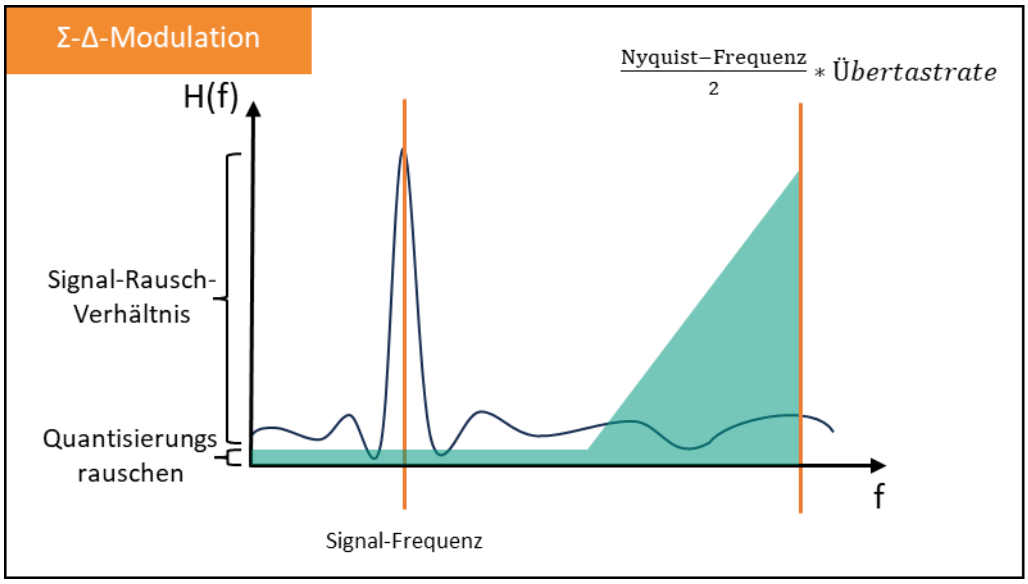

Abbildung 12: Einfluss auf das Quantisierungsrauschen bei der Verwendung von Überabtastung und Σ-Δ-Modulation 2.Ordnung

3.3.6 Komparator

Der Komparator bildet die letzte Stufe des Modulators. Ihre Aufgabe besteht darin, das integrierte Signal, das von der vorherigen Stufe des Modulators stammt, in ein digitales Signal umzuwandeln. Hierfür vergleicht der Komparator das integrierte analoge Signal direkt mit einem festgelegten Referenzwert. Auf Basis dieses Vergleichs generiert der Komparator ein binäres Ausgangssignal, das entweder den Zustand hoch (1) oder niedrig (0) annimmt. Die binäre Entscheidung gibt an, ob das integrierte Signal den Referenzwert über- oder unterschreitet. Dadurch wird die komplexe Information des analogen Signals auf eine einfache Form reduziert, die für die digitale Weiterverarbeitung geeignet ist. Außerdem erleichtert es die galvanische Trennung innerhalb eines Bauteils, da zur Übermittlung eines Bits nur ein Optokoppler benötigt wird, was ressourcen-, energie- und platzsparender ist als die Übermittlung ganzer Vektoren. [7]

3.4 Auswertung des Sigma-Delta-Modulator-Ausgangssignals

Da der Modulationsprozess ein hochfrequentes Signal mit einem breiten Spektrum an Quantisierungsrauschen erzeugt, ist eine Nachbearbeitung des Signals erforderlich. Durch die Verwendung eines Tiefpassfilters und eines Dezimators wird das Rauschen reduziert und die Abtastrate des Signals auf ein verarbeitbares Maß verringert. Die Schritte verbessern

nicht nur die Signalqualität, indem das Signal-Rausch-Verhältnis erhöht wird, sondern optimieren auch die effektive Auflösung des digitalisierten Signals. Dadurch kann das Signal an die spezifischen Anforderungen der Anwendung angepasst werden. [10]

3.4.1 Tiefpass

Durch das Noise-Shaping verschiebt der Sigma-Delta-Modulator das Quantisierungsrauschen in höhere Frequenzbereiche, um das Signal-Rausch-Verhältnis im Nutzsignalbereich zu optimieren. Das Ausgangssignal enthält jedoch trotz dieser Verbesserung hochfrequentes Rauschen, welches durch den Tiefpassfilter effizient entfernt wird (siehe Abbildung 13).

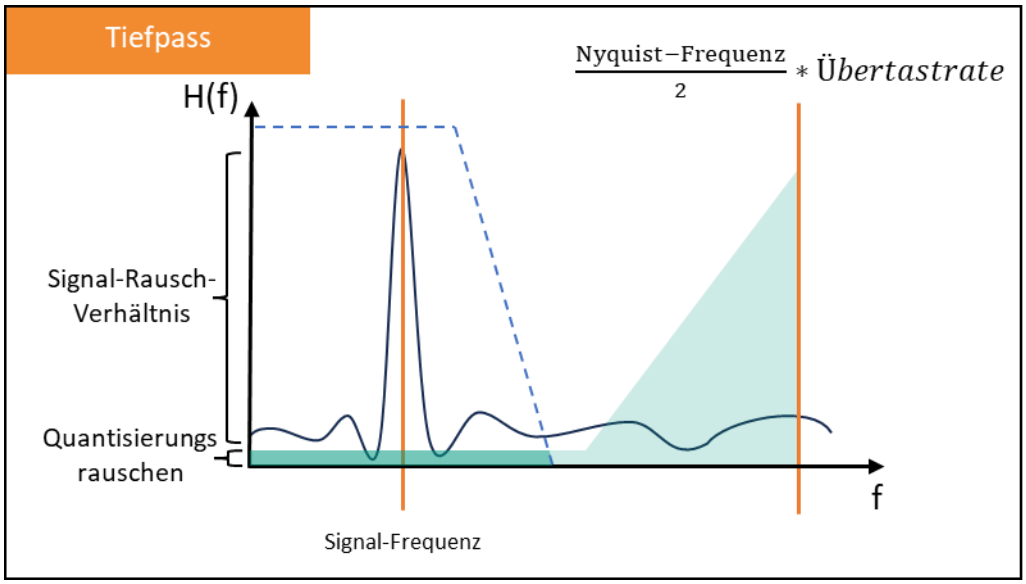

Abbildung 13: Einfluss auf das Quantisierungsrauschen bei der Verwendung von Überabtastung, Σ-Δ-Modulation 2.Ordnung und eines Tiefpass-Filters

Dieser Schritt ist notwendig, um zu vermeiden, dass es zu einer Faltung des hochfrequenten Rauschens in den Nutzbereich kommt. Zusätzlich können so die Bandbreitenanforderungen für die Anwendungen erfüllt werden und die in Kapitel 3.3.1 erwähnten Spiegelfrequenzen, verantwortlich für Aliasing-Effekte, entfernt werden.

Ein weiterer wesentlicher Vorteil des Tiefpassfilters liegt in seiner Fähigkeit, die Signalwiedergabe zu glätten. Diese Glättung führt zu einer nahezu originalgetreuen Wiedergabe des Eingangssignals, ohne die Artefakte, die durch das digitale Abtasten entstanden sind. Die Wirkung des Tiefpassfilters ist daher besonders wichtig, wenn das digitale Signal in ein analoges Signal umgewandelt werden soll. Der Filter agiert nicht nur als Rauschunterdrücker, sondern ist auch ein wesentlicher Bestandteil des Rekonstruktionsprozesses. Durch seine

Tiefpasseigenschaft trägt er dazu bei, ein Signal zu erzeugen, das dem ursprünglichen analogen Eingang so nah wie möglich kommt. [8]

3.4.2 Dezimierung

Die Dezimierungsstufe, die direkt nach dem Tiefpass folgt (siehe Abbildung 14), hat die Aufgabe, die sehr hohe Abtastrate des, aus der Modulation resultierenden, Bitstroms auf eine verwendbare Rate zu reduzieren, was die Signalverarbeitung vereinfacht. Der Tiefpass ist dafür besonders wichtig, da sich durch die Dezimierung eine geringere Abtastrate als die des überabgetasteten Signals ergibt. Um auch mit dieser mindestens das Nyquist-Shannon-Abtasttheorem zu erfüllen, dämpft der Tiefpass vorab alle Frequenzen oberhalb der neuen, durch die geringere Abtastfrequenz entstehende, Nyquist-Frequenz.

Abbildung 14: Aufbau eines Sigma-Delta-ADC

3.5 Serial Peripheral Interface (SPI)

Im Rahmen dieser Arbeit wird ein Modul verwendet, das einen SPI-Bus besitzt. Dabei handelt es sich um ein Kommunikationsprotokoll, das für die Kommunikation in eingebetteten Systemen verwendet wird, um beispielsweise Sensoren oder Peripheriegeräte an einen Mikrocontroller anzubinden. Obwohl das SPI mehrere Slaves unterstützt und einfach zu implementieren ist, benötigt es mehr Leitungen als andere serielle Protokolle und ist nicht durch ein standardisiertes Protokoll zur Fehlererkennung vor Störungen geschützt. Daher eignet es sich besonders für kurze Kommunikationsstrecken. Das SPI nutzt für die Kommunikation zwischen Master und Slaves vier Leitungen:

- MOSI: Diese ist die „Master Out Slave In“ Leitung und zuständig für die Übermittlung der Daten vom Master zum Slave.
- MISO: Ist das Gegenstück zu MOSI und bedeutet „Master In Slave Out“. Diese Leitung ermöglicht die Kommunikation vom Slave zum Master.
- SCK: Da es sich bei SPI um ein synchrones Protokoll handelt, benötigt sowohl der Master als auch die angebundenen Slaves einen gemeinsamen Takt. Dieser wird über diese Leitung mit dem Namen „Serial Clock“ übermittelt.

- ~CS: Steht für „Chip Select“ und übermittelt die Information, ob die Daten für den Slave (Chip) bestimmt sind. Jeder neue Slave braucht seine eigene Anbindung an den Master, was die Größe des Systems auf die Anzahl der IOs des Masters beschränkt.

Für die Steuerung eines Moduls, bei der nur eine unidirektionale Kommunikation benötigt wird, kann entweder die MISO- oder MOSI-Leitung weggelassen werden. Diese Form des Drei-Kabel-SPI ist üblich bei z.B. Sensoren, die nur Werte ausgeben müssen. Durch das Weglassen der MOSI-Leitung können Ressourcen gespart werden. [11]

Um den Vorschriften des SPI-Protokolls zu entsprechen, muss jeder Sendeprozess eine bestimmte Abfolge einhalten, um als solcher erfolgreich wahrgenommen zu werden. Abbildung 15 zeigt diese Abfolge für die 32-Bit SPI-Kommunikation mit dem in diesem Projekt verwendeten Modul.

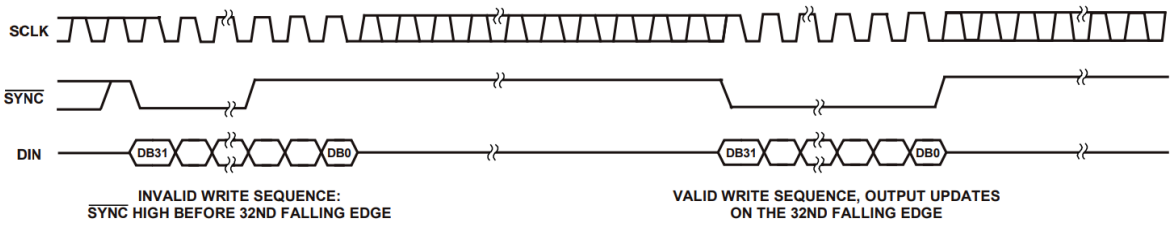

Abbildung 15: Ungültiger und Gültiger Schreibvorgang in einer 32 Bit SPI-Komunikation [12]

Das Modul nutzt das Drei-Kabel-SPI mit nur einem MOSI. Die MOSI-Leitung wird als „DIN“ bezeichnet und die ~CS-Leitung als „~SYNC“, welche Low-aktiv ist. Um einen Schreibvorgang zu starten, muss das ~SYNC-Signal mit der positiven Taktflanke von High auf Low wechseln. Ab dieser steigenden Taktflanke müssen auch die entsprechenden Datenbits gesendet werden. Bei der ersten fallenden Taktflanke wird das erste Bit vom Slave eingelesen. Durch das Schreiben bei steigenden und Lesen bei fallenden Taktflanken wird ein möglichst stabiler Zustand des Daten-Bits zum Zeitpunkt des Lesens erreicht. Das ~SYNC-Signal wird während der gesamten Datenübertragung auf dem Low-Pegel gehalten. Wenn das letzte Daten-Bit gesendet wurde, wechselt es mit der ersten steigenden Taktflanke nach dem Daten-Bit wieder auf den High-Pegel, um den Schreibprozess abzuschließen. Für 32-Bit Daten werden also 33 Takte benötigt. Bei Nichteinhaltung dieses Ablaufs wertet der Slave den Input als ungültig und löscht alle bisher eingelesenen Daten.

4 Hardware

In diesem Kapitel wird die für das Projekt ausgewählte Hardware beschrieben. Es wird ein umfassender Überblick auf wesentlichen Hardwarekomponenten, einschließlich der Verschaltungen, Anschlüsse und Kommunikationsmodule, geboten. Außerdem wird das für den Betrieb notwendige Vorwissen und die Bedienung behandelt.

4.1 AD7403

Der AD7403 ist ein isolierter Sigma-Delta-Modulator zweiter Ordnung von Analog Devices. Abbildung 16 zeigt den Aufbau des Modulators. Auf der linken Seite sind die Eingänge für die zu messende Spannung dargestellt. Die Differenz zwischen diesen beiden Eingängen kann zwischen 250 mV und -250 mV variieren. [13]

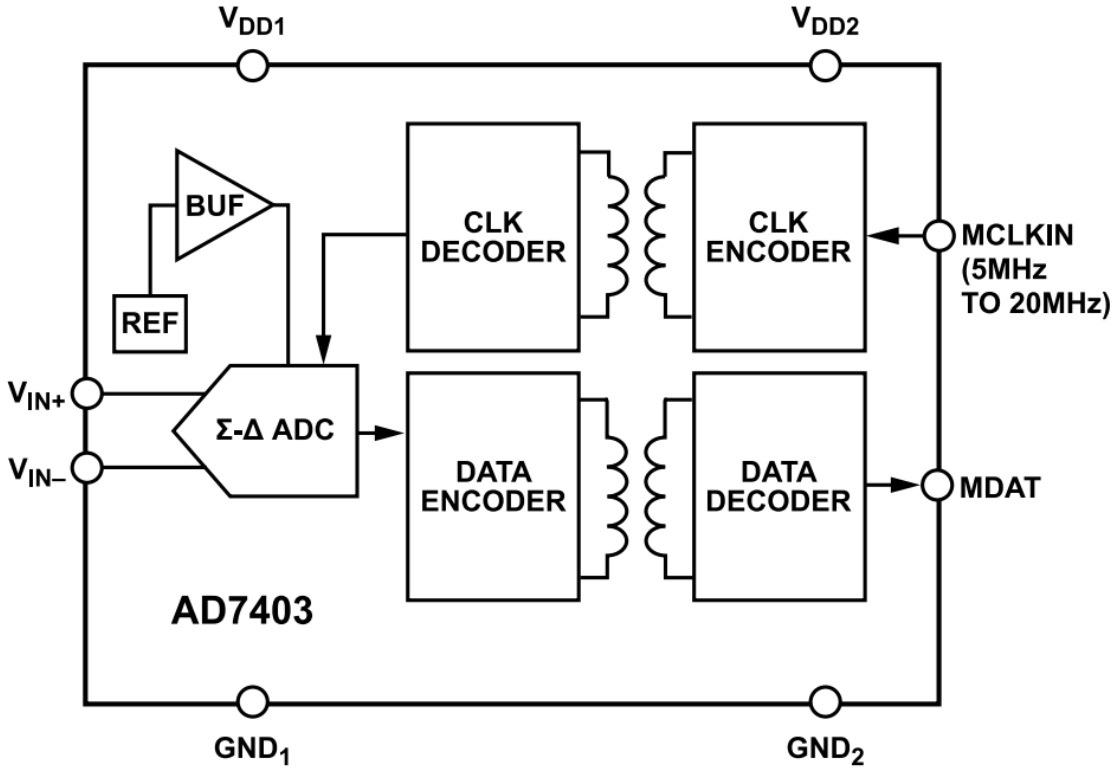

Abbildung 16: Block Diagramm des Sigma-Delta-Modulators [13]

Die Abtastrate für die Eingänge wird extern durch ein Taktsignal am Eingang „MCLKIN“ vorgegeben und kann zwischen 5 MHz und 20 MHz liegen. Um das Taktsignal von der zweiten höheren Spannung zu isolieren, wird ein digitaler Isolator eingesetzt. Dieser trennt die Motorseite, die eine hohe Spannung und große Ströme führt, von der Nutzerseite, welche bei

weniger Spannung betrieben wird, galvanisch. So wird das Unfallrisiko im Fehlerfall minimiert. Das Taktsignal wird daraufhin an den Sigma-Delta-Modulator weitergeleitet, welcher einen 1-Bit-Bitstream erzeugt. Dieser wird ebenfalls über einen digitalen Isolator übertragen. Der Differenzierer des Modulators nutzt eine interne Referenzspannung als Referenzspannung. [13]

Montiert ist das Bauteil auf dem EDPS-Board von Trenz Electronic.

4.2 TEC0053-04 EDPS Power Stage

Das Electronic Drive Power Stage (EDPS) Board von Trenz Electronic (vgl. Abbildung 17) ist für den Einsatz in Verbindung mit einem Controller-Board zur Testung von Motorsteuerungsanwendungen entworfen. Es bietet eine flexible Plattform für die Entwicklung und Testung von Antriebstechniken. Das Board eignet sich für eine Vielzahl von Motoranwendungen, da es bis zu 30 A bei 48 V verarbeiten kann. Es unterstützt 3-Phasen-BLDC-Motoren durch eine MOSFET-Leistungsstufe und ermöglicht die Strommessung auf zwei Phasen für eine Analyse und Steuerung des Motors und der Leistung. Es besteht auch die Option für eine 3-Phasen-Messung. [14]

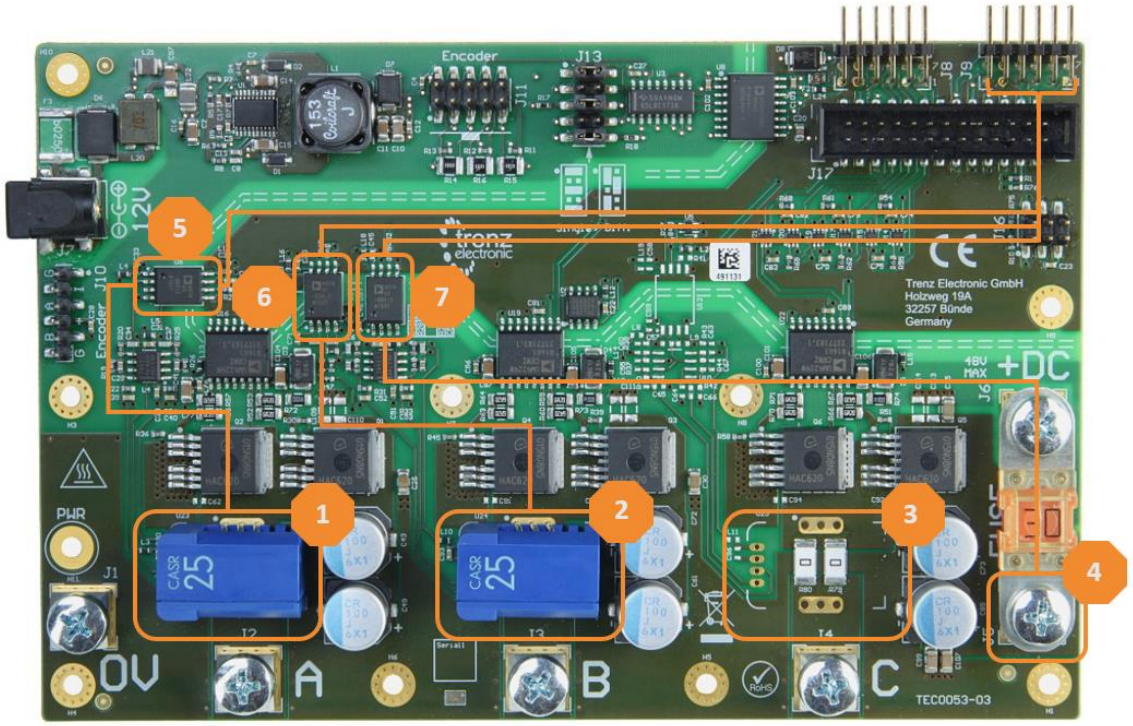

Abbildung 17: EDPS-Board [14]

Das Bauteil, das in Abbildung 17 mit den Markierungen eins und zwei versehen ist, ist ein LEM-Wandler, der für die Messung des Phasenstroms der Phasen A und B zuständig ist. Er wandelt den Motorstrom in eine Spannung zwischen 0 V und 5 V um. Wenn kein Strom durch die überwachte Leitung fließt, gibt er standardmäßig eine Spannung von 2,5 V aus, um den Nullpunkt darzustellen. Dies ist notwendig, da bei der Messung des Motorstroms Ströme in beide Richtungen fließen können. [15]

Zur Steuerung eines Motors sind lediglich zwei präzise Phasenstrommessungen ausreichend. Der dritte Strom wird durch die Differenz zwischen Null und der Addition der beiden gemessenen Phasenströmen dargestellt. Daher ist für I_W kein Wandler erforderlich und die Fläche drei in Abbildung 17 bleibt unbestückt.

