Fachhochschule Westküste
Hochschule für Wirtschaft & Technik

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Sönke Appel

Konzeption, Implementierung und Test eines Demonstrators auf FPGA-Basis zur Datenübertragung mit Millimeterwellen-Frontends

Fachhochschule Westküste
Fachbereich Technik

Hochschule für Angewandte Wissenschaften Hamburg
Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Fachhochschule Westküste - University of Applied Sciences
Faculty of Engineering

Hamburg University of Applied Sciences
Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Sönke Appel

Konzeption, Implementierung und Test eines Demonstrators auf FPGA-Basis zur Datenübertragung mit Millimeterwellen-Frontends

Masterarbeit eingereicht im Rahmen der Masterprüfung
im gemeinsamen Masterstudiengang Mikroelektronische Systeme
am Fachbereich Technik
der Fachhochschule Westküste
und
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.-Ing. Stephan Hußmann
Zweitgutachter : Prof. Dr.-Ing. Hans Peter Kölzer

Abgegeben am 24. Mai 2012

Sönke Appel

Thema der Masterarbeit

Konzeption, Implementierung und Test eines Demonstrators auf FPGA-Basis zur Datenübertragung mit Millimeterwellen-Frontends

Stichworte

FPGA, VHDL, System-Generator, Modulation, Demodulation, OOK, DPSK, inkohärent, ADC, DAC, Clock-Board

Kurzzusammenfassung

Diese Arbeit umfasst die Entwicklung eines Demonstrators auf FPGA-Basis für eine neue Generation von Millimeterwellen-Frontends. Zur Demonstration werden digital modulierte Signale über die Millimeterwellen-Frontends gesendet und empfangen. Teil der Entwicklung ist die Anbindung eines Sender-FPGAs an einen DAC, sowie eine Anbindung eines ADCs an einen Empfänger-FPGA. Eine Bitfehlerratenmessung wird zur Evaluierung der Frontends und der verwendeteten Modulationsarten verwendet. Im Zuge der Arbeit werden optimierte System-Generator-Module untersucht sowie alternative Modulationsarten, die mit den besonderen Eigenschaften der Millimeterwellen-Frontends arbeiten.

Sönke Appel

Title of the paper

Design, implementation and test of a FPGA-based demonstrator for data transmission with millimeter-wave front ends

Keywords

FPGA, VHDL, System-Generator, modulation, demodulation, OOK, DPSK, incoherent, ADC, DAC, Clock-Board

Abstract

This work includes the development of a FPGA-based demonstrator for a new generation of millimeter-wave front ends. For demonstration digital modulated signals are sent and received over the millimeter wave front-ends. Part of the development is to connect a transmitter-FPGA to an DAC, as well as to link a ADC to a receiver-FPGA. A bit error rate measurement is used for the evaluation of the front end and the different kinds of modulation. In the course of the work there are optimized System-Generator-modules examined, and alternative modulation schemes where tested that work with the special properties of the millimeter wave front ends.

Inhaltsverzeichnis

Abkürzungsverzeichnis

ADC	Analog-to-digital converter.
ASK	Amplitude Shift Keying.
AWGN	additive white gaussian noise.
BER	Bit error rate.
BMBF	Bundesministerium für Bildung und Forschung.
BP	Bandpass.
BPSK	Binary-Phase-shift keying.
CDR	Clock and Data Recovery.
CFO	Carrier Frequency Offset.
CORDIC	Coordinate Rotation Digital Computer.
CPM	Continuous-Phase-Modulation.
DAC	Digital-to-analog converter.
DBPSK	differential-Binary-Phase-shift keying.
DCI	Digitally Controlled Impedance.
DCM	Digital Clock Manager.
DPSK	differential-Phase-shift keying.
DQPSK	differential-Quarternary-Phase-shift keying.
DSP	Digitaler Signalprozessor.
ENOB	Effective number of bits.
FPGA	Field Programmable Gate Array.
FSM	Finite-state machine.
GPU	Graphics Processing Unit.
IBUFG	Dedicated Input Clock Buffer.
IBUFGDS	Differential Signaling Dedicated Input Clock Buffer and Optional Delay.

IDELAYCTRL	Tap Delay Value Control.
IODELAY	Input and Output Fixed or Variable Delay Element.
IQ-Mischer	In-phase-Quadratur-phase-Mischer.
ISI	Intersymbolinterferenz.
JTAG	Joint Test Action Group.
LFSR	linear feedback shift register.
LO	Lokal Oszillator.
LUT	Lookup table.
LVDS	Low Voltage Differential Signaling.
MILLILINK	Millimeterwellen-Drahtlos-Links.
MMIC	Monolithic microwave integrated circuit.
NCO	Numerically Controlled Oscillator.
NF	Noise floor.
OBUF	Output Buffer.
ODDR	Dedicated Dual Data Rate Output Register.
OFDM	Orthogonal Frequency Division Multiplexing.
OOK	On-Off Keying.
PAPR	peak-to-average power ratio.
PLL	Phase-Locked Loop.
PMPR	peak-to-minimum power ratio.
PMSPR	peak-to-minimum-space power ratio.
PSK	Phase-shift keying.
QAM	Quadratur-Amplituden-Modulation.
QPSK	Quarternary-Phase-shift keying.
RAM	Random-Access-Memory.
RMS	Root mean square.
ROM	Read-only memory.
RTL	Register Transfer Level.
SNR	Signal-to-noise ratio.
VCO	Voltage Controlled Oscillator.

VHDL	Very High Speed Integrated Circuit Hardware Description Language.

Teil I.

Konzeption

1. Das MILLILINK-Projekt

Millimeterwellen-Drahtlos-Links (MILLILINK) ist ein vom Bundesministerium für Bildung und Forschung (BMBF) gefördertes Projekt mit einem Konsortium mit Partnern aus Wirtschaft und Öffentlichkeit. So ist auf der öffentlichen Seite das Frauenhofer Institut für Angewandte Festkörperphysik, sowie das Karlsruhe Institute of Technology vertreten. Auf der wirtschaftlichen Seite sind die Kathrein-Werke KG, das mittelständische Unternehmen Radiometer Physics GmbH, sowie die Siemens AG vertreten.

Ziel des MILLILINK-Projekts ist die Einbindung von drahtlosen Links bzw. Funkstrecken in breitbandigen optischen Kommunikationsnetzwerken. Als Zieldatenrate sind $40\ \mathrm{Gbit/s}$ über eine Distanz von bis zu $1\ \mathrm{km}$ geplant [20].

Im Arbeitspaket 12, welches Grundlage dieser Masterarbeit ist, werden die Konzeption, die Implementierung und der Test eines Demonstrators im elektrischen Prüfumfeld gefordert. Dabei sollen die für das Projekt geforderten $40\ \mathrm{Gbit/s}$ nicht erreicht werden. Ziel des elektrischen Demonstrators ist der Einsatz höherwertiger Modulationsarten über die zu realisierende Richtfunkstrecke [20]. Höherwertige Modulationsarten bieten die Möglichkeit, die gleiche Anzahl an Bits über eine geringere Frequenzbandbreite zu übertragen. Parallel wird durch das Karlsruhe Institute of Technology ein Demonstrator mit $40\ \mathrm{Gbit/s}$ entwickelt. Dieser wird allerdings für das optische Prüfumfeld nur mit On-Off Keying (OOK) entwickelt.

2. Stand der Technik

Der Einsatz von Monolithic microwave integrated circuit (MMIC)-Technologie auf Basis von Transistoren mit hoher Ladungsträgerbeweglichkeit (HEMT) ermöglicht erstmals die Realisierung kompakter, leistungsfähiger und kostengünstiger Systeme auf Basis aktiver Elektronik. Die hohen Betriebsfrequenzen von 30 GHz bis zu 300 GHz ermöglichen darüber hinaus den Einsatz monolithisch integrierter On-Chip Antennen und moderner Methoden der Strahlungsfokussierung [20]. Der Millimeterwellenbereich der mit der MMIC-Technologie realisiert werden kann, liegt im Wellenlängenbereich von 1 mm bis 10 mm. Dies entspricht einer Frequenz von 30 GHz bis 300 GHz im freien Raum.

Zusammenfassend sind mit der MMIC-Technologie die Möglichkeiten der verwendbaren Frequenzbereiche auch in der Funkdatenübertragung erweitert worden. Es sind bereits Transistoren mit Frequenzen von bis zu 900 GHz erfolgreich entwickelt worden [11, Seite 31]. Dies ist hauptsächlich der erfolgreichen Verkleinerung der Gatelänge von Transistoren auf 20 nm zu verdanken [11, Seite 31].

Die neuen Frequenzbereiche stellen auch neue Anforderungen an eine digitale Datenverarbeitung. Gegenstand der in dieser Arbeit zu realisierenden digitalen Signalverarbeitung, sind die mit der MMIC-Technologie entwickelten Millimeterwellen-Frontends.

2.1. Millimeterwellen-Frontends

Die in Bild 2.1 dargestellten Millimeterwellen-Frontends

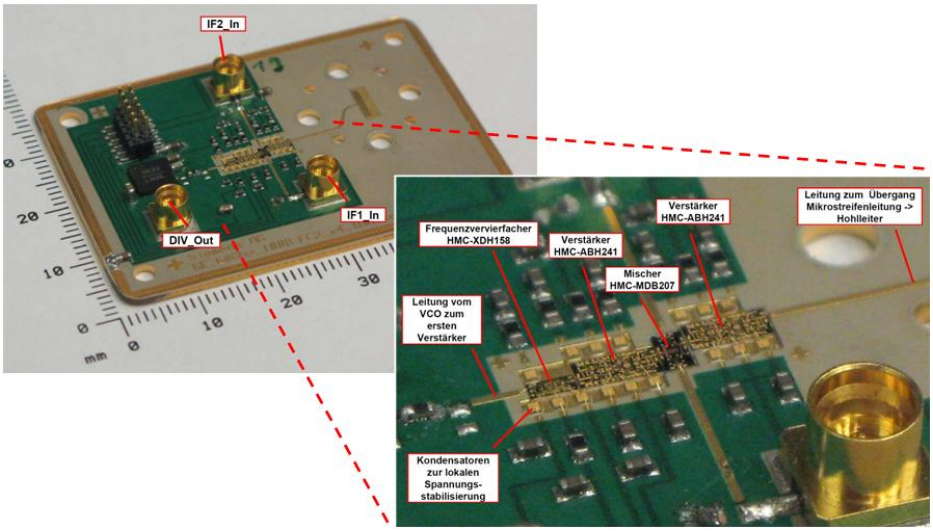


Bild 2.1.: 60 GHz-Millimeterwellen-Frontends

Quelle: Siemens AG

bestehend aus Voltage Controlled Oscillator (VCO), Frequenzvervierfacher, Oszillator-Verstärker, In-phase-Quadratur-phase-Mischer (IQ-Mischer) und Ausgangsverstärker, wie in Bild 2.2 aufgeführt,

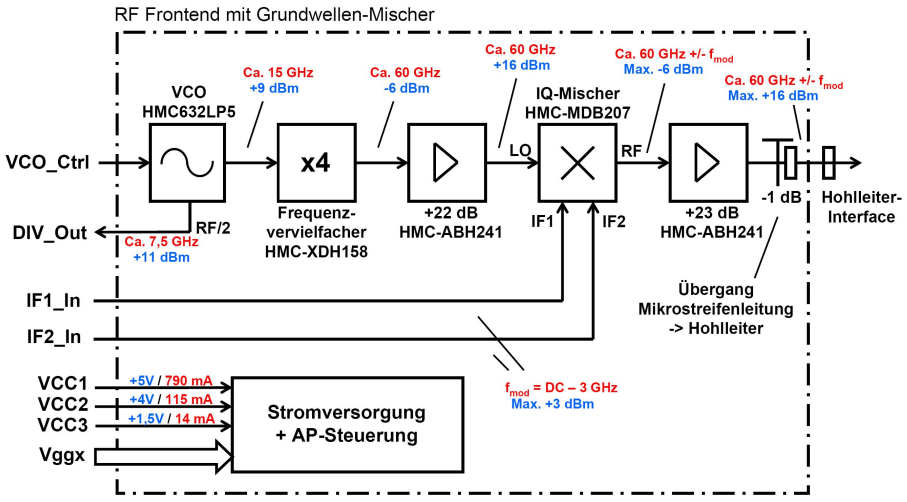


Bild 2.2.: Blockschaltbild der 60 GHz-Millimeterwellen-Frontends

Quelle: Siemens AG

werden zur Übertragung von Daten auf Funkstrecken verwendet. Die Millimeterwellen-Frontends arbeiten mit einer Mittenfrequenz von 60 GHz. Der VCO wird mit einer Regelspannung auf eine Frequenz von 15 GHz eingestellt. Diese 15 GHz werden mit einem Frequenzvervierfacher auf 60 GHz vervierfacht und mit einem Verstärker an den IQ-Mischer übergeben. Bevor das gemischte Signal an den Ausgang übertragen wird, wird dieses noch ein weiteres Mal verstärkt. Der Ausgang der Millimeterwellen-Frontends endet mit einem Hohlleiterübergang. An dem Hohlleiteranschluss soll aktuell eine Hornantenne angeschlossen werden.

2.2. Eigenschaften von Millimeterwellen-Frontends im Allgemeinen

Da Millimeterwellen-Frontends auf Frequenzen, die im Millimeterwellenbereich liegen, verstärken und mischen, kommen bei diesen besonders stark negative Eigenschaften wie Phasenrauschen und IQ-Imbalance zum Tragen. Folgend wird auf diese Eigenschaften der Reihe nach genauer eingegangen.

2.2.1. Phasenrauschen

Das Phasenrauschen ist eine Differenz der Phasenlage von einem idealen Oszillator zu einem reellen Oszillator. Sei ein idealer Oszillator

$$s_o(t) = A \cdot (2\pi f_o t) \tag{2.1}$$

mit der Amplitude A und der Frequenz f_o, so ist ein realer Oszillator

$$s_o(t) = A \cdot (2\pi f_o t + \theta(t)) + n(t). \tag{2.2}$$

Der Term $\theta(t)$ ist dabei das Phasenrauschen, welches zeitabhängig ist. Der Term $\mathrm{n}(t)$ steht für ein Grundrauschen und soll hier nicht weiter betrachtet werden. Angegeben wird das Phasenrauschen im Rauschleistungsdichtespektrum. Es hat die Maßeinheit $\mathrm{dBc/\,Hz}$ und wird für bestimmte Abstände zur Oszillatorfrequenz angegeben. Damit ist das Rauschleistungsdichtespektrum des Phasenrauschens $L\,(\Delta\omega)$ eine Funktion des Frequenzabstandes. Das Rauschleistungsdichtespektrum eines Oszillators kann nach Gleichung 2.3 [26, Seite 66] beschrieben werden. Die Gleichung beschreibt das Phasenrauschen in Abhängigkeit von der Oszillatorfrequenz ω_0 und in Abhängigkeit von dem Frequenzabstand zur Oszillatorfrequenz $\Delta\omega$. Die Oszillatorfrequenz geht mit $20\log\omega_0$ ein. Somit steigt das Rauschen mit zunehmender Oszillatorfrequenz an. Daher sind Komponenten im Millimeterwellenbereich mit starken Phasenrauschen behaftet. Um dies nochmal unabhängig vom Abstand zur Oszillatorfrequenz zu verdeutlichen, wurde die Oszillatorfrequenz in Gleichung 2.4 herausgezogen und in Gleichung 2.5 das Integral über die Frequenz gebildet.

$$L\,(\Delta\omega, \omega_0) = 10\log\left[\frac{2kT}{P_{sig}}\left(\frac{\omega_0}{2Q\Delta\omega}\right)^2\right] \tag{2.3}$$

Durch Herausziehen der Oszillatorfrequenz ω_0 erhalten wir

$$L\,(\Delta\omega, \omega_0) = 20\log\left[\omega_0\right] 10\log\left[\frac{2kT}{P_{sig}}\left(\frac{1}{2Q\Delta\omega}\right)^2\right]. \tag{2.4}$$

Anschließendes Integrieren erzeugt

$$L_g = \int L\,(\Delta\omega, \omega_0)\, d\Delta\omega = 20\log\left[\omega_0\right]\int 10\log\left[\frac{2kT}{P_{sig}}\left(\frac{1}{2Q\Delta\omega}\right)^2\right] d\Delta\omega, \tag{2.5}$$

welches die Gesamtleistung des Phasenrauschens darstellt. Des Weiteren geht mit dem Gütefaktor Q die Güte des Oszillator-Resonanzkreises ein. Die Güte eines Oszillator-

Resonanzkreises sinkt mit steigender Mittenfrequenz . Daher steigt die Rauschleistung nach Formel 2.5 mit sinkendem Gütefaktor Q.

Da Phasenrauschen das Hauptproblem bei der Übertragung mit Millimeterwellen-Frontends ist, soll das Phasenrauschen noch genauer betrachtet werden. Nach [17] lässt sich das Phasenrauschen in drei Anteile aufteilen. Als Erstes wird das Funkelrauschen (engl.: flicker noise) als $\theta_1(t)$ mit einem um den Faktor $1/f$ abfallendem Leistungsdichtespektrum aufgeführt. Ein solches Leistungsdichtespektrum wird auch rosa Leistungsdichtespektrum genannt. Des Weiteren wird rotes Rauschen (engl.: random walk or white frequency noise) als $\theta_2(t)$ mit einem mit den Faktor $1/f^2$ abfallendem Leistungsdichtespektrum erwähnt. Als Letztes wird weißes Phasenrauschen (engl.: white phase noise) als $\theta_3(t)$ aufgeführt. Das weiße Phasenrauschen hat für alle Frequenzen im Leistungsdichtespektrum einen konstanten Wert. Aus den drei Faktoren für das Phasenrauschen ergibt sich durch

$$\theta_G(t) = \theta_1(t) + \theta_2(t) + \theta_3(t) \tag{2.6}$$

das Gesamtphasenrauschen $\theta_G(t)$. Nun ist das Zeitsignal $\theta_G(t)$ ein stochastisches Signal. Stochastische Signale lassen sich am besten durch eine Varianz σ_θ^2 beschreiben. Wobei die Standardabweichung σ_θ proportional zum Jitter t_j ist. Mit

$$\sigma_\theta = 2\pi f_0 t_j \tag{2.7}$$

erhält man durch quadrieren

$$\sigma_\theta^2 = (2\pi f_0 t_j)^2 . \tag{2.8}$$

Somit errechnet sich die Varianz σ_θ^2 des Phasenrauschens aus dem Jitter t_j. In [4] ist ein Verfahren angegeben, nach dem sich das Phasenrauschen in ein Jitter t_j umrechnen lässt. Die hier angegebenen Formeln können später herangezogen werden, wenn das Phasenrauschen der Millimeterwellen-Frontends bekannt ist. Aus den Phasenrauschen kann somit eine Varianz σ_θ^2 des Phasenrauschens berechnet werden. Aus der Varianz σ_θ^2 können Rückschlüsse auf die Leistungsfähigkeit verschiedener Modulationsarten geschlossen werden, wie noch in Kapitel 3.5 gezeigt wird.

Um umgekehrt die Leistungsfähigkeit verschiedener Modulationsarten bei verschiedenem Phasenrauschen zu testen, wäre ein variables Phasenrauschen nötig. Eine Oszillatorquelle mit variablem Phasenrauschen steht aber nicht zur Verfügung. Dies wäre auch hinfällig, da in den Millimeterwellen-Frontends sogenannte *Jittercleaner* integriert sind. Jittercleaner minimieren den Jitter einer Sinusquelle und verringern somit das Phasenrauschen einer Sinusquelle. Ein Verfahren um ein Phasenrauschen anzunähern, ist ein Carrier Frequency Offset (CFO). Dabei wird eine definierte Frequenzabweichung zwischen Sender- und Empfänger-Millimeterwellen-Frontends angelegt. Dies ist aber nur eine Möglichkeit um eine Tendenz zu erkennen, da die Phasenabweichung bei einem CFO gleichverteilt ist. Außerdem erzeugt ein CFO zusätzliche Probleme, da das empfangene Signal nicht mehr exakt zurück in das Basis-

band gemischt wird. Dennoch soll in dieser Arbeit dieses Verfahren genutzt werden, um eine Tendenz der Phasenrauschresistenz der verschiedenen Modulationsarten zu erkennen.

2.2.2. IQ-Imbalance

Unter einer IQ-Imbalance versteht man die Imbalance zweier IQ-Mischerkanäle, dem Inphase-Kanal und dem Quadratur-Kanal. Im Idealfall sollen beide Kanäle die gleiche Amplitudenverstärkung über alle Frequenzen aufweisen. Eine Differenz der Amplitudenverstärkung nennt man Amplituden-IQ-Imbalance. Auch sollen beide Kanäle eine Phasenlage von $90°$ über alle Frequenzen zueinander haben. Eine Abweichung der Phasenlage von den gewollten $90°$ nennt man Phasen-IQ-Imbalance. Ein IQ-Mischer wird weiter unten im Kapitel 5.2 genauer beschrieben.

Zwei negative Aspekte entstehen durch die IQ-Imbalance. Zum einen wird bei einer Seitenbandunterdrückung die Dämpfung der Unterdrückung beeinflusst. Dieses Phänomen wird ausführlich in [7, Seite 4 f] beschrieben. Aus [5, Seite 5] ist die aufschlussreiche Grafik 2.3,

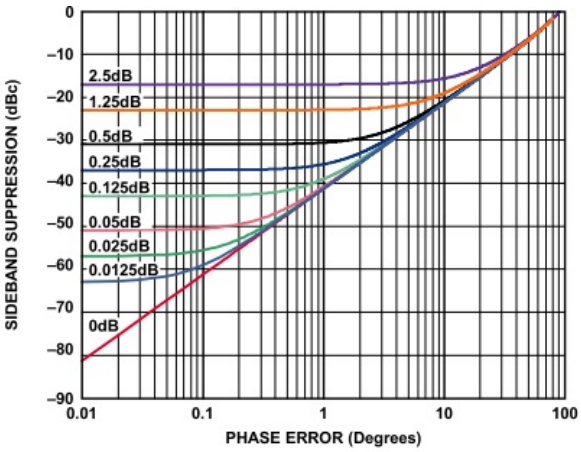


Bild 2.3.: Seitenbandunterdrückung in Abhängigkeit der Phasen-IQ-Imbalance bei verschiedenen Amplituden-IQ-Imbalancen

Quelle: Analoge Devices: AN:1039

in der die Seitenbandunterdrückung in Abhängigkeit der Phasen-IQ-Imbalance bei verschiedenen Amplituden-IQ-Imbalancen dargestellt wird. Aus der Grafik ist erkennbar, dass bei

nur 1° Phasen-IQ-Imbalance die Dämpfung der Seitenbandunterdrückung auf $\approx -40\,\mathrm{dBc}$ steigt und bei einem halben dB Amplituden-IQ-Imbalance die Dämpfung der Seitenbandunterdrückung auf $\approx -30\,\mathrm{dBc}$ steigt. Der zweite negative Effekt ist die Verschiebung der Punkte im Konstellationsdiagramm, wie in [5, Seite 3] verdeutlicht wird. Verschiedene Modulationen und deren Konstellationsdiagramme werden im Kapitel 3.5 aufgeführt.

Bei den Millimeterwellen-Frontendsist die IQ-Imbalance durch die große Bandbreite besonders hoch. Betrachtet man beispielsweise die Kurven aus einem Datenblatt eines gängigen Millimeterwellen-Frontend, abgebildet in Bild 2.4, so erkennt man eine starke IQ-Imbalance

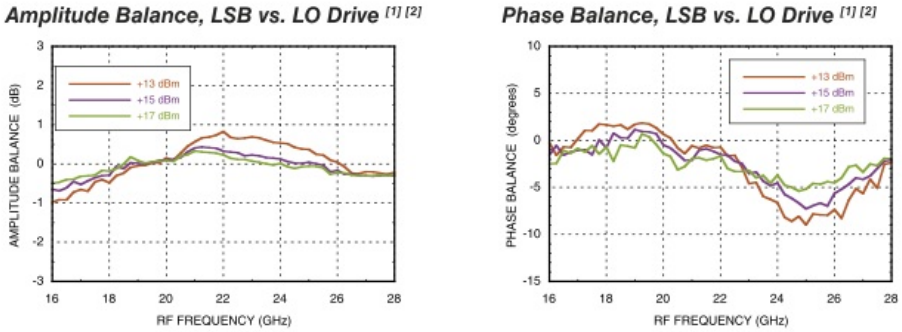


Bild 2.4.: IQ-Imbalance eines GaAs MMIC I/Q MIXER

Quelle: http://www.hittite.com HMC1041LC4

über die Frequenz. Somit ist die Amplituden- und die Phasen-IQ-Imbalance eine Funktion der Frequenz. Eine konstante Phasen- und/oder Amplituden-Imbalance lässt sich mittels Digital-to-analog converter (DAC) und Analog-to-digital converter (ADC) ausgleichen, da viele DAC und ADC die Möglichkeit bieten, eine Phasendifferenz und eine Verstärkung zwischen den beiden I- und Q-Kanälen einzustellen. Da die Phasen- und Amplituden-Imbalance nicht konstant ist, lässt sich diese Methode der Korrektur nicht anwenden.