Die Bitstreams, die durch die ADCs erzeugt werden, stehen dem Nutzer als Rohdaten am Anschluss J9 zur Verfügung. Die genaue Beschreibung des Anschlusses J9 ist in Abbildung 18 zu finden. Eine Übersicht auf die Pinbelegung von J9 ist in Abbildung 19 dargestellt.

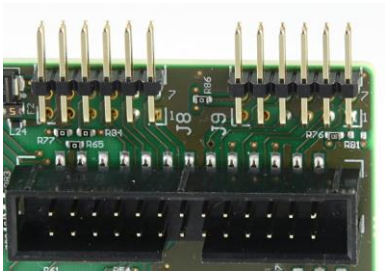

Abbildung 18: Anschlüsse des EDPS für das Arty Z7 [14]

Signal names	Connector J8	Connector J9	Connector J17
Supply Voltage ADC "raw" Signal from EDPS (usable with FPGA IP)		Pin 7: SDIV - from DC_LINK (Fused Motor Supply Voltage)	Pin 24: SDIV - from DC_LINK (Fused Motor Supply Voltage)
ADC Clock Signal to EDPS		Pin 1: SCLK	Pin 23: SCLK
Encoder Digital Signals from EDPS		Pin 8: ENC_A Pin 9: ENC_B Pin 10: ENC_I	Pin 20: ENC_A Pin 18: ENC_B Pin 16: ENC_I
Motor Current ADC "raw" Signals from EDPS (usable with FPGA IP)		Pin 2: SDI1 - Current Ch.A Pin 3: SDI2 - Current Ch.B Pin 4: SDI3 - Current Ch.C	Pin 19: SDI1 - Current Ch.A Pin 17: SDI2 - Current Ch.B Pin 15: SDI3 - Current Ch.C

Abbildung 19: Ausschnitt aus der Tabelle der Pinbelegungen des EDSP-Boards [14]

Gemäß des Kits, aus dem das EDPS-Board stammt, kann an den Anschlüssen J8 und J9 über Pmod-Header das Entwicklerboard Arty Z7 angeschlossen werden (siehe Abbildung 20).

4.3 Arty Z7 FPGA Board

Das Arty Z7 FPGA Board, welches in Abbildung 20 zu sehen ist, ist ein Entwicklerboard, das von Digilent herausgegeben wurde und auf dem Zynq-7000 System on Chip (SoC) von Xilinx basiert. Das SoC integriert zwei ARM-Cortex-A9 Prozessorkerne mit einer konfigurierbaren FPGA-Fabrik. Die Zwei-Kerne-Konfiguration ermöglicht eine effiziente Verarbeitung von Anwendungen, während der FPGA-Teil des SoC eine flexible Anpassung an spezifische Hardwareanforderungen erlaubt. Das Board besitzt verschiedene Schnittstellen und Anschlüsse, wie zum Beispiel Gigabit Ethernet, USB, HDMI und Pmod-Anschlüsse. Diese werden nicht nur genutzt, um mit dem EDPS-Board zu kommunizieren, sondern auch um den DAC auf dem Pmod-Modul zu bedienen. [16]

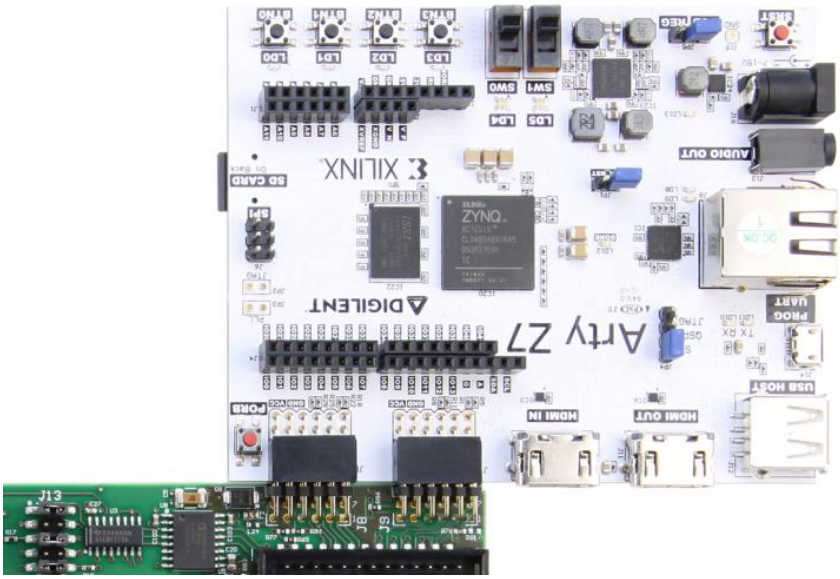

Abbildung 20: Arty Z7 angeschlossen an das EDPS [14]

4.4 Pmod-DA4

Das Pmod DA4 (Abbildung 21) ist ein Modul zur Digital-Analog-Wandlung. Es basiert auf dem AD5628 Chip von Analog Devices und verfügt über acht unabhängige 12-Bit-Digital-Analog-Wandler. Diese ermöglichen die Umwandlung von digitalen Signalen in acht separate analoge Ausgangssignale. Die 12-Bit-Auflösung jedes DACs bietet ausreichende Präzision für die Umsetzung der digitalen Eingangswerte in analoge Spannungen. Die Ausgangsspannungsbereiche dieser Signale werden durch die Versorgungsspannung des Moduls bestimmt, welche typischerweise zwischen 2,7 V und 5,5 V liegt. [12]

Abbildung 21: Pmod-DA4 [12]

Die Kommunikation mit dem Modul erfolgt über eine SPI-Schnittstelle, die mittels einer standardmäßigen 6-Pin-Pmod-Steckverbindung angebunden wird. Zum Betrieb des Moduls werden jedoch nur fünf der sechs Pins benötigt, wie Tabelle 1 zeigt. Es fehlt eine MISO-Verbindung, die zur Steuerung des Moduls nicht benötigt wird.

Tabelle 1: Pinbelegung des Pmod-DA4 [12]

Pin	Signal	Description
1	~CS	Chip Select
2	MOSI	Master-Out-Slave-In
3	(NC)	Not Connected
4	SCLK	Serial Clock
5	GND	Power Supply Ground
6	VCC	Power Supply (3.3V/5V)

Zur Ansteuerung des DA4 wird eine Serie von 32-Bit verwendet, welche sich, wie in Abbildung 22 zu sehen ist, aus insgesamt fünf Abschnitten zusammensetzt.

DB31 (MSB)				Command Bits				Address Bits			
X	X	X	X	C3	C2	C1	C0	A3	A2	A1	A0
Data Bits											
D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
DB0 (LSB)											
X	X	X	X	X	X	X	X				

Abbildung 22: Zusammensetzung einer 32-Bit Serie für das Ansteuern eines Pmod-DA4 [12]

Die ersten vier Bits werden als sogenannte „Don't care"-Bits bezeichnet, da ihr Inhalt keinen Einfluss auf die Konfiguration oder Steuerung des DACs hat. Sie sind dennoch erforderlich, um die 32-Bit-Kommunikation einzuhalten, die sich im Laufe der Zeit bei Pmod-Modulen etabliert hat.

Die darauffolgenden vier Bits werden als „Command"-Bits bezeichnet, da sie verwendet werden, um das DA4 zu konfigurieren. Alle möglichen Bit-Folgen sind in Abbildung 23 dargestellt.

Command				
C3	C2	C1	C0	Description
0	0	0	0	Write to Input Register n
0	0	0	1	Update DAC Register n
0	0	1	0	Write to Input Register n, update all (software $\overline{\text{LDAC}}$)
0	0	1	1	Write to and update DAC Channel n
0	1	0	0	Power down/power up DAC
0	1	0	1	Load clear code register
0	1	1	0	Load $\overline{\text{LDAC}}$ register
0	1	1	1	Reset (power-on reset)
1	0	0	0	Set up internal REF register
1	0	0	1	Reserved
–	–	–	–	Reserved
1	1	1	1	Reserved

Abbildung 23: Command-Bit Konfigurationen zum Einstellen des DA4 [12]

Jeder der acht DACs auf dem Pmod DA4 kann unabhängig angesteuert werden. Dadurch ist es möglich, acht separate analoge Ausgangssignale gleichzeitig zu erzeugen. Zur Zuordnung der Daten zu den DACs werden die „Address"-Bits benötigt (siehe Abbildung 24).

Address (n)				
A3	A2	A1	A0	Selected DAC Channel
0	0	0	0	DAC A
0	0	0	1	DAC B
0	0	1	0	DAC C
0	0	1	1	DAC D
0	1	0	0	DAC E
0	1	0	1	DAC F
0	1	1	0	DAC G
0	1	1	1	DAC H
1	1	1	1	All DACs

Abbildung 24: Adress-Bit-Kombinationen für das separate und gemeinsame Ansteuern der DACs [12]

Nach den ersten 12 Bits folgen die 12 „Daten"-Bits. Hier wird ein Wert zwischen null und 4095 in binärer Schreibweise eingefügt, der die Spannung am Ausgang des DACs zwischen 0 V und der Versorgungsspannung festlegt.

Die letzten 12 Bits sind, außer in einer Situation, erneut „Don't care"-Bits. Die Ausnahme bildet in diesem Fall die Command-Bit-Kombination „1000", welche den AD5628 während der Verwendung veranlasst, das Bit an der Position 0 als Einstellung für die Referenzspannung zu übernehmen. Standardmäßig ist der AD5628 so eingestellt, dass er eine externe Referenzspannung verwendet. Auf dem DA4 existiert jedoch keine externe Referenzspannung. Daher ist es bei jeder Inbetriebnahme notwendig, die Einstellung durch den Wechsel des letzten Bits von 0 auf 1 auf die interne Referenz umzustellen. [13]

5 Aufbau

In diesem Kapitel wird der Aufbau und die Funktionsweise der Tiefpass- und Dezimator-Module beschrieben, die in VHDL realisiert wurden, um den Bitstream aus dem Sigma-Delta-Modulator auszuwerten. Außerdem werden das, für die Kommunikation verwendete SPI-Modul und die zur Überprüfung notwendigen Testbenches beschrieben.

5.1 Filterstruktur

Für die Auswertung des Bitstreams, der durch den AD7403 erzeugt wird, empfiehlt der Hersteller die Verwendung eines Sinc-Filters. Dieser Tiefpassfilter basiert auf der mathematischen Funktion des Sinus Cardinales. Zur Beschreibung des Filters wird die Übertragungsfunktion in der z-Domäne verwendet, wie in Formel (5) dargestellt. Dabei steht M für die im Filter integrierte Dezimierungsrate und K für die Filterordnung.

$$H[z] = \left(\frac{1}{M} \times \frac{1 - z^{-M}}{1 - z^{-1}}\right)^{K} \quad (5)$$ **[17]**

Der SINC-Filter eignet sich besonders für die Hardwareimplementierung und somit auch für die Auswertung des Bitstreams im FPGA. Im Gegensatz zu anderen Filtern benötigt er keine Multiplikation, was das Design vereinfacht, und Kosten spart. Multiplikatoren beanspruchen mehr Ressourcen als die für die Integration und Differenzierung benötigten Addierer und Subtrahierer.

Ein Filter, dessen Ordnung die des Sigma-Delta-Modulators übersteigt, bietet eine effektivere Unterdrückung der Frequenzen, die durch das Noise Shaping des Modulators in höhere Frequenzbereiche verlagert wurden. Der Filter muss diese Frequenzen dämpfen können, bevor sie durch den Prozess der Dezimierung potenziell ins Basisband gefaltet werden könnten, was Aliasing zur Folge hätte. Da der AD7403 ein Modulator zweiter Ordnung ist, bedeutet das, dass der SINC-Filter mindestens dritter Ordnung sein muss. [17]

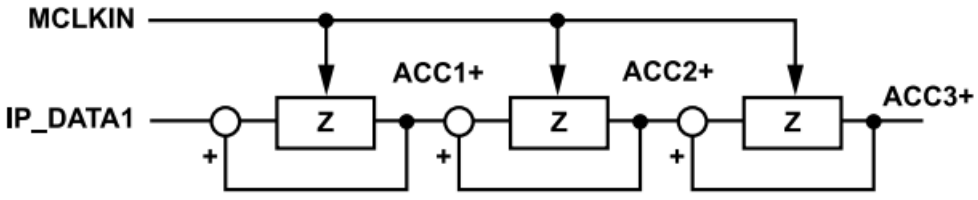

Abbildung 25: Aufbau der Integratoren des SINC-Filters dritter Ordnung [13]

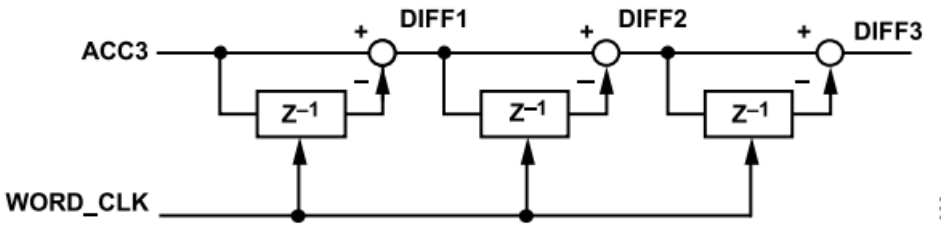

Abbildung 26: Aufbau der Differenzierer des SINC-Filters dritter Ordnung [13]

Zur Implementation des Filters reichen drei Stufen mit Integratoren (Siehe Abbildung 25) und drei Stufen mit Differenzierern (Siehe Abbildung 26) aus. Bei den mit „Z" beschrifteten Komponenten handelt es sich um Flip-Flops die als Verzögerungsglieder wirken [6].

Durch das Einfügen des, in Abbildung 31 zu sehenden, Flip-Flops zwischen den Integrator- und Differenzierer-Stufen ist es möglich, die beiden Hälften des Filters mit unterschiedlichen Frequenzen zu betreiben. Dadurch wird eine Dezimierung innerhalb des Filters ermöglicht. Diese ist notwendig, da die Daten bis zu diesem Zeitpunkt noch immer mit einer hohen Abtastrate weit oberhalb der Nyquistfrequenz in den Filter geleitet werden. Mittels des weiteren Flip-Flops kann die Abtastfrequenz direkt im Filter auf eine geringere Frequenz von 20 MHz/Dezimierungsrate verringert werden. Die Wahl der Dezimierungsrate hängt dabei vom Verwendungszweck ab, da sie die Wiedergabequalität des Signals beeinflusst. Eine höhere Dezimierungsrate nutzt mehr Integrierte Samples, was die Auflösung verbessert, aber gleichzeitig die Reaktionszeit des Filters verschlechtert. Denn eine höhere Anzahl an integrierten Samples repräsentiert den gemessenen Wert genauer, da mehr Daten verwendet werden, um das Signal an der Stelle zu beschreiben. Zusätzlich nimmt dadurch das Quantisierungsrauschen durch eine feinere Auflösung ab, was das SNR und folglich auch ENOB erhöht. Da neue Samples jedoch mit der Frequenz der Abtastrate zur Verfügung gestellt werden, erhöht sich gleichzeitig die benötigte Zeit um den gleichen Faktor. Für diese Arbeit werden drei verschiedene Auflösungsstufen benötigt (Siehe Kapitel 2.2).

5.1.1 Dezimierungsraten

Um die Dezimierungsrate für die verschiedenen Auflösungen festzulegen, ist es wichtig, die Anforderungen an die Reaktionszeit zu berücksichtigen. Bei der Kurzschlussabschaltung ist eine möglichst schnelle Reaktionszeit erforderlich. Die Auflösung kann dabei vernachlässigt werden, da der Strom im Falle eines Kurzschlusses so schnell ansteigt, dass die Situation auch mit einer schlechten Auflösung erkannt werden kann. Für den Kurzschlussfall wird

empfohlen, die Abschaltzeit so schnell wie möglich, jedoch spätestens nach 3 µs, einzustellen, um mögliche Schäden an der Hardware zu vermeiden.

Der Modulator arbeitet mit einer Frequenz von 20 MHz, wodurch alle 0,05 µs ein Sample erstellt wird. Wenn beispielsweise 32 Samples verwendet werden, dauert es 1,6 µs, bis alle notwendigen Samples zur Repräsentation des Signals gesammelt sind (siehe Tabelle 2). Da es sich bei dem Filter jedoch um einen Filter der dritten Ordnung handelt, müssen die Daten dreimal durch den Filter propagieren, um am Ausgang des Filters korrekt repräsentiert zu werden. (Siehe Abbildung 27)

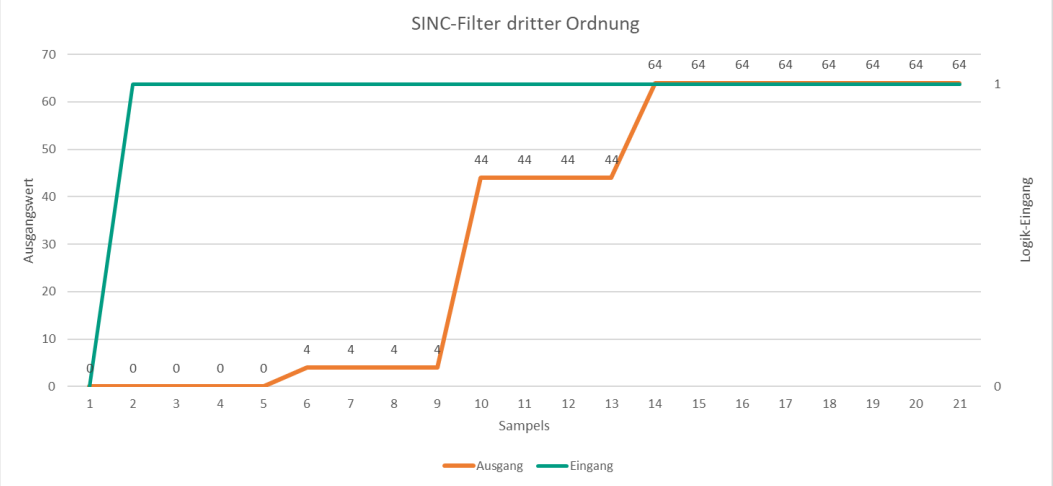

Abbildung 27: Verlauf einer an einem SINC-Filter anliegenden Sprung-Funktion bei einer Dezimierungsrate von 4 und einer Auflösung von 64

Aufgrund dessen erhöht sich die Dauer auf 4,8 µs, was für eine Kurzschlussabschaltung zu spät wäre. Es ist jedoch nicht bekannt, auf welcher Hardware die Abschaltung stattfinden soll und wie lange die Umsetzung des Abschaltens dauern wird. Daher empfiehlt es sich, die Dezimierung mit der schnellsten Erkennung, aber ausreichender Verstärkung zur Erkennung zu wählen. Um die Verstärkung des Filters in Abhängigkeit von der Dezimierungsstufe zu berechnen, wird die Gleichung aus Formel (2) verwendet. Dabei beschreibt M die Dezimierungsrate und n die Ordnung des Filters. Durch das Umrechnen der Verstärkung in dB mit der Formel (7) kann durch Formel (8) die Verstärkung in einer Menge an Bits dargestellt werden. Formel (8) verwendet dafür die logarithmische Eigenschaft von dB, da es bei einer Erhöhung von 6 dB zu einer Verdopplung des Wertes kommt. Eine Verdopplung wird, in der Bit-Darstellung, durch die Ergänzung eines weiteren Bits erzielt.

$$Gain = M^n \quad (6)\ \textbf{[8]}$$

$$Gain(dB) = 20 \times log_{10}(Gain) \quad (7)\ \textbf{[4]}$$

$$Bits = \frac{Gain(dB)}{6} \quad (8)$$

Tabelle 2: Auswirkung der Dezimierungsrate auf die Verarbeitungsdauer des Filters

M	Verstärkung	Verstärkung (Bits)	Dauer pro Filterstufe (µs)	Gesamtdauer (µs)
2	8	3	0,1	0,3
4	64	6	0,2	0,6
8	512	9	0,4	1,2
16	4096	12	0,8	2,4
32	32768	15	1,6	4,8
64	262144	18	3,2	9,6
128	2097152	21	6,4	19,2
256	16777216	24	12,8	38,4

Wie Tabelle 2 zeigt, erhöht sich die Auflösung, bei einer Verdopplung der Dezimierungsrate, immer nur um das Achtfache, während die Gesamtdauer exponentiell zunimmt (Siehe Abbildung 28). Das zeigt, dass sich für die Verarbeitung besonders geringe Dezimierungsraten eignen, da sie nicht nur weniger Ressourcen beanspruchen, sondern auch eine schnelle Verarbeitung bei guten Auflösungen ermöglichen. Eine übermäßige Erhöhung der Dezimierungsrate ist selbst für die Verwendung in der FOC ungeeignet, da der Zugewinn an Informationen schnell durch die Dauer überboten wird.

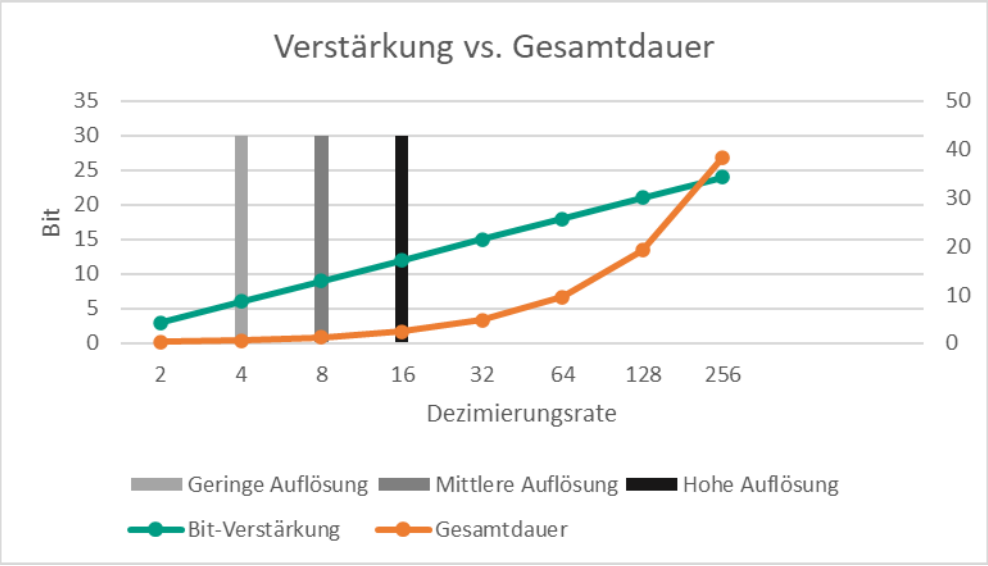

Abbildung 28: Verlauf der Verstärkung im Vergleich zur Erhöhung der Dauer

Tabelle 2 zeigt, dass eine Dezimierungsrate von vier eine Verstärkung von 64 besitzt und damit ausreichend ist, um einen Kurzschluss zu erkennen. Die Dauer von 0,6 µs ist kurz genug, um genug Spielraum für langsame Hardware zu bieten. Daher eignet sich eine Dezimierungsrate von vier für die Kurzschluss-Erkennung.

Im Vergleich zur Kurzschlusserkennung steht für die Überstromüberwachung mehr Zeit zur Verfügung. Ihr Ziel ist es, den Regelkreis am Überschwingen zu hindern. Da Schäden erst durch eine längere Überstrombelastung entstehen, kann eine höhere Auflösungsstufe gewählt werden. Eine Dezimierungsrate von acht bietet mit einer Verstärkung von 512 nicht nur eine achtmal höhere Verstärkung als die Kurzschlussüberwachung, sondern ist mit 1,2 µs auch schnell genug. In diesem Fall ist eine hohe Geschwindigkeit erwünscht, da der Filter später doppelt verwendet werden kann.

Für die Steuerung eines Motors mittels FOC wird jedoch eine deutlich höhere Auflösung benötigt. Die zeitliche Vorgabe ergibt sich dabei durch das pulsweitenmodulierte (PWM) Signal, das mit einer Frequenz von 16 kHz vorgegeben wird. Um mindestens einmal pro Periode messen zu können, muss die Datenverarbeitung spätestens nach 62,5 µs abgeschlossen sein.

Der Zeitpunkt dieser Messung muss sorgfältig gewählt werden, um sicherzustellen, dass interne Schaltvorgänge die Messung nicht beeinflussen. Während des Pegelwechsels der PWM treten normalerweise Überschwingungen auf. (Siehe Abbildung 29)

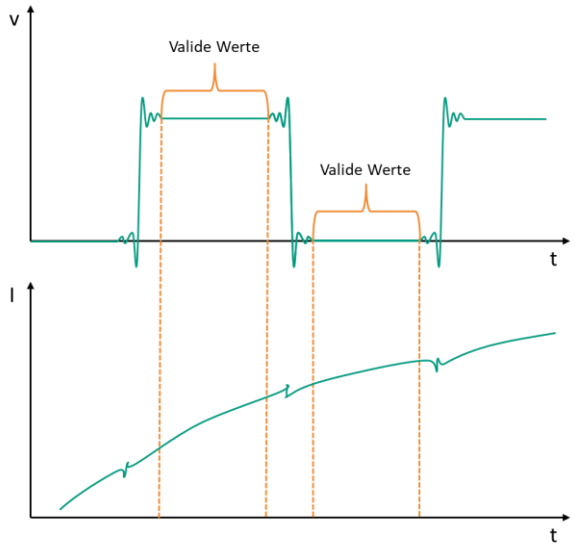

Abbildung 29: Vereinfachte Darstellung des Einflusses der PWM-Schaltvorgänge auf den gemessenen Strom

Um sicherzustellen, dass diese kurzen Spitzen bei der Strommessung nicht fehlinterpretiert werden, ist es wichtig, in einem schaltvorgangsfreien Bereich zu messen. Hierfür gibt es zwei Möglichkeiten (siehe Abbildung 30): Eine davon ist, den Filter schnell genug zu machen, sodass immer mindestens eine Messung im validen Bereich liegt. Dadurch stehen später deutlich mehr Daten zur Verfügung, die jedoch für die Verwendung in einer FOC von den ungültigen getrennt werden müssen. Eine zweite Option besteht darin, explizit nach dem Abklingen der Überschwingungen einmal zu messen. Hierfür benötigt der Filter jedoch ein externes Trigger-Signal, welches zu diesem Zeitpunkt nicht verfügbar ist. Daher wird in dieser Arbeit Option eins verfolgt.

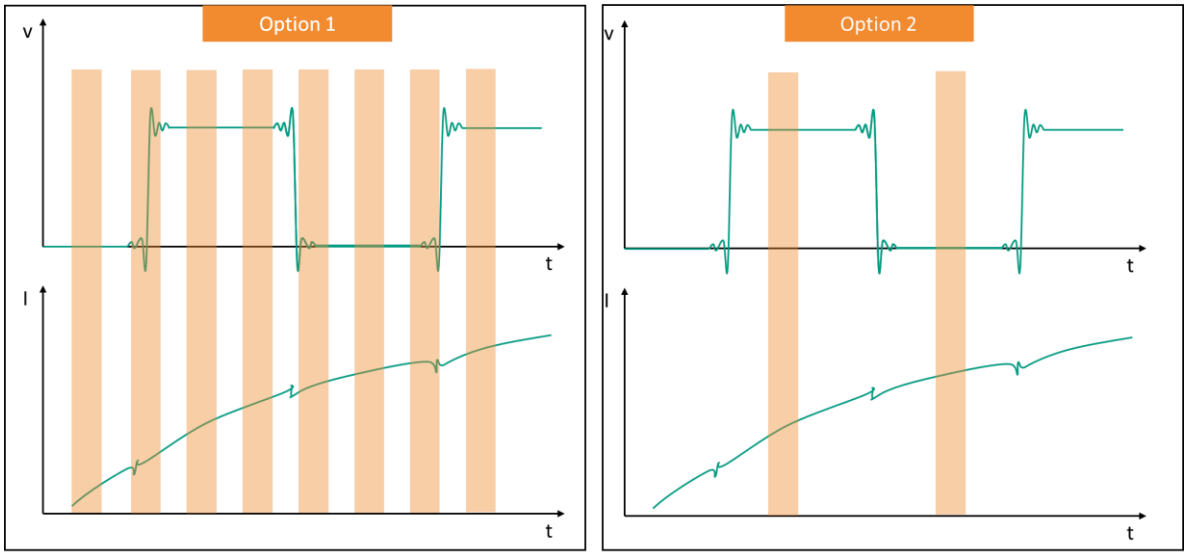

Abbildung 30: Optionen zur verzerrungsfreien Strommessung

Um sicherzustellen, dass ausreichend Messdaten zum Zeitpunkt der Verarbeitung vorhanden sind, muss der Filter mehrere Messungen innerhalb der 62,5 µs Periodendauer verarbeiten. Da eine Auflösung von 4096 Stufen ausreicht, um eine funktionstüchtige FOC zu betreiben, eignet sich eine Dezimierung von 16. Es ist jedoch zu beachten, dass dieser Filter doppelt verwendet werden soll, was zu einer zeitlichen Verdopplung der Abstände zwischen den Messpunkten führt. Dadurch liegen nur alle 4,8 µs neue Daten über die Messung am Ausgang des Filters an, was 13 Auswertungen pro Periode entspricht. Abbildung 30 zeigt die Doppelnutzung bereits in Aktion. Während der Orange markierten Bereiche wird einer der Phasenströme ausgewertet. Direkt im Anschluss geschieht dasselbe mit dem andern Phasenstrom in dem unmarkierten Bereich. Durch diesen schnellen Wechsel kann trotz der Doppelnutzung eine hohe Datenrate erreicht werden.

Um die Dezimierungsraten erfolgreich im Filter umzusetzen, ist es wichtig zu verstehen, dass dieser auf der Grundlage der Modulo-Arithmetik arbeitet. Dabei handelt es sich um eine Vorgehensweise, die das Überlaufen bei begrenzten Wortbreiten nutzt. Dadurch kann der Filter nicht nur effizienter realisiert werden, sondern es wird auch ein versehentliches Überschreiten der Hardwarekapazität durch schnell ansteigende Werte innerhalb der Integratoren verhindert. [17]

In Tabelle 3 wird die Modulo-Arithmetik veranschaulicht. Der Eingang des Filters wird mit einer Sprungfunktion angesteuert und als „ADCIN" bezeichnet. „ADDin"-„ADD2" sind die getakteten Signale der Addierer-Ausgänge, die in Abbildung 31 zu sehen sind, und „SUB12"-„SUB31" sind die Subtrahierer-Eingänge der Differenzierer. „ADCout" wird verwendet, um den Ausgang des Filters zu simulieren. Jeder Übergang in die nächste Zeile zeigt die Änderungen bei einer positiven Taktflanke. Die Bit-Breite des Filters wird dabei mithilfe der Formel (9) berechnet.

$$Busbreite_{Filter} = 1 + n \times \log_2 M \qquad (9)\ [17]$$

Daher lauten die Busbreiten für die Auflösungsstufen wie folgt:

- Kurzschlusserkennung: 7 Bit
- Überstrommessung: 10 Bit
- Strommessung für die FOC: 13 Bit

Bei diesem Filter mit einer Dezimierungsrate von vier ergibt sich so eine interne Busbreite von sieben Bits mit 127 als höchsten Wert. Wenn eine der Filterstufen 127 überschreitet, wird nur der Überlauf dargestellt. [17]

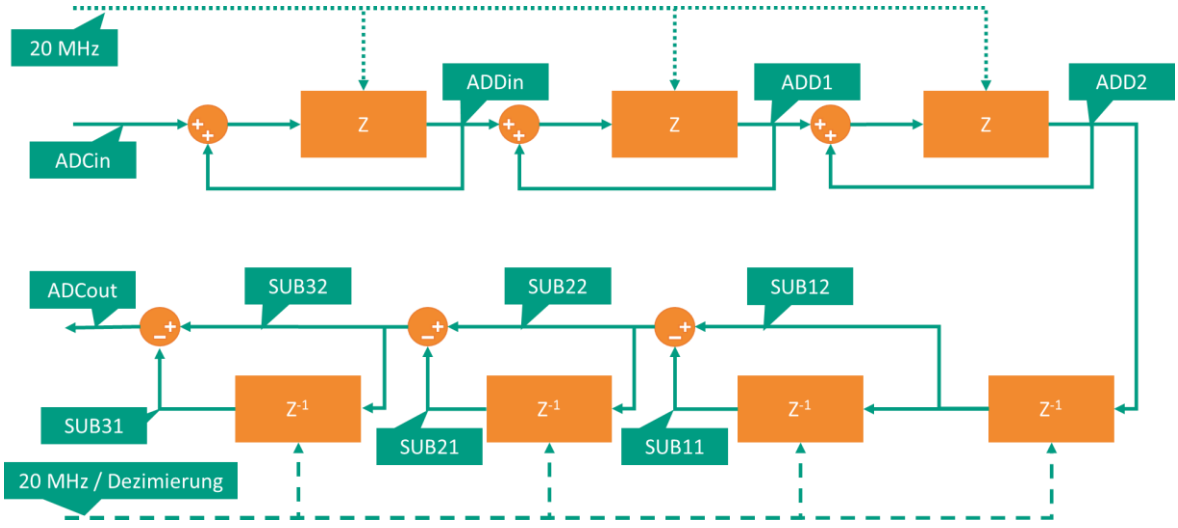

Abbildung 31: Blockdiagramm des SINC-Filters inklusive der Signalbenennung

Besonders gut lassen sich in Tabelle 3 die Dezimierung und die dreistufige Filterung betrachten. Die ersten Änderungen an den Subtrahierern finden vier Takte nach dem Eingang der ersten Eins statt. Mit einem Ausgangswert von vier weichen sie zu Beginn deutlich vom tatsächlichen Wert ab. Erst nach acht weiteren Takten und somit zwei weiteren Ausführungen der Differenzierungsstufe wird der Ausgangswert das gemessene Maximum repräsentieren. Dieses Maximum wird aufgrund der Modulo-Arithmetik trotz einer kontinuierlichen Erhöhung von „ADDin" nicht überschritten, sondern nach dem Erreichen konstant gehalten.

Tabelle 3: Verlauf der internen Signale eines SINC-Filters dritter Ordnung mit einer Dezimierungsstufe von vier bei einer Ansteuerung durch eine Sprung-Funktion.

ADCin	ADDin	ADD1	ADD2	SUB12	SUB11	SUB22	SUB21	SUB32	SUB31	ADCout
0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0
1	2	1	0	0	0	0	0	0	0	0
1	3	3	1	0	0	0	0	0	0	0
1	4	6	4	0	0	0	0	0	0	0
1	5	10	10	4	0	4	0	4	0	4
1	6	15	20	4	0	4	0	4	0	4
1	7	21	35	4	0	4	0	4	0	4
1	8	28	56	4	0	4	0	4	0	4
1	9	36	84	56	4	52	4	48	4	44
1	10	45	120	56	4	52	4	48	4	44
1	11	55	37	56	4	52	4	48	4	44
1	12	66	92	56	4	52	4	48	4	44
1	13	78	30	92	56	36	52	112	48	64
1	14	91	108	92	56	36	52	112	48	64
1	15	105	71	92	56	36	52	112	48	64
1	16	120	48	92	56	36	52	112	48	64
1	17	8	40	48	92	84	36	48	112	64
1	18	25	48	48	92	84	36	48	112	64
1	19	43	73	48	92	84	36	48	112	64
1	20	62	116	48	92	84	36	48	112	64

5.2 VHDL-Designs

Dieses Kapitel befasst sich mit der Umsetzung des Filters und aller anderen benötigten Module in VHDL.

5.2.1 Sinc-Filter

Um die Integratorstufe in der Hardware zu erzeugen, ist ein Prozess erforderlich. Dieser Prozess wird durch ein Ereignis bei einem der Signale in der Sensibilitätsliste ausgelöst: „clk20" als Clock-Signal oder „rst" für einen asynchronen Reset. Wenn ein solches Ereignis

auftritt, überprüft das Design, ob das „rst"-Signal derzeit ein Low-Signal (logische Null) ausgibt, wie in Listing 1 dargestellt.

```
process(clk20,rst)
begin
------------------------------------------  asyncroner Reset
    if (rst = '0') then
        ADDin <= "0000000";
        ADD1  <= "0000000";
        ADD2  <= "0000000";
        SUB11 <= "0000000";
        SUB12 <= "0000000";
        SUB21 <= "0000000";
        SUB31 <= "0000000";
        M <= "00";
```

Listing 1: Anfang des Prozesses und asynchroner Reset aus dem VDHL-Design für den SINC-Filter

Dadurch werden alle Signale des Prozesses auf einen vordefinierten Wert gesetzt. Diese Anwendung ermöglicht das asynchrone Zurücksetzen des Designs und wird in allen folgenden Modulen auf die gleiche Art und Weise umgesetzt. Wenn der Pegel des „rst"-Signals jedoch eine logische Eins repräsentiert, wird auf eine steigende Taktflanke des Clock-Signals geprüft (Siehe Listing 2).

```
 elsif rising_edge(clk20) then
---------------------------------------  Integrator
      if (ADCin = '1') then
          ADDin <= ADDin + 1;
      end if;

      ADD1 <= ADD1 + ADDin;
      ADD2 <= ADD2 + ADD1;
```

Listing 2: Beginn des Taktflanken gesteuerten Prozessabschnittes aus dem VDHL-Design für den SINC-Filter

An dieser Stelle beginnt die Umsetzung der ersten Integrationsstufe. Wie aus Tabelle 3 hervorgeht, genügt es für die erste Integrationsstufe, den Wert bei einer anliegenden Eins zu inkrementieren. Die if-Anweisung übernimmt diese Aufgabe. Bei einer Null bleibt „ADDin" unverändert. Direkt danach werden die beiden weiteren Integrierer kaskadiert. Diese sind direkt unterhalb der beendeten if-Anweisung zu finden. Die kombinatorische und getaktete Logik der Integratoren kann, zur besseren Übersichtlichkeit, in einem Prozess kombiniert werden. Es ist anzumerken, dass die Daten durch das Takten diskret und nicht kontinuierlich,

wie beim analogen Gegenstück, verarbeitet werden. Dies spielt jedoch keine Rolle, da die Daten des Modulators mit demselben Takt zur Verfügung gestellt werden.

Die Ausgänge der Integratoren werden entweder auf den Addierer des nächsten Integrators oder, wie im Falle des dritten, auf ein Flip-Flop geführt. Das Flip-Flop, das in Abbildung 25 zwischen „ADD2"und „SUB12" als Schalter fungiert, wird mit den Flip-Flops der Differenzierer im Prozess zusammengefasst (vgl. Listing 3).

```
------------------------------------------  Differenzierer Teil 1
            if (M = "11") then
                SUB12 <= ADD2;
                SUB11 <= SUB12;
                SUB21 <= SUB22;
                SUB31 <= SUB32;
            end if;
```

Listing 3: Flip-Flops des Schalters und der Differenzierer aus dem VDHL-Design für den SINC-Filter mit einer Dezimierung von vier

Für die Dezimierung müssen diese Flip-Flops seltener als die Integratoren ausgeführt werden. Hierfür wird das Signal „M" verwendet, das die Anzahl der durchgeführten Integrationen zählt und jedes vierte Mal die Flip-Flops des Schalters und der Differenzierer auslöst. „M" ist ein weiteres Beispiel für die Verwendung der Modulo-Arithmetik. Durch das Zählen bis zur binären drei und den darauffolgenden Überlauf auf null entstehen insgesamt vier Zustände, ohne dass eine Zurücksetzung des Zähler-Signals erforderlich ist. Im Filter bildet die Inkrementierung des Zählers auch die letzte Aktion im Prozess. Danach folgt lediglich die kombinatorische Logik, in diesem Fall die Subtraktion, der Differenzierer (Siehe Listing 4).

```
------------------------------------------  Counter Inkrementieren für das Dezimieren
            M <= M + 1;
        end if;
    end process;

------------------------------------------  Differenzier Teil 2
    SUB22  <= SUB12 - SUB11;
    SUB32  <= SUB22 - SUB21;
    ADCout <= SUB32 - SUB31;
```

Listing 4: Zählerinkrementierung und Kombinatorik des Filters.