Generell gibt es zwei Möglichkeiten eine sichere Datenübertragung bei IQ-Imbalance zu erreichen. Zum einen kann ein Mehrträgersystem verwendet werden. Bei einem Mehrträgersystem werden die Daten auf sehr viele Träger moduliert. Je größer die Anzahl der Träger, desto schmaler die einzelnen Frequenzbänder. Wählt man die Anzahl der Träger so groß und dadurch die Frequenzbänder so schmal, dass die IQ-Imbalance der einzelnen Frequenzbänder annähernd konstant sind, kann wiederum die IQ-Imbalance der einzelnen Frequenzbänder leicht durch Verzögerung und Verstärkung behoben werden. Leider verursacht eine große Anzahl an Trägern wiederum ein schlechtes peak-to-average power ratio (PAPR). Das PAPR wird genauer in Kapitel 3.5 beschrieben und behandelt.

Die zweite Möglichkeit einer über die Frequenz variablen IQ-Imbalance entgegen zu wirken, ist die Implementierung einer Kanalentzerrung. Die Kanalentzerrung kann als Filter, der die Amplitudenverstärkung für alle Frequenzen angleicht und die Phasenlage auf einen konstanten Wert bringt, verstanden werden. Das Design eines solchen Filters ist insbesondere wegen der großen Bandbreite, eine schwer zu lösende Aufgabe.

2.2.3. Weitere Kenneigenschaften

Des Weiteren sollen hier einige Eigenschaften der Millimeterwellen-Frontends aufgeführt werden, die zu weiteren Beschränkungen und Anforderungen an den Aufbau einer Übertragungsstrecke führen.

Rauschzahl

Die Rauschzahl (engl.: Noise Figure) ist das Verhältnis aus Eingangsrauschverhältnis zu Ausgangsrauschverhältnis. Somit ein Verhältnis zweier Verhältnisse. Mit S als Signalleistung und N als Rauschleistung lässt sich mit

$$F = \frac{\frac{S_{\text{in}}}{N_{\text{in}}}}{\frac{S_{\text{out}}}{N_{\text{out}}}} \tag{2.9}$$

das Verhältnis von Eingangs- zu Ausgangs-Signal-to-noise ratio (SNR) bestimmen. Drückt man Formel 2.9 in dB aus, lässt sich die Rauschzahl vereinfacht durch

$$F_{[\text{dB}]} = \text{SNR}_{\text{in},[\text{dB}]} - \text{SNR}_{\text{out},[\text{dB}]} \tag{2.10}$$

darstellen. Dies führt auch gleichzeitig zur Bedeutung der Rauschzahl. Denn durch Umstellung zu

$$\text{SNR}_{\text{out},[\text{dB}]} = \text{SNR}_{\text{in},[\text{dB}]} - F_{[\text{dB}]} \tag{2.11}$$

erkennt man, dass die Rauschzahl das Ausgangs-SNR verringert. Je größer die Rauschzahl, desto schlechter das Signal- zu Rausch-Verhältnis am Ausgang der Millimeterwellen-Frontends.

Bei den Millimeterwellen-Frontends ist die Rauschzahl besonders hoch da ein erhöhtes Funkelrauschen auftritt. Das Funkelrauschen ist proportional zur Frequenz. Die Mittenfrequenz ist bei den Millimeterwellen-Frontends wie bereits erwähnt besonders hoch. Somit produzieren die Millimeterwellen-Frontendsdurch die hohe Betriebsfrequenz eine hohe Rauschzahl.

Linearität

Die Linearität der Millimeterwellen-Frontends bezieht sich hauptsächlich auf die in den Millimeterwellen-Frontends integrierten Verstärkern. Unter Linearität versteht man bei Verstärkern das konstante Verhältnis von Eingangsleistung zu Ausgangsleistung. Ideal wäre ein Verstärker, der für alle Eingangsleistungen die gleiche Verstärkung produziert. Ein realer Verstärker erreicht einen Sättigungspunkt, über den dieser nicht verstärken kann. Als Kenngröße ist der $1\,\mathrm{dB}$-compression-point $P1\mathrm{dB}$ angegeben. Der $P1\mathrm{dB}$ ist der Punkt, an dem die Verstärkung um $1\,\mathrm{dB}$ geringer als die Kleinsignalverstärkung ist.

Wird ein Eingangssignal nichtlinear verstärkt, so verzerrt sich das Ausgangssignal. Das Ausgangssignal müsste dann gegebenenfalls entzerrt werden. Im schlechtesten Fall, wenn die Verzerrung zu groß ist, kann das Signal nicht mehr rekonstruiert werden. Neben der Verzerrung eines Signals kann durch nichtlineare Verstärkung eine Intermodulation des Signals erfolgen. In diesem Fall werden die unterschiedlichen Spektralanteile in einem Signal miteinander gemischt und führen zu weiteren unerwünschten Spektralanteilen. Beschrieben wird diese Eigenschaft auch durch die Intercept-Points. Genaueres zu den Intercept-Points findet sich in [16, Seite 8].

In [30] wird darauf aufmerksam gemacht, dass Leistungsverstärker insbesondere bei hohen Frequenzen ein stark nichtlineares Verhalten zeigen.

Sendeausgangsleistung

Die Sendeleistung ist die abgegebene Leistung am Ausgang der Millimeterwellen-Frontends. Die maximale Sendeausgangsleistung wird mit dem Saturated Output Power Parameter P_{Sat} angegeben.

Die Sendeausgangsleistung wirkt sich direkt auf den Kanalgewinn aus. Vereinfacht ist der Kanalgewinn die Addition von Sendeleistung, Antennengewinn und der Empfängerempfindlichkeit weniger der Freiraumdämpfung. Der Kanalgewinn ist äquivalent zur empfangenen Leistung. Bei der Übertragung addiert sich ein Rauschen auf das Signal. Dieses Rauschen hat eine bestimmte Leistung. Somit bestimmt die Sendeleistung das SNR im Empfänger.

Damit die Millimeterwellen-Frontends auf so hohen Frequenzen, wie den der Millimeterwelle operieren können, werden diese in besonders kleiner Bauform entwickelt. Wie bereits einleitend erwähnt, beträgt die Gatelänge aktueller Transistoren $20\,\mathrm{nm}$. Kleinere Bauformen

bedeuten aber auch geringere Spannungen und Ströme. Dies bedeutet wiederum eine geringere Ausgangsleistung bzw. Sendeleistung. Des Weiteren erkennt man durch Betrachten der Formel für die Freiraumdämpfung

$$F[\mathrm{dB}] = 20 \log \left(\frac{4\pi df}{c_0} \right)^2, \tag{2.12}$$

dass die Freiraumdämpfung bei steigender Frequenz ansteigt. Dies wiederum resultiert in einem geringen Kanalgewinn und einer geringeren empfangenen Leistung. Wie oben gezeigt sinkt dadurch das SNR im Empfänger.

2.3. Digitale Datenverarbeitung

Für die digitale Datenverarbeitung gibt es verschiedene Techniken. Zu den leistungsstärksten Techniken gehören Digitaler Signalprozessor (DSP)s, Field Programmable Gate Array (FPGA)s und neuerdings auch Graphics Processing Unit (GPU)s. Je nach Anwendung hat jede Technik ihren Vorteil. Es ist geplant, langfristig mehrere Sender und Empfänger parallel zu betreiben, um verschiedene Kanäle abzudecken. Hierbei bietet ein FPGA den besonderen Vorteil, dass das Design dupliziert werden kann und mehrfach nebeneinander im FPGA angeordnet wird. Eine GPU eignet sich vor allen Dingen bei Berechnungen aus dem Teilbereich der Bildverarbeitung und eignet sich somit nicht für das aktuelle Thema. DSPs sind zwar sehr rechenstark, aber für einen hohen Grad an Parallelisierung nicht geeignet. Zur Verarbeitung des digitalen Datenstroms der aktuellen Aufgabenstellung eignet sich somit ein FPGA am besten.

Das aktuellste FPGA-Produkt von Altera ist der Stratix 5. In der Ausführung des Stratix 5 5SGXBB stehen die meisten Ressourcen zur Verfügung. Es sind 359.200 ALMs (Adaptive logic module) integriert. ALMs werden von Altera als Lookup table (LUT)-basierende Ressourcen mit erweiterten effizienten Logik-Anwendungen definiert. Ein ALM beinhaltet zwei adaptive LUTs und kann 8 Eingänge und 8 Ausgänge lesen und treiben. Unabhängig können aber nur 6 Eingänge durch ein 6-Eingangs-LUT oder durch zwei 5-Eingangs-LUT verarbeitet werden. Des Weiteren stehen 952.000 äquivalente LEs zur Verfügung. LEs sind logische Bausteine, die ein 4-Eingangs-LUT, ein programmierbares Register und ein Carry-Anschluss beinhaltet. Des Weiteren sind 1.436.800 Register adressierbar. Der Block-Random-Access-Memory (RAM) ist in 2.640 20 kb-Blöcke eingeteilt und kann insgesamt 52 Mb adressieren. Zusätzlich sind 707 18 Bit · 18 Bit-Multiplizierer integriert, wobei der spezielle Signalverarbeitungs-FPGA Startix V 5SGSD8 3.926 dieser Multiplizierer beinhaltet. Als Letztes sind 352 27 Bit · 27 Bit-DSP-Blocks integriert, wobei der spezielle Signalverarbeitungs-FPGA Startix V 5SGSD8 1, 936 solcher DSP-Blöcke beinhaltet. [2]

Das Konkurrenzprodukt von Xilinx der Virtex 7 in der XCV2000T Ausführung enthält 1.954.560 Logic Cells. Logic Cells sind Bausteine, die ein 4-Eingangs-LUT, ein programmierbares Register und eine Direktverbindung zur Nachbarzelle beinhalten. Diese sind vergleichbar mit den LEs bei Altera. Damit beinhaltet das Spitzenprodukt von Xilinx mehr als doppelt so viele Logic Cells bzw. LEs wie das Spitzenprodukt von Altera. Das Pendant zum ALM ist bei Xilinx das CLB (configurable logic block). Davon sind 305.400 sogenannte CLB-Slices enthalten. Die CLB-Slices beinhalten ein 6-Eingangs-LUT, welches auch zu zwei 5-Eingangs-LUT konfiguriert werden kann. Den LUT sind acht Register nachgeschaltet. Zwischen 25% bis 50% dieser CLBs können auch als 32 Bit-Schieberegister verwendet werden. Damit liegt Xilinx bei der Anzahl der CLBs leicht hinter dem Altera Produkt. Dies ist auch bei den 2.584 18 kb oder 1.292 36 kb Block-RAM der Fall, der sich zu 46,5 Mb adressierbaren RAM zusammenfassen lässt. Wobei bei Xilinx Wortbreiten von 72 Bit bei 1 kb Adressraum und bei Altera nur 20 Bit bei 1 kb Adressraum zulässig sind. Bei Xilinx stehen beim Spitzenprodukt mehr DSP-Slices, mit 2.160 Stück, zur Verfügung. Der spezielle Signalverarbeitungs-FPGA XC7VX980T bietet sogar 3.600 DSP-Slices. Ein DSP-Slice beinhaltet einen 25 Bit · 18 Bit-Multiplizierer und einen 48 Bit Akkumulator, welche mit einer Taktrate von bis zu 741 MHz betrieben werden kann. Alternativ kann ein DSP-Slice so betrieben werden, dass vier 12 Bit Addierer, Subtrahierer oder Akkumulatoren verwendet werden können. [39]

Als absolute Innovation gilt die neue ZYNQ-Familie bei Xilinx. Der ZYNQ ist eine Plattform auf einem Chip, die neben einem FPGA einen ARM-Prozessor, sowie weitere Peripherie beinhaltet. Der ARM-Prozessor ist ein von der britischen Firma ARM Limited entwickelter 32 Bit Prozessor. ARM steht für Advanced RISC Machines. Wobei RISC für Reduced Instruction Set Computer steht. Bei dem ZYNQ steht ein 800 MHz Dual-Core-RISC-Prozessor im Zentrum der Plattform. Der ARM-Prozessor ist von einem FPGA umgeben. Der ARM-Prozessor hat eine große Anzahl an Schnittstellen zum FPGA, sowie den vollen Umfang diesen zu konfigurieren. Die Philosophie die dahinter steht ist folgende: Die administrative Rechenleistung findet auf dem ARM-Prozessor statt. Benötigt der Prozessor einen erweiterten Befehlssatz, so konfiguriert er im laufendem Betrieb die gewünschte Operation in den FPGA. So können rechenaufwändige Operationen auf den FPGA ausgelagert werden. Da bei ZYNQ nicht mehr der FPGA das Hauptaugenmerk des Chips ist, wird die ZYNQ-Familie bei Xilinx unter dem Synonym Extensible Processing Plattform (EPP) geführt. [36]

Auch in der Signalverarbeitung gibt es bei Xilinx und Altera eine weitere Innovation. Xilinx hat in Kooperation mit MathWorks ein Designtool für digitale Signalverarbeitung, mit den Namen *System Generator*, herausgebracht. Mit System Generator werden unter der Entwicklungsumgebung MATLAB-Simulink Hardwareblöcke miteinander zu einer Signalverarbeitung verbunden. Das so entstandene System kann unter MATLAB-Simulink simuliert werden und direkt in die Hardware synthetisiert werden. Damit eröffnet sich die Möglichkeit, Designs umfangreich zu simulieren. Außerdem lassen sich durch die MATLAB-Simulink-Umgebung sehr

schnell Systeme erstellen. Damit setzen Xilinx, Altera und MathWorks neue Maßstäbe in der Erstellung und in der Möglichkeit der Simulation von Hardwarebeschreibungscode [33]. Von Altera heißt das Designtool *DSP Builder*.

3. Systementwurf

Im Systementwurf soll die prinzipielle Gesamtstruktur des Demonstratoraufbaus festgelegt werden. Dabei wird zu Beginn auf die mögliche Struktur eines Mehrträgersystems in Kapitel 3.1 eingegangen. In Kapitel 3.2 *Inkohärente Empfänger* wird die Festlegung auf einen inkohärenten Empfänger begründet. Nachdem diese grundlegenden Überlegungen abgeschlossen sind, kann der grobe Aufbau entworfen werden. In Kapitel 3.3 *Hardwareentwurf* wird die zu verwendende Hardware detaillierter festgelegt und anschließend die Struktur der Hardware definiert. In einem weiteren Kapitel, dem 3.4 *Implementierungsentwurf* werden die zu implementierenden Module festgelegt.

Im MILLILINK-Projekt soll ein sehr hoher Datenstrom übertragen werden. Das Ziel liegt im Bereich von bis zu $40\,\mathrm{Gbit/s}$. Für das elektrische Prüfumfeld ist dies nicht gefordert. Im elektrischen Prüfumfeld werden die Millimeterwellen-Frontends mit einem elektrischen Datenstrom betrieben und nicht, wie später vorgesehen, in einem optischen Datenstrom. Im elektrischen Prüfumfeld soll primär untersucht werden, in wie weit sich höherwertige Modulationsverfahren für die Übertragung mit den Millimeterwellen-Frontends eignen. Durch höherwertige Modulationsverfahren lässt sich die spektrale Effizienz steigern. Auf die verschiedenen Modulationsarten und die spektrale Effizienz wird genauer in Kapitel 3.5 eingegangen. Umgekehrt kann im elektrischen Prüfumfeld die Leistungsfähigkeit der verschiedenen Modulationsarten bei unterschiedlichen Millimeterwellen-Frondend-Parametern, wie dem Phasenrauschen, untersucht werden. Es ist auch von Interesse, wie hoch der maximale Datendurchsatz im elektronischen Prüfumfeld ist. Ein so hoher Datendurchsatz wie im optischen Betrieb kann nur schwerlich im elektrischen Betrieb erreicht werden. Um den Datendurchsatz im elektrischen Prüfumfeld so groß wie möglich zu halten, kann an verschiedenen Punkten skaliert werden. Zum einen kann, wie oben angedeutet, die spektrale Effizienz durch den Grad der Modulation erhöht werden. Des Weiteren können schnellere ADCs und DACs verwendet werden. Letztendlich kann auch die Anzahl der Träger erhöht werden, wie im folgenden Kapitel dargestellt.

3.1. Mehrträgersysteme

In einem Mehrträgersystem wird der zu übertragende hochbitratige Datenstrom auf mehrere niederbitratige Datenströme parallelisiert. Jeder einzelne Datenstrom wird auf einem eigenen Träger moduliert. Die Träger sind dabei so angeordnet, dass die einzelnen Signalbänder der modulierten Träger sich nicht überschneiden. Alle modulierten Träger werden auf einen Kanal gegeben. Im Empfänger werden die einzelnen Signalbänder, wie in Bild 3.1 dargestellt, wieder separiert.

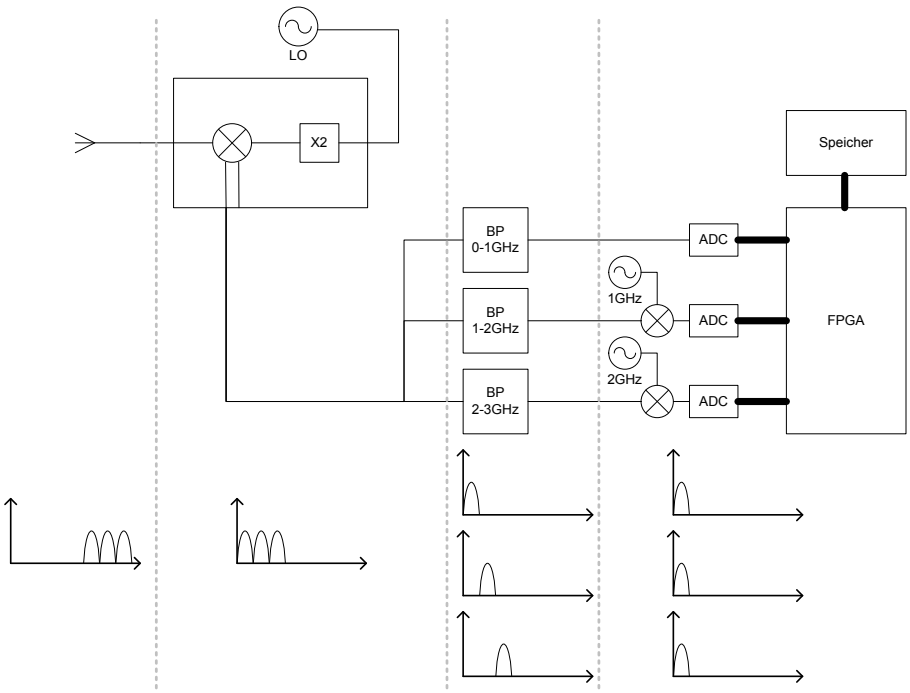


Bild 3.1.: Analoge-FDM

Die Antenne empfängt ein Mehrträgersignal. Dieses Mehrträgersignal wird in eine Zwischenfrequenz gemischt. Anschließend wird jeder Kanal mit Hilfe eines Bandpass (BP)es separiert. Der erste Kanal befindet sich bereits im Basisband und kann direkt mittels ADC in den FPGA eingelesen werden. Die anderen beiden Kanäle müssen erst ins Basisband gemischt werden. Die vom FPGA empfangenen Daten werden im FPGA demoduliert und in den angeschlossenen Speicher geschrieben. Pro zusätzlichen Träger werden ein weiterer ADC, BP und Mischer benötigt.

Im aktuellen Demonstrator soll für den ersten Aufbau nur ein Träger implementiert werden. Sollten weitere Träger implementiert werden, kann dies durch Duplizieren des Designs und Hinzufügen von Mischern, ADCs und Bandpässen erreicht werden.

3.2. Inkohärente Empfänger

Unter einem inkohärenten Empfänger versteht man einen Empfänger, dem das Trägersignal nicht bekannt ist. Bei einem kohärenten Empfänger wird das Trägersignal durch eine Trägerrückgewinnung zurückgewonnen. Dabei ist zwischen drei verschiedenen Frequenzen zu unterscheiden:

1. Das Hochfrequenz-Trägersignal ist ein Sinussignal, mit dem die Millimeterwellen-Frontends von der Hochfrequenz in die Zwischenfrequenz mischen, et vice versa.
2. Das Zwischenfrequenz-Trägersignal ist ein Sinussignal, mit dem von der Zwischenfrequenz in das Basisband gemischt wird, et vice versa.
3. Der Symboltakt ist die Taktrate, mit der die einzelnen Symbole übertragen werden. Ein Rückgewinnung dieses Signals wird Symboltaktrückgewinnung oder Taktrückgewinnung genannt.

Eine Symboltaktrückgewinnung ist immer zwingend notwendig, da ohne Symboltaktrückgewinnung die Daten nicht zurückgewonnen werden können. Eine Symboltaktrückgewinnung wird in Kapitel 5.4 besprochen. Somit ist ein Empfänger, an dem die Zwischenfrequenz- und die Hochfrequenz-Mischersignale nicht zurückgewonnen werden, ein inkohärenter Empfänger, auch wenn in diesem der Symboltakt zurück gewonnen wird.

Ist der Empfänger inkohärent realisiert, so muss die Demodulation inkohärent stattfinden (siehe Glossar). Des Weiteren wird das Übertragungssystem in ein nichtlineares System überführt. Ein nichtlineares Übertragungssystem benötigt eine nichtlineare Kanalentzerrung [21, Seite 321].

Um einen inkohärenten Empfänger in einen kohärenten Empfänger zu überführen, muss wie bereits erwähnt, eine Trägerrückgewinnung erfolgen. Da das Signal zweimal gemischt wird, müsste theoretisch für jede Mischstufe eine Trägerrückgewinnung erfolgen. Es kann aber auch nur in der Zwischenfrequenz eine Trägerrückgewinnung erfolgen, welche die inkohärente Mischung im Hochfrequenzsignal ausgleicht. Dadurch steigen die Anforderungen an die Trägerrückgewinnung in der Zwischenfrequenz. Dieses Verfahren wird meist bevorzugt, da eine Trägerrückgewinnung in der Hochfrequenz sehr schwierig bis unmöglich ist. Ein Trägerrückgewinnung in der Hochfrequenz ist deshalb so schwierig, da für die Trägerrückgewinnung die doppelte Frequenz erzeugt wird, um daraus wiederum das Trägersignal zu gewinnen. Dieses Verfahren heißt *squaring Loop*. Die doppelte Frequenz kann aber nicht erzeugt werden, da die Platinen, auf denen die Millimeterwellen-Frontends aufgebaut sind, keine Signale der doppelten Trägerfrequenz führen können. Eine Alternative bietet die Costas Loop, die aber nur für ein Binary-Phase-shift keying verwendet werden kann. Das Binary-Phase-shift keying wird in Kapitel 3.5 und die Costas Loop und weitere Regelkreise werden in Kapitel 5.4 genauer betrachtet. Außerdem mischt die Costas Loop das Hochfrequenzsignal direkt

ins Basisbandsignal, was wiederum nicht gewünscht ist, da die Millimeterwellen-Frontends AC-gekoppelt sind und in diesem Zuge das Basisband verzerren und nicht vollständig verarbeiten. Somit ist eine Trägerrückgewinnung in der Hochfrequenz nicht möglich.

Die Realisierung der Trägerrückgewinnung in der Zwischenfrequenz wäre prinzipiell leicht zu lösen, jedoch ist durch die erhöhten Anforderungen durch die nicht erfolgte Trägerrückgewinnung in der Hochfrequenz anzunehmen, dass die Leistungsfähigkeit der Trägerrückgewinnung stark eingeschränkt ist. Die Leistungsfähigkeit der Übertragung hängt damit stark von der Trägerrückgewinnung ab. Aus diesem Grund wird auf eine Trägerrückgewinnung verzichtetet und bei der Implementierung werden Modulationsverfahren verwendet, die keine kohärente Demodulation benötigen.

Des Weiteren lassen sich bei einem System, bei dem keine Trägerrückgewinnung erfolgt, Messreihen in Abhängigkeit eines Phasenrauschens bzw. einer CFO aufnehmen. Im Teil III wird die Bitfehlerrate in Abhängigkeit der CFO aufgenommen.

3.3. Hardwareentwurf

Es wird ein Millimeterwellen-Frontend als Sender und ein Millimeterwellen-Frontend als Empfänger verwendet. Bild 3.2 zeigt dies in Form eines Blockschaltbildes.

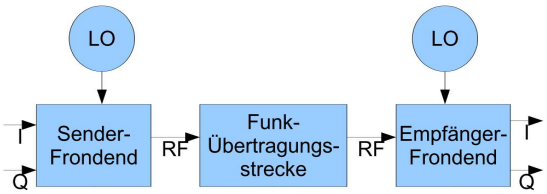


Bild 3.2.: Übertragungsstrecke mit 60 GHz-Millimeterwellen-Frontends

Die Millimeterwellen-Frontends sind auch für den bidirektionalen Betrieb ausgelegt. Allerdings soll im Demonstrator ausschließlich die sichere Datenübertragung demonstriert werden. Dazu ist nur eine unidirektionale Datenübertragung nötig. Die Millimeterwellen-Frontends mischen, wie bereits erwähnt, auf einer Mittenfrequenz von 60 GHz. Für das MILLILINK-Projekt sind 240 GHz-Millimeterwellen-Frontends geplant. Jedoch werden diese nicht bis zum Ende dieser Masterarbeit verfügbar sein. Daher werden die 60 GHz-Millimeterwellen-Frontends verwendet.

Für eine bessere Übersicht wird der Hardwareentwurf in einen Sender- und in einen Empfänger-Hardwareentwurf unterteilt.

3.3.1. Sender-Hardwareentwurf

Bild 3.3 zeigt ein Blockschaltbild des Sender-Hardwareentwurfs.

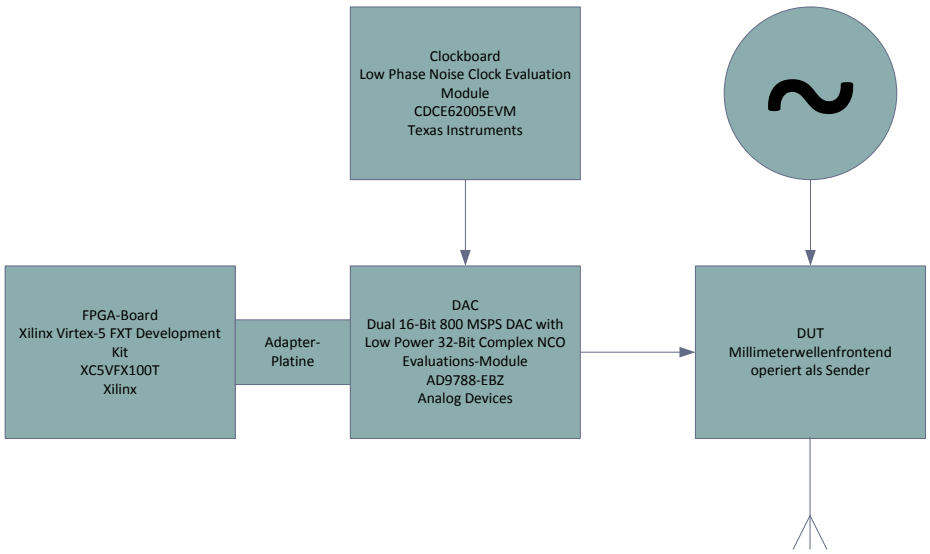


Bild 3.3.: Sender-Hardwareentwurf

Daten werden auf einem FPGA-Board erzeugt und über eine Adapter-Platine an ein DAC übergeben. Der DAC wird von einem Clock-Board mit einem Takt versorgt. Das analoge Ausgangssignal wird an das Millimeterwellen-Frontends übergeben. Das Millimeterwellen-Frontends, welches mit einem Oszillator verbunden ist, sendet das Hochfrequenzsignal über eine Antenne an den Empfänger.

Als FPGA-Board wird ein Virtex-5 FX100T Evaluations-Board von Avnet [9] verwendet. Ein Vorteil dieses Avnet-Boards ist der leistungsfähigere EPX-Stecker [8] für individuelle Adapter-Platinen. Dieser ist für single-ended in- und outputs auf bis zu $200\,\mathrm{MHz}$ und für differential in- und outputs auf bis zu $750\,\mathrm{MHz}$ spezialisiert. Der auf dem FPGA-Board verwendete FPGA ist ein Virtex-5 FX100T. Dabei steht FX für High-performance embedded systems with advanced serial connectivity. Dieser FPGA ist mit einem PowerPC 440 ausgestattet. Somit kann im Zuge der Entwicklung bei Bedarf auf einen Hard-Core zurückgegriffen werden.

Das DAC-Board *Dual 12-/14-/16-Bit 800 MSPS DAC with Low Power 32-Bit Complex NCO (AD9788-EBZ)* [3] hat zwei Kanäle mit einer 16-Bit Auflösung und einer Ausgangs-Quantisierungsrate von $800\,\mathrm{MSPS}$. Somit können komplexe Modulationsarten implementiert werden. Die hohe Bitrate kommt durch eine 8-fache Interpolation im DAC zustande. Am Eingang können maximal $100\,\mathrm{MSPS}$ übertragen werden. Die Wahl für diesen DAC hat

zwei Gründe. Einerseits kommt dieser DAC zum Zeitpunkt der Masterarbeit aus dem mittleren Preissegment und andererseits hat dieser einen integrierten komplexen Numerically Controlled Oscillator (NCO) mit IQ-Mischer. Somit kann das Basisbandsignal aus dem Basisband in ein leicht höheres Band gemischt werden, um AC-gekoppelte Frontends betreiben zu können.

Das verwendete Clock-Board *Five/Ten Output Clock Generator/ Jitter Cleaner With Integrated Dual VCOs* [31] bietet die Möglichkeit ein Taktsignal mit sehr geringen Jitter zu erzeugen, welcher die Anforderungen an den DAC erfüllt.

3.3.2. Empfänger-Hardwareentwurf

Bild 3.4 zeigt ein Blockschaltbild des Empfänger-Hardwareentwurfs.

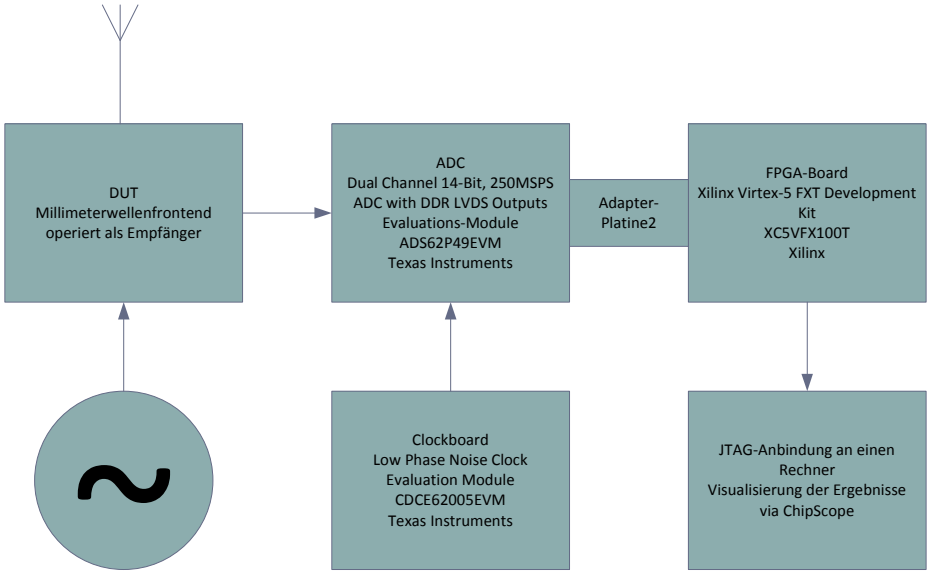


Bild 3.4.: Empfänger-Hardwareentwurf

Das Hochfrequenzsignal wird über die Antenne empfangen und mittels Millimeterwellen-Frontends in die Zwischenfrequenz gemischt. Das runtergemischte Zwischenfrequenzsignal wird an den ADC übergeben, welcher durch ein Clock-Board mit einem Taktsignal versorgt wird. Der ADC übergibt das digitalisierte Signal an den Empfänger-FPGA weiter. Im FPGA wird das digitale Signal ins Basisband gemischt und weiter verarbeitet. Über eine Joint Test Action Group (JTAG)-Schnittstelle können Daten aus dem Empfänger-FPGA mit einem Rechner ausgelesen werden. ChipScope (siehe Glossar) ermöglicht das Auslesen der Daten.

Auch im Empfänger wird das gleiche Virtex-5 FX100T Evaluations-Board von Avnet [9] wie im Sender verwendet. Auch das verwendete Clock-Board [31] ist das Gleiche wie beim Sender.

Das gewählte ADC-Board *Dual Channel 14-/12-Bit, 250-/210-MSPS ADC With DDR LVDS and Parallel CMOS Outputs (ADS62P49-EMV)* [32] bietet eine maximale Abtastrate von bis zu $250\,\mathrm{MSPS}$. Dies ist ca. die dreifache Abtastrate des Sender-DAC, bezogen auf die nicht interpolierte Symbolrate. Je höher die Abtastrate im ADC ist, desto genauer kann die Phase bei der Taktrückgewinnung ermittelt werden. Die Daten werden mittels Low Voltage Differential Signaling (LVDS) an den FPGA übermittelt. Dies ist nötig, da solche hohe Datenraten wegen der nötigen Bandbreite nicht mehr single-ended übertragen werden können [25]. Es handelt sich auch um ein Dual Channel ADC, um komplexe Modulationsverfahren verwenden zu können. Zwar können auch mit einem ein Kanal-ADC komplexe Modulationsverfahren verwendet werden, dazu ist aber die doppelte Abtastrate nötig. Die Wahl dieses ADC erfolgte nach ähnlichen Kriterien, wie schon bei dem DAC. Auch hier handelt es sich um ADC aus dem mittleren Preissegment. Ein weiterer Vorteil ist, dass sich dieser ADC sehr einfach seriell über den FPGA programmieren lässt.

Zur Visualisierung der Messergebnisse wurde ChipScope via JTAG-Schnittstelle gewählt. Der FPGA hat keine Peripherie die eine Visualisierung zulässt. Daher ist dies die preisgünstigste und am schnellsten umsetzbare Lösung.

3.4. Implementierungsentwurf

Für eine bessere Übersicht wird auch der Implementierungsentwurf in einen Sender-Implementierungsentwurf und in einen Empfänger-Implementierungsentwurf unterteilt.

3.4.1. Sender-Implementierungsentwurf

Bild 3.5 zeigt ein Blockschaltbild des Sender-Implementierungsentwurfs. Daten werden mittels Pseudo-Zufallsgenerator erzeugt und zur Modulation weitergegeben. Mit einem Pseudo-Zufallsgenerator lassen sich mit n-Registern Zufallsfolgen der Länge $2^n - 1$ erzeugen [1]. Die modulierten Daten, beschrieben in Kapitel 3.5 und 7, werden mit einem Impulsformer, beschrieben in Kapitel 5.1, bearbeitet, damit diese das erste und zweite Nyquist-Theorem einhalten [21, Seite 234 ff]. Eine DAC-Schnittstelle, beschrieben in Kapitel 4.1.1, übermittelt die Daten an den DAC. In dem verwendeten DAC wird das Signal interpoliert, bevor es mittels Mischer und NCO aus dem Basisband in eine Zwischenfrequenz gemischt wird. Der Aufbau des NCO und der Mischer ist in [3, Seite 40] beschrieben. Das Zwischenfrequenzsignal ist gleichanteilsfrei und wird an das Millimeterwellen-Frontends übergeben.

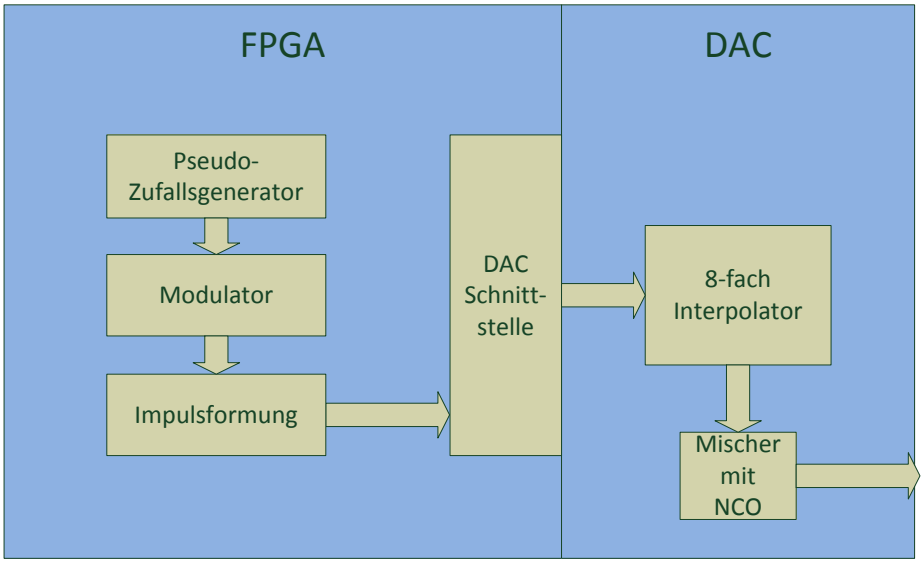


Bild 3.5.: Sender-Implementierungsentwurf

3.4.2. Empfänger-Implementierungsentwurf

Bild 3.6 zeigt ein Blockschaltbild des Empfänger-Implementierungsentwurfs. Das empfangene und vom ADC digitalisierte Zwischenfrequenzsignal wird mit einem im FPGA implementierten Mischer und NCO zurück ins Basisband gemischt. Der Mischer wird in Kapitel 5.2 beschrieben. Der NCO wird in [37] beschrieben. Aus dem Basisbandsignal wird mit einer Taktrückgewinnung, beschrieben in Kapitel 5.4, der Takt des Senders zurückgewonnen. Mit diesen Takt und dem Basisbandsignal wird der Demodulator, beschrieben in Kapitel 7, der das Basisbandsignal demoduliert, betrieben. Gleichzeitig wird der gleiche Pseudo-Zufallsgenerator wie im Sender mit dem zurückgewonnen Takt versorgt. Ein Vergleicher entscheidet, ob die demodulierten Daten mit dem des Pseudo-Zufallsgenerator übereinstimmen. Sind die Daten verglichen, wird mittels Bitfehlerratenmessung, beschrieben in Kapitel 6, über die JTAG-Schnittstelle die Bitfehlerrate angezeigt. Neben dem Empfänger ist noch eine ADC-Programmiereinheit implementiert, beschrieben in Kapitel 4.2. Diese sorgt dafür, dass der ADC beim Hochfahren des FPGA richtig konfiguriert wird.

3.5. Auswahl der Modulationsarten

Die Modulation ist ein Vorgang, in dem eine Nachricht einem Trägersignal aufmoduliert wird. Dem Trägersignal kann die Nachricht auf verschiedene Trägersignaleigenschaften aufmoduliert werden. So kann die Nachricht auf die Phase, die Amplitude oder auf die Frequenz

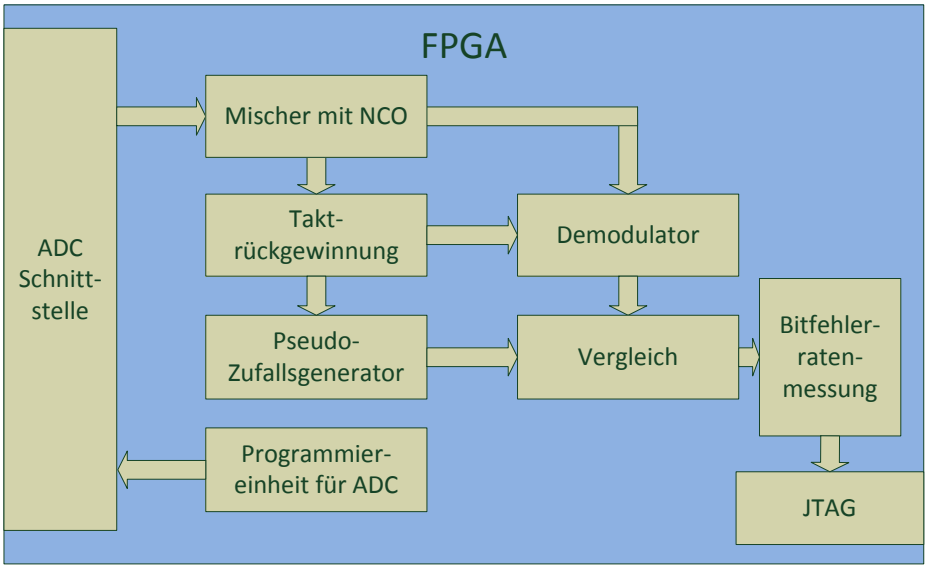


Bild 3.6.: Empfänger-Implementierungsentwurf

moduliert werden. Dadurch wird eine hochfrequente Übertragung eines niederfrequenten Signals möglich. So können die Ausbreitungsvorteile einer höherliegenden Frequenz genutzt oder viele Signale auf verschiedenen Frequenzen übertragen werden.

Ein Trägersignal

$$U_t = A_t \sin(2\pi f_t t + \theta_t) \tag{3.1}$$

mit den Trägereigenschaften Amplitude A_t, Frequenz f_t und Phase θ_t wird mit der Nachricht, die dem Signal $s(t)$ entspricht, moduliert. Bei der Amplitudenmodulation wird das Signal mit der Amplitude des Trägers zu

$$U_{\text{mod,amp}} = s(t) \cdot A_t \sin(2\pi f_t t + \theta_t) \tag{3.2}$$

moduliert. Bei der Frequenzmodulation wird das Signal mit der Frequenz des Trägers zu

$$U_{\text{mod,freq}} = A_t \sin(2\pi f_t \cdot s(t) \cdot t + \theta_t) \tag{3.3}$$

moduliert. Bei der Phasenmodulation wird das Signal mit der Phase des Trägers zu

$$U_{\text{mod,phase}} = A_t \sin(2\pi f_t t + \theta_t \cdot s(t)) \tag{3.4}$$

moduliert. Dies sind nicht die gängigen Notationen, jedoch verdeutlichen diese Schreibweisen die verschiedenen Modulationsarten auf eine einfache Weise.

Die Modulation im Demonstratoraufbau erfolgt digital. Bei der digitalen Modulation erfolgt eine Signalraumzuordnung der binären Daten auf den Träger, das sogenannte Mapping. Je nachdem, wie hochwertig die gewählte Modulationsart ist, werden mehrere binäre Daten in einem Schritt mittels Signalraumzuordnung gemappt. Es kann rein reell oder auch komplex gemappt werden. Bild 3.7 verdeutlicht die Signalraumzuordnung.

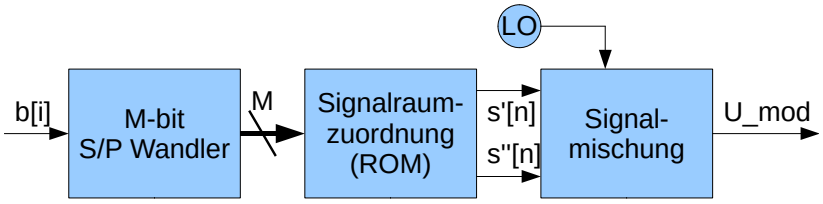


Bild 3.7.: Blockschaltbild einer digitalen Modulation

Ein binäres Signal $b[i]$ wird durch einen Seriell-zu-Parallel-Wandler in ein M-bit breites Wort gewandelt. Das M-bit breite Wort wird mittels Read-only memory (ROM) auf den komplexen Signalraum gemappt. Das gemappte Signal $\bar{s}[n] = s'[n] + j \cdot s''[n]$ wird Basisbandsignal genannt. Daher wird der Prozess des Parallelisieren der binären Daten im Zusammenhang mit der Signalraumzuordung auch als digitale Basisbandmodulation verstanden. Das Basisbandsignal wird dem Trägersignal Lokal Oszillator (LO) aufmoduliert. Eine genaue Betrachtung der Signalmischung von Trägersignal und Basisbandsignal erfolgt in Kapitel 5.2. Wie das Trägersignal im Mischer moduliert wird, wird schon durch die Basisbandmodulation entschieden. Die digitale Basisbandmodulation bietet eine Vielzahl von Spezialfällen der drei oben beschriebenen Modulationen. Die verschiedenen Basisbandmodulationen lassen sich durch Betrachten eines Konstellationsdiagramms visualisieren. In Kapitel 3.5.2 werden die gängigsten digitalen Basisbandmodulationen vorgestellt, im Konstellationsdiagramm dargestellt und die Vor- und Nachteile aufgezählt. Um die Vor- und Nachteile aufzählen zu können, werden zuvor die Eigenschaften der Basisbandmodulationen definiert und mathematisch beschrieben.

3.5.1. Kenneigenschafter digitaler Basisbandmodulationen

Die Anzahl der Bits pro Symbol gibt an, wie viele Bits in einem Taktzyklus übertragen werden. Diese Angabe ist unabhängig von der Taktrate. Die Multiplikation der Bits pro Symbol mit der Taktrate ergibt die Symbolrate. In der Literatur wird die spektrale Effizienz als Symbolrate pro Bandbreite des Signals definiert [21, Seite 242]. Jedoch ist die Bandbreite des Signals von dem nachgeschalteten Impulsformungsfilters, näher betrachtet in Kapitel 5.1, abhängig. Hier sollen die Basisbandmodulationen unabhängig von dem nachgeschalteten Impulsformungsfilters klassifiziert werden.

Der Scheitelfaktor (engl.: peak-to-average power ratio (PAPR)) oder auch als Crestfaktor bekannt, ist ein Maß, welches das Verhältnis von der Maximalleistung zur Durchschnittsleistung, in der Form

$$\mathrm{PAPR} = \frac{|x|_{\mathrm{peak}}}{x_{\mathrm{rms}}}, \tag{3.5}$$

beschreibt. Berechnet wird das PAPR für die verschiedenen Modulationsarten aus

$$\mathrm{PAPR} = \frac{s_{\mathrm{max}}^2 \cdot 2^M}{\sum_{l=1}^{2^M} \{s[l]^2\}}. \tag{3.6}$$

Wobei s_{max}^2 für Symbolleistung des Symbols mit der maximalen Leistung steht und $s[l]^2$ für die Symbolleistung eines Symbols steht. 2^M ist die Anzahl der unterschiedlichen Symbole. Ein hohes PAPR wirkt sich negativ auf die Übertragung aus, da die Frontends die Daten durchschnittlich mit einer geringeren Leistung, im Verhältnis zum maximal möglichen Leistung, übertragen.

Des Weiteren soll das Leistungsverhältnis vom Maximalleistung zur Minimalleistung aufgeführt werden. Dabei ist 0 nicht als Minimalleistung gültig. Wir wollen diese Größe als peak-to-minimum power ratio (PMPR), in der Form

$$\mathrm{PMPR} = \frac{|x|_{\mathrm{peak}}}{|x|_{\mathrm{minimum}}} \quad \text{mit} \quad |x|_{\mathrm{minimum}} \neq 0, \tag{3.7}$$

definieren. Berechnet wird das PMPR für die verschiedenen Modulationsarten aus

$$\mathrm{PMPR} = \frac{s_{\mathrm{max}}^2}{s_{\mathrm{min}}^2} \quad \text{mit} \quad s_{\mathrm{min}} \neq 0. \tag{3.8}$$

Ein hohes PMPR wirkt sich negativ auf die Übertragung aus, da das Symbol mit der geringsten Leistung bei größeren PMPR näher an das Grundrauschen (engl.: Noise floor (NF)) gelangt. Ein hohes PMPR bedeutet eine geringe Leistung für das Symbol mit der geringsten Leistung. Ist die Leistung des Symbols mit der geringsten Leistung geringer als das Grundrauschen, geht das Symbol im Grundrauschen unter.

Als drittes Leistungsverhältnis wird die Maximalleistung zum minimalen Leistungsabstand der Symbole zueinander als peak-to-minimum-space power ratio (PMSPR), in der Form

$$\mathrm{PMSPR} = \frac{|x|_{\mathrm{peak}}}{|x|_{\mathrm{minimum,space}}}, \tag{3.9}$$

definiert. Berechnet wird das PMPR für die verschiedenen Modulationsarten aus

$$\mathrm{PMSPR} = \frac{s_{\mathrm{max}}^2}{s_{\mathrm{min,space}}^2}. \tag{3.10}$$

Ein hohes PMSPR wirkt sich negativ auf die Übertragung aus, da bei geringem Leistungsabstand der Symbole zueinander die Symbolübergänge im Kanalrauschen untergehen können.

Die Formeln für die theoretische Bitfehlerrate (engl.: Bit error rate (BER)) werden je direkt im Kapitel 3.5.2 bei den Basisbandmodulationen angegeben, da jede Modulationsart eine eigene Formel benötigt. Die theoretische Bitfehlerrate wird bei definiertem Verhältnis von $\frac{E_b}{N_0}$, der Annahme, dass die Daten über einen additive white gaussian noise (AWGN)-Kanal gesendet werden und im Empfänger ein idealer Entscheider implementiert ist, berechnet. Das Verhältnis $\frac{E_b}{N_0}$ (eng.: energy per bit to noise power spectral density ratio) ist proportional zum SNR, in der Form

$$\mathrm{SNR} = \frac{E_b}{N_0} \cdot \frac{f_b}{B}, \tag{3.11}$$

mit B als Signalbandbreite, E_b als Signalenergie pro Bit, N_0 als Rauschleistung pro $1\,\mathrm{Hz}$ und f_b als Netto-Datenrate. Das Verhältnis $\frac{E_b}{N_0}$ ist damit unabhängig von der Bandbreite B. Ein AWGN-Kanal wird näher im Glossar beschrieben.