Diese Struktur und Funktionsweise bleibt über die verschiedenen Filtergrößen hinweg identisch. Lediglich die Busbreiten und Zählervariablen müssen entsprechend angepasst werden.

5.2.2 Erweiterter SINC-Filter

Um einen Motor mittels FOC zu steuern, ist es notwendig, den Stromfluss aller drei Phasen zu ermitteln. Dazu genügt es, wie zuvor erwähnt, zwei der drei Ströme zu messen. Zur Auswertung der Messungen werden dementsprechend auch zwei Filter benötigt. Da die Filter für beide Ströme identisch wären, bietet es sich zur Ressourcenschonung an, einen Filter für beide Ströme zu verwenden. Um sicherzustellen, dass beide Messungen denselben Filter verwenden, müssen die Daten nacheinander durch den Filter geleitet werden. Dadurch verdoppelt sich der zeitliche Abstand zwischen den Messungen. Um dennoch die Zeitvorgaben einzuhalten, muss das Design schnell genug sein. Dies ist bei den Dezimierungsstufen acht und 16, wie in Kapitel 5.1.1 erläutert, gewährleistet. Der Filter zur Kurzschlusserkennung mit einer Dezimierungsrate von vier ist jedoch nicht für die Doppelverwendung geeignet. Dies liegt zum einen an den zeitlichen Anforderungen und zum anderen an dem dadurch entstehenden Datenverlust. Normalerweise kann bereits nach der ersten Differenziererausführung am Ausgang des Filters eine Abweichung vom vorherigen Datenpunkt vermutet werden (vgl. Tabelle 3). Bei der Doppelverwendung des Filters entfallen diese Informationen, da die vorherigen Werte innerhalb des Filters von der Verarbeitung der vorherigen Strommessung stammen. Aufgrund der dritten Ordnung des Filters entsprechen die Werte am Ausgang erst nach den vollen drei Ausführungen der Differenzierer der tatsächlichen Messung.

Um die Filter für die Doppelverwendung anzupassen, müssen die Ein- und Ausgänge des Filters zum richtigen Zeitpunkt auf verschiedene Anschlüsse geschaltet werden. Hierfür wird ein weiteres Zählsignal namens „changeCNT“ verwendet, dass bei jeder Ausführung der Differenziererstufe inkrementiert wird. Es wird erst nach sechs Dezimiererausführungen auf null gesetzt. Anhand dieses Signals kann festgestellt werden, welche Ein- und Ausgangsverschaltung verwendet werden soll und ob der aktuelle Input bereits dreimal durch den Filter ausgeführt wurde. (Siehe Abbildung 32).

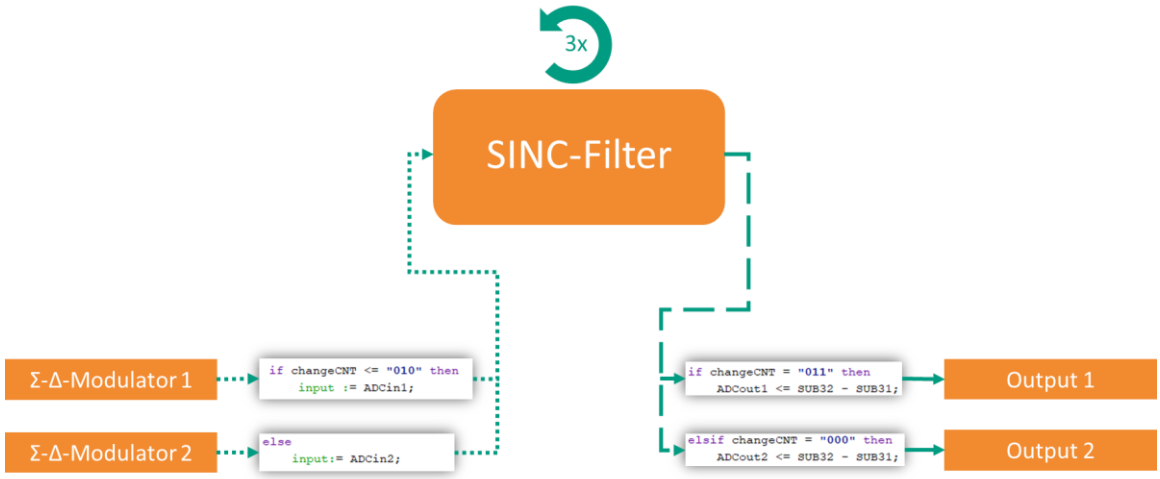

Abbildung 32: Verschaltung des SINC-Filters zur Doppelverwendung

Die Bitstreams der Modulatoren werden durch den Filter auf Variablen geführt, um eine sofortige Verwendung der Daten im Prozess zu ermöglichen. Dies ist notwendig, um die Daten rechtzeitig für die Differenzierung zur Verfügung zu haben. Wenn der Input ein Signal wäre, würde es immer einen Takt hinterherhängen und somit das Ergebnis verfälschen. Wenn der Input eines neuen Modulators vollständig integriert und an die Differenzierer weitergegeben wird, schiebt der Output des vorherigen Modulators einmalig die aktuellen Inhalte der Differenzier raus. Dadurch werden nicht verwertbare Filterausgaben übersprungen und es wird immer nur der vollständig verarbeitete Wert ausgegeben. Dieses Vorgehen wird sowohl beim Filter mit einer Dezimierungsrate von acht als auch beim Filter mit einer Dezimierungsrate von 16 ohne Änderungen verwendet.

5.2.3 Debounce-Module

Um Fehleingaben aufgrund von Schalterprellen zu vermeiden, wird der aktuelle Zustand des Reset-Tasters in einem Vektor im Debounce-Modul gespeichert. Hierfür wurde eine leicht abgeänderte Version eines Moduls aus einem anderen Design [18] verwendet. Listing 5 zeigt den entsprechenden Ausschnitt aus dem VHDL-Design.

```
-- von rechts Eintakten
    input_vec <= input_vec(14 downto 0) & input_d;
```

Listing 5: Tasten-Input einschieben

Im Vergleich zum ursprünglichen Design wird hier anstelle eines fünf Bit langen Vektors ein 16 Bit langer Vektor verwendet. Dadurch erhöht sich die Verzögerung, aber auch die Zuverlässigkeit, da mehr Bits betrachtet werden, bevor das Modul eine Entscheidung trifft.

Die Positionen 0 bis 14 des Input-Vektors werden im Signal „input_vec“ gespeichert. Der neue Input wird von rechts, wie in Abbildung 33 dargestellt, in den Vektor eingeschoben. Weil der Zynq mit einem Takt von 125 MHz arbeitet, werden für den Vorgang 128 ns benötigt. [18]

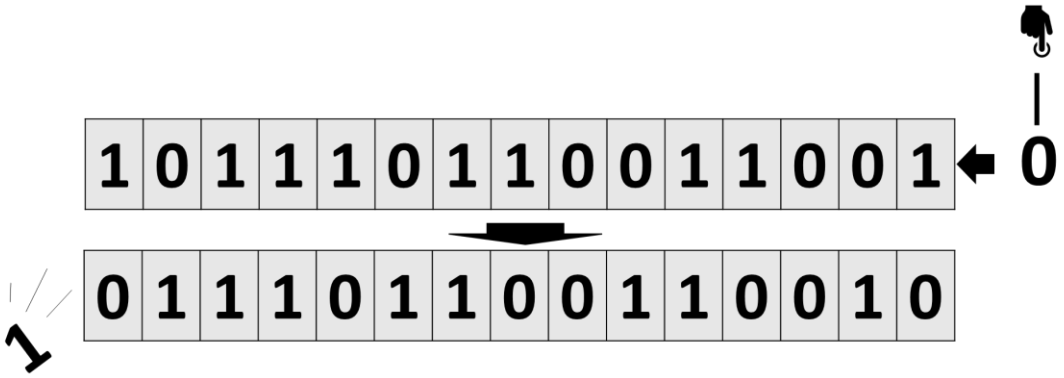

Abbildung 33: Nachrückverfahren des Inputvektors

Somit wird der Inputverlauf im Vektor abgebildet. Dadurch kann erkannt werden, dass der Taster, bei einem mit Einsen gefüllten Vektor, tatsächlich geschlossen ist und nicht prellt. Somit befindet sich der Taster im blau markierten Bereich von Abbildung 34. [18]

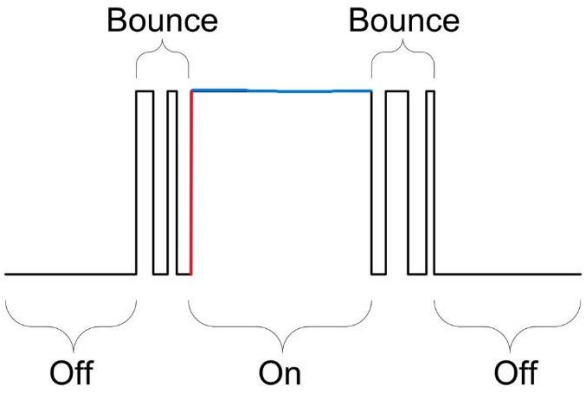

Abbildung 34: Tasterzustände [18]

Wenn ein Modul mit dem blauen Bereich arbeitet, wird für die Dauer des Drückens eine Eins ausgegeben. Für das Design ist nur, die in Abbildung 34 zu findende, rot markierte Flanke relevant, da nur kurzzeitig ein Signal zum Takt übermittelt wird. Um diesen Fall zu erkennen wird, anstatt den Zustand nur bei einem Vektor aus Einsen auszugeben, gibt das Modul einen Takt früher eine erkannte Flanke aus. Umgesetzt wird das Ganze durch eine Case-Abfrage, die den Vektor mit allen möglichen Optionen abgleicht und entsprechend handelt (siehe Listing 6). Sowohl die Information über den aktuellen Zustand als auch die positive

Taktflanke werden immer ausgegeben. Diese Informationen werden bei der nächsten positiven Taktflanke von den anderen Prozessen übernommen, wodurch der Input auch synchronisiert wird. [18]

```
-- Zustaende zuweisen
case input_vec is
    when "1111111111111111" =>
    rise_d <= '0';
    output_d <= '1';
    -- rising edge
    when "0111111111111111" =>
    rise_d <= '1';
    output_d <= '1';
    --alles andere
    when others =>
    rise_d <= '0';
    output_d <= '0';
end case;
```

Listing 6: Case-Abfrage im Debounce-Modul

5.2.4 SPI-Module

Die Aufgabe des SPI-Moduls besteht darin, die Daten aus dem Filter entsprechend der Skalierung des DAC anzupassen und gemäß dem SPI-Protokoll zu übermitteln.

Das erste Signal, das in dem Modul verarbeitet wird, ist das 20 MHz Taktsignal, mit dem das Modul arbeitet. Dieses wird direkt auf einen Ausgang gelegt, um auch dem SPI-Slave zur Verfügung zu stehen. Wie in Kapitel 4.4 erklärt, muss bei jeder Inbetriebnahme des DA4-Moduls die verwendete Referenzspannung einmal auf „Intern“ gestellt werden. Zum Zeitpunkt der ersten Ausführung des Moduls liegt dieser Befehl bereits als Initialisierungswert auf dem 32-Bit-Signal „data“ vor (siehe Listing 7). Sobald alle 32-Bit gesendet wurden, wird das Signal für das Senden der Messwerte vorbereitet.

```
signal data: std_logic_vector (31 downto 0):= "00001000000000000000000000000001";
```

Listing 7: Initialisierung des „data“-Signals

Um das zu erreichen, müssen die ersten vier „don't care“-Bits mit Nullen aufgefüllt werden. Zusätzlich werden die Command-Bits auf „0011“ konfiguriert, damit die Daten-Bits das Register des adressierten DAC überschreiben und die Änderungen sofort ausgeführt werden (siehe Abbildung 23). Danach werden auch die letzten acht „don't care“-Bits durch Nullen ersetzt. Für die Übermittlung der Messwerte müssen an diesen Bits zukünftig keine Änderungen mehr vorgenommen werden.

Der DAC erwartet für die Datenbits eine Bit-Folge mit insgesamt zwölf Bits als Ausgabewert. Keiner der Filterausgänge nutzt derzeit eine Darstellung mit zwölf Bits. Um sicherzustellen,

dass die Daten später korrekt dargestellt werden, müssen sie skaliert werden. Hierfür wird die Formel (10) verwendet.

$$y = x \times \frac{2^{12} - 1}{Gain} \tag{10}$$

Um die Formel in VHDL umzusetzen, werden die Vektoren mit den Messergebnissen zunächst mit 4095 multipliziert werden. Hierfür dient das Signal „scale“, ein zwölf Bit langer Vektor, der nur mit Einsen gefüllt ist. Durch die Multiplikation entsteht ein Vektor mit einer Größe von n +12 Bit. Dieser wird einem neuen Signal zugewiesen, dessen Name sich wie folgt zusammensetzt: „M“+ Dezimierung+ Nummerierung+ „_scaled“ (siehe Listing 8).

Für eine der Kurzschlussstrom-Filter ergibt sich so z.B. ein 19 Bit langer Vektor mit dem Namen „M41_scaled“.

```
elsif rising_edge(clk20)then

    M41_scaled <= std_logic_vector(unsigned(M41) * unsigned(scale));
    M42_scaled <= std_logic_vector(unsigned(M42) * unsigned(scale));
    M81_scaled <= std_logic_vector(unsigned(M81) * unsigned(scale));
    M82_scaled <= std_logic_vector(unsigned(M82) * unsigned(scale));
    M161_scaled <= std_logic_vector(unsigned(M161) * unsigned(scale));
    M162_scaled <= std_logic_vector(unsigned(M162) * unsigned(scale));
```

Listing 8: Multiplikation der Vektoren zwecks Skalierung

Um aus diesem den Skalierten Wert zu erhalten, muss er noch durch die maximale Verstärkung (Gain) des Filters geteilt werden. In dem Beispiel des Kurzschlussstrom-Filters also durch 64. Um den zusätzlichen Schritt des Schiebens für die Division zu sparen, greift das Design, wie in Listing 9 zu sehen ist, immer nur auf die entsprechenden Bits zu. Eine Division durch 64 würde durch eine sechsfache Schiebung nach rechts erzielt werden. Folglich fangen die skalierten Werte an Position sechs an und enden zwölf Bits später an Position 17 des Vektors. Für die anderen Filter sieht die Skalierung ähnlich aus, mit der Ausnahme der angepassten Division.

Die Daten- und Adressbits werden kontinuierlich durchrotiert, unabhängig davon, wie schnell die Filter neue Daten zur Verfügung stellen (siehe Listing 9). Hierfür wird die Case-Anweisung verwendet, welche nacheinander die einzelnen Kombinationen abläuft. Am Ausgang der DACs zeigen sich somit nur Unterschiede, die durch die verschiedenen Auflösungen entstehen und nicht etwa durch priorisiertes Senden.

```
if (cnt = 32) then
    data (31 downto 24)<= "00000011";
    data (7 downto 0)<= "00000000";

    Case in_select is
     when 0 => data(23 downto 8) <= "0000" & M41_scaled(17 downto 6);
     when 1 => data(23 downto 8) <= "0001" & M42_scaled(17 downto 6);
     when 2 => data(23 downto 8) <= "0010" & M81_scaled(20 downto 9);
     when 3 => data(23 downto 8) <= "0011" & M82_scaled(20 downto 9);
     when 4 => data(23 downto 8) <= "0100" & M161_scaled(23 downto 12);
     when 5 => data(23 downto 8) <= "0101" & M162_scaled(23 downto 12);
    end case;

    if (in_select = 5) then
        in_select <= 0;
    else
        in_select <= in_select+1;
    end if;

    cnt <= 0;
else
    data <= data;
    cnt <= cnt + 1;
end if;
```

Listing 9: Änderung der zu sendenden Daten

Die Übermittlung der Daten an den Eingang des Slaves wird in Listing 10 dargestellt. Dazu muss das Slave-Select-Signal für mindestens einen Takt auf Eins gesetzt werden. Dies geschieht, wenn die Daten vollständig gesendet wurden und der Zähler wieder auf Null steht. Im nächsten Takt wird das Slave-Select-Signal dann wieder auf Null gesetzt und der Sendeprozess beginnt erneut. Bei dieser Form der Datenausgabe wird bei jeder steigenden Taktflanke nacheinander eines der Bits aus dem „data“-Signal ausgegeben. Jeder der sechs Sendeprozesse benötigt somit insgesamt 33 Takte. Die Ausgabe eines DACs wird also erst nach 9,9 µs aktualisiert. Das ist mehr als viermal langsamer als der Filter für die FOC, welcher mit 2,4 µs Latenz der langsamste der Filter ist. Daher eignet sich diese Form der Datenausgabe lediglich zu Demonstrationszwecken.

```
if (cnt = 0) then
    ss <= '1';
else
    ss <= '0';
    MOSI <= data(32-cnt);
end if;
```

Listing 10: Der Sendevorgang im SPI-Modul

5.3 Blockdesign

Das Blockdesign ist eine vereinfachte Möglichkeit, um die zuvor erzeugten Module zu verschalten. Dabei werden die Module durch Blöcke repräsentiert und durch das Ziehen einer Leitung einfach verbunden. Dadurch wird das aufwendige Einbinden und Verbinden in textbasierter Weise erleichtert. Eine Darstellung des Blockdesigns dieses Projekts ist in Abbildung 35 zu sehen. Die Leitungen sind farblich hervorgehoben, um die Identifizierung zu erleichtern.

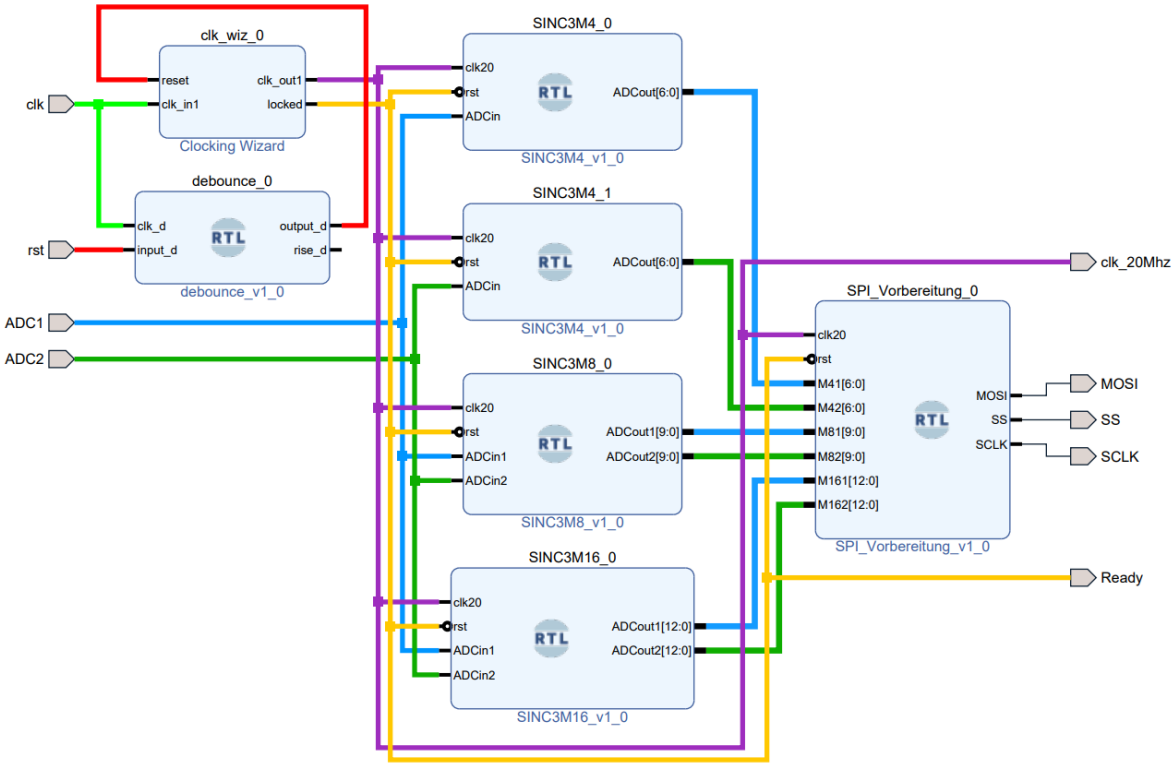

Abbildung 35: Blockdiagramm des finalen Designs

Die linke Hälfte des Designs enthält alle benötigten Eingänge, einschließlich des clock-Signals, an das der 125 MHz Takt des Arty Z7 angeschlossen wird, des Reset-Signals vom Taster des Arty Boards und der beiden Eingänge für die auf dem EDPS-Board befindlichen Modulatoren.