Von besonderem Interesse ist das Verhalten der verschiedenen Basisbandmodulationen im Zusammenhang mit dem Phasenrauschen. In [17] wird eine Formelsammlung zum Thema Leistungsfähigkeit verschiedener Basisbandmodulationen unter Berücksichtigung von Phasenrauschen angegeben. Da es sich um eine ganze Formelsammlung handelt, die für einzelnen Basisbandmodulationen spezielle Formeln enthält, sollen die Formeln direkt im Kapitel 3.5.2 aufgeführt werden. Die Performance der verschiedenen Modulationsarten bei ausgewählten Verhältnissen von $\frac{E_b}{N_0}$ und in Abhängigkeit des Phasenrauschens soll berechnet und bewertet werden. Das Phasenrauschen ist ein zeitabhängiger stochastischer Prozess, aus dem sich eine Varianz berechnen lässt. In [17] wurde dazu ein Modell entwickelt, mit dem sich das Phasenrauschen $\theta(t)$ in eine Varianz σ_θ^2 von einer Phasenabweichung θ_{offset} umrechnen lässt. Es wird angenommen, dass die Phasenabweichung θ_{offset} normalverteilt mit der Varianz σ_θ^2 auftritt. Für jede Phasenabweichung lässt sich eine Bitfehlerwahrscheinlichkeit berechnen. Durch

$$\mathrm{P}_b(e|\sigma_\theta) = \int_{-\infty}^{\infty} \mathrm{P}_b(e|\theta_{\mathrm{offset}})\, p(\theta_{\mathrm{offset}}|\sigma_\theta)\, \mathrm{d}\theta_{\mathrm{offset}} \tag{3.12}$$

lässt sich die Bitfehlerwahrscheinlichkeit bei gegebener Varianz σ_θ^2 berechnen. $p(\theta_{\mathrm{offset}}|\sigma_\theta)$ ist dabei die Wahrscheinlichkeitsverteilung der Phasenabweichung θ_{offset} bei gegebener Varianz σ_θ^2 unter der Bedingung, dass die Phasenabweichung normalverteilt ist. $\mathrm{P}_b(e|\theta_{\mathrm{offset}})$ ist die Bitfehlerwahrscheinlichkeit bei gegebener Phasenabweichung.

Des Weiteren werden spezielle Eigenschaften, die sich zum Teil aus den oben definierten Eigenschaften ableiten lassen, besprochen. Gleichanteilsfreiheit ist eine dieser Eigenschaften. Bestimmte Unempfindlichkeiten, wie beispielsweise unempfindlich gegen Phasenrau-

schen, unempfindlich gegen Trägerfrequenzabweichung (engl.: CFO) und Trägerphasenabweichung sind weitere Eigenschaften.

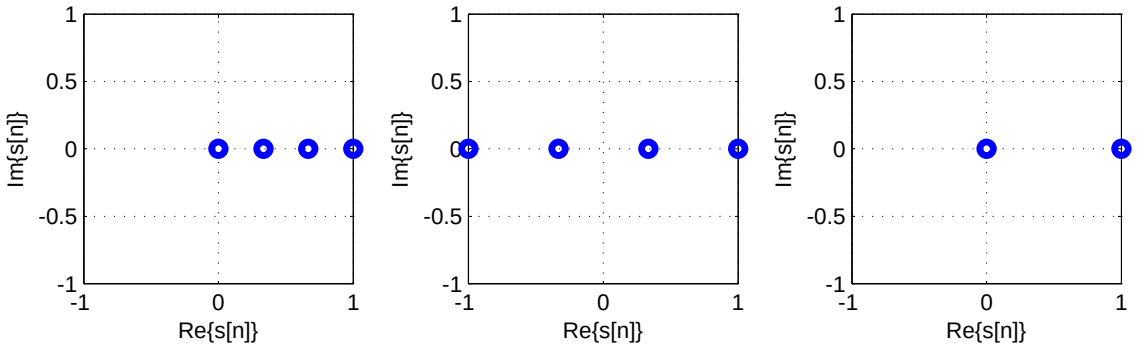
Da nun die Kenneigenschaften der digitalen Basisbandmodulationen bekannt sind, können die gängigen Basisbandmodulationen klassifiziert werden.

3.5.2. Klassifikation digitaler Basisbandmodulationen

In diesen Kapitel werden zuerst alle gängigen Modulationsarten vorgestellt, im Konstellationsdiagramm dargestellt und wichtige spezielle Formeln mit angegeben. Sind alle Modulationsarten vorgestellt, werden die Modulationsarten miteinander verglichen. Dazu wird am Ende des Kapitels die Tabelle 3.1 herangezogen. Die Kurven zur Bitfehlerwahrscheinlichkeit in Abhängigkeit des Verhältnisses $\frac{E_b}{N_0}$ 3.12 und in Abhängigkeit des Phasenrauschens 3.13 und A.1 werden anschließend zum Vergleich herangezogen. Dabei ist von besonderem Interesse, bei welcher Varianz σ_θ^2 die Leistungsfähigkeit der Phasenmodulationen gegenüber Amplitudenmodulationen zurückfallen.

Amplitude Shift Keying

Das Amplitude Shift Keying (ASK) ist eine Amplitudenmodulation. Je nachdem, ob der Signalraum auch negative Werte zulässt, kann das ASK auch als hybride Modulation von Phase und Amplitude verstanden werden. In Bild 3.8 sind drei verschiedene Arten des ASK im Konstellationsdiagramm visualisiert worden.



(a) 4-ASK ohne negative Werte (b) 4-ASK mit negativen Werten (c) OOK

Bild 3.8.: Konstellationsdiagramme verschiedener ASK

Im Konstellationsdiagramm 3.8(a) ist eine 4-ASK ohne negative Werte dargestellt. Die Information des Signals wird ausschließlich auf die Amplitude moduliert. In der Phase des Signals steckt kein Informationsgehalt. Ganz anders ist dies bei der 4-ASK mit negativen Werten, dargestellt im Konstellationsdiagramm 3.8(b). Hier wird die Information auf die Amplitude und die Phase moduliert. Die negativen Werte können als $180°$ Phasensprünge interpretiert werden. Diese Art der Amplitudenmodulation ist eine hybride Modulation aus Phasenmodulation und Amplitudenmodulation. Das dritte Konstellationsdiagramm 3.8(c) zeigt eine Sonderform der ASK, den On-Off Keying (OOK). Beim OOK wird ein Bit auf die Amplitude 1 oder 0 gemappt. Das Trägersignal wird mit der Bitfolge ein- und ausgeschaltet. Dies ist die einfachste Art der Basisbandmodulation. Mit Formel

$$\mathrm{P}_b|_{\mathrm{OOK}}(e) = \mathrm{Q}\left(\sqrt{\frac{E_b}{N_0}}\right) \tag{3.13}$$

aus [24, Kapitel 4.4] lässt sich die Bitfehlerwahrscheinlichkeit für das OOK berechnen. Die $\mathrm{Q}(x)$-Funktion ist zu

$$\mathrm{Q}(x) = \frac{1}{2} \cdot \mathrm{erfc}\left(\frac{x}{\sqrt{2}}\right) \tag{3.14}$$

definiert. Da beim OOK die Information nicht auf die Phase moduliert wird, ist das OOK unempfindlich gegenüber Phasenrauschen. Für jede Phasenabweichung gilt daher Formel 3.13. Da sich der Abstand der Symbole zueinander bei 4-ASK ohne negativen Werten drittelt, lässt sich aus Formel 3.13 die Bitfehlerwahrscheinlichkeit für 4-ASK ohne negativen Werten zu

$$\mathrm{P}_b|_{4-\mathrm{ASK,pos}}(e) = \mathrm{Q}\left(\sqrt{\frac{E_b}{N_0} \cdot \frac{1}{3}}\right) \tag{3.15}$$

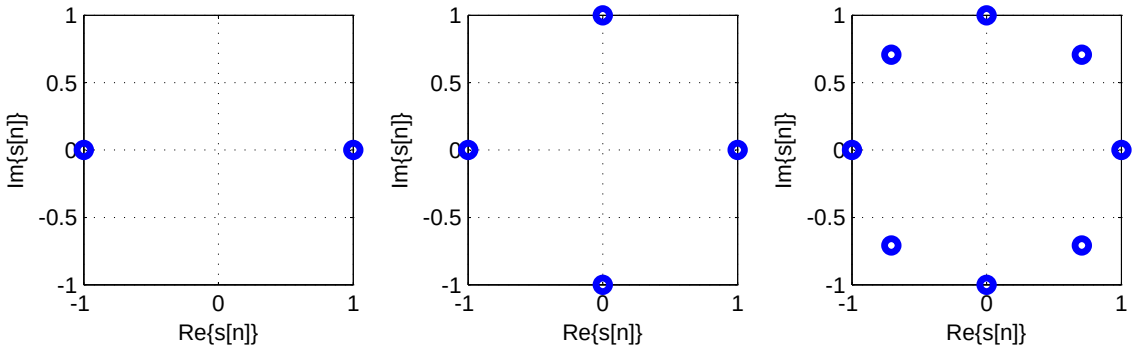
ableiten. Auch hier gilt wieder, dass für jede Phasenabweichung die Formel 3.15 gilt. In [21, Seite 386] ist die Formel

$$\mathrm{P}_b|_{4-\mathrm{ASK,neg}}(e) = \frac{1}{2} \cdot \mathrm{erfc}\left(\sqrt{\frac{1}{4} \cdot \frac{E_b}{N_0}}\right) \tag{3.16}$$

zur Berechnung der Bitfehlerwahrscheinlichkeit von 4-ASK mit negativen Werten angegeben. Eine Formel zur Berechnung der Bitfehlerwahrscheinlichkeit bei gegebener Phasenabweichung wurde nicht gefunden.

Phase Shift Keying

Das Phase-shift keying (PSK) ist eine Phasenmodulation. In Bild 3.9 sind die drei gängigsten Phasenmodulationen angegeben.



(a) BPSK (b) QPSK (c) 8-PSK

Bild 3.9.: Konstellationsdiagramme verschiedener PSK

Das Binary-Phase-shift keying (BPSK) ist abgebildet im Konstellationsdiagramm 3.9(a). Das Quarternary-Phase-shift keying (QPSK) ist abgebildet im Konstellationsdiagramm 3.9(a). Eine 8-PSK ist abgebildet im Konstellationsdiagramm 3.9(a). Alle drei Modulationen sind bei Normalverteilung gleichanteilsfrei. Eine QPSK kann sowohl wie in 3.9(b) dargestellt, als auch um $\pi/4$ gedreht werden. Mit Formel

$$P_b|_{\mathrm{BPSK}}(e) = 0{,}5 \cdot \operatorname{erfc}\left(\sqrt{\frac{E_b}{N_0}}\right) \tag{3.17}$$

aus [21, Seite 374], die identisch ist zu

$$P_b|_{\mathrm{QPSK}}(e) = 0{,}5 \cdot \operatorname{erfc}\left(\sqrt{\frac{E_b}{N_0}}\right) \tag{3.18}$$

aus [21, Seite 378], lassen sich die Bitfehlerwahrscheinlichkeiten für das BPSK und das QPSK berechnen. Für die Bitfehlerwahrscheinlichkeiten des 8-PSK gibt [21, Seite 380] folgende Näherungslösung an:

$$\begin{aligned} P_b|_{8-\mathrm{PSK}}(e) = &\frac{1}{3}\operatorname{erfc}\left(\sqrt{3\frac{E_b}{N_0}}\sin\left(\frac{\pi}{2}\right)\right)\left[1-\frac{1}{8}\operatorname{erfc}\left(\sqrt{3\frac{E_b}{N_0}}\sin\left(\frac{3\pi}{8}\right)\right)\right] \\ &+\operatorname{erfc}\left(\sqrt{3\frac{E_b}{N_0}}\sin\left(\frac{3\pi}{8}\right)\right) \end{aligned} \tag{3.19}$$

Um die Bitfehlerwahrscheinlichkeit von BPSK bei gegebener Phasenabweichung zu berechnen, wird Formel

$$P_b|_{\mathrm{BPSK}}(e|\theta_{\mathrm{offset}}) = Q\left(\sqrt{2\frac{E_b}{N_0}}\cos(\theta_{\mathrm{offset}})\right) \tag{3.20}$$

aus [17] verwendet. Im selben Paper findet sich die Formel

$$\begin{aligned} P_b|_{\mathrm{QPSK}}(e|\theta_{\mathrm{offset}}) = &\frac{1}{2}Q\left(\sqrt{2\frac{E_b}{N_0}}\{\cos(\theta_{\mathrm{offset}}) + \sin(\theta_{\mathrm{offset}})\}\right) \\ &+ \frac{1}{2}Q\left(\sqrt{2\frac{E_b}{N_0}}\{\cos(\theta_{\mathrm{offset}}) - \sin(\theta_{\mathrm{offset}})\}\right) \end{aligned} \tag{3.21}$$

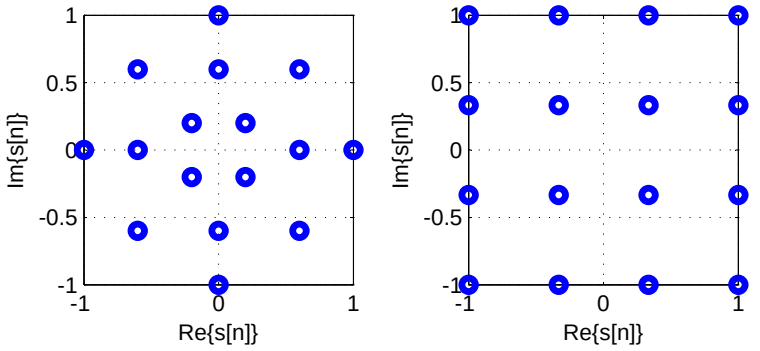
zur Berechnung der Bitfehlerwahrscheinlichkeit bei QPSK. Die Formel

$$\begin{aligned} P_b|_{8-\mathrm{PSK}}(e|\theta_{\mathrm{offset}}) = &\frac{1}{6}\mathrm{erfc}\left(\sqrt{3\frac{E_b}{N_0}}\sin\left(\frac{\pi}{8} - \theta_{\mathrm{offset}}\right)\right) \\ &+ \frac{1}{6}\mathrm{erfc}\left(\sqrt{3\frac{E_b}{N_0}}\sin\left(\frac{\pi}{8} + \theta_{\mathrm{offset}}\right)\right) \end{aligned} \tag{3.22}$$

zur Berechnung der Bitfehlerwahrscheinlichkeit bei 8-PSK kommt aus [21, Seite 388].

Hybride Modulationen

Neben dem reinen Amplituden- und Phasenmodulationen existieren hybride Modulationen, die die Information in Phase und Amplitude modulieren. In Bild 3.10 sind die zwei bekanntesten hybriden Modulationsarten dargestellt.



(a) 16-ASK-PSK (b) 16-QAM

Bild 3.10.: Konstellationsdiagramme hybrider Modulationsarten

Eine 16-ASK-PSK ist im Konstellationsdiagramm 3.10(a) und eine 16-Quadratur-Amplituden-Modulation (QAM) im Konstellationsdiagramm 3.10(b) dargestellt. Einzig zur Berechnung der Bitfehlerwahrscheinlichkeit der 16-QAM wurde in [21, Seite 387] die Formel

$$P_b|_{16-\mathrm{QAM}}(e) = \frac{3}{8}\,\mathrm{erfc}\left(\frac{2}{5}\frac{E_b}{N_0}\right) \tag{3.23}$$

gefunden. Um die Bitfehlerwahrscheinlichkeit bei 16-QAM in Abhängigkeit der Phasenabweichung zu berechnen, können die [17, Formeln 14 bis 22] verwendet werden. Da sich die Berechnung über mehrere Seiten erstrecken würde, soll diese hier nicht angegeben werden.

Differential-Phase-shift keying

Eine Sonderform der Phasenmodulation ist die differential-Phase-shift keying (DPSK). Bei der DPSK werden die Daten auf die Phasendifferenz gemappt. Bei einem zweistufigen DPSK der differential-Binary-Phase-shift keying (DBPSK) wird eine logische 0 als Phasenstillstand codiert und eine logische 1 als $180°$-Phasensprung. Dieses Verfahren hat den Vorteil, dass die absolute Phasenabweichung über alle empfangenen Symbole im Empfänger nicht bekannt sein muss. Die Bitfehlerwahrscheinlichkeit bei DBPSK verschlechtert sich gegenüber der BPSK, wie

$$P_b|_{\mathrm{DBPSK}}(e) = 2 \cdot P_b|_{\mathrm{BPSK}}(e) = \mathrm{erfc}\left(\sqrt{\frac{E_b}{N_0}}\right), \tag{3.24}$$

aus [21, Seite 385] zeigt. In [17] findet sich dazu die Formel

$$\begin{aligned} \mathrm{P}_b|_{\mathrm{DBPSK}}(e|\theta_{\mathrm{offset}}) &= \frac{1}{2}\left[1-\cos\left(\theta_{\mathrm{offset}}\right)\mathrm{I}_{\mathrm{e}}\left(\sin\left(|\theta_{\mathrm{offset}}|\right), \frac{E_b}{N_0}\right)\right], |\theta_{\mathrm{offset}}| \leq \pi/2 \\ &= -\mathrm{P}_b|_{\mathrm{DBPSK}}(e|\pi-\theta_{\mathrm{offset}}), \pi/2 < \theta_{\mathrm{offset}} \leq \pi \\ &= -\mathrm{P}_b|_{\mathrm{DBPSK}}(e|\pi+\theta_{\mathrm{offset}}), -\pi < \theta_{\mathrm{offset}} \leq -\pi/2 \end{aligned} \tag{3.25}$$

mit

$$\mathrm{I}_{\mathrm{e}}\left(k,x\right) = \frac{1}{\sqrt{1-k^2}} - \frac{1}{\pi}\int_0^{\pi} \frac{\exp^{-x(1-k\cos(\theta))}}{1-k\cos(\theta)}\mathrm{d}\theta, \tag{3.26}$$

aus [27], zur Berechnung der Bitfehlerwahrscheinlichkeit in Abhängigkeit der Phasenabweichung. Da bei Phasenabweichung größer $\pi/2$ die Bitfehlerwahrscheinlichkeit bei $0{,}5$ liegt, gilt die Berechnung der Bitfehlerwahrscheinlichkeit in Abhängigkeit der Varianz σ_θ^2 nach Formel 3.12 nicht, sondern

$$\mathrm{P}_b(e|\sigma_\theta) = \int_{-\pi}^{\pi} \mathrm{P}_b(e|\theta_{\mathrm{offset}})p(\theta_{\mathrm{offset}}|\sigma_\theta)\mathrm{d}\theta_{\mathrm{offset}} + P(|\theta_{\mathrm{offset}}| > \pi/2) \tag{3.27}$$

mit

$$P(|\theta_{\mathrm{offset}}| > \pi/2) = \mathrm{erfc}\left(\sqrt{\frac{\pi^2}{8\sigma_\theta^2}}\right). \tag{3.28}$$

Es zeigt sich, dass für den Fall, dass $\theta_{\mathrm{offset}} = 0$, Formel 3.25 nicht zum identischen Ergebnis kommt, wie Formel 3.24. Beide Formeln kommen aus unterschiedlichen Quellen. Weiter unten zeigt sich, dass der Formelsatz für DBPSK für kleine Varianzen σ_θ^2 nicht auf den richtigen Grenzwert strebt, jedoch für mittlere und große Varianzen σ_θ^2 plausible Ergebnisse liefert. Da uns interessiert, bei welcher Varianz σ_θ^2 die Leistungsfähigkeit der Phasenmodulationen gegenüber Amplitudenmodulationen zurückfallen, ist diese Einschränkung unerheblich.

Eine Sonderform der DPSK ist wiederum die 3-DPSK, dargestellt im Konstellationsdiagramm 3.11.

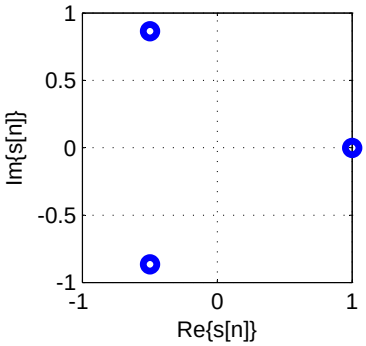


Bild 3.11.: Konstellationsdiagramm einer 3-DPSK

Bei der 3-DPSK wird ein ständiger Phasensprung erzeugt. Zur Übertragung einer logischen 0 wird ein $120°$-Phasensprung und zur Übertragung einer logischen 1 wird ein $240°$-Phasensprung erzeugt. Die 3-DPSK ist der in der Literatur [21] erwähnten Continuous-Phase-Modulation (CPM) sehr ähnlich und kann sowohl zu den Phasenmodulationen, als auch zu den Frequenzmodulationen gezählt werden, da die $120°$-Phasensprünge zu einer niedrigen Frequenz und die $240°$-Phasensprünge zu einer höheren Frequenz umgerechnet werden können. Ein wesentlicher Vorteil bei der 3-DPSK ist die Tatsache, dass das Signal bei einem Zustandswechsel nicht durch den Ursprung des Konstellationsdiagramms läuft. Bei der nachfolgenden Mischung des Basisbandsignals in die Zwischenfrequenz durch Multiplikation mit dem Träger bedeutet dies, dass zu jedem Zeitpunkt ein Signal übertragen wird. Es kommt nicht zu Leistungseinbrüchen in der Übertragung, da der Träger niemals mit dem Wert 0 multipliziert wird. In [21, Seite 277 f] ist das Problem der Leistungseinbrüche als Amplitudenschwankung einer PSK thematisiert. Eine daraus resultierende, nicht konstante Einhüllende resultiert in höhere Anforderungen an die Millimeterwellen-Frontends.

Die konstante Einhüllende des Sendesignals ist für die einfache und kostengünstige Endverstärkerrealisierung wichtig, da Leistungsverstärker insbesondere bei hohen Trägerfrequenzen ein stark nichtlineares Verhalten zeigen und daher Modulationsverfahren ohne konstante Einhüllende weniger geeignet sind.

Bernhard Spinnler [30]

Des Weiteren sind die Anforderungen an die Realisierung der Symboltaktrückgewinnung geringer, da zu jedem Symboltakt ein Phasensprung erzeugt wird. Bei jedem Phasensprung kann die, in einer Symboltaktrückgewinnung realisierten Phase-Locked Loop (PLL), Phaseninformation zur Regelung gewonnen werden. Symboltaktrückgewinnung und PLLs werden im Kapitel 5.4 genauer betrachtet.

Vergleich der Modulationsarten

Alle oben vorgestellten Modulationsarten werden in Tabelle 3.1 zusammenfassend aufgeführt. Die aus den im Kapitel 3.5.1 vorgestellten Formeln zur Berechnung der Kenneigenschafter digitaler Basisbandmodulationen sind Basis der in Tabelle 3.1 aufgeführten Parameter. Mit dem File A.1 wurden die Parameter in der Tabelle berechnet. Um ein Bit pro

	$\frac{\text{Bits}}{\text{Symbol}}$	PAPR	PMPR	PMSPR	Modulationsart
BPSK	1	1	1	0,25	Phase
DBPSK	1	1	1	0,25	Phase
3-DPSK	1	1	1	0,34	Frequenz/Phase
OOK	1	2	1	1	Amplitude
QPSK $\frac{\pi}{2}$	2	1	1	0,5	Phase
QPSK	2	1	1	0,5	Phase
DQPSK	2	1	1	0,5	Phase
4-ASK mit neg. Werten	2	1,8	9	2,25	Phase und Amplitude
4-ASK ohne neg. Werten	2	2,57	9	9	Amplitude
8-PSK	3	1	1	1,71	Phase
16-QAM	4	1,8	9	4,5	Phase und Amplitude
16-ASK-PSK	4	1,85	12,5	6,25	Phase und Amplitude

Tabelle 3.1.: Parametervergleich verschiedener Basisbandmodulationen

Symbol zu übertragen, würden nach Tabelle 3.1 BPSK bzw. DBPSK, wegen des besseren PMSPR am besten geeignet sein. Um zwei Bits pro Symbol zu übertragen, würden nach Tabelle 3.1 QPSK bzw. differential-Quarternary-Phase-shift keying (DQPSK) am besten geeignet sein, da alle drei Parameter einen niedrigeren Wert als beide 4-ASKs aufweisen. Bei den 16-stufigen Modulationsarten, bei denen 4 Bits pro Symbol übertragen werden, weist die 16-QAM bessere Parameter als die 16-ASK-PSK auf. Da an dieser Stelle das PAPR als wichtiger Entscheidungsparameter thematisiert wird, bietet es sich an, eine Anmerkung zur modernen Übertragungstechnik Orthogonal Frequency Division Multiplexing (OFDM) zu machen. Bei der OFDM wird ein hochbitratiges Signal zu einem niederbitratigen parallelen Signal umgeformt und auf orthogonal zueinander liegenden Trägern moduliert. Dabei liegt die Anzahl der Träger bei einigen Zehn bis einigen Tausend. Das daraus resultierende PAPR

bei einer OFDM entspricht dabei der Trägeranzahl multipliziert mit dem PAPR der gewählten Basisbandmodulation [21, Seite 618]. Daher ist die Verwendung einer OFDM nur möglich, wenn die Frontends einen genügend großen Dynamikbereich aufweisen und der Kanalgewinn groß genug ist. Dies können die Millimeterwellen-Frontends nicht leisten. Aus diesem Grund wurde und wird die OFDM nicht weiter thematisiert.

Die aus Tabelle 3.1 gewonnen Erkenntnisse decken sich mit den in Bild 3.12 dargestellten Kurven.