Der 125 MHz Takt versorgt in diesem Design nur zwei Module: das Debounce-Modul und den „Clocking Wizard“. Um die Verzögerung zwischen der Betätigung des Tasters und dem Ausführen des Resets zu minimieren, empfiehlt es sich, das Debounce-Modul mit dem schnellstmöglichen Takt zu betreiben. Wenn das Debounce-Modul eine Null ausgibt, sollten

die Signale innerhalb des Designs auf ihre Initialisierungswerte zurückgesetzt werden. Dafür ist es ausreichend, den „Clocking Wizzard" zurückzusetzten.

Beim „Clocking Wizzard" handelt es sich um einen IP-Block der Firma Xilinx. Er ermöglicht, mittels Mixed-Mode-Clock-Manager (MMCM) oder Phase-Locked-Loops (PLLs) das präzise Erzeugen diverser Takte. Die Hardware gewährleistet, durch ständiges Regeln, eine zuverlässige und stabile Quelle für Takte, die geringere oder höhere Frequenzen als der Ursprungstakt besitzen, ohne dass dabei eine Phasenverschiebung gegenüber dem Ursprungstakt auftritt. Auch temperatur- oder versorgungsspannungsbedingte Einflüsse werden berücksichtigt und entsprechend ausgeglichen. Für dieses Projekt genügt es, den Eingangstakt wie in Abbildung 36 dargestellt auf 125 MHz zu setzen und den Output auf die gewünschten 20 MHz zu konfigurieren (vgl. Abbildung 37).

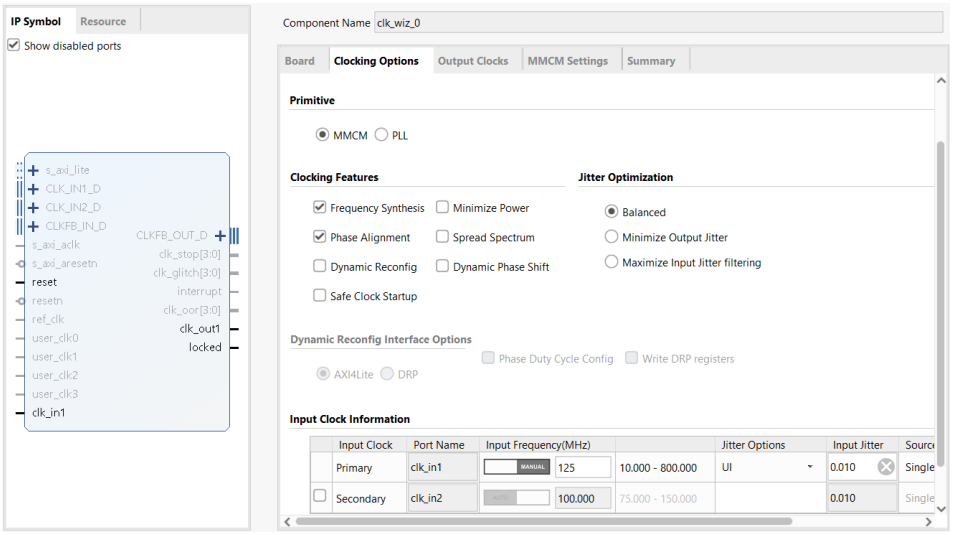

Abbildung 36: "Clocking Wizzard" Konfiguration des Inputs

The phase is calculated relative to the active input clock.

Output Clock	Port Name	Output Freq (MHz)		Phase (degrees)		Duty Cycle (%)
		Requested	Actual	Requested	Actual	Requested
clk_out1	clk_out1	20	20.00000	0.000	0.000	50.000
clk_out2	clk_out2	100.000	N/A	0.000	N/A	50.000
clk_out3	clk_out3	100.000	N/A	0.000	N/A	50.000
clk_out4	clk_out4	100.000	N/A	0.000	N/A	50.000
clk_out5	clk_out5	100.000	N/A	0.000	N/A	50.000
clk_out6	clk_out6	100.000	N/A	0.000	N/A	50.000
clk_out7	clk_out7	100.000	N/A	0.000	N/A	50.000

Abbildung 37: "Clocking Wizzard" Konfiguration des Outputs

In diesem Design wird der 20 MHz Takt für die Filter, die SPI-Kommunikation und die Modulatoren erzeugt, wie in Abbildung 35 durch die lila Markierung gekennzeichnet. Wenn der

„Clocking Wizard“ gestartet wird, benötigt er einige Takte, um das Signal am Ausgang auf 20 MHz zu stabilisieren. Sobald der Ziel-Takt erreicht ist, wechselt das Signal am Ausgang „locked“ des Blocks von Null auf Eins, wie in Abbildung 35 durch die gelbe Markierung gekennzeichnet. Dieses Signal erfüllt in dem Design mehrere Zwecke. Während des gesamten Einschwingvorgangs des „Clocking Wizards“ darf keines der anderen Module aktiv sein, da der unsaubere Takt des „Clocking Wizards“ zu Timing-Problemen führen kann. Daher dient das Signal für die anderen Module als Reset. Die weiteren Module des Designs beginnen erst mit dem Betrieb, wenn der Takt den Vorgaben entspricht. Vor dem Betrieb wird immer ein Reset durchgeführt. Um sicherzustellen, dass jedes interne Signal einen vordefinierten Zustand hat, werden diese während der Initialisierung gesetzt.

Da das „locked“-Signal des Wizards für die Resets der anderen Module verantwortlich ist, reicht es aus, diesen im Falle eines Resets zurückzusetzen. Wenn ein Reset auftritt, wird die Takterzeugung des Wizards unterbrochen und der Pegel des „locked“-Signals ändert sich entsprechend. Folglich werden alle weiteren Module nach einer kurzen Verzögerung zurückgesetzt.

Das „locked“-Signal repräsentiert außerdem den Betriebszustand des Designs und wird daher auf einen Ausgang des Designs geführt. Dieser Ausgang wird später an eine LED des Arty Boards angeschlossen.

Abbildung 35 zeigt vier verschiedene Filter in der mittleren Spalte. Ganz oben befinden sich die Filter für die Kurzschlusserkennung mit den Bezeichnungen „M4_0“ und „M4_1“. Da sich dieser Filter, wie in Kapitel 5.2.2 erklärt, nicht für die Doppelverwendung eignet, werden zwei Instanzen des Filters verwendet. Beide Filter erhalten neben dem 20 MHz Takt und dem Reset-Signal auch die Daten der entsprechenden Modulatoren. Der obere Filter ist für die Verarbeitung des Bit-Streams des ersten Modulators zuständig und der untere für den des zweiten. Auf die Eingänge der doppelt verwendeten Filter, werden jeweils beide Modulatorausgänge geführt. Anschließend werden die Daten an das SPI-Modul weitergeleitet, das die für das Protokoll notwendigen MOSI-, ~CS- und SCLK-Signale erzeugt und nach außen führt.

Das Blockdesign wird mittels eines Wrappers wieder in VHDL-Text umgewandelt. Vivado erzeugt automatisch eine Quelldatei, welche die visuelle Darstellung der Einbindung und Verknüpfung der Blöcke widerspiegelt.

5.4 Testbench

Um die Funktion der einzelnen und des gesamten VHDL-Moduls vor der Umsetzung auf der Hardware zu überprüfen, werden Testbenches verwendet. Mit ihrer Hilfe lässt sich das Design Testvektoren stimulieren und die Ergebnisse visuell darstellen. Die Simulation ist notwendig, um mögliche Fehler frühzeitig zu erkennen und direkt im Design zu beheben.

Die erste Testbench prüft das Design ohne das SPI-Modul. Das Design wird mit einem künstlichen 125 MHz Takt und einem Bit-Stream versorgt. Das Verhalten des Filters lässt sich besonders gut charakterisieren, indem er mit einer Sprungfunktion angesteuert wird. Dadurch wird die Sprungantwort dargestellt. An den Eingängen werden daher zwei um 180° versetzte Sprungfunktionen angelegt (Sieh Listing 11).

```
process begin
    rstT <= '0';
    ADC2T <= '1';
    ADC1T <= '0';
    wait for 10000ns;
    ADC2T <= '0';
    ADC1T <= '1';
    wait for 5000ns;
    rstT <= '1';
    wait for 5000ns;
end process;
```

Listing 11: Testbench der Filter

Zunächst wird der Reset auf einen inaktiven Zustand gesetzt. Anschließend werden zwei Bitstreams erzeugt, die für jeweils 10 µs entweder eine Eins oder Null ausgeben. Danach tauschen die beiden Bitstreams für 5 µs die Pegel, bis schließlich für weitere 5 µs ein Reset ausgeführt wird (Siehe Abbildung 38). Aufgrund der Doppelverwendung des Filters müssen diese beim Hin- und Herschalten immer zwischen den beiden Maxima wechseln. Damit wird vor allem der Wechselvorgang und die Reaktionszeit des Filters getestet. Auch die Präzision der Wertwiedergabe wird überprüft.

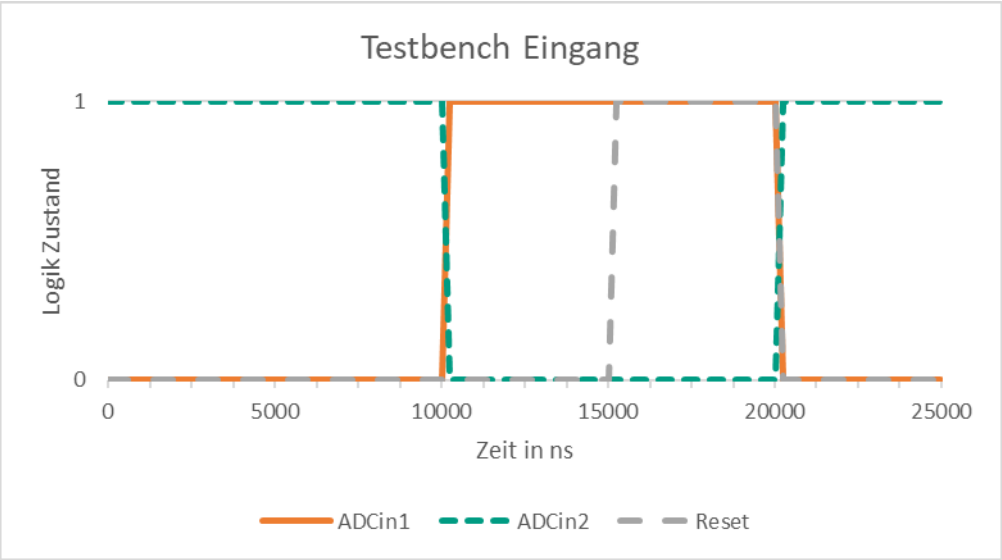

Abbildung 38: Verlauf der Testbench zur Prüfung der Filter

Anschließend wird das SPI-Modul getestet. Hierfür wird es mit Maximum- oder Minimum-Filterausgaben versorgt, um die Daten später besser wiederfinden zu können. Außer dem 20 MHz Takt wird für dieses Modul nichts weiter benötigt. Es sollte die in Listing 12 dargestellten Daten zusammen mit ihrer Adresse nacheinander senden.

```
M41T <= "1000000";
M42T <= "0000000";
M81T <= "1000000000";
M82T <= "0000000000";
M161T <= "1000000000000";
M162T <= "0000000000000";
```

Listing 12: Versorgung des SPI-Moduls in der Testbench

Zuletzt wird das Design als Ganzes getestet. Dazu genügen die Vorgabe des Modulator-Bitstreams und das Verhalten des Resets neben dem Takt. Der Test erfolgt ähnlich wie bei den anderen Modulen (vgl Listing 13). Ein konstanter Input und ein abrupter Wechsel des Pegels heben mögliche Fehler besonders hervor. Die Werte werden im Format der SPI-Kommunikation ausgegeben.

```
process
begin

    rstT <= '0';
    ADC2T <= '1';
    ADC1T <= '0';
    wait for 10000ns;
    ADC2T <= '0';
    ADC1T <= '1';
    wait for 5000ns;
    rstT <= '1';
    wait for 5000ns;
end process;
end Behavioral;
```

Listing 13: Versorgung des gesamten Designs in der Testbench

5.5 Constraint-Datei

Die Constraint-Datei dient der Zuweisung der Ein- und Ausgänge des Designs an die hardwarespezifischen Komponenten. In der Regel wird die Constraint-Datei vom Hersteller bereitgestellt. Für die Verwendung der internen Clock-Signale des Boards muss die Zeile, die in Listing 14 zu sehen ist, entkommentiert werden. Der Name des im Design verwendeten Eingangs muss in den Klammern nach der Anweisung „get_ports" eingetragen werden.

```
## Clock Signal
set_property -dict { PACKAGE_PIN H16    IOSTANDARD LVCMOS33 } [get_ports {clk}]; #IO_L13P_T2_MRCC_35 Sch=SYSCLK
```

Listing 14: Zuweisung des Clock-Eingangs

Das Gleiche wird auch mit den anderen Ein- und Ausgängen des Designs durchgeführt. Der Ausgang „Ready", der vom Signal „Locked" gesetzt wird, wird auf eine der LEDs auf dem Arty Z7 Board geleitet (Siehe Listing 15).

```
## LEDs
set property -dict { PACKAGE PIN R14    IOSTANDARD LVCMOS33 } [get ports { Ready }]; #IO L6N T0 VREF 34 Sch=LED0
```

Listing 15: Zuweisung des Ready-Ausgangs

Der Eingang für den asynchronen Reset wird auf einen der Taster des Arty Z7 geführt (vgl. Listing 16)

```
set_property -dict { PACKAGE_PIN L19    IOSTANDARD LVCMOS33 } [get_ports { rst }]; #IO_L9P_T1_DQS_AD3P_35 Sch=BTN3
```

Listing 16: Zuweisung des Reset-Eingangs

Bei der Zuweisung der Ausgänge für das SPI-Modul ist Tabelle 1 zu beachten. Um alle Module später über die Pmod-Header miteinander verbinden zu können, wird der Pmod-Header JA verwendet (vgl. Listing 17). Das EDPS-Board bietet daneben ausreichend Platz.

```
## Pmod Header JA
set_property -dict { PACKAGE_PIN Y18   IOSTANDARD LVCM 333 } [get_ports { SS }]; #IO_L17P_T2_34 Sch=JA1_P (Pin 1)
set_property -dict { PACKAGE_PIN Y19   IOSTANDARD LVCMOS33 } [get_ports { MOSI }]; #IO_L17N_T2_34 Sch=JA1_N (Pin 2)
#set_property -dict { PACKAGE_PIN Y16   IOSTANDARD LVCMOS33 } [get_ports {  }]; #IO_L7P_T1_34 Sch=JA2_P (Pin 3)
set_property -dict { PACKAGE_PIN Y17   IOSTANDARD LVCMOS33 } [get_ports { SCLK }]; #IO_L7N_T1_34 Sch=JA2_N (Pin 4)
```

Listing 17: Zuweisung der SPI-Ausgänge

Die restlichen Ein- und Ausgänge werden gemäß Abbildung 19 auf den Pmod-Header JB gelegt (Siehe Listing 18).

```
## Pmod Header JB
set_property -dict { PACKAGE_PIN W14   IOSTANDARD LVCMOS33 } [get_ports { clk_20Mhz }]; #IO_L8P_T1_34 Sch=JB1_P (Pin 1)
set_property -dict { PACKAGE_PIN Y14   IOSTANDARD LVCMOS33 } [get_ports { ADC1 }]; #IO_L8N_T1_34 Sch=JB1_N (Pin 2)
set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33 } [get_ports { ADC2 }]; #IO_L1P_T0_34 Sch=JB2_P (Pin 3)
```

Listing 18: Zuweisung der Anschlüsse der Modulatoren

5.6 Versuchsaufbau

Das VHDL-Design wird nach der Synthese, Implementierung und Bitstream-Generierung über den Hardware-Manager in Vivado auf das Arty Board geladen. Bevor das EDPS-Board zum Testen des Designs an das Arty Board angeschlossen werden kann, müssen jedoch noch kleine Änderungen vorgenommen werden.

Das Ziel des Versuchs ist es, eine Spannung am Eingang der Sigma-Delta-Wandler zu erzeugen, die in der Motorstrommessung verwendet werden. Diese befinden sich in der Schaltung auf der linken Seite, wie in Abbildung 39 dargestellt.

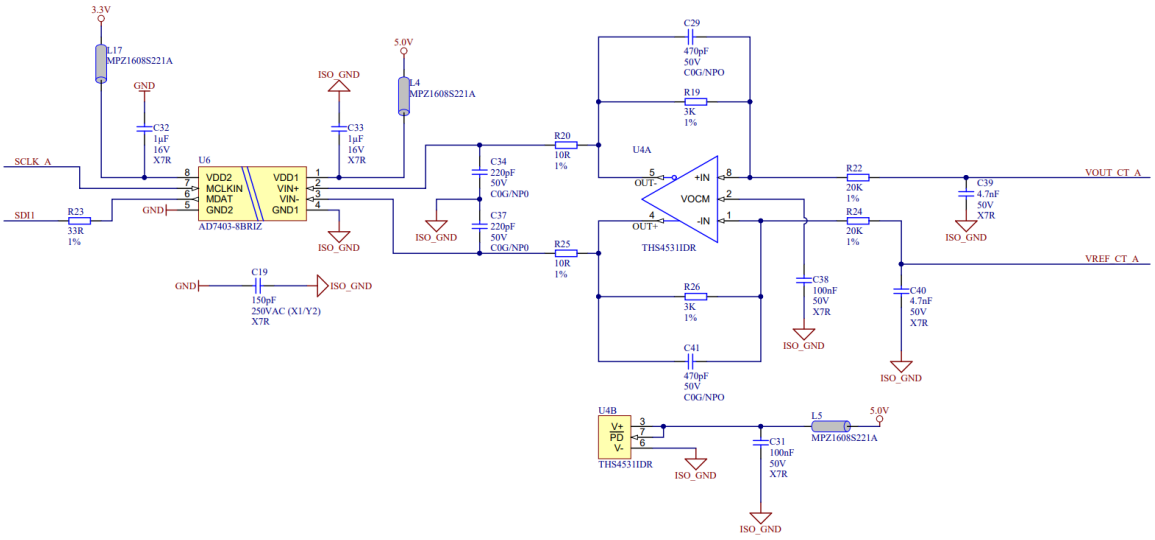

Abbildung 39: Sigma-Delta-Wandler für die Messung der Phase Iu inklusive Signalanpassung [19]

Um die Spannung an den Wandler-Eingängen künstlich zu erzeugen, reicht es nicht aus, diese an die Eingänge anzulegen, da die Wandler mit einer Spannung zwischen ±250 mV

arbeiten. Es ist nur bedingt möglich, so kleine Spannungen direkt aus einer Quelle einzuspeisen, um die Hardware zuverlässig zu testen. Zusätzlich wird die an den Eingängen anliegende Spannung durch weitere Komponenten geregelt. Um dies zu vermeiden, sollte die Spannung an den Anschlüssen „VOUT“ und „VREF“ reguliert werden. Diese Spannung wird von den LEM-Wandlern erzeugt und kann zwischen 0 V und nahezu 5 V variieren. Daher eignet sich dieser Punkt besonders gut, um einen normalen Betrieb zu simulieren. Um mögliche Schäden oder Eingriffe des Bauteils bei der Einspeisung zu vermeiden, empfiehlt es sich, die LEM-Wandler vom EDPS-Board zu entfernen. Die Bauteilgröße erleichtert diesen Prozess. An die „VOUT“ und „VREF“ Anschlüsse auf der Platine werden Kabel zum Anlegen von Spannungen angelötet.

Zur Überprüfung der Funktion des Designs unter realen Bedingungen wird der in Abbildung 40 gezeigte Versuchsaufbau verwendet.

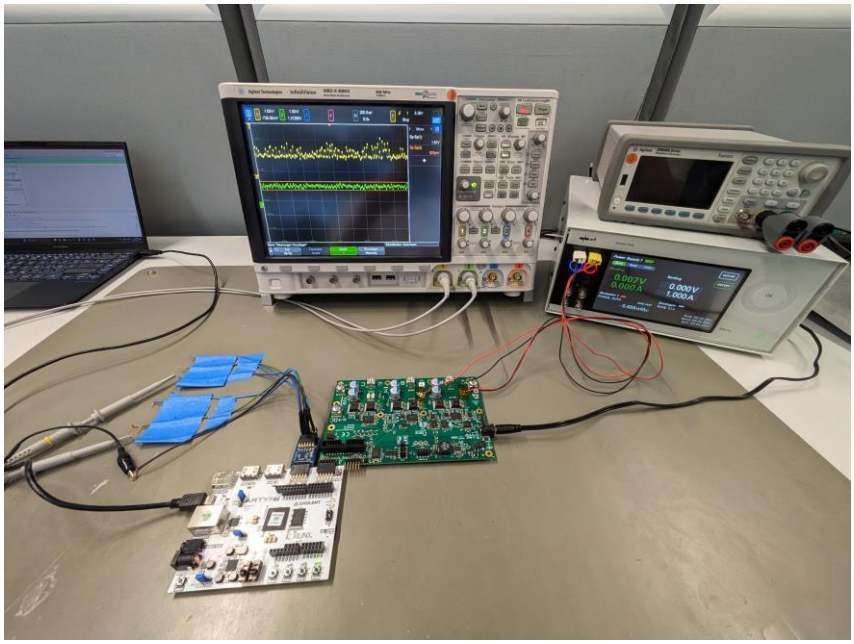

Abbildung 40: Versuchsaufbau zur Überprüfung des Designs

Das modifizierte EDPS-Board wird über den J9-Ausgang mit dem JB-Pmod-Header des Arty Z7 verbunden. Der Pmod-Header JA des Arty Z7 wird durch den DA4 belegt. Da dieser nur eine der beiden Reihen belegt, ist es wichtig sicherzustellen, dass er gemäß der Belegung in der Constraint-Datei in der oberen Reihe steckt. Die verbundenen Boards werden gemäß Abbildung 41 an weitere Geräte angeschlossen. Ein USB-Kabel ist an den USB-Eingang des Arty Z7-Boards angeschlossen, um eine Verbindung zum Rechner herzustellen, auf dem der

Hardware-Manager von Vivado ausgeführt wird. Der USB-Anschluss versorgt das Board zusätzlich mit Strom. Um die generierten Signale einfacher zu messen, werden Leitungen gemäß der Abbildung 41 am DA4-Modul angeschlossen. Die Pinbelegung ergibt sich aus den im SPI-Modul verwendeten Adressen. Das Signal wird über die Leitungen mit einem Oszilloskop gemessen. So können nicht nur der Pegel, sondern auch zeitliche Veränderungen und die Qualität der Signale betrachtet werden. Im ersten Teil des Versuchs wird am Eingang des Sigma-Delta-Wandlers ein fester Spannungspegel eingestellt, welcher von einer Spannungsquelle eingespeist wird. Zur besseren Überprüfung der verschiedenen Auflösungsstufen kann die Spannungsquelle durch einen Funktionsgenerator ersetzt werden. Eine vereinfachte Darstellung des Aufbaus ist in Abbildung 41 zu sehen.

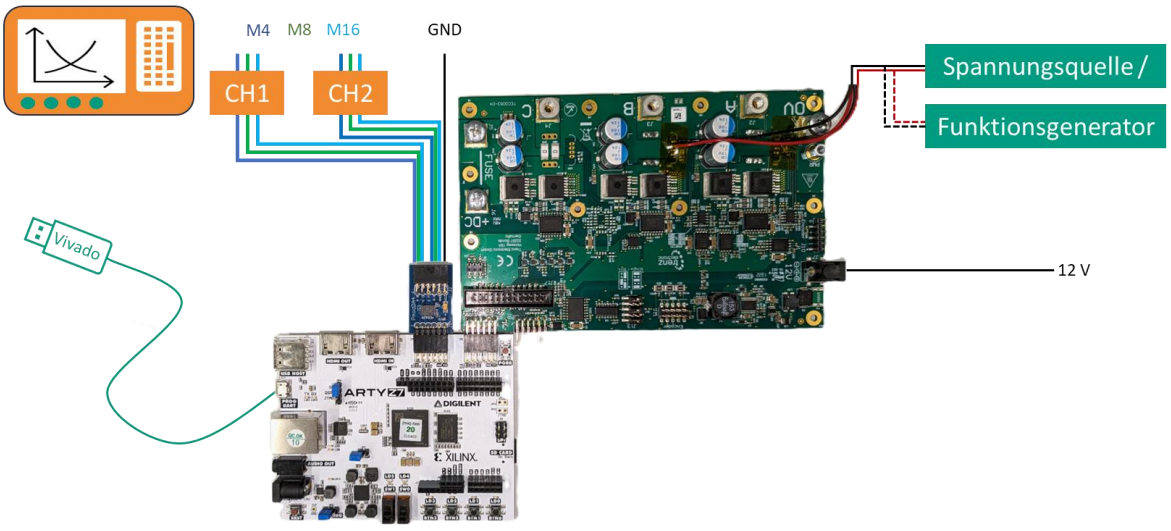

Abbildung 41: Versuchsaufbau (vereinfacht dargestellt)

6 Ergebnisse

In diesem Kapitel werden die Ergebnisse der Simulationen und des Versuchsaufbaus dargelegt. Das Ziel ist es, die Funktion des Designs durch die Darstellung der aus den Simulationen gewonnenen Daten und der gemessenen Daten zu verifizieren. Dabei wird besonderer Wert auf die Übereinstimmungen und Unterschiede zwischen den simulierten und realen Ergebnissen gelegt.

6.1 Simulationen

Abbildung 42 zeigt die Ergebnisse der Simulation mit der ersten Testbench. Diese testet das Design ohne das SPI-Modul, so dass nur die Filtereigenschaften Einfluss auf die Ausgabe haben.

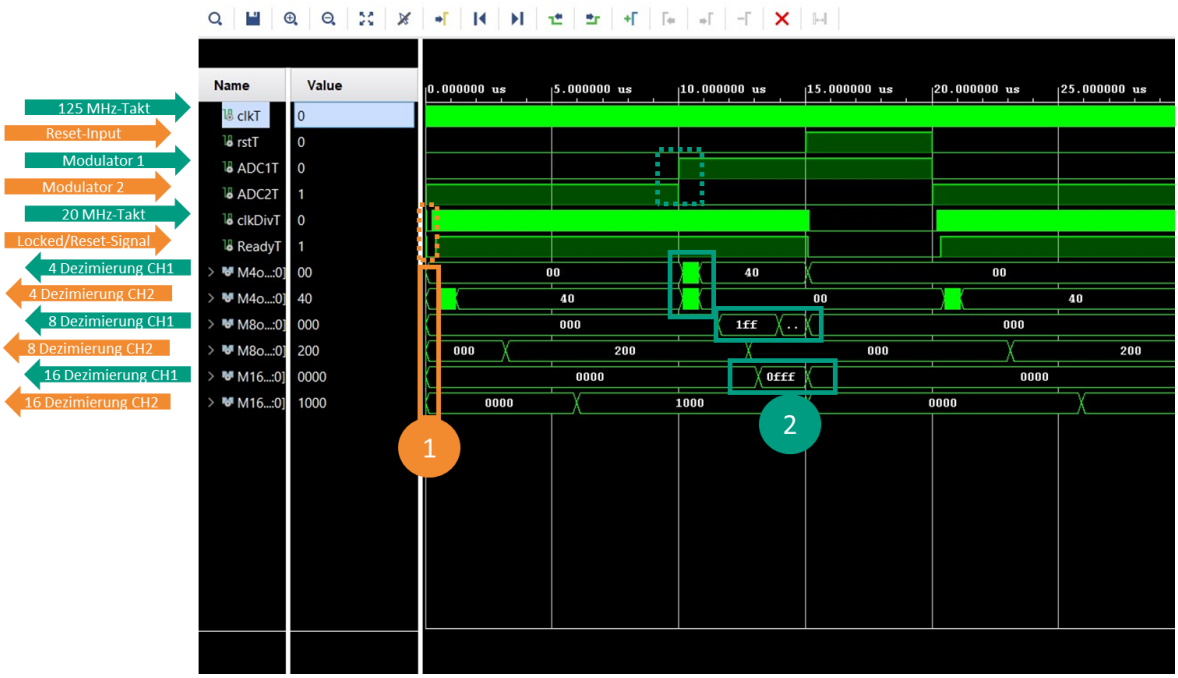

Abbildung 42: Simulationsergebnisse der Testbench zum Überprüfen der Filter

Als erstes Signal ist der 125 MHz Takt zu sehen, gefolgt vom Reset-Signal, den beiden Bitstreams, dem Takt aus dem „Clocking Wizard" sowie seinem Locked-Signal. Zum Schluss folgen die sechs Filterausgaben.

In der ersten Situation, in welcher der „Clocking Wizard" den Takt noch nicht bestätigt hat, werden alle Ausgaben auf ihren definierten Ausgangswert gesetzt. So wird keines der Module mit einem unsauberen Takt betrieben und es wird ein anfänglicher Reset ausgeführt, um undefinierte Zustände zu vermeiden. Erst nachdem das Locked-Signal gesetzt ist, das in

diesem Fall als „Ready“ bezeichnet wird, beginnen die Filter mit der Auswertung. Kurz darauf haben alle Filter ihren endgültigen Wert eingenommen.

Die Ergebnisse in der Simulation werden als hexadezimale Zahlen dargestellt. Die #40 des Filters mit einer Dezimierungsrate von vier entspricht folglich dem maximalen Gain des Filters, nämlich 64. Für die anderen Filter gilt dasselbe mit den Ergebnissen von 512 und 4096. Die einzige Ausnahme ergibt sich zum Zeitpunkt des Pegelwechsels der Eingänge in Situation zwei. Kurz nach dem Pegelwechsel nehmen die ersten Kanäle der doppelt verwendeten Filter, M8 und M16, kurzzeitig einen Wert nahe des Zielwerts an. Zu dieser Situation kommt es, da das Einlesen für die Filter auch während des Pegelwechsels stattfindet. Der ausgegebene Wert setzt sich also aus einer zufälligen Menge von Nullen und Einsen zusammen. Nachdem der Pegel am Eingang des Filters wieder konstant ist, nimmt dieser auch wieder den entsprechenden Ausgangswert an. Auch die Filter M4 durchlaufen nach dem Pegelwechsel Änderungen. Situation zwei markiert die drei Stufen, die die M4 Filter benötigen, um den neuen Ausgang zu repräsentieren. Zur besseren Darstellung dieser Situation übergeben die doppelt verwendeten Filter in Abbildung 43 nicht nur die Endwerte, sondern auch die sonst nicht sichtbaren Zwischenschritte. Dadurch lassen sich die drei Übergangsstufen des Filters betrachten.

Abbildung 43: Simulationsergebnisse der Testbench zum Überprüfen der Filter mit der Ausgabe aller Werte

Das orange markierte Signal vom ersten Kanal des M8-Filters zeigt, wie der Filter auf den internen Ein-/Ausgangswechsel reagiert. Bis zu dem Zeitpunkt des Wechsels bei 1,6 µs, auf dem unteren Zeitstrahl, war der M8-Filter noch an den Eingang des zweiten Kanals gebunden. Folglich ist der Filter zu diesem Zeitpunkt noch vollständig mit dessen Daten gefüllt,

weshalb am Ausgang des Filters das Maximum von #200 (512) anliegt. Ab diesem Zeitpunkt jedoch, wechselt der Eingang des Filters von Kanal 2 auf Kanal 1, weshalb bei 2 µs der Ausgang vom Maximum (512) auf #1c8 (456) wechselt. Dieser Zwischenschritt beschreibt die erste der drei Stufen des Filters. Nach acht weiteren Abtastungen und der Ausführung der Dezimierungsstufe sinkt der Wert erneut. Zu diesem Zeitpunkt ist er mit #078 (120) schon deutlich näher am Zielwert von null. Die platzierten Cursors zeigen dabei eine Dauer von 400 ns für diesen Schritt, welcher auf der linken Hälfte der Abbildung identisch ist. Bei einem Takt mit einer 50 ns Periodendauer entspricht dies also genau den eingestellten 8 Takten. Schließlich wurde nach der dritten Ausführung der Dezimierung der Endwert Null erreicht. Das gleiche Verhalten lässt sich ebenfalls für dem doppelt verwendeten M16-Filter betrachten, dessen Abtastungen doppelt so lange andauert.

Die Ergebnisse haben bestätigt, dass die Zwischenschritte immer eine Mischung aus den Eingangswerten beider Kanäle sind und sich somit nicht für die Verarbeitung eigenen. Das Design gibt daher immer nur den dritten und endgültigen Wert aus. Die obersten beiden Filterausgaben werden für die Kurzschlussabschaltung verwendet. Es sind keine Zwischenschritte zu erkennen, da die Filter nicht doppelt verwendet wurden.

Abbildung 44 zeigt, was beim Reset mit dem Design geschieht. Das locked-Signal und damit das gesamte Design werden bereits 125 ns nach der Betätigung auf Null zurückgesetzt. Die Verzögerung wird größtenteils durch das Debounce-Modul verursacht. Dieses benötigt 15 Takte am Eingang, um das Signal zu bewerten. Da das Modul mit dem 125 MHz Takt arbeitet, ergibt sich eine Dauer von 15 × 8 ns = 120 ns. Nach Ablauf von 120 Nanosekunden wird diese Information an den „Clocking Wizard“ weitergeleitet. Dieser reagiert 5 Nanosekunden später und passt das „locked“-Signal an.

Abbildung 44: Simulationsergebnisse der Testbench zum Überprüfen der Filter zum Zeitpunkt eines Resets

Als nächstes wird das SPI-Modul separat getestet. Das Simulationsergebnis in Abbildung 45 zeigt vier vollständige Sendevorgänge. Zunächst wechselt das ~CS-Signal von Eins auf Null. Gleichzeitig wird eine Null gesendet. Nach vier weiteren Nullen wird im Bereich der „Command"-Bits die erste Eins ausgegeben. Danach folgen, bis auf die letzte Stelle, nur Nullen. Nach dem letzten Bit wechselt das ~CS-Signal wieder auf eine Eins, was den Sendeprozess beendet. Der resultierende Vektor lautet „00001000000000000000000000000001" und wird zu Beginn gesendet, um das Modul auf die interne Referenz zu stellen. Das Protokoll wurde eingehalten und der Sendeprozess erfolgreich abgeschlossen. Die anderen Sendevorgänge verlaufen ähnlich. Mit dem Unterschied, dass sie an den vorgesehenen Stellen ihre Adresse und die zwölf Datenbits haben. Der blaue und der lila Bereich zeigen außerdem, wie die Vektoren „1000000" und „1000000000" als Darstellung des maximalen Filterwerts skaliert werden. Beide werden zu einer Folge von zwölf Einsen skaliert, was dem maximal einstellbaren Wert des DACs entspricht. Auch die Nullen im roten Bereich werden richtig skaliert.

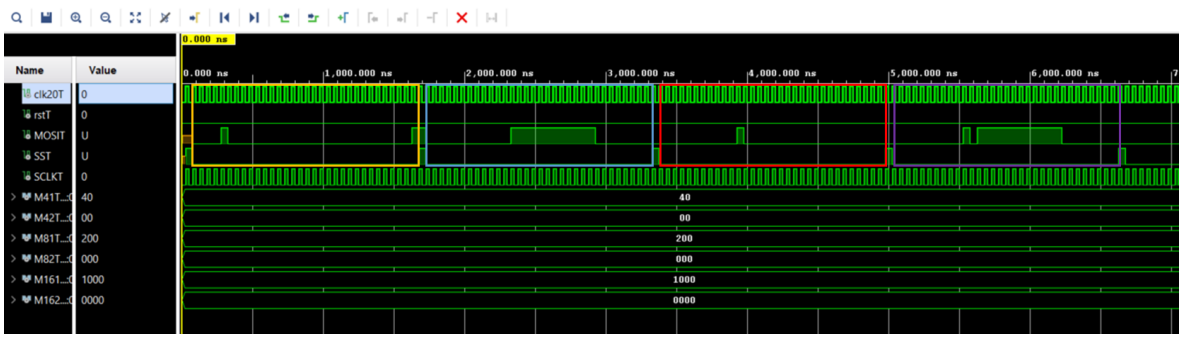

Abbildung 45: Simulationsergebnisse der Testbench zum Überprüfen des SPI-Moduls

Zuletzt wird das Design als Ganzes getestet. Im Vergleich zu den Simulationsergebnissen in Abbildung 45 wurden der SPI-Kommunikation in Abbildung 46 noch die Command-Bits „0011" hinzugefügt. Da Abbildung 45 gezeigt hat, dass das Timing eingehalten wird und die „Command"-Bits gesendet wurden, wenn auch nur als „0000", sind die Ergebnisse aus Abbildung 45 weiterhin korrekt. Die Änderung ist notwendig, da ohne sie zwar ein Betrieb möglich wäre, jedoch keine Änderung am Ausgang des DACs verursacht werden würde.

Gemäß der Testbench wird das Design zu Beginn der Simulation auf Kanal 1 nur mit Nullen und auf Kanal 2 nur mit Einsen versorgt. Da dies dem invertierten Input aus der SPI-Überprüfung entspricht, lassen sich in Abbildung 46, richtigerweise, die Ausgaben inklusive der neuen Command-Bits und invertierten Daten-Bits betrachten. Zwischen den positiven Impulsen des CS-Signals, vergehen jeweils 1600 ns, was bei einem 20 MHz Taktsignal, mit einer Periodendauer von 50 ns, genau den 32-Bits entspricht. Folglich ist sowohl die Verarbeitung durch die Filter korrekt, sowie auch die Vorbereitung und Übermittlung des SPI-Moduls.

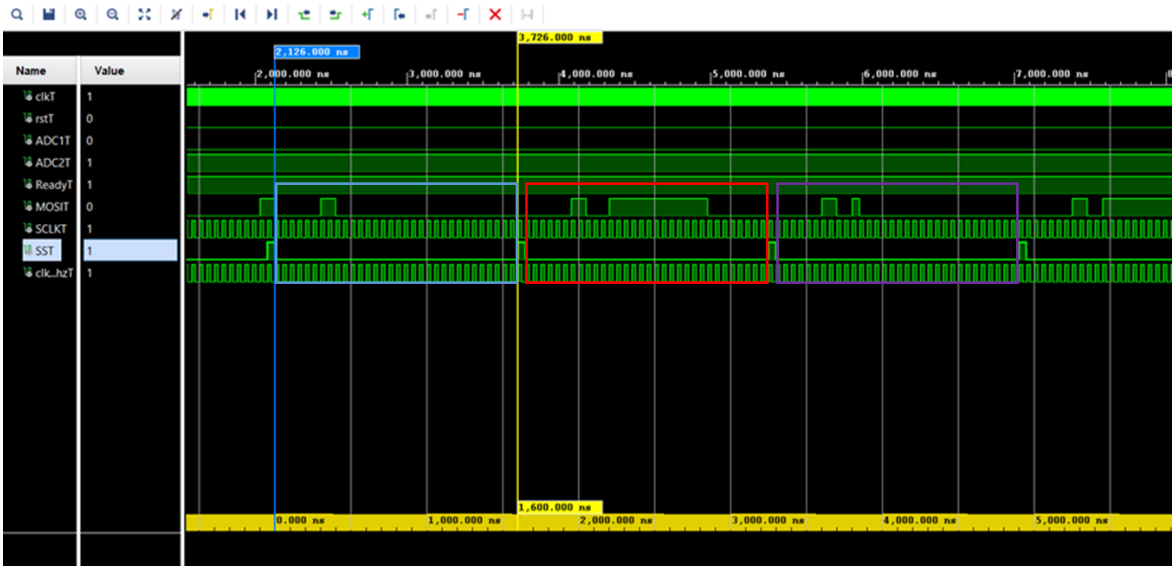

Abbildung 46: Simulationsergebnisse der Testbench zur Überprüfung des gesamten Designs

6.2 Messungen

Nach den Simulationen folgen die Messungen. Bei der Inbetriebnahme des Versuchsaufbaus zeigte sich, dass selbst bei einer Spannung von 0 V am Eingang der Modulatoren ein Unterschied zwischen den Ausgaben der Kanäle am DAC besteht. Das Signal, das vom ersten Modulator stammt und in Abbildung 47 als gelbe Kurve zu sehen ist, weist eine große Menge an Rauschen auf. Im Gegensatz zum anderen Kanal nimmt das Rauschen nicht ab,

sondern zu, selbst bei Erhöhung der Dezimierungsrate. Das Rauschen erstreckt sich mit einem Spitze-Spitze-Wert von 2,3 V fast über den gesamten 3 V Darstellungsbereich des DACs.