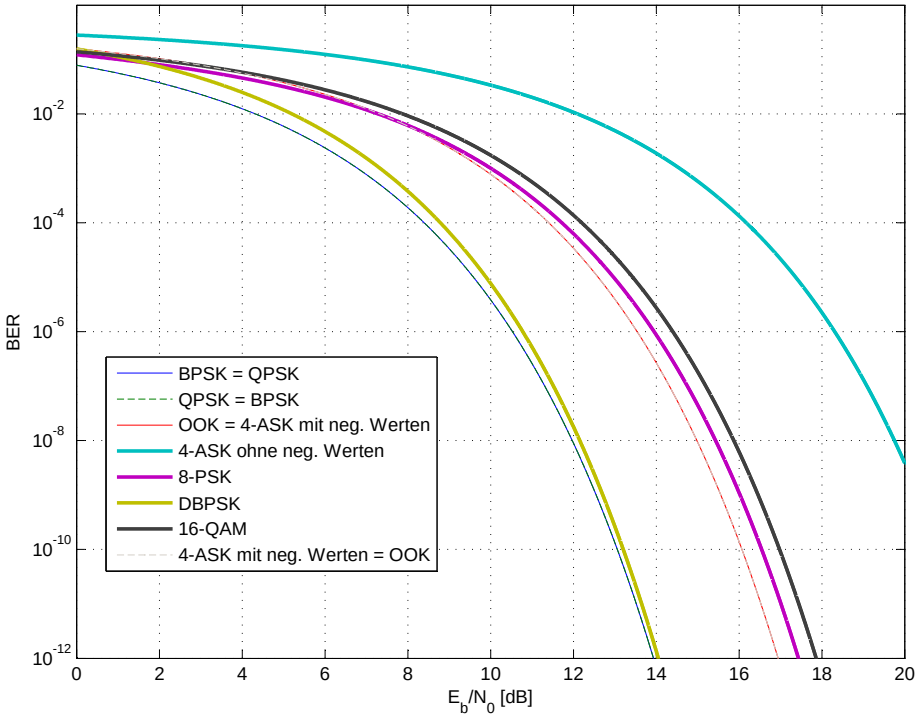
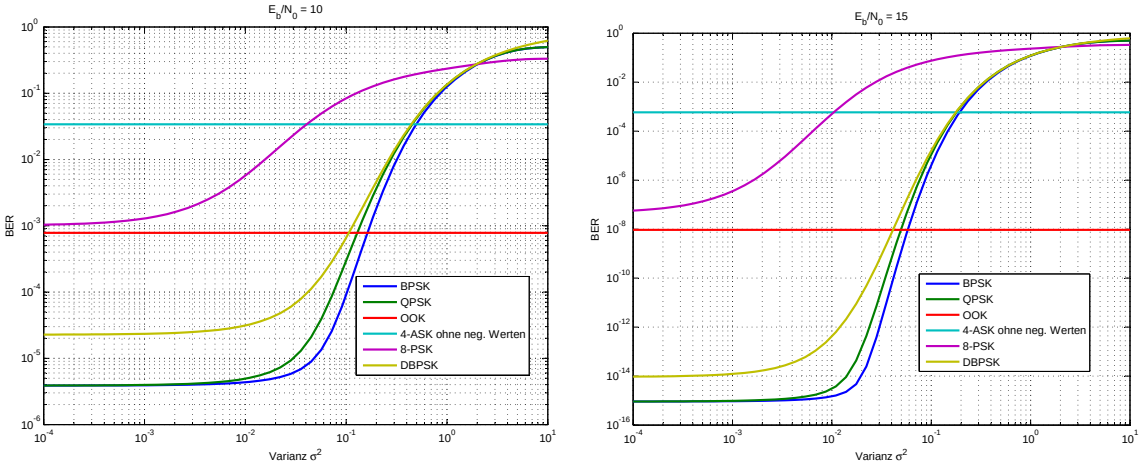


Bild 3.12.: Bitfehlerwahrscheinlichkeit in Abhängigkeit des Leistungsverhältnisses E_b/N_0

In Bild 3.12 wurden die oben angegebenen Formeln zur Berechnung der Bitfehlerwahrscheinlichkeit ohne Berücksichtigung einer Phasenabweichung in einem Plot verarbeitet. Im Anhang befindet sich das File A.5 zur Erzeugung dieses Plots. Jede Kurve wurde in Abhängigkeit des Verhältnisses $\frac{E_b}{N_0}$ dargestellt. Auch hier zeigt sich, dass das BPSK, DBPSK und QPSK die niedrigsten Bitfehlerwahrscheinlichkeiten produzieren. Bezeichnend ist, dass selbst QPSK eine geringere Bitfehlerwahrscheinlichkeit als OOK aufweist, obwohl mit QPSK zwei Bits statt einem Bit pro Symbol übertragen werden. Die 4-ASK ohne negativen Werten weist die schlechtesten Bitfehlerwahrscheinlichkeiten auf.

Wird jedoch das Phasenrauschen mit in die Betrachtung der Bitfehlerwahrscheinlichkeit genommen, so verschiebt sich die Leistungsfähigkeit zugunsten des OOK und der 4-ASK ohne negativen Werten. Basis der Berechnung waren die oben angegebenen Formeln zur Berechnung der Bitfehlerwahrscheinlichkeit in Abhängigkeit der Phasenabweichung. Zur Erzeugung dieses Plots wurden die Files A.6, A.2, A.3 und A.4 geschrieben.



(a) $E_b/N_0 = 10\,\mathrm{dB}$ (b) $E_b/N_0 = 15\,\mathrm{dB}$

Bild 3.13.: Bitfehlerwahrscheinlichkeit in Abhängigkeit der Phasenabweichungsvarianz

Im Bild 3.13 stellen die Kurven 3.13(a) und 3.13(b) die Bitfehlerwahrscheinlichkeit der verschiedenen Modulationsarten in Abhängigkeit der Varianz σ_θ^2 des Phasenrauschens bei zwei verschiedenen $\frac{E_b}{N_0}$-Verhältnissen dar. Bei einem Verhältnis von $\frac{E_b}{N_0} = 10\,\mathrm{dB}$ kreuzt die Kurve des BPSK, DBPSK und des QPSK die Kurve des OOK im Bereich von $\sigma_\theta^2 = 10^{-1}$. Bei einem Verhältnis von $\frac{E_b}{N_0} = 15\,\mathrm{dB}$ verschiebt sich das Schneiden der Kurven auf ca. $\sigma_\theta^2 = 0.4 \cdot 10^{-2}$. Die Kurve des 4-ASK ohne negativen Werten wird erst bei einer Varianz von $\sigma_\theta^2 = 0.4 \cdot 10^{-1}$ bzw. bei $\sigma_\theta^2 = 0.2 \cdot 10^{-1}$ geschnitten. Ab den Schnittpunkten wird die Leistungsfähigkeit des OOK und des 4-ASK ohne negativen Werten besser als bei den Phasenmodulationen. Es zeigt sich außerdem, dass die 8-PSK wie zu erwarten, besonders anfällig gegenüber Phasenrauschen ist. Der Vollständigkeit halber ist im Anhang noch ein Plot A.1 für ein Verhältnis von $\frac{E_b}{N_0} = 20\,\mathrm{dB}$ angehängt.

Leider ist zum Zeitpunkt der Masterarbeit noch keine Varianz σ_θ^2 für das Phasenrauschen der Millimeterwellen-Frontends bekannt. Daher kann aus den oben gemachten Untersuchungen keine Entscheidung getroffen werden, was die optimale Basisbandmodulation ist. Es bietet

sich allerdings an, für den Demonstrator ein Amplitudenmodulationsverfahren und Phasenmodulationsverfahren zu implementieren, da sich daraus ableiten lässt, wie verlässlich sich mittels der Millimeterwellen-Frontends amplitudenmodulierte und phasenmodulierte Daten übertragen lassen. So kann bei einem Phasenmodulationsverfahren ermittelt werden, ob bei Übertragung über die Millimeterwellen-Frontends überhaupt Information auf die Phase des Trägers moduliert werden kann. Anhand eines Amplitudenmodulationsverfahrens kann auf die Anzahl der möglichen Stufen geschlossen werden. Als Amplitudenmodulationsverfahren wird das OOK ausgewählt, da dieses Verfahren resistent gegenüber Phasenrauschen, Trägerfrequenzabweichung und Trägerphasenabweichung vom Sender zum Empfänger ist. Als Phasenmodulationsverfahren wird eine 3-DPSK ausgewählt, da dieses Verfahren geringere Amplitudenschwankung als eine herkömmliche DBPSK aufweist. Das Modulationsverfahren soll differentiell sein, da eine Phasenfehlerkorrektur im Empfänger entfällt.

Teil II.

Implementierung

4. Anbindung der Hardware

Eine gesicherte Datenübertragung ist ausschließlich möglich, wenn die Hardware richtig angebunden ist. Dazu muss sichergestellt werden, dass alle Komponenten fehlerfrei miteinander arbeiten. Um dies zu gewährleisten, muss nicht nur auf ein sauberes Design, sondern auch auf Simulationen und auf ausführliche Tests geachtet werden. Im Folgenden werden die Anbindungen, Simulationen und Tests der einzelnen Komponenten detailliert beschrieben.

4.1. Anbindung des Sender-FPGAs an den DAC

Für die Anbindung des Sender-FPGAs an den DAC muss sowohl eine Adapterplatine entworfen und hergestellt werden, als auch ein Very High Speed Integrated Circuit Hardware Description Language (VHDL)-Interface auf dem FPGA implementiert werden.

4.1.1. Adapterplatine

Die Adapterplatine verbindet das Sender-FPGA-Board mit dem DAC-Board. Das Design der Adapterplatine garantiert die Signalintegrität. Dazu wird die Adapterplatine auf einer zweilagigen *FR4*-Platine aufgebaut. Für die Daten werden zweimal 16 single-ended Streifenleitungen benötigt. Für den Clock wird ein LVDS-Streifenleitungspaar benötigt. Die Adapterplatine wird auf der FPGA-Seite mit einem EPX-Stecker und auf der DAC-Seite mit zwei Hochfrequenzsteckern der Firma *TE Connectivity* (Produktnummer: 6469169-1) abgeschlossen. Diese Stecker können auch LVDS führen. Als Basismaterial wurde eine *FR4*-Platine mit einer Substratstärke von $0{,}5\,\mathrm{mm}$, einer relativen Permittivität von $4{,}4$, einer relativen Permeabilität von 1 und einer Kupferschichtstärke von $35\,\mu\mathrm{m}$ verwendet. Für die Breite der benötigten $50\,\Omega$ Mikrostreifenleitung kann Formel 4.1 [6]

$$Z_0\,(\Omega) = \frac{87}{\sqrt{\epsilon_r + 1{,}41}} \ln\left[\frac{5{,}98H}{0{,}8W + T}\right] \tag{4.1}$$

zu

$$W = 1{,}25 \left(\frac{5{,}98H}{e^{\left\{ \frac{Z_0\sqrt{\epsilon_r}+1{,}41}{87} \right\}}} - T \right) \tag{4.2}$$

umgestellt werden. Durch Einsetzen der durch das Basismaterial gegebenen Parameter

$$W = 1{,}25 \left(\frac{5{,}98 \cdot 0{,}5\,\text{mm}}{e^{\left\{ \frac{50\,\Omega\sqrt{4{,}4}+1{,}41}{87} \right\}}} - 0{,}035\,\text{mm} \right) \tag{4.3}$$

ergibt sich eine Mikrostreifenleitungsbreite von

$$W = 0{,}89\,\text{mm}. \tag{4.4}$$

Der Pin-Abstand vom *TE Connectivity* ist allerdings $1{,}5\,\text{mm}$. Beim Platinendesign ist es nicht möglich eine Mikrostreifenbreite von $0{,}89\,\text{mm}$ zu routen. Neben dieser Einschränkung ist auch nicht genügend Platz auf der Platine um solch breite Leiterbahnen zu routen. Daher wird für alle Datenleitungen die Breite auf $0{,}3\,\text{mm}$ reduziert. Dies entspricht nach Formel 4.1 einem Wellenwiederstand von $86{,}1\,\Omega$. Eine solche Fehlanpassung ist irrelevant, da es sich um eine verhältnismäßig kurze Adapterplatine im Verhältnis zur Wellenlänge handelt. Die Grundschwingung der Datenleitung ist $80\,\text{MHz}$. Dies entspricht nach

$$\lambda = \frac{c_0}{f} \frac{1}{\sqrt{\mu_r \epsilon_r}} \tag{4.5}$$

einer Wellenlänge von $1{,}8\,\text{m}$.[1] Es gilt: Solange ein Bauteil eine bauliche Länge von weniger als $\frac{1}{10}$ der Wellenlänge der Grundschwingung hat, ist dieses Bauteil als konzentriertes Bauteil zu betrachten [14, Seite 18 ff]. Die längste Leiterbahn ist $180\,\text{mm}$ lang und somit ein konzentriertes Bauteil. Bei konzentrierten Bauteilen muss nicht auf die Eigenschaften eines Bauteils, wie der Leiterbahnenwiderstand, im Bezug auf die Wellengleichungen geachtet werden, da sich diese nur sehr gering auswirken. Das LVDS-Streifenleitungspaar wird mit der berechneten $0{,}89\,\text{mm}$ Mikrostreifenbreite geroutet. Bild 4.1 zeigt das fertige Layout der Adapterplatine. Bei einer 2-Layer-Platine ist es leider nicht möglich eine Kreuzung der Datenbahnen mit dem Clockleitungspaar zu verhindern. Dies ist auch im Bild 4.1 zu sehen. In einer ersten Entwicklung wurde diese Kreuzung produziert. Bei der Verifikation der Adapterplatine wurden die Augendiagramme für die Datenleitungen aufgenommen. Die Augendigramme für die Datenleitungen, die eine Kreuzung mit dem Clock erfahren haben, waren komplett geschlossen. Aus diesem Grunde wurde künstlich ein dritter Layer an der Stelle der Kreuzung aufgebaut. Bild 4.2 zeigt die Änderung an der Adapterplatine. Durch die Änderung ist zwischen Datenleitungen und Clockleitungen eine Groundfläche erzeugt worden. Bild 4.3 zeigt die fertig produzierte Adapterplatine. Es wurden außerdem auf der

[1] Dieser Wert wurde mit ϵ_r statt mit $\epsilon_{r,eff}$ berechnet. In der Realität wird die Wellenlänge somit noch etwas länger sein.

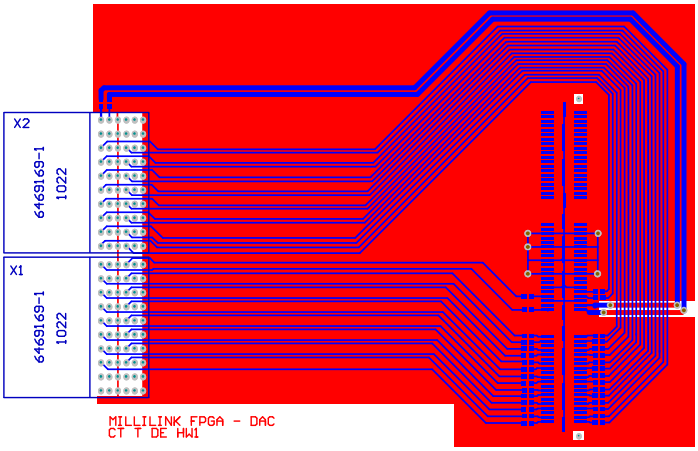


Bild 4.1.: Layout der Adapterplatine

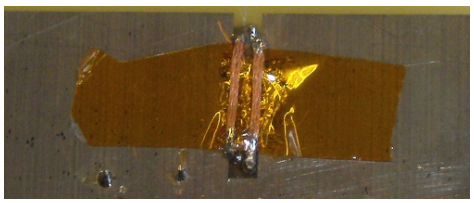


Bild 4.2.: Änderungen an der Adapterplatine

Senderseite Serienwiderstände eingelötet. Dies ist notwendig, da die Treiber im FPGA einen Innenwiderstand von $13\,\Omega$ haben. Es ist zwar möglich durch Digitally Controlled Impedance (DCI) [35, Seite 220 ff] den Innenwiderstand der Treiber digital kontrolliert zu verändern. Es ist jedoch dazu nötig, einen Referenzwiderstand an den FPGA anzuschließen. Dieser Referenzwiderstand ist bei dem gewählten FPGA-Board nicht angeschlossen. Somit bleibt nur die Möglichkeit, bei jedem einzelnen Datenleitungstreiber einen Serienwiderstand von $37\,\Omega$ einzulöten. Der Treiberwiderstand von $13\,\Omega$ in Serie mit dem eingelöteten $37\,\Omega$ Widerstand ergibt $50\,\Omega$ Gesammtwiderstand. Die Serienwiderstände haben noch ein weiteres mal deutlich zur Verbesserung der Augendiagramme geführt, da die Reflexionen auf der Datenleitung stark reduziert wurden.

Zur Verifikation der Adapterplatine wurde für alle Datenleitungen Augendiagramme aufgenommen. Bild 4.4 zeigt ein Augendiagramm, aus dem erkenntlich ist, dass die Signalintegrität gewährleistet ist.

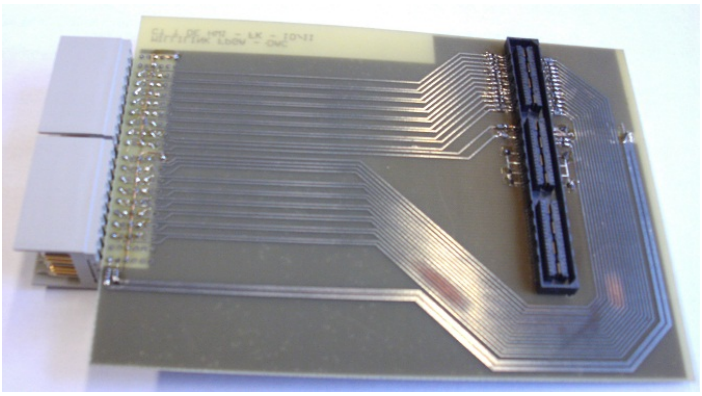


Bild 4.3.: DAC-Adapterplatine

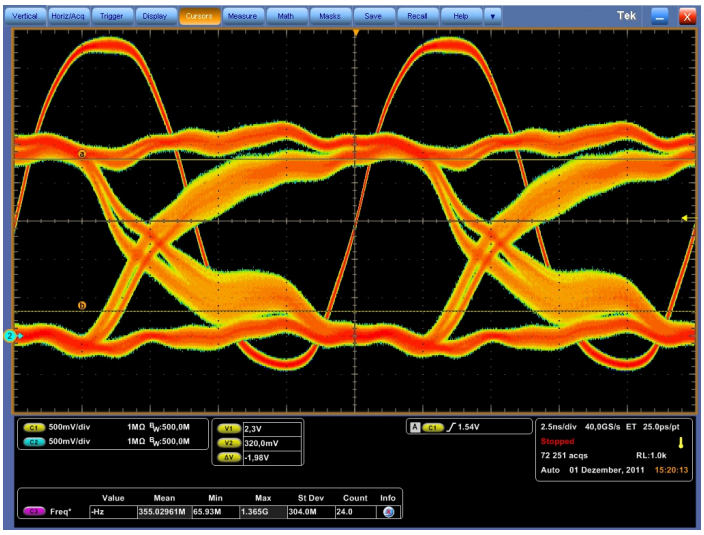


Bild 4.4.: Augendiagramm der Datenleitungen vom Sender-FPGA-Board zum DAC-Board

Das zweite Signal im Hintergrund ist das Taktsignal des DAC. Dort wo das Augendiagramm am weitesten geöffnet ist wird das Signal abgetastet. Zur Einstellung des Abtastzeitpunktes wird am Ende des Folgenden Kapitels eingegangen.

4.1.2. VHDL-Interface

Das VHDL-Interface verbindet die digitalen Datenpfade mit der Adapterplatine. Das VHDL-Interface sorgt dafür, dass die logischen Signale in die richtigen Spannungen und Ströme

überführt werden und vice versa. Außerdem wird das Timming der Signale zueinander kontrolliert.

Der DAC sendet einen differenziellen Takt. Der Takt ist phasengleich mit dem Abtasttakt, mit dem der DAC die empfangenen Daten abtastet. Der FPGA muss in dieser Taktrate die beiden Datenbusse mit Daten versorgen. Je Datenbuß werden 16 Bit parallel übertragen. Eine der Datenbusse dient später als In-Phase-Datenbuß und der zweite Datenbuß dient später als Quadratur-Phase-Datenbuß.

Zur Erzeugung eines VHDL-Interface werden sogenannte *Primitives* benötigt. Dies sind Hardwarebausteine auf dem FPGA, die durch Instantiierung im VHDL-Code angesprochen werden [38]. Im Bild 4.5 ist das Blockschaltbild für den Aufbau des VHDL-Interface abgebildet.

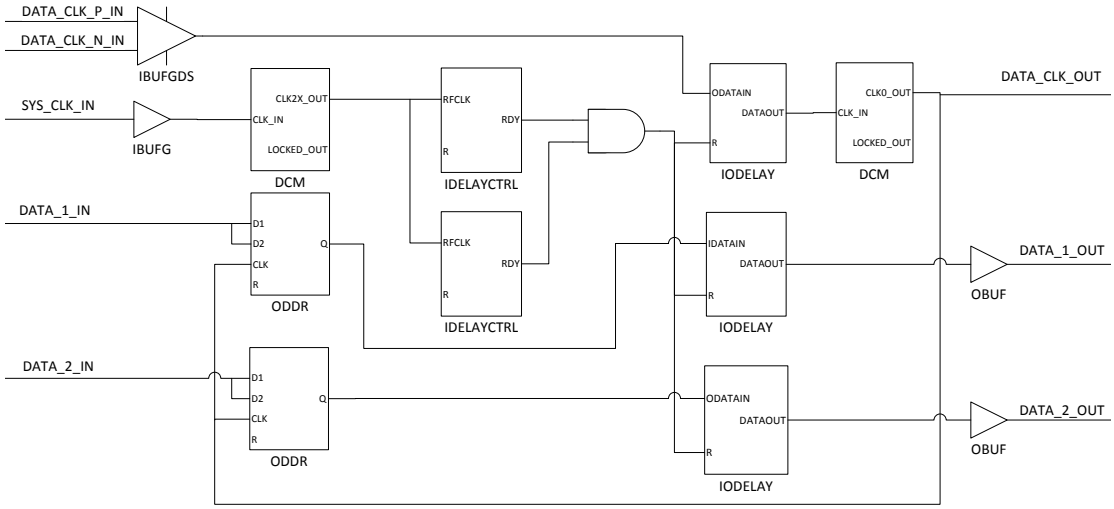


Bild 4.5.: DAC-Interface Blockschaltbild

Der empfangene, differenzielle Daten-Takt (DATA_CLK_P_IN) und (DATA_CLK_N_IN) wird mittels Differential Signaling Dedicated Input Clock Buffer and Optional Delay (IBUFGDS)-Primitiv in ein single-ended Signal überführt. Dieser sorgt für die richtige Terminierung des differenziellen Signals. In diesem Primitiv kann festgelegt werden, welche Logikpegel gelten. Der empfangene und umgewandelte Daten-Takt wird einem Input and Output Fixed or Variable Delay Element (IODELAY)-Primitiv zugeführt. Dieses Primitiv kann den Daten-Takt um einen definierten Zeitpunkt verzögern. Auf die Einstellung der Verzögerungen im allen IODELAY-Primitives wird weiter unten eingegangen. Der verzögerte Daten-Takt wird einer Digital Clock Manager (DCM)-Einheit zugeführt. Eine DCM-Einheit ist eine durch den IP CORE Generator & Architecture Wizard von Xilinx erzeugte Architektur, die ein bis mehrere Primitives enthält. Mit einer DCM kann der Takt-Jitter reduziert werden. Es können auch

verschiedene Takte vom Eingangstakt abgeleitet werden, wie doppelter Takt, halber Takt oder Phasenversetzter Takt. Im aktuellen DCM wird nur der Takt-Jitter reduziert. Hinter der DCM wird das Daten-Taktsignal aus dem VHDL-Interface geführt. Der Daten-Takt ist fertig aufbereitet und kann für die weitere Datenverarbeitung im FPGA verwendet werden (DATA_CLK_OUT).

Vom FPGA erzeugte Daten (DATA_1_IN) und (DATA_2_IN) werden an Dedicated Dual Data Rate Output Register (ODDR)-Primitives übergeben. ODDR-Primitives übernehmen Eingangsdaten an (D1) mit der nächsten steigenden Taktflanke und Eingangsdaten an (D2) mit der nächsten fallen Taktflanke und geben diese an (Q) aus. Somit könne Daten mit der doppelten Taktrate übertragen werden. Im aktuellen Design dienen die ODDR-Primitives nicht zur Erzeugung einer Dual-Data-Rate. Eingang (D1) und (D2) werden zusammen geschaltet, um eine Single-Data-Rate zu erzeugen. Der Grund für die Verwendung der ODDR-Primitives ist folgender: Die ODDR-Primitives erzeugen steilere Flanken, so wie ein Flip-Flop direkt am FPGA-Ausgang. Dies hat den Vorteil, dass beim Ändern der Signallaufwege bei der Datenerzeugung, das eingestellte Timing zwischen Taktleitung und Datenleitung konstant bleibt. Dies ist ein entscheidender Punkt bei der Erzeugung eines Interfaces. Das Verhalten des Interface soll unabhängig vom Design der Signalaufarbeitung sein. Durch das Verwenden der ODDR-Primitives ist dies gewährleistet. Die Daten, die durch die ODDR-Primitives zum Daten-Takt synchronisiert wurden, werden an zwei IODELAY-Primitives übergeben. In diesem IODELAY-Primitives werden alle Datenpfade zueinander so verzögert, so dass diese, am Ende der Adapterplatine, synchron ankommen. Auf die Einstellung der Verzögerungen in allen IODELAY-Primitives wird weiter unten eingegangen. Die verzögerten Datensignale werden mittels Output Buffer (OBUF)-Primitives ausgegeben. In den OBUF-Primitives können die Signalpegel, die Flankensteilheit und der Treiberstrom festgelegt werden.

Die IODELAY-Primitives benötigen für den Betrieb Tap Delay Value Control (IDELAYCTRL)-Primitives. IDELAYCTRL-Primitives geben dem IODELAY-Primitives ein Freigabesignal, sobald alle Takte stabil anliegen. Das Zusammenspiel zwischen IODELAY-Primitive und IDELAYCTRL-Primitive sorgt auch für die definierten Verzögerungen. Der für die IDELAYCTRL-Primitive benötigte Referenztakt dient dem IODELAY-Primitive als Referenz für die eingestellten Verzögerungen. Es ist leider nicht ersichtlich, warum dieses Referenzsignal nicht direkt an die IODELAY-Primitive angeschlossen wird. Diese Eigenart zum Design kommt aus den Vorgaben von Xilinx [35, Seite 325 ff]. Die IDELAYCTRL-Primitives benötigen für den Betrieb einen $200\,\mathrm{MHz}$ Referenztakt. Daher wird ein On-Board $100\,\mathrm{MHz}$ System-Takt über ein Dedicated Input Clock Buffer (IBUFG)-Primitive dem Interface zugeführt. Das IBUFG-Primitive ist ein, speziell für Takte definierter, Eingangsbuffer. Diese sind sehr phasengenau. Am IBUFG-Primitive können die gültigen Logikpegel definiert werden. Der System-Takt wird über ein DCM-Primitive von $100\,\mathrm{MHz}$ auf $200\,\mathrm{MHz}$ verdoppelt. Dieser $200\,\mathrm{MHz}$-Takt wird den IDELAYCTRL-Primitives als Referenztakt zugeführt. Im aktuellen Design werden zwei IDELAYCTRL-Primitives benötigt, da die verzögerten Ein- und Ausgänge auf zwei verschie-

denen Bänken platziert sind [35, Seite 339 ff]. Durch ein Und-Gatter werden die Ready-Signale der IDELAYCTRL-Primitives zu einem gemeinsamen Ready-Signal verbunden. Dieses gemeinsame Ready-Signal wird den IODELAY-Primitives zugeführt.

In dem Blockschaltbild ist die Reset-Logik nicht eingezeichnet worden. Dies hätte das Blockschaltbild unnötigerweise unübersichtlich gemacht. Die Resetlogik sorgt dafür, dass alle Primitives in der richtigen Reihenfolge freigegeben werden. Als Erstes wird das DCM-Primitive freigegeben, welches den Referenztakt für die IDELAYCTRL-Primitives erzeugt. Diese werden fünf Takte nach dem Einrasten der DCM-Einheit freigegeben. Weitere sechs Takte später wird das DCM-Primitive für den Daten-Takt freigegeben. Direkt darauf folgend, werden die ODDR-Primitives freigeschaltet. Die Reset-Logik ist als Schieberegister realisiert worden. Dies spart Logik und ist eine sehr leicht nachvollziehbare Methode, um eine zeitliche Abfolge darzustellen.

Der gesamte VHDL-Code ist im Anhang B.1 nachzulesen. Bild 4.6 zeigt die Register Transfer Level (RTL)-Schematic, die durch Synthetisieren des VHDL-Codes erzeugt worden ist.

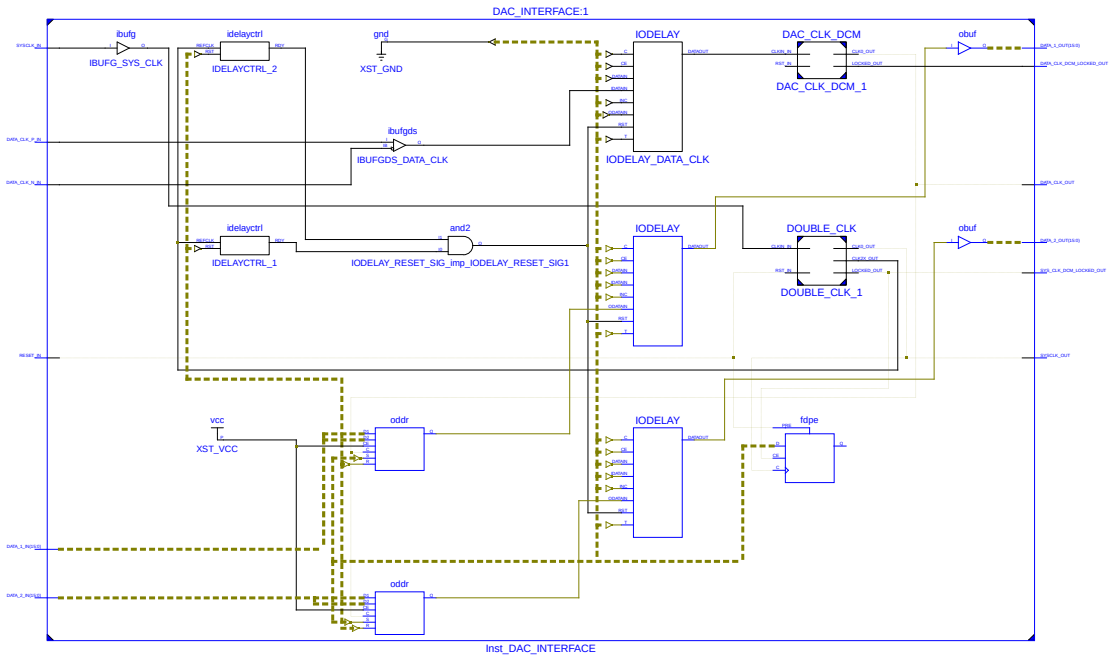


Bild 4.6.: DAC-Interface RTL-Schematic

Einstellung der Verzögerungen in den IODELAY-Primitives

Damit alle Datensignale phasensynchron am DAC anliegen, müssen die Verzögerungen in den IODELAY-Primitives eingestellt werden. Die Verzögerungen berechnen sich aus den Längen der einzelnen Datenpfade, der relativen Permittivität und relativen Permeabilität des Basismaterials, sowie der Lichtgeschwindigkeit. Mit Formel 4.6 kann die relative Zeit t_r berechnet werden, die ein Signal für die relative Strecke l_r benötigt. Die relative Strecke l_r ist die relative Länge zur längsten Strecke.

$$t_r = \frac{l_r \sqrt{\epsilon_r \mu_r}}{c_0} \tag{4.6}$$

Aus den berechneten relativen Zeiten t_r können die Delay Taps berechnet werden. Delay Taps sind definierte Zeiteinheiten, die im IODELAY-Primitive als Verzögerung eingestellt werden können. Die IODELAY-Primitives im Virtex 5 erzeugen bei einer Referenzfrequenz von $200\,\mathrm{MHz}$ pro Delay-Tap eine Verzögerung t_{tap} von $78\,\mathrm{ps}$ [34, Seite 44]. Mit Formel 4.7 kann die Anzahl der Delay Taps berechnet werden.

$$\mathrm{Taps} = \mathrm{round}\left(\frac{t_r}{t_{\mathrm{tap}}}\right) \tag{4.7}$$

Der MATLAB-Code B.2 erzeugt aus den eingegebenen Datenpfadlängen ein UCF-File, in dem die Delay Taps abhängig von den Datenpfadlängen kofiguriert sind. Zeile 95 bis Zeile 128 im UCF-File B.3 zeigt die Matlab-Ausgabe. Zeile 130 legt die Anzahl der Delay Taps für die Daten-Takt-Leitung fest. Hierfür wurde das Augendiagramm 4.7 am Dateneingang des DAC gemessen. Über das Augendiagramm wurde der Daten-Takt am Ausgang des DAC gelegt. Nun wurde die Anzahl der Delay Taps so lange erhöht, bis der Abtastzeitpunkt in der Mitte des Datenbits liegt. Diese Messung wurde für alle Datenleitungen wiederholt. Da die Datenleitungen synchronisiert wurden, ist der Abtastzeitpunkt bei allen Datenleitung exakt an der selben Stelle.

Nachdem die Funktionsfähigkeit des DAC-Interface festgestellt wurde, ist es abschließend nötig, die zeitkritischen Primitives am aktuellen Design örtlich fest zu platzieren. Dies ist nötig, da sich bei einer Änderung des Designs, wie dem Hinzufügen einer Datenverarbeitung die örtlichen Platzierungen ändern könnten. Eine Änderung der örtlichen Platzierung würde zu einer Änderung des Zeitverhalten zwischen Datenleitung und Taktleitung führen. Das veränderte Zeitverhalten führt wiederum zu unerwarteten Fehlern. Mit dem Befehl *LOC* können die Primitives örtlich platziert werden. Die aktuelle Platzierung der Primitives kann im *FPGA Editor* nachgeschaut werden. Die entnommenen Werte wurden im UCF-File nachgetragen. Zeile 86 bis Zeile 93 im UCF-File B.3 zeigt die Festlegung der Primitives via LOC-Befehl.

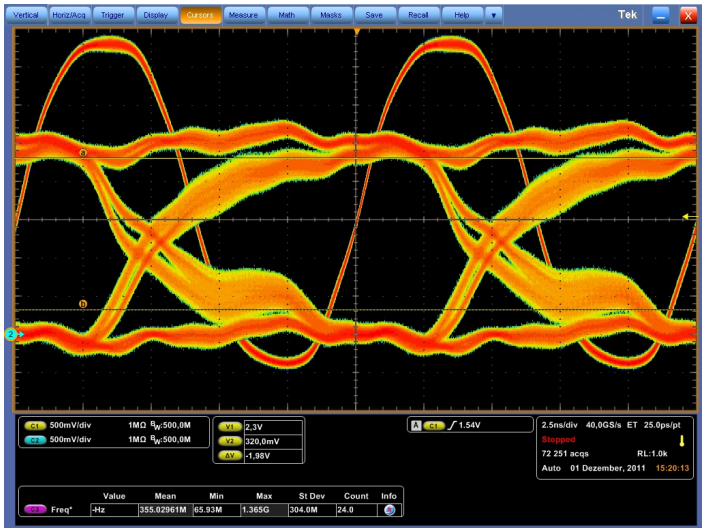


Bild 4.7.: Augendiagramm der Datenleitungen mit Daten-Takt

4.2. Anbindung des ADCs an den Empfänger-FPGA

Bei der Anbindung des ADCs an den Empfänger-FPGA kann auf ein bereits fertiges Interface zurückgegriffen werden, da das genutzte FPGA-Board schon in Kombination mit dem genutzten ADC in einem anderen Projekt verwendet wurde. Eine Adapterplatine und ein VHDL-Interface ist bereits entwickelt und ausgiebig getestet worden. Aus diesem Grunde wird nicht näher auf die verwendete Adapterplatine und das verwendete VHDL-Interface eingegangen. Jedoch bietet der ADC die Möglichkeit seriell programmiert zu werden [32, Seite 15]. Die Implementierung dieses Programmiermodul wird folgend aufgeführt.

Der Vorteil einer Programmierung des ADC-Board durch das Empfänger-FPGA ist, dass das ADC-Board nicht für jeden Test neu konfiguriert werden muss, sondern die Konfiguration als serieller Datenstrom nach Einschalten des Empfänger-FPGAs vom Empfänger-FPGA gesendet wird. Drei Signale werden für die Programmierung benötigt. Dies sind ein serielles Datensignal, ein Taktsignal und ein Takt-Enable-Signal. Bild 4.8 zeigt ein strukturelles Blockschaltbild vom Programmiermodul. Ein detailiertes Blockschaltbild befindet sich im Anhang C.1. Das Programmiermodul ist modular aufgebaut worden. Dies hat den Vorteil, dass jedes Modul einzeln auf Funktion getestet werden kann. Module können ausgetauscht und wiederverwendet werden. VHDL ist eine hoch modulare Hardwarebeschreibungssprache und sollte auch so implementiert werden [18, Seite 380] [19, Seite 157 ff]. Folgend werden die Module näher beschrieben.

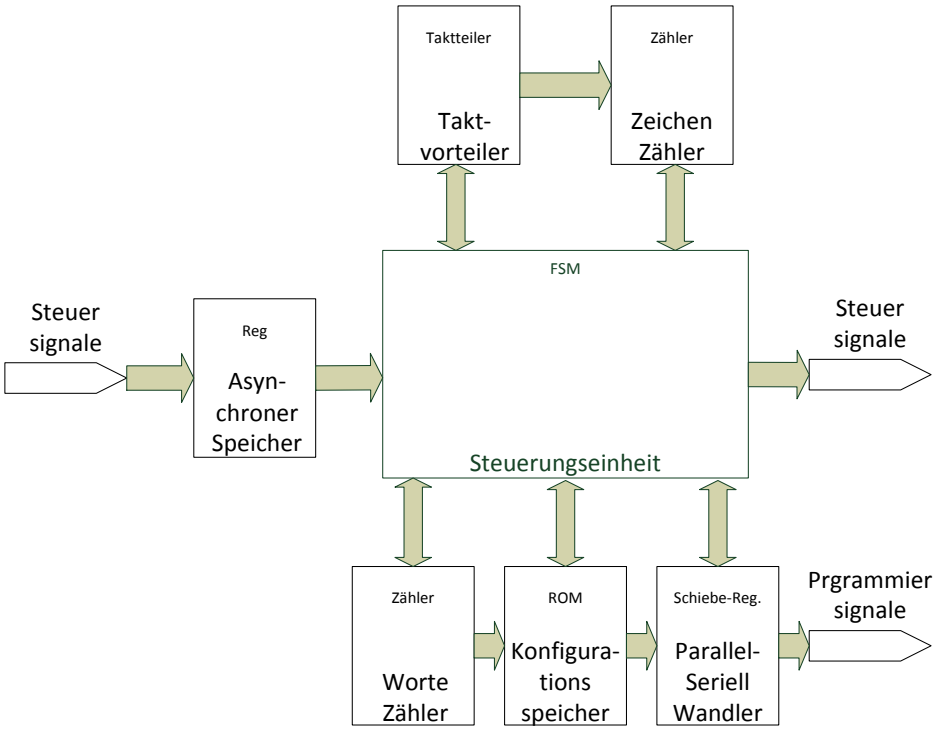


Bild 4.8.: Strukturelles Blockschaltbid für das ADC-Programmiermodul

Ein Taktvorteiler teilt den Systemtakt von $100\,\mathrm{MHz}$ auf den Programmiertakt. Der ADC benötigt einen Programmiertakt der zwischen wenigen Hertz bis zu $20\,\mathrm{MHz}$ liegt [32, Seite 18]. Der Taktvorteiler erzeugt einen Programmiertakt von $\frac{100\,\mathrm{MHz}}{128} \approx 781\,\mathrm{kHz}$. Nach jeder vollen Periode wird ein Ein-Systemtakt langes Steuerungssignal gesendet. Dieses Steuerungssignal wird an ein Zeichenzählermodul und an eine Steuerungseinheit übergeben. Im Anhang befindet sich der VHDL-Code C.1 des Taktvorteilers. Mit ModelSim und einem *Do-File* wurde die Funktion des Taktvorteilers verifiziert. Das Do-File C.2 zur Erzeugung einer Simuli und das Simulationsergebnis C.2 sind ebenfalls im Anhang hinterlegt.

Ein Zeichenzähler zählt die Anzahl der gesendeten Zeichen. Ein Programmierwort besteht aus einer $8\,\mathrm{Bit}$ Adresse und einem $8\,\mathrm{Bit}$ langen Datenwort. Zusammen ist ein Programmierwort somit $16\,\mathrm{Bit}$ lang. Der Zeichenzähler sendet nach 16 Zeichen ein Ein-Systemtakt langes Steuerungssignal an die Steuerungseinheit. Der Zeichenzähler wird durch das Steuerungssignal des Taktvorteilers und durch die Steuerungseinheit gesteuert. Im Anhang befindet sich der VHDL-Code C.3 des Zeichenzählers. Mit ModelSim und einem *Do-File* wurde die Funktion des Zeichenzählers verifiziert. Das Do-File C.4 zur Erzeugung einer Simuli und das Simulationsergebnis C.3 sind ebenfalls im Anhang hinterlegt.

Ein Wortezähler zählt die Anzahl der gesendeten Programmierworte. Bis die Programmierung fertig ist, müssen 19 Programmierworte vom Modul gesendet werden. Der Wortezäh-

ler sendet nach 19 Programmierworten ein Ein-Systemtakt langes Steuerungssignal an die Steuerungseinheit. Der aktuelle Zählerstand wird an einen ROM, der als Konfigurationsspeicher dient, übergeben. Der Wortezähler wird durch die Steuerungseinheit gesteuert. Im Anhang befindet sich der VHDL-Code C.5 des Wortezählers. Mit ModelSim und einem *Do-File* wurde die Funktion des Wortezählers verifiziert. Das Do-File C.6 zur Erzeugung einer Simuli und das Simulationsergebnis C.4 sind ebenfalls im Anhang hinterlegt.

Ein Konfigurationsspeicher, realisiert als ROM , beinhaltet die Adressen und die Daten mit denen der ADC programmiert werden soll. Der Wortezähler dient dem Konfigurationsspeicher als Adresszuweisung. Adressen und Daten werden am Ausgang des Konfigurationsspeicher an einen Paralell-Seriell-Wandler übergeben. Die Adressen und Daten, die im Konfigurationsspeicher enthalten sind, wurden im Konfigurationsspeicher initialisiert. Zur Erzeugung der Initialisierung wurde ein MATLAB-Skript geschrieben C.8. Das MATLAB-Skript erzeugt eine TXT-Datei, aus der die Initialisierung in den VHDL-Code kopiert werden kann. Im Anhang befindet sich der VHDL-Code C.7 des Konfigurationsspeichers. Mit ModelSim und einem *Do-File* wurde die Funktion des Konfigurationsspeichers verifiziert. Das Do-File C.9 zur Erzeugung einer Simuli und das Simulationsergebnis C.5 sind ebenfalls im Anhang hinterlegt.

Ein Parallel-Seriell-Wandler schiebt die einzelnen Bits mit dem Programmiertakt auf eine serielle Datenleitung. Gesteuert wird der Parallel-Seriell-Wandler durch die Steuerungseinheit. Die parallelen Daten kommen aus dem Konfigurationsspeicher. Im Anhang befindet sich der VHDL-Code C.10 des Parallel-Seriell-Wandlers. Mit ModelSim und einem *Do-File* wurde die Funktion des Parallel-Seriell-Wandlers verifiziert. Das Do-File C.11 zur Erzeugung einer Simuli und das Simulationsergebnis C.6 sind ebenfalls im Anhang hinterlegt.

Ein asynchroner Speicher, realisiert als Register, speichert ein externes Steuerungssignal zwischen. Dieses Steuerungssignal stößt den Programmierprozess an. Asynchrone Speicher werden immer dann eingesetzt, wenn die Eingangssignale nicht taktsynchron zu erwarten ist. Im aktuellen Fall handelt es sich um einen externen Push-Button. Asynchrone Speicher verhindern außerdem, dass die Steuerungseinheit, meist realisiert durch eine Finite-state machine (FSM), keine Steuerungssignale verpassen. Um einen asynchronen Speicher zu realisieren, wird der asynchrone Eingang auf das Enable-Signal des Registers, sowie der Signalzustand *high* dauerhaft auf den Dateneingang gelegt. Im Anhang befindet sich der VHDL-Code C.12 des asynchronen Speichers. Mit ModelSim und einem *Do-File* wurde die Funktion des asynchronen Speichers verifiziert. Das Do-File C.13 zur Erzeugung einer Simuli und das Simulationsergebnis C.7 sind ebenfalls im Anhang hinterlegt.

Eine Steuerungseinheit, realisiert als FSM, sorgt dafür, dass alle Module im Einklang miteinander funktionieren. Im Bild 4.9 ist das Zustandsdiagramm der FSM dargestellt. Die FSM ist als Moore-Automat realisiert worden. Diese Maßnahme macht den Zustandsautomaten im vorhersehbaren Verhalten sicherer, da keine Eingangssignale direkt auf den Ausgang wirken

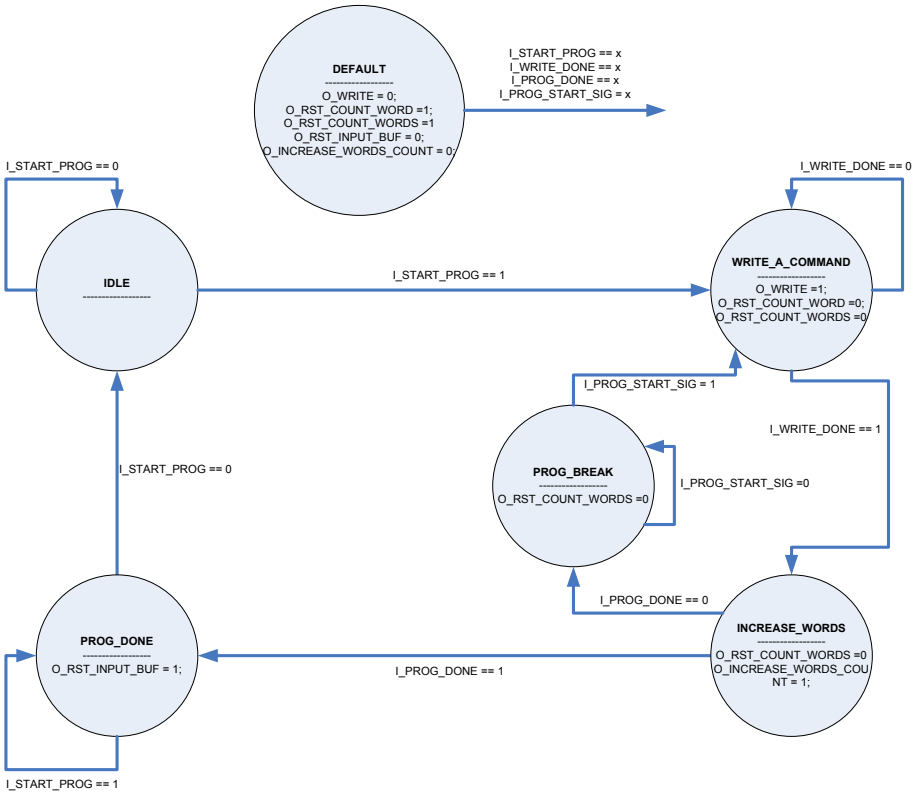


Bild 4.9.: Zustandsdiagramm der Steuerungseinheit

können [19, Seite 117 f]. Der Zustandsautomat hat 5 Zustände. Vom IDLE-Zustand startet der Automat durch Drücken des Programmierbuttons (I_START_PROG) und wechselt in den WRITE_A_COMMAND-Zustand. Ist ein Programmierwort komplett gesendet worden, sendet das oben beschriebene Zeichenzähler-Modul das Steuerungssignal (I_WRITE_DONE). Der Zustandsautomat wechselt in den INCREASE_WORDS-Zustand. Am Steuerungssignal (I_PROG_DONE) ist erkenntlich, ob bereits alle Programmierwörter gesendet worden sind. Ist dies der Fall, wechselt der Zustandsautomat in den PROG_DONE-Zustand. Ist dies nicht der Fall, wechselt der Zustandsautomat in den PROG_BREAK-Zustand. Im PROG_BREAK-Zustand wartet der Zustandsautomat einen weiteren Programmiertakt, signalisiert durch das Steuerungssignal (I_PROG_STARTSIG), ab, bevor wieder in den WRITE_A_COMMAND-Zustand gewechselt wird. Dies ist nötig, da nach einigen Programmierbefehlen eine Pause gewährleistet werden muss [32, Seite 20]. Ist der Zustandsautomat im PROG_DONE-Zustand, sendet der Zustandsautomat an den asynchronen Speicher ein Reset-Signal (0_RST_INPUT_BUF). Vom PROG_DONE-Zustand wird erst in den IDLE-Zustand gewechselt, sobald der asynchrone Speicher gelöscht ist. Dies ist durch das Steuerungssignal (I_START_PROG) erkenntlich. Der VHDL-Code der Steuerungseinheit C.14 zeigt die Umsetzung des Zustandsdiagramms. Im VHDL-Code wurde der Zustandsautomat als Zwei-

Prozess-Zustandsautomat realisiert, da dies dem RTL-Modellierungsstil entspricht [19, Seite 346]. Der VHDL-Code des Zustandsautomaten wurde, eingebettet im Top-Level-Design, simuliert. Dies ermöglicht die natürliche Erzeugung einer Stimuli. Damit dies möglich ist, muss im Top-Level-Design für alle untergeordneten Komponenten die Konfiguration zur Auswahl von Modellarchitekturen festgelegt werden [19, Seite 163]. Im VHDL-Code C.16 des Top-Level-Designs ist dies in Zeile 115 bis 121 vorgenommen worden. Außerdem müssen im Do-File die einzelnen Komponenten geladen werden C.15. Bei der Simulation der Steuerungseinheit wurde auf viele verschiedene Funktionen geachtet. Das dargestellte Simulationsergebnis im Bild C.8 zeigt, dass der Zustandsautomat vom Anfang bis zum Ende alle Zustände durchläuft. Dargestellt ist hier nur der Übersprung vom letzten Zustand zurück in den IDLE-Zustand. Außerdem ist zu sehen, dass alle Steuerungsignale empfangen und gesendet werden. Eine genaue Betrachtung erfolgt im Anhang.