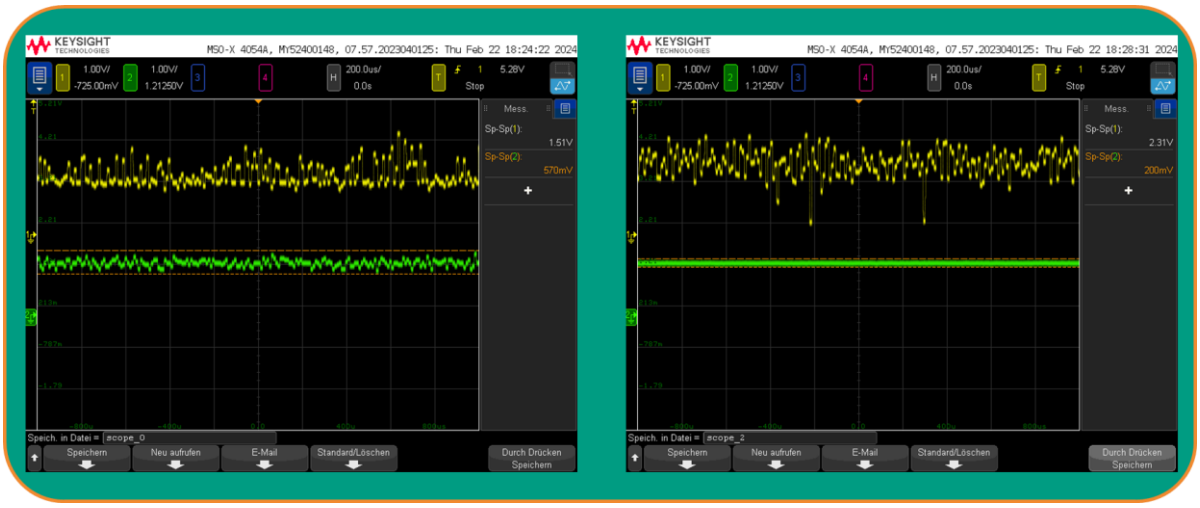

Abbildung 47: Oszillogramm der Ausgänge beider Kanäle am DAC nach der Verarbeitung durch die Filter mit einer Dezimierungsrate von 4 (links) und von 16 (rechts)

Durch den Wechsel der Eingänge in der Constraint Datei kann jedoch mit Abbildung 48 gezeigt werden, dass der Fehler nicht in der Verarbeitung, sondern in der Hardware liegt.

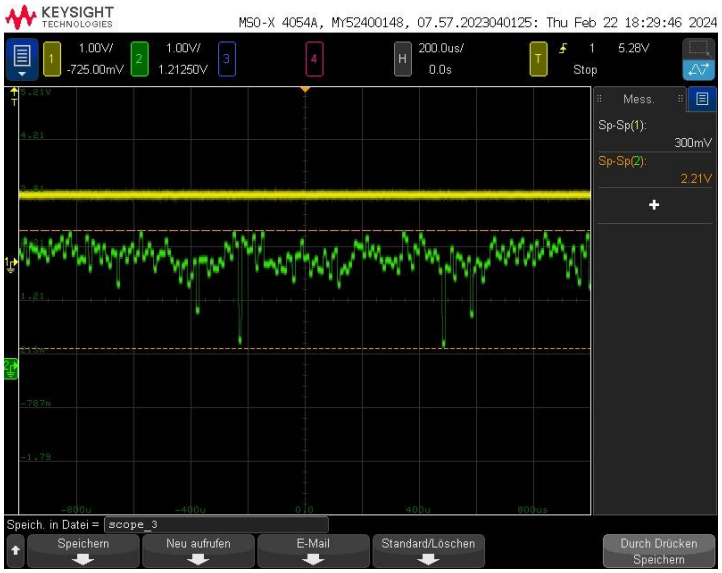

Abbildung 48: Rauschen bei getauschten Eingängen

Bei Betrachtung der Modulator-Signale in Kombination mit dem Takt wird deutlich, dass es bei einem der Modulatoren zu Signallaufzeitproblemen kommt. Die Modulator-Ausgabe, die in Abbildung 49 als grüner Verlauf zu sehen ist, ist zeitlich so verzögert, dass es während der Datenübernahme bei steigenden Taktflanken zu Pegelwechseln kommt.

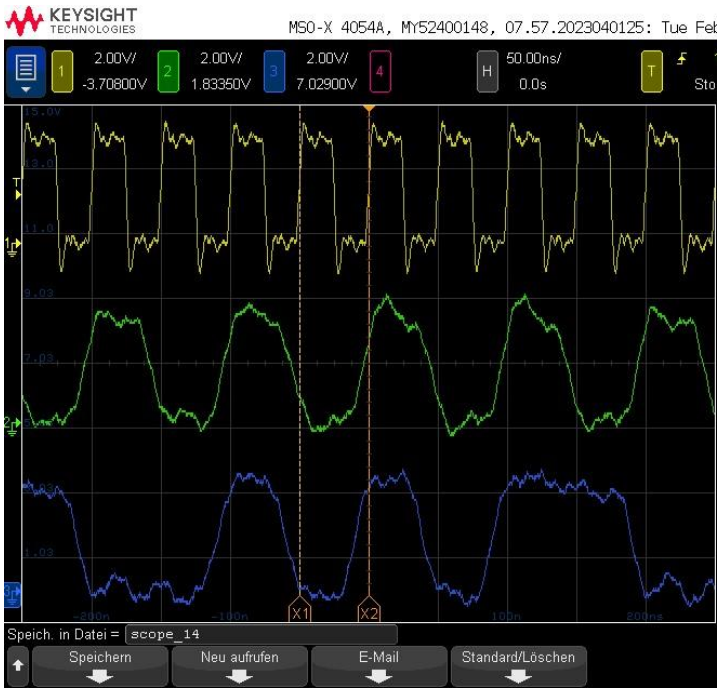

Abbildung 49:Timing-Probleme bei der Verwendung der Modulatoren

Auch der Ausgang des anderen Modulators ist zeitlich verzögert. Da diese Verzögerung jedoch mit dem Takt übereinstimmt, wirkt sie sich nicht auf die ausgelesenen Werte aus.

Eine Verkürzung der Leiterbahnen durch Eingriff in das Platinen-Layout könnte die Verzögerung verkürzen. Eine Anpassung der Leiterbahnen auf eine gemeinsame Länge hingegen würde die zeitlichen Unterschiede zwischen den Ausgangssignalen aufheben.

Da ein solcher Eingriff jedoch nicht möglich ist, bietet das VHDL-Design eine weitere Möglichkeit, indem es eine Verzögerung der Signale um wenige Nanosekunden vorsieht. Die Änderung erfordert, dass das Signal nicht mit 20 MHz, sondern mindestens mit 80 MHz eingelesen wird. Bei einer Frequenz von 80 MHz werden die Werte 12,5 ns nach dem Eintreten der positiven Taktflanke des 20 MHz Takts eingelesen und für die nächste positive Taktflanke des 20 MHz Takts bereitgestellt. Dadurch würde es zu einer späteren Erfassung der aktuellen Messdaten kommen. Diese kann jedoch durch die Reaktionszeit der Filter kompensiert werden.

Da das Design in Kombination mit dieser Hardware nicht mit maximaler Geschwindigkeit betrieben werden muss, reicht es aus, den Takt von 20 MHz auf 10 MHz zu halbieren. Durch die Halbierung des Taktes werden die Pegel der Ausgänge doppelt so lange gehalten und die Signalstabilität nimmt zu. Dadurch erhöht sich die Anzahl der erfolgreichen Einlese-Vorgänge. Die Änderung erfolgt im „Clocking Wizard" des Designs. Die dadurch entstehenden

Signale sind ausreichend zur Funktionskontrolle und für Demonstrationszwecke, entsprechen jedoch nicht der tatsächlichen Geschwindigkeit.

Abbildung 50 zeigt, dass die Halbierung des Taktes funktioniert. Das Rauschen des Modulators ist nun niedrig genug, um klare Werte zu erkennen. Dennoch besteht ein Unterschied von 300 mV zwischen den beiden Ausgangsspannungen. Dies kann durch die in Abbildung 39 gezeigte Schaltung und die bedingten Abweichungen ihrer Bauteile verursacht werden. Kleine Unterschiede in den Widerstandswerten können zu erheblichen Änderungen am Eingang des Modulators führen, insbesondere wenn diese durch den OP verstärkt werden. Es handelt sich hierbei nicht um einen Fehler, sondern um eine Größe, die bei der späteren Auswertung des Signals berücksichtigt werden muss.

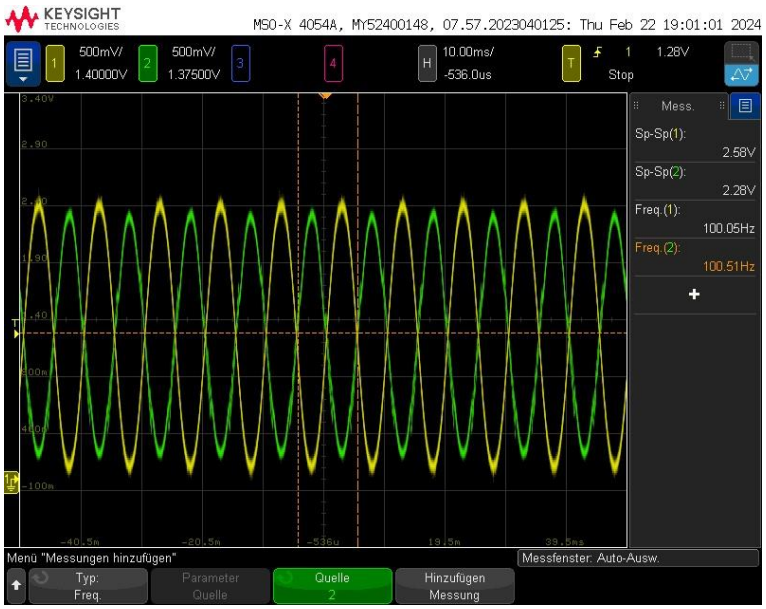

Abbildung 50: Ausgangsspannungen am DAC beider Kanäle mit 10 Mhz und einer Dezimierung von 16 bei einer Ansteuerung durch zwei phasenverschoben Sinus Kurven

Bei genauerer Betrachtung zeigt sich jedoch in Abbildung 51 weiterhin die Problematik mit der Latenz. Sobald es zu häufigen und schnellen Wechseln zwischen Nullen und Einsen kommt, nimmt das Rauschen zu. Der schnelle Wechsel tritt besonders um den Wendepunkt herum auf, da der Modulator am Wendepunkt eine 50%ige Aufteilung zwischen Nullen und Einsen produziert.

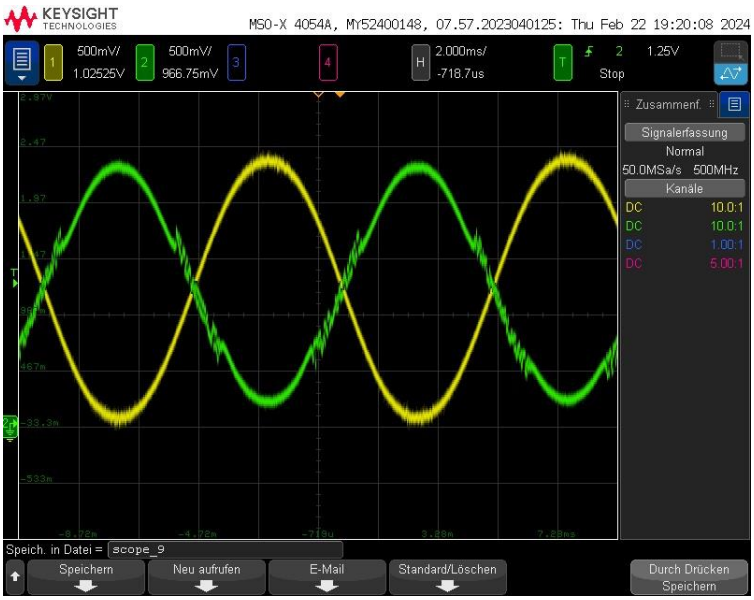

Abbildung 51: Rauschen trotz der Takt Halbierung

Für die Erstellung von Abbildung 52 wurden nur die Ergebnisse des voll funktionsfähigen Modulators verwendet. Die Abbildung zeigt die Unterschiede zwischen den verschiedenen Dezimierungsstufen auf. Obwohl die Filter vor der Ausgabe mehr als achtmal so schnell sind, ist klar erkennbar, dass eine Dezimierungsrate von vier bei einer Auflösung von 64 Stufen nicht geeignet ist, um einen Motor präzise zu steuern, aber ausreicht, um den Verlauf der Spannung, die den Strom repräsentiert, frühzeitig zu erkennen. Im Gegensatz dazu zeigt sich die Repräsentation mit einer Auflösung von 4095 Stufen als präziser mit einem deutlich geringeren Rauschanteil.

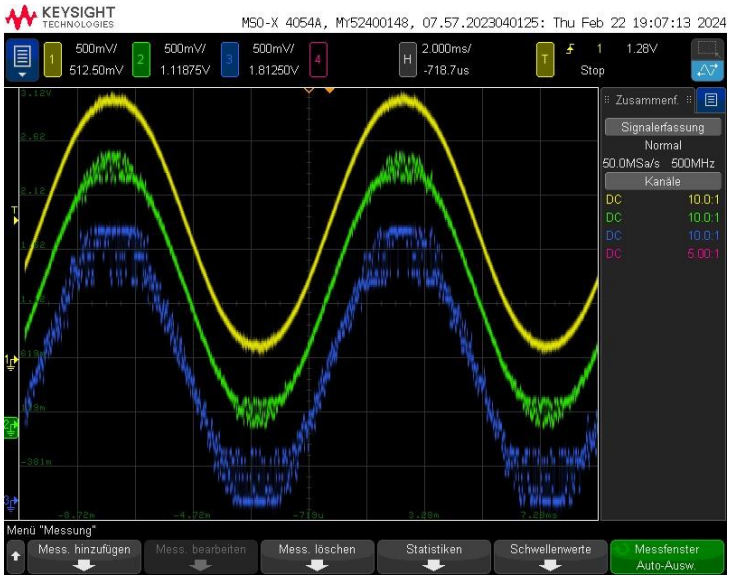

Abbildung 52: Ausgangsspannungen des DACs vom zweiten Kanal in allen Auflösungsstufen (Gelb = 4095; Grün = 512; Blau = 64)

Folglich verhält sich das Design nach kleinen Anpassungen wie erwartet. Die Auflösungsstufen sind für ihre Verwendungszwecke richtig gewählt und die Ausgänge repräsentieren den skalierten Wert des Eingangs.

7 Ausblick

Die Entwicklung des Sinc3-Filters zur Auswertung eines Sigma-Delta-Wandlers stellt einen der ersten Schritte für eine präzisen Motorsteuerung dar. Die erreichte Genauigkeit und Effizienz in der Signalumwandlung bieten eine solide Grundlage für die Weiterentwicklung und Implementierung in komplexeren Steuerungssystemen.

Zukünftige Entwicklungsarbeiten könnten sich darauf konzentrieren, den Filter in ein FOC-System zu integrieren, um Elektromotoren präziser steuern zu können.

Ein bedeutender Vorteil der Verwendung des entwickelten Sinc3-Filters in einem FOC-System besteht darin, dass die Qualität der Strom- und Spannungssignale, die vom Sigma-Delta-Wandler geliefert werden, angepasst werden können. Dadurch könnte eine präzisere Steuerung des Motors erreicht werden, da der Filter effektiv Rauschen und unerwünschte Frequenzkomponenten unterdrückt. Durch diese Maßnahmen würde die Effizienz und Leistung des Motors gesteigert und gleichzeitig der Energieverbrauch reduziert werden.

Außerdem könnte untersucht werden, wie man die Anpassungsfähigkeit des Filters optimieren kann, um ihn für verschiedene Motortypen und Anwendungsbedingungen maßzuschneidern. Hierbei ist es wichtig, die Filterparameter und das Design gegebenenfalls aufgrund von hardwarebedingten Latenzen anzupassen.

8 Literaturverzeichnis

[1] Homofacienes, „Homofacienes," 20 02 2018. [Online]. Available: https://www.homofaciens.de/technics-electric-motors-synchronous-motor_ge.htm. [Zugriff am 12 02 2024].

[2] P. D.-I. R. Fischer, Elektrische Maschinen, München: Carl Hanser Verlag, 2013.

[3] Texas Instruments Europa, „ti," 02 1998. [Online]. Available: https://www.ti.com/lit/an/bpra073/bpra073.pdf. [Zugriff am 20 02 2024].

[4] A. S. W. S. H. N. Alan V. Oppenheim, Signals and Systems, Prentice Hall, 1996.

[5] R. W. S. Alan V. Oppenheim, Zeitdiskrete Signalverarbeitung, München: Oldenbourg Verlag, 1999.

[6] S. Finke, „Umsetzung eines Error-Feedback Modulators mit Verilog als Basisaufbau für ein FPGA Synthesizer," FH Dortmund, Dortmund, Deutschland, 2023.

[7] S. /. S. R. /. T. G. C. Pavan, Understanding Delta-Sigma Data Converters, Wiley & Sons Ltd, 2017.

[8] R. G. Lyons, Understanding Digital Signal Processing, Prentice Hall, 2010.

[9] R. Schreier, „oregonstate," 03 2008. [Online]. Available: https://classes.engr.oregonstate.edu/eecs/spring2021/ece627/Lecture%20Notes/2nd%20&%20Higher-Order2.pdf. [Zugriff am 10 03 2024].

[10] Texas Instruments, „ti.com," 2011. [Online]. Available: https://www.ti.com/lit/an/slyt438/slyt438.pdf?ts=1710117436473&ref_url=https%253A%252F%252Fwww.google.com%252F. [Zugriff am 22 02 2024].

[11] J. Axelson, Serial Port Complete: COM Ports, USB Virtual COM Ports, and Ports for Embedded Systems, Lakeview Research, 2007.

[12] Digilent, „digilent.com," Digilent, 28 11 2016. [Online]. Available: https://digilent.com/reference/_media/reference/pmod/pmodda4/pmodda4_rm.pdf. [Zugriff am 13 02 2024].

[13] Analog Devices, „analog.com," Analog Devices, 22 12 2023. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/ad7403.pdf. [Zugriff am 14 02 2024].

[14] A. Errapart, „trenz-electronics," 26 10 2017. [Online]. Available: https://wiki.trenz-electronic.de/display/PD/EDPS+User+Manual. [Zugriff am 18 02 2024].

[15] LEM, „LEM.com," 02 2009. [Online]. Available: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwikmKi1ws6EAxXmg_0HHZ2BCrIQFnoECA0QAQ&url=https%3A%2F%2Fwww.lem.com%2Fen%2Ffile%2F5636%2Fdownload&usg=AOvVaw0q5Eg5m4C0WOQzevlwre7u&opi=89978449. [Zugriff am 18 02 2024].

[16] Digilent , „digilent.com," [Online]. Available: https://digilent.com/reference/programmable-logic/arty-z7/reference-manual. [Zugriff am 06 02 2024].

[17] T. H. Miroslav Oljaca, „ti.com," 06 2003. [Online]. Available: https://www.ti.com/lit/an/sbaa094/sbaa094.pdf?ts=1709055288348&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FADS1204%253Futm_source%253Dgoogle%2526utm_medium%253Dcpc%2526utm_campaign%253Dasc-dc-null-44700045496406805_prodfolderdynamic-cpc-pf-goog. [Zugriff am 12 02 2024].

[18] J. Noss, „Framework für den SoC XC7Z020-1CLG400C (PWM-Generator)," Fh Dortmund, Dortmund, Deutschland, 2023.

[19] trenz electronic, „trenz-electronik.de," trenz elektronik, 27 06 2019. [Online]. Available: https://shop.trenz-electronic.de/trenzdownloads/Trenz_Electronic/Development_Boards/TEC0053/REV04/Documents/SCH-TEC0053-04.PDF. [Zugriff am 18 02 2024].