Im Top-Level-Design des Programmiermoduls werden alle Komponenten deklariert und instanziiert . Es wird auch die Konfiguration zur Auswahl von Modellarchitekturen vorgenommen. Im Anhang befindet sich der VHDL-Code C.16 des Top-Level-Designs. Mit ModelSim und einem *Do-File* wurde die Funktion des Top-Level-Designs verifiziert. Das Do-File C.17 zur Erzeugung einer Simuli und das Simulationsergebnis C.9 sind ebenfalls im Anhang hinterlegt.

Das RTL-Schematic, abgebildet im Bild 4.10, gilt als letzte Verifikation des Moduls, bevor dieses getestet wurde. Die Signalleitungen wurden durch das Synthesetool so miteinander

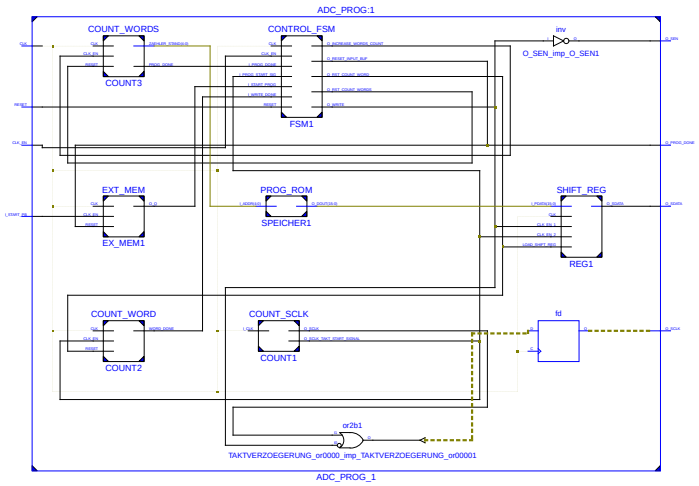


Bild 4.10.: RTL-Schematic des Programmiermoduls (Top Level Design)

verbunden, wie dies nach dem detaillierten Blockschaltbild C.1 bezweckt wurde. Bild 4.11 zeigt eine Messung der Programmierleitung mittels Oszilloskop. Die unterste Kurve ist die

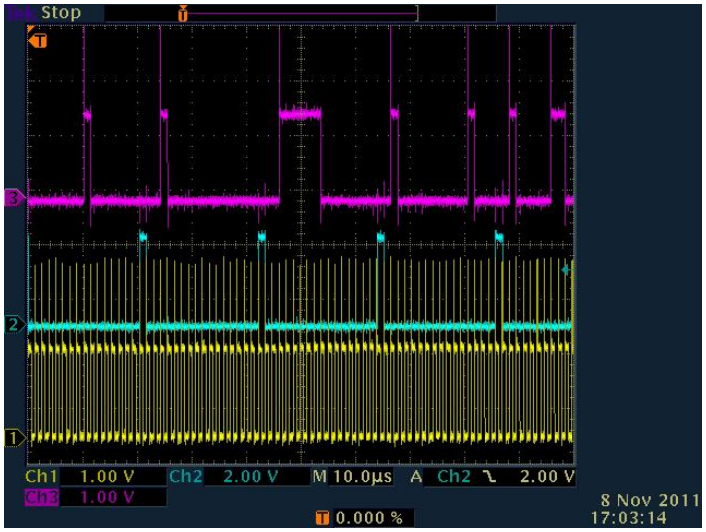


Bild 4.11.: Messung der Programmierleitung mittels Oszilloskop

Messung des Programmiertakts. Die mittlere Kurve ist die Messung des Enable-Signals. Die obere Kurve ist die Messung des seriellen Datensignals. Durch Vergleich der gesendeten Daten mit den bei der Messung aufgenommen Daten wurde festgestellt, dass die Programmierung ordnungsgemäß funktioniert. Einzig die starken Überschwinger beim Programmiertakt und dem Datensignal sind unerwünscht. Dies liegt wieder daran, dass Treiber und Datenpfade nicht denselben Widerstand haben. Durch Einlöten von Sender-Widerständen auf der Adapterplatine wurde dieses Problem, wie in Bild 4.12 zu sehen ist, behoben. Zur besseren Verdeutlichung wurde in die unterste Kurve, den Programmiertakt, hinein gezoomt. Es ist zu erkennen, dass keine Überschwinger mehr vorkommen. Um zu zeigen, dass das ADC-Interface mit der Programmierung ordnungsgemäß funktioniert, wurde ein Sinussignal mit dem ADC ausgenommen. Mit ChipScope wurden die vom FPGA ausgenommenen Daten ausgelesen. Bild 4.13 zeigt einen Screen Shoot von ChipScope bei der Datenaufnahme.

4.3. Versorgung des DAC und des ADC mit einem Clock

Für die Versorgung des ADCs und des DACs wurde das Clockboard *CDCE62005-Eval* [31] von *Texas Instruments* verwendet. Dieses erreicht unter Verwendung des internen Referenztakts einen minimalen Root mean square (RMS)-Jitter von $0.405\ \mathrm{ps}$ [31, Seite 19]. Mit

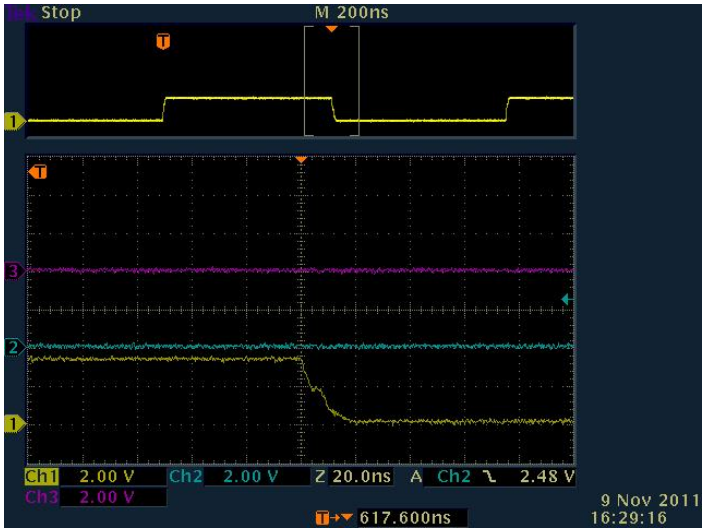


Bild 4.12.: Messung der Programmierleitung mittels Oszilloskop mit Sender-Widerständen

Formel 4.8 kann der RMS-Jitter in ein SNR am Ausgang eines idealen ADCs bzw. DACs umgerechnet werden [22, Seite 2.63 f].

$$\mathrm{SNR} = 20 \log_{10} \left[\frac{1}{2\pi f_a t_j} \right] \tag{4.8}$$

Dabei ist t_j der RMS-Jitter und f_a die Frequenz einer analogen Sinusquelle, betrieben im Wandlervollausschlag (fullscale), am Wandlereingang. Da nach der Formel 4.8 der SNR mit steigender Frequenz sinkt, ist in die Formel die maximale verwendete Frequenz einzutragen. Der ADC arbeitet mit einer Abtastrate von $240\,\mathrm{MHz}$. Somit ist die maximal zulässige Frequenz nach dem Nyquist-Shannon-Abtasttheorem $120\,\mathrm{MHz}$. Der DAC kann am Ausgang zwar eine weitaus höhere Frequenz ausgeben, aber diese können dann nicht vom ADC aufgenommen werden. Somit sollte die höchste Frequenz die am DAC ausgegeben wird auch maximal $120\,\mathrm{MHz}$ sein. Die hohe Ausgangsrate am DAC kommt verminderten Spezifikationsanforderungen an dem Rekonstruktionsfilter zu gute. Durch Einsetzen der maximalen verwendeten Frequenz und der RMS-Jitter-Performance des Clockboards ergibt sich mit Formel 4.8 ein SNR von

$$\mathrm{SNR} = 20 \log_{10} \left[\frac{1}{2\pi \cdot 120\,\mathrm{MHz} \cdot 0.405\,\mathrm{ps}} \right] = 70.3\,\mathrm{dB}. \tag{4.9}$$

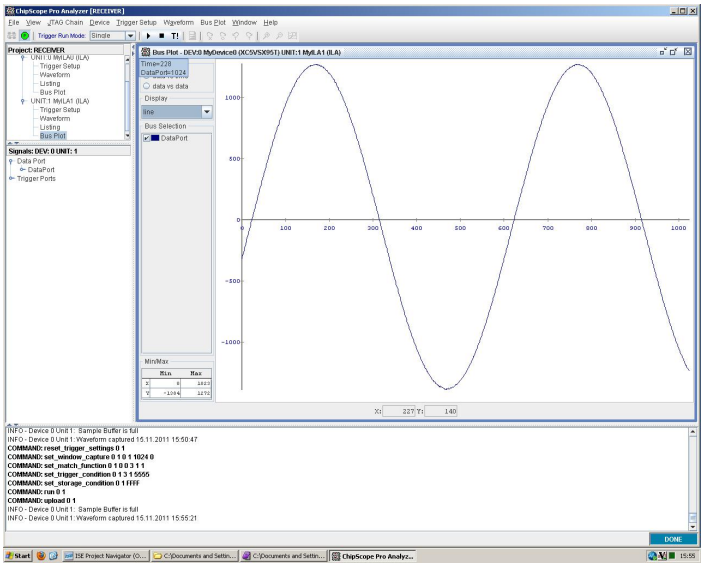


Bild 4.13.: Sinussignal abgetastet mit dem ADC und ausgelesen mit ChipScope

Damit ist das Ausgangssignal auf mehr als $\frac{1}{1000}$ genau. Mit Formel 4.10 wird der SNR in die Effective number of bits (ENOB) umgerechnet [22, Seite 2.64].

$$\mathrm{ENOB} = \frac{\mathrm{SNR} - 1.76}{6.02} = \frac{70.3 - 1.76}{6.02} = 11.4 \tag{4.10}$$

Die berechneten $11.4\,\mathrm{Bit}$ erreichen zwar nicht die maximale Leistungsfähigkeit der Wandler, jedoch ist ein SNR von $70.3\,\mathrm{dB}$ ausreichend.

5. Implementierung der digitalen Signalaufarbeitung

In diesem Kapitel werden alle nötigen Schritte aufgeführt, die für eine Datenübertragung via Funkstrecke neben der Modulation und der Hardwareanbindung nötig sind. Dazu gehört die Impulsformung der gesendeten Datensymbole 5.1, das Rekonstruktionsfilter und das Anti-Aliasing-Filter 5.3, die Mischung des Basisbandsignals in eine Zwischenfrequenz und vice versa 5.2, sowie die Takt und Datenrückgewinnung 5.4.

5.1. Impulsformung

Eine Impulsformung formt einen Impuls so, dass er bandbegrenzt ist und zeitlich benachbarte Impulse nicht beeinflusst. Die einzelnen zu übertragenden Daten werden in wertediskreten Symbolimpulsen codiert. Das Übersprechen eines Symbolimpulses zu einem zeitlich folgenden Symbolimpuls ist dabei unerwünscht. Dieses zeitliche Übersprechen wird als Intersymbolinterferenz (ISI) bezeichnet.

Wie bereits einleitend erwähnt, wird eine kontrollierte Impulsformung benötigt um ISI zu verhindern. Um ISI zu verhindern muss ein Impulsformungsfilter $h[n \cdot T]$, mit T als Symboldauer, entworfen werden, welches zum Zeitpunkt $n = 0$ des Symbolimpulses den Wert des Symbolimpulses überträgt und für alle vorhergegangenen und folgenden Symbolimpulse $n \neq 0$ den Wert des Symbolimpulses zu 0 setzt. Bedingung 5.1 verdeutlicht dies.

$$h[n \cdot T] = \begin{cases} 1; & n = 0 \\ 0; & n \neq 0 \end{cases} \tag{5.1}$$

Wie [21, Seite 234 ff] zeigt, führt diese Bedingung entweder zu einem idealen Tiefpass oder zu einem so genannten Raised-Cosine-Filter. Der ideale Tiefpass ist nicht realisierbar, da dieses eine unendliche Impulsantwort hat. Das Raised-Cosine-Filter hat eine Impulsantwort nach Formel 5.2.

$$h(t) = \frac{1}{T} \cdot \frac{\sin\left(\pi \frac{t}{T}\right)}{\pi \frac{t}{T}} \cdot \frac{\cos\left(\pi r \frac{t}{T}\right)}{1 - \left(2\pi r \frac{t}{T}\right)^2} \tag{5.2}$$

Dabei ist t die Zeit. Für den Zeitpunkt $t = 0$ ist der Faktor $\frac{\sin\left(\pi \frac{t}{T}\right)}{\pi \frac{t}{T}}$ zu 1 definiert. Der Faktor r ist der sogenannte *Roll-off-Faktor*. Der Roll-off-Faktor bestimmt die Bandbreite des Signals bzw. den Übergang vom Durchlassbereich zum Sperrbereich. Dies verdeutlicht auch die Übertragungsfunktion des Filters 5.3.

$$H[j\omega] = \begin{cases} 1 & \text{für } \frac{|\omega|}{\omega_N} \leq 1 - r \\ \frac{1}{2}\left[1 + \cos\left\{\frac{\pi}{2r}\left(\frac{\omega}{\omega_N} - (1-r)\right)\right\}\right] & \text{für } 1 - r \leq \frac{|\omega|}{\omega_N} \leq 1 + r \\ 0 & \text{für } \frac{|\omega|}{\omega_N} \geq 1 + r \end{cases} \tag{5.3}$$

Ein solches Raised-Cosine-Filter ist im Interpolationsfilter des DAC integriert[3, Seite 38]. Der verwendete Roll-off-Faktor ist leider nicht im Datenblatt angegeben. Jedoch lässt folgende Simulation des Interpolationsfilter auf einen Roll-off-Fakor von $r \approx 0{,}2$ schließen. Mit MATLAB/Simulink wurde das Interpolationsfilter simuliert. Bild 5.1 zeigt den Simulationsaufbau. Die Koeffizienten aus dem Datenblatt [3, Seite 38] wurden in die *INTERPOLATION*s-

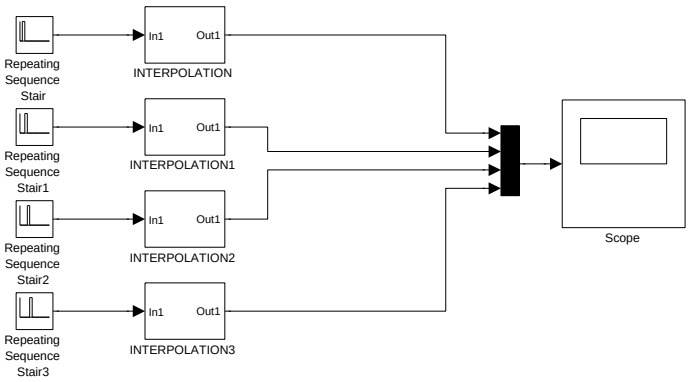


Bild 5.1.: MATLAB/Simulink Modell zur Simulation des Interpolationsfilter

Blöcke implementiert. Jeder Block wird mit einem Symbolimpuls gespeist. Die Symbolimpulse sind je um einen Takt versetzt. Die Ausgänge der *INTERPOLATION*s-Blöcke werden mit einem Multiplexer übereinander gelegt und an das Scope übergeben. Die resultierende Kurve, aufgenommen mit dem Scope, wird in Bild 5.2 dargestellt. Vier farblich zu unterscheidende Kurven repräsentieren je eine Impulsantwort. Jede Impulsantwort erreicht einen Maximalwert von 1. Jeweils sind zu diesem Zeitpunkt alle anderen Impulsantworten genau 0. Somit wird jedes Symbol zum Zeitpunkt des Symbols durch kein vorangegangenes und durch kein folgendes Symbol beeinflusst.

Durch Aufnahme eines Augendiagramms kann auch festgestellt werden, ob sich die Symbole gegenseitig beeinflussen. Findet keine Beeinflussung statt, so ist das Augendiagramm vertikal maximal geöffnet. Die horizontale Öffnung wird durch den Roll-off-Faktor bestimmt. Je

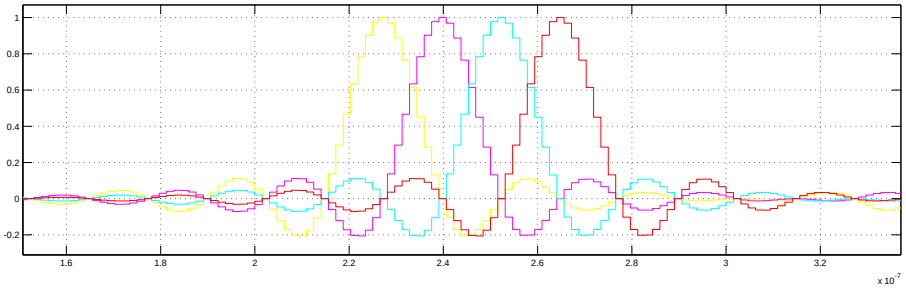


Bild 5.2.: MATLAB/Simulink Ergebnis zur Simulation des Interpolationsfilter

kleiner der Roll-off-Faktor, je geringer die benötigte Bandbreite und je höher die Anforderung an die spätere Takt- und Datenrückgewinnung. Bild 5.3 zeigt das MATLAB/Simulink Modell zur Aufnahme eines Augendiagramms. Zwei Datenpfade sind ausgeführt worden. Ein Daten-

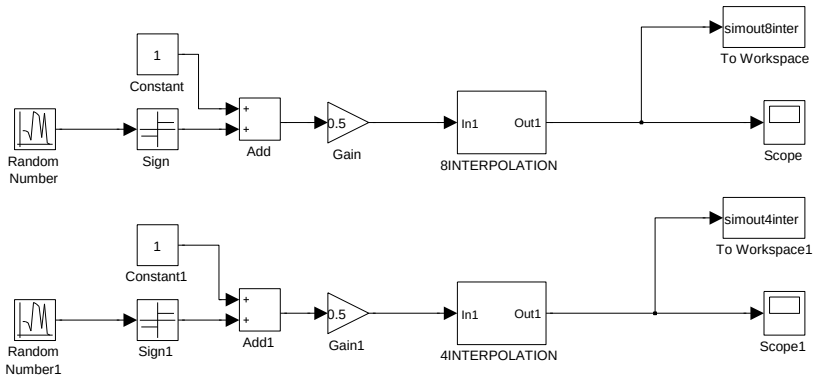
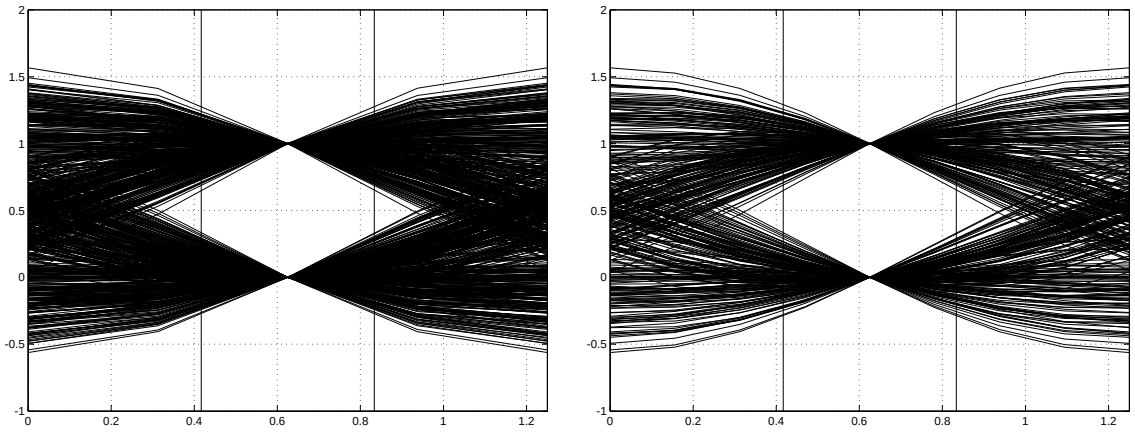


Bild 5.3.: MATLAB/Simulink Modell zur Augendiagrammssimulation des Interpolationsfilters

pfad für eine 8-fache Interpolation und ein Datenpfad für eine 4-fache Interpolation. Dies kann wahlweise im DAC eingestellt werden. Ein Zufallsgenerator erzeugt ein Zufallssignal. Dieses Zufallssignal wird mit einem Vorzeichenoperator auf den Wert -1 und 1 gemappt. Durch Addition mit 1 und anschließender Multiplikation mit dem Faktor $0{,}5$ wird eine Zufallsfolge mit den Werten 0 und 1 erzeugt. Diese Zufallsfolge wird je an die Interpolationsfilter übergeben. Das Ausgangssignal der Interpolationsfilter wird an den MATLAB-Workspace übergeben. In MATLAB kann mit dem MATLAB-Code D.1 aus den Daten ein Augendiagramm erzeugt werden. Bild 5.4 zeigt das Augendiagramm der MATLAB/Simulink Simulation.



(a) 4-fach Interpolation (b) 8-fach Interpolation

Bild 5.4.: MATLAB/Simulink Augendiagrammssimulation der Interpolationsfilter

Wie erwartet ist das Auge vertikal maximal geöffnet. Die horizontale Augenöffnung kann dagegen auf die Hälfte geschätzt werden. Durch Vergleich mit den in [21, Seite 240] angegebenen Augendiagrammen, kann der Roll-off-Faktor auf $r \approx 0{,}2$ geschätzt werden.

5.2. Mischung

Unter Mischung versteht man einen Signalverarbeitungsprozess, der zwei Signale unterschiedlicher Frequenz in neue Signale mit Differenzfrequenzen und Summenfrequenzen transformiert. In der Kommunikationstechnik werden mit diesem Verfahren Signalbänder in einen anderen Frequenzbereich geschoben. Beispielsweise, wie im konkreten Fall, ein Basisbandsignal in ein Zwischenfrequenzsignal.

Die Millimeterwellen-Frontends können keine Signale mit Gleichanteil übertragen, da diese AC gekoppelt sind und im niederfrequenten Bereich eine sehr starke IQ-Imbalance vorweisen. Die Basisbandsignale müssen in eine Zwischenfrequenz gemischt werden, bevor diese an die Millimeterwellen-Frontends übergeben werden können. Durch Mischung der Basisbandsignale in eine Zwischenfrequenz wird das Signalband aus dem niederfrequenten Bereich in einen höheren Frequenzbereich geschoben. Das Signalband in der Zwischenfrequenz kann anschließend mit einer besseren IQ-Balance-Performance übertragen werden.

5.2.1. Mathematischer Hintergrund

Die Mischung erfolgt mit Hilfe einer Multiplikation im Zeitbereich. Anhand eines Beispielssignals mit der Frequenz f_e, welches mit der Mischfrequenz f_0 in eine Zwischenfrequenz gemischt wird, soll der mathematische Hintergrund erläutert werden.

Es sei

$$U_e = A_e \cos\left(t 2\pi f_e\right) \tag{5.4}$$

das Eingangssignal in einen Mischer und

$$U_0 = 1 \cos\left(t 2\pi f_0\right) \tag{5.5}$$

dass Mischsignal, welches auch als Lokaloszillator bezeichnet wird. Durch Multiplikation der beiden Signal erhält man, durch den trigonometrischen Zusammenhang

$$\cos\left(\alpha\right) \cdot \cos\left(\beta\right) = \frac{1}{2}\left[\cos\left(\alpha - \beta\right) + \cos\left(\alpha + \beta\right)\right], \tag{5.6}$$

das Ausgangssignal

$$U_a = U_e \cdot U_0 = \frac{1}{2} \cdot A_e \cdot \left[\cos\left(t 2\pi (f_0 - f_e)\right) + \cos\left(t 2\pi (f_0 + f_e)\right)\right]. \tag{5.7}$$

Das Ausgangssignal spaltet sich in zwei Terme auf. Der erste Term $\frac{1}{2} \cdot A_e \cdot \cos\left(t 2\pi (f_0 - f_e)\right)$ wird als unteres Seitenband U_u bezeichnet und der zweite Term $\frac{1}{2} \cdot A_e \cdot \cos\left(t 2\pi (f_0 + f_e)\right)$ wird als oberes Seitenband U_o bezeichnet. Wie ersichtlich, steckt in dem Ausgangssignal durch die zwei Terme, Redundanz. Dies ist unerwünscht, da der volle Informationsgehalt in einem Seitenband enthalten ist und zwei Seitenbänder doppelt so viel Bandbreite in einem Übertragungskanal benötigen, wie ein Seitenband. Ein Seitenband könnte durch Filterung unterdrückt werden. Es zeigt sich jedoch, dass es eine geschicktere Möglichkeit gibt. Wird der Lokaloszillator und das Eingangssignal um $90°$ verzögert, so, dass das Eingangssignal nun

$$U_{e,-90°} = A_e \sin\left(t 2\pi f_e\right) \tag{5.8}$$

sei und der Lokaloszillator nun

$$U_{0,-90°} = 1 \sin\left(t 2\pi f_0\right) \tag{5.9}$$

sei, dann ergibt sich durch Multiplikation der beiden Signale, mit dem trigonometrischen Zusammenhang

$$\sin\left(\alpha\right) \cdot \sin\left(\beta\right) = \frac{1}{2}\left[\cos\left(\alpha - \beta\right) - \cos\left(\alpha + \beta\right)\right], \tag{5.10}$$

das Ausgangssignal

$$U_{a,-90^\circ} = U_{e,-90^\circ} \cdot U_{0,-90^\circ} = \frac{1}{2} \cdot A_e \cdot [\cos(t2\pi(f_0 - f_e)) - \cos(t2\pi(f_0 + f_e))]. \quad (5.11)$$

Durch Addition oder Subtraktion des Ausgangssignals U_a mit dem um 90° verzögerten Ausgangssignal $U_{a,-90^\circ}$ erhält man eine Seitenbandunterdrückung. Denn

$$\begin{aligned} U_a + U_{a,-90^\circ} =& \frac{1}{2} \cdot A_e \cdot [\cos(t2\pi(f_0 - f_e)) + \cos(t2\pi(f_0 + f_e))] + \\ & \frac{1}{2} \cdot A_e \cdot [\cos(t2\pi(f_0 - f_e)) - \cos(t2\pi(f_0 + f_e))] \\ =& A_e \cdot \cos(t2\pi(f_0 - f_e)) = U_u \end{aligned} \quad (5.12)$$

und

$$\begin{aligned} U_a - U_{a,-90^\circ} =& \frac{1}{2} \cdot A_e \cdot [\cos(t2\pi(f_0 - f_e)) + \cos(t2\pi(f_0 + f_e))] - \\ & \frac{1}{2} \cdot A_e \cdot [\cos(t2\pi(f_0 - f_e)) - \cos(t2\pi(f_0 + f_e))] \\ =& A_e \cdot \cos(t2\pi(f_0 + f_e)) = U_o. \end{aligned} \quad (5.13)$$

Durch Addition erhält man das untere Seitenband U_u und durch Subtraktion erhält man das obere Seitenband U_o.