9 Bildverzeichnis

10 Listing

11 Tabellenverzeichnis

12 Abkürzungsverzeichnis

VHDL	Very High Speed Integrated Circuit Hardware Description Language
FPGA	Field Programmable Gate Array
DAC	Digital-Analog-Wandler
ADC	Analog-Digital-Wandler
FOC	Field Oriented Control
SPI	Serial Peripheral Interface
MOSI	Master-Out-Slave-In
MISO	Master-In-Slave-Out
SNR	Signal-Noise-Ratio
ENOB	Effective Number of Bits

13 Anhang

13.1 VHDL-Design des SINC3 M4 Filters

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.std_logic_unsigned.all;

use IEEE.numeric_std.all;

entity SINC3M4 is
   port(
      clk20:     in std_logic;
      rst:       in std_logic;
      ADCin:     in std_logic;

      ADCout:    out std_logic_vector (6 downto 0)
      );
end SINC3M4;

architecture Behavioral of SINC3M4 is

   signal ADDin, ADD1, ADD2:   std_logic_vector (6 downto 0):= "0000000";

   signal SUB11, SUB12, SUB21, SUB22, SUB31, SUB32:   std_logic_vector (6 downto 0):=
"0000000";

   signal M: std_logic_vector (1 downto 0) := "00";

begin

   process(clk20,rst)
   begin
-------------------------------------------- asyncroner Reset
      if (rst = '0') then
         ADDin <= "0000000";
         ADD1  <= "0000000";
         ADD2  <= "0000000";
         SUB11 <= "0000000";
         SUB12 <= "0000000";
         SUB21 <= "0000000";
         SUB31 <= "0000000";
         M <= "00";

      elsif rising_edge(clk20) then
-------------------------------------------- Integrator
         if (ADCin = '1') then
```

```
            ADDin <= ADDin + 1;
         end if;

         ADD1 <= ADD1 + ADDin;
         ADD2 <= ADD2 + ADD1;
------------------------------------------------ Differenzierer Teil 1
         if (M = "11") then
            SUB12 <= ADD2;
            SUB11 <= SUB12;
            SUB21 <= SUB22;
            SUB31 <= SUB32;
         end if;
------------------------------------------------ Counter Inkrementieren für das Dezimieren
         M <= M + 1;
      end if;
   end process;

------------------------------------------------ Differenzier Teil 2
   SUB22  <= SUB12 - SUB11;
   SUB32  <= SUB22 - SUB21;
   ADCout <= SUB32 - SUB31;

end Behavioral;
```

13.2 VHDL-Design des SINC3 M8 Filters

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;

entity SINC3M8 is
   port(
      clk20:  in std_logic;
      rst:    in std_logic;
      ADCin1: in std_logic;
      ADCin2: in std_logic;

      ADCout1: out std_logic_vector (9 downto 0);
      ADCout2: out std_logic_vector (9 downto 0)
      );
end SINC3M8;

architecture Behavioral of SINC3M8 is

   signal ADDin, ADD1, ADD2: std_logic_vector (9 downto 0):= "0000000000";

   signal SUB11, SUB12, SUB21, SUB22, SUB31, SUB32: std_logic_vector (9 downto 0):=
"0000000000";

   signal M: std_logic_vector (2 downto 0):= "000";

begin
```

```
   process(clk20,rst)

   variable input: std_logic := '0';
   variable changecnt: std_logic_vector (2 downto 0) := "000";

   begin

------------------------------------------------------------ asyncroner Reset
      if (rst = '0') then
         ADDin      <= "0000000000";
         ADD1       <= "0000000000";
         ADD2       <= "0000000000";
         SUB11      <= "0000000000";
         SUB12      <= "0000000000";
         SUB21      <= "0000000000";
         SUB31      <= "0000000000";
         M          <= "000";
         ADCout1    <= "0000000000";
         ADCout2    <= "0000000000";
         changeCNT   := "000";
         input      := '0';
------------------------------------------------------------ Input Verschaltung
      elsif rising_edge(clk20) then

------------------------------------------------------------ Differenzierer Teil 1

         if (M = "111") then
            SUB12 <= ADD2;
            SUB11 <= SUB12;
            SUB21 <= SUB22;
            SUB31 <= SUB32;

------------------------------------------------------------ Counter für den Output-Wechsel (Verlauf)
            --if changeCNT <= "011" and changeCNT > "000" then
            if changeCNT = "011" then
               ADCout1 <= SUB32 - SUB31;
            elsif changeCNT = "000" then
               ADCout2 <= SUB32 - SUB31;
            end if;

         changeCNT := changeCNT+1;

         end if;
------------------------------------------------------------ Counter für den Input-Wechsel
         if changeCNT = "110" then
           changeCNT := "000";
         end if;

         if changeCNT <= "010" then
            input := ADCin1;
         else
```

```
            input:= ADCin2;
         end if;

---------------------------------------------------------- Integrator
         if (input = '1')then
            ADDin <= ADDin + 1;
         end if;


         ADD1 <= ADD1 + ADDin;
         ADD2 <= ADD2 + ADD1;

---------------------------------------------------------- Counter Inkrementieren für den Dezimator
         M <= M + 1;

      end if;
   end Process;

---------------------------------------------------------- Differenzier Teil 2

    SUB22  <= SUB12 - SUB11 ;
    SUB32  <= SUB22 - SUB21 ;

end Behavioral;
```

13.3 VHDL-Design des SINC3 M16 Filters

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;

entity SINC3M16 is
   port(
      clk20:   in std_logic;
      rst:     in std_logic;
      ADCin1:  in std_logic;
      ADCin2:  in std_logic;

      ADCout1: out std_logic_vector (12 downto 0);
      ADCout2: out std_logic_vector (12 downto 0)
      );
end SINC3M16;

architecture Behavioral of SINC3M16 is

   signal input: std_logic:= '0';

   signal ADDin, ADD1, ADD2:   std_logic_vector (12 downto 0):= "0000000000000";
```

```
   signal SUB11, SUB12, SUB21, SUB22, SUB31, SUB32:    std_logic_vector (12 downto 0):=
"0000000000000";

   signal M: std_logic_vector (3 downto 0):= "0000";

begin



   process(clk20,rst)

   variable input: std_logic := '0';
   variable changecnt: std_logic_vector (2 downto 0) := "000";

   begin

------------------------------------------------------------ asyncroner Reset
      if (rst = '0') then
         ADDin      <= "0000000000000";
         ADD1       <= "0000000000000";
         ADD2       <= "0000000000000";
         SUB11      <= "0000000000000";
         SUB12      <= "0000000000000";
         SUB21      <= "0000000000000";
         SUB31      <= "0000000000000";
         M          <= "0000";
         ADCout1    <= "0000000000000";
         ADCout2    <= "0000000000000";
         changeCNT   := "000";
         input      := '0';

      elsif rising_edge(clk20) then
------------------------------------------------------------ Differenzierer Teil 1
         if (M = "1111") then
            SUB12 <= ADD2;
            SUB11 <= SUB12;
            SUB21 <= SUB22;
            SUB31 <= SUB32;

------------------------------------------------------------ Counter für den Output-Wechsel (Verlauf)
            --if changeCNT <= "011" and changeCNT > "000" then
            if changeCNT = "011" then
               ADCout1 <= SUB32 - SUB31;
            elsif changeCNT = "000" then
               ADCout2 <= SUB32 - SUB31;
            end if;

         changeCNT := changeCNT+1;
         end if;
------------------------------------------------------------ Counter für den Intput-Wechsel

         if changeCNT = "110" then
```

```
          changeCNT := "000";
        end if;

        if changeCNT <= "010" then
          input := ADCin1;
        else
          input:= ADCin2;
        end if;

------------------------------------------------------------ Integrator
        if (input = '1')then
          ADDin <= ADDin + 1;
        end if;

        ADD1 <= ADD1 + ADDin;
        ADD2 <= ADD2 + ADD1;

        M <= M + 1;
      end if;
   end Process;

------------------------------------------------------------ Differenzier Teil 2
    SUB22  <= SUB12 - SUB11;
    SUB32  <= SUB22 - SUB21;


end Behavioral;
```

13.4 VHDL-Design des Debounce-Moduls

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity debounce is
          Port (
                    clk_d : in std_logic;
          input_d : in std_logic;
          output_d : out std_logic := '0';
                    rise_d : out std_logic := '0');

end debounce;

architecture Behavioral of debounce is
          signal input_vec   : std_logic_vector(15 downto 0);
begin

          process (clk_d)
          begin

      if rising_edge(clk_d) then
```

```
                -- von rechts Eintakten
                input_vec <= input_vec(14 downto 0) & input_d;

                -- Zustaende zuweisen
                case input_vec is
            when "1111111111111111" =>
            rise_d <= '0';
            output_d <= '1';
            -- rising edge
            when "0111111111111111" =>
            rise_d <= '1';
            output_d <= '1';
            --alles andere
            when others =>
            rise_d <= '0';
            output_d <= '0';
                            end case;
                end if;
        end process;

end Behavioral;
```

13.5 VHDL-Design des SPI-Moduls

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity SPI_Vorbereitung is
  Port (
    clk20: in std_logic;
    rst: in std_logic;
    M41,M42: in std_logic_vector (6 downto 0);
    M81,M82: in std_logic_vector (9 downto 0);
    M161,M162: in std_logic_vector (12 downto 0);

    MOSI: out std_logic;
    SS: out std_logic;
    SCLK: out std_logic
    );
end SPI_Vorbereitung;

architecture Behavioral of SPI_Vorbereitung is
signal M41_scaled, M42_scaled: std_logic_vector (18 downto 0);
signal M81_scaled, M82_scaled: std_logic_vector (21 downto 0);
signal M161_scaled, M162_scaled: std_logic_vector (24 downto 0);
signal scale: std_logic_vector (11 downto 0):= "111111111111";
signal cnt: integer range 0 to 32:= 0;
signal in_select: integer range 0 to 5:= 0;
signal data: std_logic_vector (31 downto 0):= "00001000000000000000000000000001";
```

```
begin

SCLK <= clk20;

 process(clk20, rst)
  begin
    if (rst = '0') then
      M41_scaled <= "00000000000000000000";
      M42_scaled <= "00000000000000000000";
      M81_scaled <= "00000000000000000000000";
      M82_scaled <= "00000000000000000000000";
      M161_scaled <= "00000000000000000000000000";
      M162_scaled <= "00000000000000000000000000";
      data <= "00001000000000000000000000000001";
      scale <= "111111111111";
      cnt <= 0;
      in_select <= 0;

    elsif rising_edge(clk20)then

      M41_scaled <= std_logic_vector(unsigned(M41) * unsigned(scale));
      M42_scaled <= std_logic_vector(unsigned(M42) * unsigned(scale));
      M81_scaled <= std_logic_vector(unsigned(M81) * unsigned(scale));
      M82_scaled <= std_logic_vector(unsigned(M82) * unsigned(scale));
      M161_scaled <= std_logic_vector(unsigned(M161) * unsigned(scale));
      M162_scaled <= std_logic_vector(unsigned(M162) * unsigned(scale));

      if (cnt = 0) then
        ss <= '1';
      else
        ss <= '0';
        MOSI <= data(32-cnt);
      end if;

      if (cnt = 32) then
        data (31 downto 24)<= "00000011";
        data (7 downto 0)<= "00000000";

        Case in_select is
         when 0 => data(23 downto 8) <= "0000" & M41_scaled(17 downto 6);
         when 1 => data(23 downto 8) <= "0001" & M42_scaled(17 downto 6);
         when 2 => data(23 downto 8) <= "0010" & M81_scaled(20 downto 9);
         when 3 => data(23 downto 8) <= "0011" & M82_scaled(20 downto 9);
         when 4 => data(23 downto 8) <= "0100" & M161_scaled(23 downto 12);
         when 5 => data(23 downto 8) <= "0101" & M162_scaled(23 downto 12);
        end case;

        if (in_select = 5) then
          in_select <= 0;
        else
          in_select <= in_select+1;
```

```
            end if;

            cnt <= 0;
          else
            data <= data;
            cnt <= cnt + 1;
          end if;

       end if;

     end process;
```

13.6 Testbench-Design zur Überprüfung der Filter

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity SINC3Testbench is
--  Port ( );
end SINC3Testbench;

architecture Behavioral of SINC3Testbench is
component BAblockdesign_wrapper
   port(
      clk, rst, ADC1, ADC2 : in std_logic;
      M4out1, M4out2 : out std_logic_vector(6 downto 0);
      M8out1, M8out2 : out std_logic_vector(9 downto 0);
      M16out1, M16out2 : out std_logic_vector(12 downto 0);
      clk_20Mhz : out std_logic;
      Ready : out std_logic
      );
end component;

signal clkT, rstT, ADC1T, ADC2T,clkDivT,ReadyT : std_logic := '0';
signal M4out1T, M4out2T : std_logic_vector(6 downto 0):= "0000000";
signal M8out1T, M8out2T : std_logic_vector(9 downto 0):= "0000000000";
signal M16out1T, M16out2T : std_logic_vector(12 downto 0):= "0000000000000";

begin

DUT: BAblockdesign_wrapper port map
(
 clk => clkT, clk_20Mhz => clkDivT, Ready => ReadyT, rst => rstT, ADC1 => ADC1T, ADC2 => ADC2T,
M4out1 => M4out1T, M4out2 => M4out2T, M8out1 => M8out1T, M8out2 => M8out2T, M16out1
=> M16out1T, M16out2 => M16out2T
);

clkT <= not clkT after 4 ns;
```

```
process begin
   rstT <= '0';
   ADC2T <= '1';
   ADC1T <= '0';
   wait for 10000ns;
   ADC2T <= '0';
   ADC1T <= '1';
   wait for 5000ns;
   rstT <= '1';
   wait for 5000ns;
end process;

end Behavioral;
```

13.7 Testbench-Design zur Überprüfung des SPI-Moduls

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity SPI_Testbench is
--  Port ( );
end SPI_Testbench;



architecture Behavioral of SPI_Testbench is

component SPI_Vorbereitung is
  Port (
   clk20: in std_logic;
   rst: in std_logic;
   M41,M42: in std_logic_vector (6 downto 0);
   M81,M82: in std_logic_vector (9 downto 0);
   M161,M162: in std_logic_vector (12 downto 0);

   MOSI: out std_logic;
   SS: out std_logic;
   SCLK: out std_logic
   );
end component;

signal clk20T, rstT, MOSIT, SST, SCLKT: std_logic:= '0';
signal M41T, M42T: std_logic_vector (6 downto 0);
signal M81T, M82T: std_logic_vector (9 downto 0);
signal M161T, M162T: std_logic_vector (12 downto 0);

begin

DUT: SPI_Vorbereitung port map
```

```
(
   clk20 => clk20T, rst => rstT, MOSI => MOSIT, SS => SST, SCLK => SCLKT, M41 => M41T, M42 =>
M42T, M81 => M81T, M82 => M82T, M161 => M161T, M162 => M162T
);

clk20T <= not clk20T after 25ns;

rstT <= '1';

M41T <= "1000000";
M42T <= "0000000";
M81T <= "1000000000";
M82T <= "0000000000";
M161T <= "1000000000000";
M162T <= "0000000000000";

end Behavioral;
```

13.8 Testbench-Design zur Überprüfung des vollständigen Designs

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;



entity SINC3SPITestbench is
--  Port ( );
end SINC3SPITestbench;

architecture Behavioral of SINC3SPITestbench is

component BAblockdesign_wrapper is
  port (
    ADC1 : in STD_LOGIC;
    ADC2 : in STD_LOGIC;
    MOSI : out STD_LOGIC;
    Ready : out STD_LOGIC;
    SCLK : out STD_LOGIC;
    SS : out STD_LOGIC;
    clk : in STD_LOGIC;
    clk_20Mhz : out STD_LOGIC;
    rst : in STD_LOGIC
  );
end component;

signal clkT, rstT, ADC1T, ADC2T, ReadyT, MOSIT, SCLKT, SST, clk_20MhzT : std_logic := '0';

begin

DUT: BAblockdesign_wrapper port map
(
```

```
 clk => clkT, clk_20Mhz => clk_20MhzT, Ready => ReadyT, rst => rstT, ADC1 => ADC1T, ADC2 =>
ADC2T, SS => SST, MOSI => MOSIT, SCLK => SCLKT
);

clkT <= not clkT after 4ns;

process
begin

   rstT <= '0';
   ADC2T <= '1';
   ADC1T <= '0';
   wait for 10000ns;
   ADC2T <= '0';
   ADC1T <= '1';
   wait for 5000ns;
   rstT <= '1';
   wait for 5000ns;
end process;
end Behavioral;
```

13.9 Constraint-Datei

```
## Clock Signal
set_property -dict { PACKAGE_PIN H16    IOSTANDARD LVCMOS33 } [get_ports {clk}];
#IO_L13P_T2_MRCC_35 Sch=SYSCLK
#create_clock -add -name sys_clk_pin -period 8.00 -waveform {0 4} [get_ports { CLK }];#set

## Switches
#set_property -dict { PACKAGE_PIN M20    IOSTANDARD LVCMOS33 } [get_ports { sw[0] }];
#IO_L7N_T1_AD2N_35 Sch=SW0
#set_property -dict { PACKAGE_PIN M19    IOSTANDARD LVCMOS33 } [get_ports { sw[1] }];
#IO_L7P_T1_AD2P_35 Sch=SW1

## RGB LEDs
#set_property -dict { PACKAGE_PIN L15    IOSTANDARD LVCMOS33 } [get_ports { led4_b }];
#IO_L22N_T3_AD7P_35 Sch=LED4_B
#set_property -dict { PACKAGE_PIN G17    IOSTANDARD LVCMOS33 } [get_ports { led4_g }];
#IO_L16P_T2_35 Sch=LED4_G
#set_property -dict { PACKAGE_PIN N15    IOSTANDARD LVCMOS33 } [get_ports { led4_r }];
#IO_L21P_T3_DQS_AD14P_35 Sch=LED4_R
#set_property -dict { PACKAGE_PIN G14    IOSTANDARD LVCMOS33 } [get_ports { led5_b }];
#IO_0_35 Sch=LED5_B
#set_property -dict { PACKAGE_PIN L14    IOSTANDARD LVCMOS33 } [get_ports { led5_g }];
#IO_L22P_T3_AD7P_35 Sch=LED5_G
#set_property -dict { PACKAGE_PIN M15    IOSTANDARD LVCMOS33 } [get_ports { led5_r }];
#IO_L23N_T3_35 Sch=LED5_R

## LEDs
set_property -dict { PACKAGE_PIN R14    IOSTANDARD LVCMOS33 } [get_ports { Ready }];
#IO_L6N_T0_VREF_34 Sch=LED0
```

```
#set_property -dict { PACKAGE_PIN P14    IOSTANDARD LVCMOS33 } [get_ports { led[1] }];
#IO_L6P_T0_34 Sch=LED1
#set_property -dict { PACKAGE_PIN N16    IOSTANDARD LVCMOS33 } [get_ports { led[2] }];
#IO_L21N_T3_DQS_AD14N_35 Sch=LED2
#set_property -dict { PACKAGE_PIN M14    IOSTANDARD LVCMOS33 } [get_ports { led[3] }];
#IO_L23P_T3_35 Sch=LED3

## Buttons
#set_property -dict { PACKAGE_PIN D19    IOSTANDARD LVCMOS33 } [get_ports { commit_0 }];
#IO_L4P_T0_35 Sch=BTN0
#set_property -dict { PACKAGE_PIN D20    IOSTANDARD LVCMOS33 } [get_ports { btn[1] }];
#IO_L4N_T0_35 Sch=BTN1
#set_property -dict { PACKAGE_PIN L20    IOSTANDARD LVCMOS33 } [get_ports { btn[2] }];
#IO_L9N_T1_DQS_AD3N_35 Sch=BTN2
set_property -dict { PACKAGE_PIN L19    IOSTANDARD LVCMOS33 } [get_ports { rst }];
#IO_L9P_T1_DQS_AD3P_35 Sch=BTN3

## Pmod Header JA
set_property -dict { PACKAGE_PIN Y18   IOSTANDARD LVCMOS33 } [get_ports { SS }];
#IO_L17P_T2_34 Sch=JA1_P (Pin 1)
set_property -dict { PACKAGE_PIN Y19   IOSTANDARD LVCMOS33 } [get_ports { MOSI }];
#IO_L17N_T2_34 Sch=JA1_N (Pin 2)
#set_property -dict { PACKAGE_PIN Y16   IOSTANDARD LVCMOS33 } [get_ports {  }];
#IO_L7P_T1_34 Sch=JA2_P (Pin 3)
set_property -dict { PACKAGE_PIN Y17   IOSTANDARD LVCMOS33 } [get_ports { SCLK }];
#IO_L7N_T1_34 Sch=JA2_N (Pin 4)
#set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports { M41[4] }];
#IO_L12P_T1_MRCC_34 Sch=JA3_P (Pin 7)
#set_property -dict { PACKAGE_PIN U19   IOSTANDARD LVCMOS33 } [get_ports { M41[5] }];
#IO_L12N_T1_MRCC_34 Sch=JA3_N (Pin 8)
#set_property -dict { PACKAGE_PIN W18   IOSTANDARD LVCMOS33 } [get_ports { M41[6] }];
#IO_L22P_T3_34 Sch=JA4_P (Pin 9)
#set_property -dict { PACKAGE_PIN W19   IOSTANDARD LVCMOS33 } [get_ports { ADC1Out }];
#IO_L22N_T3_34 Sch=JA4_N (Pin 10)

## Pmod Header JB
set_property -dict { PACKAGE_PIN W14   IOSTANDARD LVCMOS33 } [get_ports { clk_20Mhz }];
#IO_L8P_T1_34 Sch=JB1_P (Pin 1)
set_property -dict { PACKAGE_PIN Y14   IOSTANDARD LVCMOS33 } [get_ports { ADC1 }];
#IO_L8N_T1_34 Sch=JB1_N (Pin 2)
set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33 } [get_ports { ADC2 }];
#IO_L1P_T0_34 Sch=JB2_P (Pin 3)
```

14 Datenträger