Im aktuellen Projekt wird ein voller IQ-Mischer (In-Phase-Quadratur-Phase-Mischer) verwendet. Ein voller IQ-Mischer erzeugt ein In-Phase und einen Quadratur-Phase-Signal. Dies ist nötig, damit ein weiterer Mischer das Zwischenfrequenzsignal in ein noch höheres Frequenzband mischen kann. Im aktuellen Projekt mischen die Millimeterwellen-Frontends das Zwischenfrequenzsignal in den Millimeterwellenbereich. Das In-Phase-Signal wurde bereits mit Formel 5.4 bis 5.12 hergeleitet. Durch Änderung der Notation der Eingangssignale von $U_{e,-90^\circ}$ zu $\mathrm{Im}\{\bar{U}_e\}$ und U_e zu $\mathrm{Re}\{\bar{U}_e\}$ erhält man

$$U_I = \mathrm{Re}\{\bar{U}_e\} \cdot \cos(t2\pi f_0) + \mathrm{Im}\{\bar{U}_e\} \cdot \sin(t2\pi f_0). \quad (5.14)$$

Die Herleitung für das Quadratur-Phase-Signal erfolgt ähnlich und resultiert in

$$U_Q = \mathrm{Re}\{\bar{U}_e\} \cdot \sin(t2\pi f_0) - \mathrm{Im}\{\bar{U}_e\} \cdot \cos(t2\pi f_0). \quad (5.15)$$

Um aus dem Zwischenfrequenzsignal wieder ein Basisbandsignal zu erhalten, wird das In-Phase-Signal und das Quadratur-Phase-Signal wie folgt erweitert:

$$\mathrm{Re}\{\bar{U}_e\} = U_I \cdot \cos(t2\pi f_0) + U_Q \cdot \sin(t2\pi f_0) \quad (5.16)$$

Der folgende Beweis soll dies zeigen: Durch Einsetzen von Formel 5.14 und Formel 5.15 in 5.16 ergibt sich

$$\begin{aligned}\mathrm{Re}\left\{\bar{U}_e\right\} = &\left[\mathrm{Re}\left\{\bar{U}_e\right\} \cdot \cos\left(t2\pi f_0\right) + \mathrm{Im}\left\{\bar{U}_e\right\} \cdot \sin\left(t2\pi f_0\right)\right] \cdot \cos\left(t2\pi f_0\right) + \\ &\left[\mathrm{Re}\left\{\bar{U}_e\right\} \cdot \sin\left(t2\pi f_0\right) - \mathrm{Im}\left\{\bar{U}_e\right\} \cdot \cos\left(t2\pi f_0\right)\right] \cdot \sin\left(t2\pi f_0\right).\end{aligned} \tag{5.17}$$

Durch Ausmultiplizieren erhält man

$$\begin{aligned}\mathrm{Re}\left\{\bar{U}_e\right\} = &\mathrm{Re}\left\{\bar{U}_e\right\} \cdot \cos^2\left(t2\pi f_0\right) + \mathrm{Im}\left\{\bar{U}_e\right\} \cdot \sin\left(t2\pi f_0\right) \cdot \cos\left(t2\pi f_0\right) + \\ &\mathrm{Re}\left\{\bar{U}_e\right\} \cdot \sin^2\left(t2\pi f_0\right) - \mathrm{Im}\left\{\bar{U}_e\right\} \cdot \sin\left(t2\pi f_0\right) \cdot \cos\left(t2\pi f_0\right).\end{aligned} \tag{5.18}$$

Der zweite und vierte Term lösen sich gegenseitig auf. Der erste und dritte Term werden zu

$$\mathrm{Re}\left\{\bar{U}_e\right\} = \mathrm{Re}\left\{\bar{U}_e\right\} \cdot \left[\cos^2\left(t2\pi f_0\right) + \sin^2\left(t2\pi f_0\right)\right]. \tag{5.19}$$

Der trigonometrische Zusammenhang

$$\cos^2\alpha + \sin^2\alpha = 1 \tag{5.20}$$

führt zu

$$\mathrm{Re}\left\{\bar{U}_e\right\} = \mathrm{Re}\left\{\bar{U}_e\right\}. \tag{5.21}$$

Damit ist bewiesen, was zu beweisen war.

Für den Imaginärteil gilt

$$\mathrm{Im}\left\{\bar{U}_e\right\} = U_I \cdot \sin\left(t2\pi f_0\right) - U_Q \cdot \cos\left(t2\pi f_0\right). \tag{5.22}$$

Der folgende Beweis soll dies zeigen: Durch Einsetzen von Formel 5.14 und Formel 5.15 in 5.22 ergibt sich

$$\begin{aligned}\mathrm{Im}\left\{\bar{U}_e\right\} = &\left[\mathrm{Re}\left\{\bar{U}_e\right\} \cdot \cos\left(t2\pi f_0\right) + \mathrm{Im}\left\{\bar{U}_e\right\} \cdot \sin\left(t2\pi f_0\right)\right] \cdot \sin\left(t2\pi f_0\right) - \\ &\left[\mathrm{Re}\left\{\bar{U}_e\right\} \cdot \sin\left(t2\pi f_0\right) - \mathrm{Im}\left\{\bar{U}_e\right\} \cdot \cos\left(t2\pi f_0\right)\right] \cdot \cos\left(t2\pi f_0\right).\end{aligned} \tag{5.23}$$

Durch Ausmultiplizieren erhält man

$$\begin{aligned}\mathrm{Im}\left\{\bar{U}_e\right\} = &\mathrm{Re}\left\{\bar{U}_e\right\} \cdot \cos\left(t2\pi f_0\right) \cdot \sin\left(t2\pi f_0\right) + \mathrm{Im}\left\{\bar{U}_e\right\} \cdot \sin^2\left(t2\pi f_0\right) - \\ &\mathrm{Re}\left\{\bar{U}_e\right\} \cdot \cos\left(t2\pi f_0\right) \cdot \sin\left(t2\pi f_0\right) + \mathrm{Im}\left\{\bar{U}_e\right\} \cdot \cos^2\left(t2\pi f_0\right).\end{aligned} \tag{5.24}$$

Der erste und dritte Term lösen sich gegenseitig auf. Der zweite und vierte Term werden zu

$$\mathrm{Im}\left\{\bar{U}_e\right\} = \mathrm{Im}\left\{\bar{U}_e\right\} \cdot \left[\sin^2\left(t2\pi f_0\right) + \cos^2\left(t2\pi f_0\right)\right]. \tag{5.25}$$

Der trigonometrische Zusammenhang 5.20 führt zu

$$\mathrm{Im}\left\{\bar{U}_e\right\} = \mathrm{Im}\left\{\bar{U}_e\right\}. \tag{5.26}$$

Damit ist bewiesen, was zu beweisen war.

5.2.2. Realisierung der Mischung vom Basisband in die Zwischenfrequenz

Eine Signalmischung kann sowohl analog als auch digital erfolgen. Die digitale Mischung hat den Vorteil, dass der 90° Phasenversatz vom Oszillator genauer erfolgen kann. Die Genauigkeit des Phasenversatzes ist ausschließlich durch die Wortlänge des Phasenakkumulators begrenzt. Der verwendete DAC hat einen integrierten NCO, mit dem das digitale Signal in die Zwischenfrequenz gemischt werden kann, bevor es in ein analoges Signal gewandelt wird [3, Seite 40]. Hierzu interpoliert der ADC das Eingangssignal zuvor wahlweise 2, 4 oder 8-fach. Bild 5.5 zeigt ein OOK Spektrum am Ausgang des DAC bei 8-fach Interpolation und Mischung auf $64{,}3\,\mathrm{MHz}$.

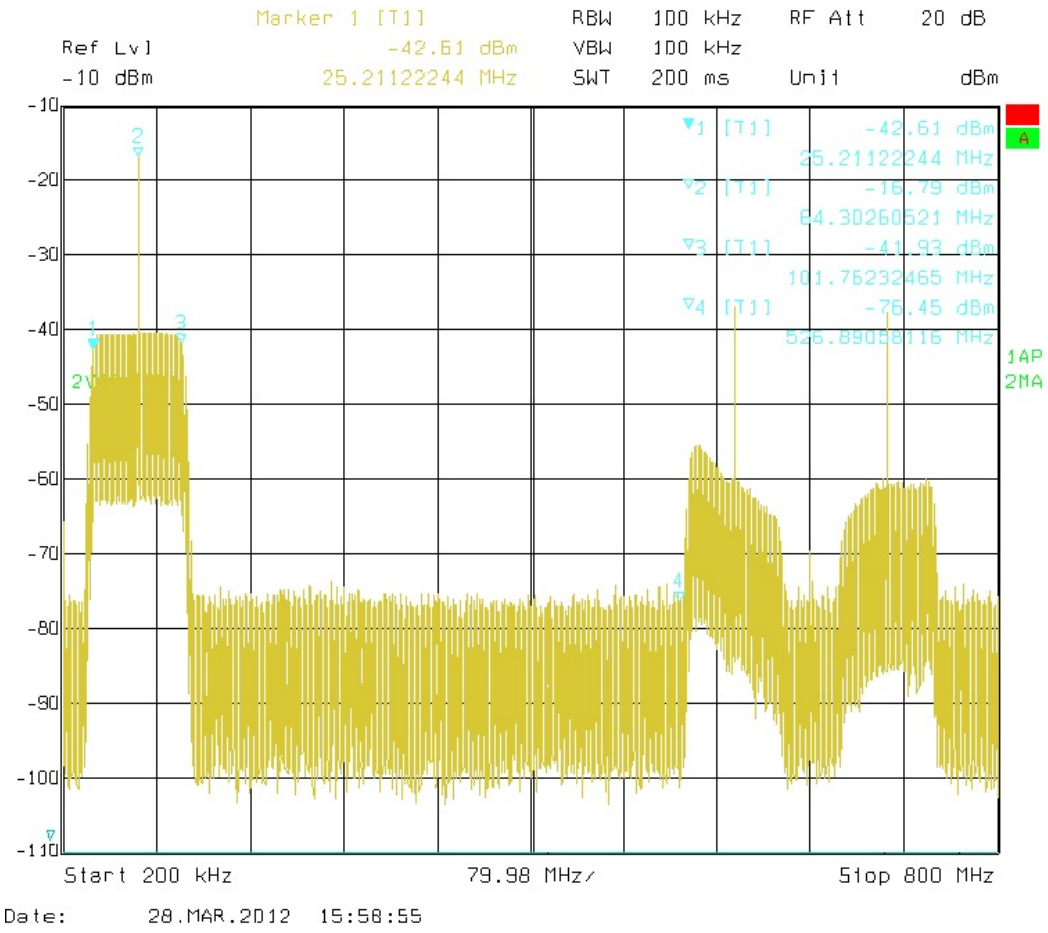


Bild 5.5.: OOK Spektrum am Ausgang des DAC bei 8-fach Interpolation und Mischung auf 64,3 MHz

Das Signalband geht von 25,2 MHz bis 101,8 MHz. Das erste Image fängt bei 526,9 MHz an. Ein Rekonstruktionsfilter müsste eine Passbandeckfrequenz von 101,8 MHz und eine maximale Dämpfung bei 526,9 MHz aufweisen. Der erste Aufbau wird jedoch mit 4-fach Interpolation durchgeführt. Im Kapitel 5.3 wird darauf genauer eingegangen. An dieser Stelle soll erst die Realisierung der Mischung von der Zwischenfrequenz in das Basisband erläutert werden.

5.2.3. Realisierung der Mischung von der Zwischenfrequenz in das Basisband

Der verwendete ADC hat keinen integrierten NCO. Somit muss die Mischung von der Zwischenfrequenz in das Basisband im FPGA implementiert werden. Formel 5.16 und Formel

5.22 wurden mit Hilfe von System Generator implementiert. Bild 5.6 zeigt das implementierte Modell.

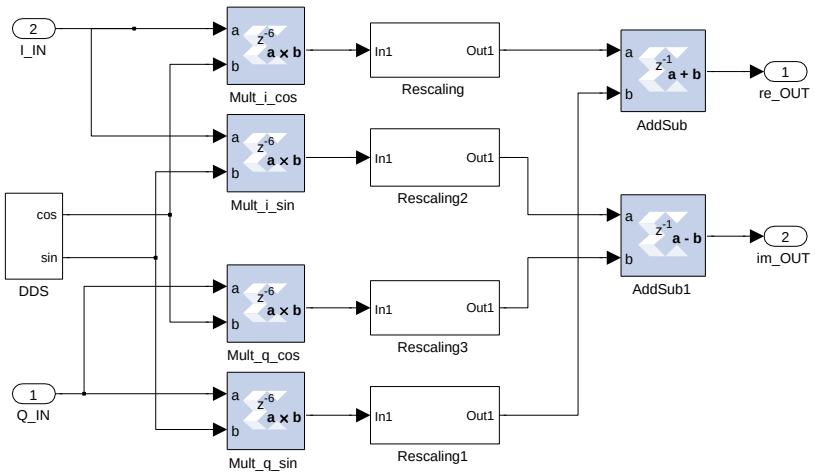
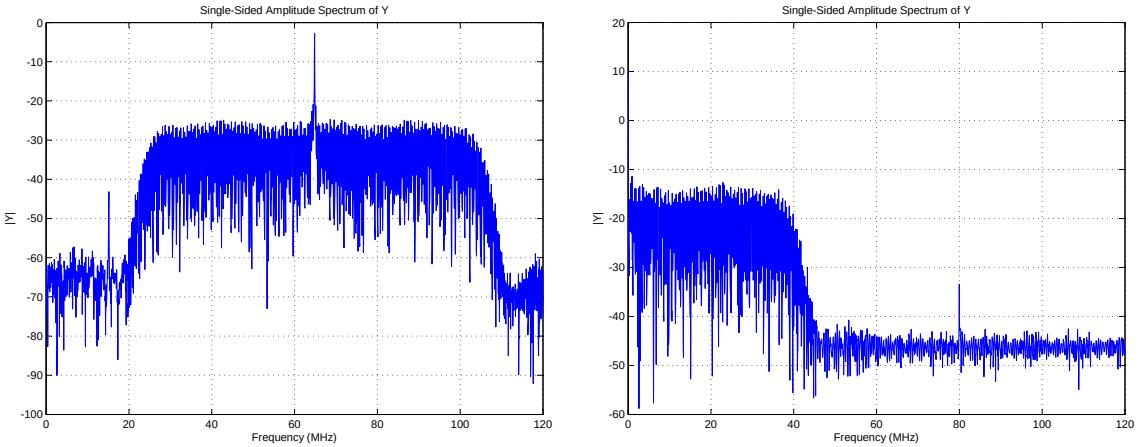


Bild 5.6.: In System Generator implementierter Mischer

Hinter den Multiplizierern sind *Rescaling*-Blöcke eingefügt worden. Diese *Rescaling*-Blöcke schneiden die unteren Bits nach einer Multiplikation ab. Dies ist nötig, da die Ausgangswortlänge der Summe der beiden Eingangswortlängen entspricht. Die Datenpfade sind auf 14 Bit Genauigkeit festgelegt worden, da der ADC Daten mit 14 Bit Wortlänge liefert. Bild 5.7 zeigt das simulierte OOK In-Phase-Spektrum am *I_IN* Eingang des Mischers und das resultierende OOK Realteil-Spektrum am *re_OUT* Ausgang des Mischers.



(a) *I_IN* Eingang des Mischers

(b) *re_OUT* Ausgang des Mischers

Bild 5.7.: OOK Spektren am Ein- und Ausgang des im FPGA implementierten Mischers

Durch Vergleich der beiden Spektren ist erkenntlich, dass der Mischer das Eingangssignal von der Zwischenfrequenz in das Basisband mischt. Am Ausgangsspektrum ist, neben dem Basisbandsignal bei $80\,\mathrm{MHz}$, der unterdrückte Symboltakt erkenntlich. In der Simulation ist somit gezeigt worden, dass das Mischermodul funktioniert. Damit dies auch in der Hardware funktioniert, müssen Rekonstruktionsfilter und Anti-Aliasing-Filter hinter den DAC bzw. vor den ADC geschaltet werden. Darauf wird im folgenden Kapitel eingegangen.

5.3. Rekonstruktionsfilter und Anti-Aliasing-Filter

Rekonstruktionsfilter sorgen für eine Wandlung eines diskreten Signals in ein analoges Signal. Ein Rekonstruktionsfilter kann auch als Interpolationsfilter verstanden werden, da es bestimmt, wie die Werte zwischen zwei diskreten Werten interpoliert werden. Ohne Rekonstruktionsfilter würde sich das Spektrum des DAC-Ausgangssignals unendlich vorsetzen. Dies ist unerwünscht, da die Bandbreite des zu übertragenen Signals so gering wie möglich gehalten werden soll.

Anti-Aliasing-Filter bereiten ein analoges Signal für eine Digitalisierung vor. Ein Anti-Aliasing-Filter begrenzt die Bandbreite eines analogen Signals. Der theoretische Hintergrund zum Aliasing und die daraus resultierenden Anforderung an ein Anti-Aliasing-Filter werden in Kapitel 5.3.2 betrachtet.

5.3.1. Theorie der Rekonstruktion

Um die Notwendigkeit des Rekonstruktionsfilters zu erklären, lässt sich die Theorie der Abtastung heranziehen. Folgend wird dargestellt, wie ein analoges Signal digitalisiert wird und anschließend dieses digitalisierte Signal wieder komplett analog rekonstruiert wird. Ein analoges Signal $s(t)$ mit dem Spektrum $S(f)$ wird durch einen idealen Abtaster digitalisiert.

$$s_a(t) = s(t) \cdot \sum_{n=-\infty}^{\infty} \delta(t - nT) \tag{5.27}$$

Das daraus entstehende Spektrum lautet:

$$S_a(f) = S(f) * \left[\frac{1}{T} \sum_{n=-\infty}^{\infty} \delta\left(f - \frac{n}{T}\right) \right] \tag{5.28}$$

Das Ausgangssignal eines DACs entspricht dabei dem zu repräsentierenden Signal $s(t)$, welches durch einen realen Abtaster mit Rechteckfunktion der Breite T diskretisiert wurde.

$$s_{\mathrm{DAC,out}}(t) = \mathrm{rect}\left(\frac{t}{T}\right) * \left[s(t) \sum_{n=-\infty}^{\infty} \delta(t - nT) \right] \tag{5.29}$$

Das daraus entstehende Spektrum lautet:

$$S_{\mathrm{DAC,out}}(f) = T \cdot \mathrm{si}\left(\pi \frac{f}{f_a}\right) \cdot \left[S(f) * 1/T \sum_{n=-\infty}^{\infty} \delta(f - nf_a) \right] \tag{5.30}$$

Die Multiplikation mit $\mathrm{si}\left(\pi \frac{f}{f_a}\right)$ lässt das Spektrum zur Abtastfrequenz 0 werden und lässt das Spektrum in der Unendlichkeit abklingen. Die Faltung mit dem Spektrum Dirac-Impuls lässt das Spektrum $S(f)$ unendlich oft wiederholen. Sogenannte Images entstehen.

$$S_{\mathrm{DAC,out}}(f) = T \cdot \mathrm{si}\left(\pi \frac{f}{f_a}\right) \cdot 1/T \sum_{n=-\infty}^{\infty} S(f - nf_a) \tag{5.31}$$

Bild 5.8 zeigt das Spektrum zu Formel 5.31 unter Vernachlässigung der Multiplikation mit $\mathrm{si}\left(\pi \frac{f}{f_a}\right)$.

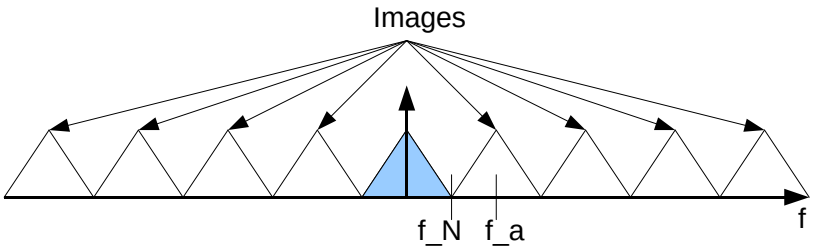


Bild 5.8.: Images eines DAC-Ausgangsspektrums

Um das im Bild schraffierte Signal $S(f)$ zu rekonstruieren, müssen alle Images unterdrückt werden. Ein Filter, welches diese Aufgabe erfüllt, müsste ein ideales Rechteckfilter nach Bild 5.9 sein.

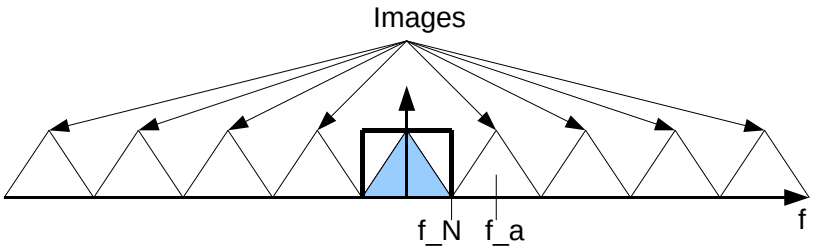


Bild 5.9.: Filteranforderung zur Unterdrückung der Images eines DAC-Ausgangsspektrum

Ein solches ideales Rechteckfilter existiert nicht. Um ein Filterdesign möglich zu machen, wird das digitale Signal vorher interpoliert und digital gefiltert. Das digitale Interpolationsfilter wurde bereits in Kapitel 5.1 erklärt. Das interpolierte und digital gefilterte DAC-Ausgangssignal ist in Bild 5.10 dargestellt.

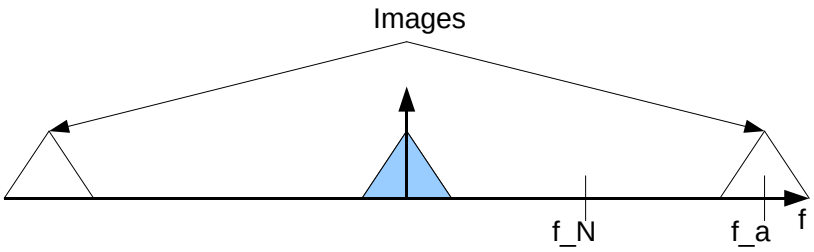


Bild 5.10.: Images eines DAC-Ausgangsspektrum mit 4-fach-Interpolation

Der Abstand zwischen Nutzsignal $S(f)$ und Images vergrößert sich. Die Anforderung an das Rekonstruktionsfilter sinken dadurch und sind im Bild 5.11 dargestellt.

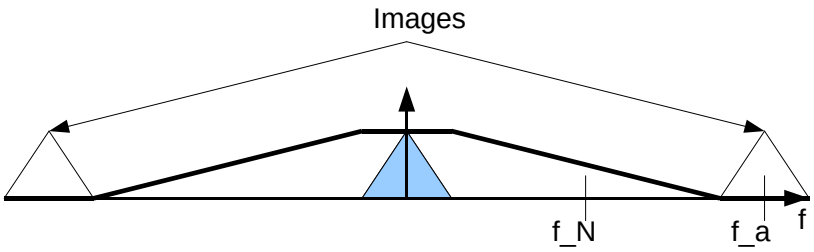


Bild 5.11.: Filteranforderung zur Unterdrückung der Images eines DAC-Ausgangsspektrum mit 4-fach-Interpolation

Der vergrößerte Übergangsbereich vom Durchgangsbereich zu Sperrbereich erleichtert das Filterdesign. Auf die Realisierung des Rekonstruktionsfilter wird weiter unten im Kapitel 5.3.3 eingegangen. Zunächst soll noch auf den Einfluss der Multiplikation mit $\mathrm{si}\left(\pi\frac{f}{f_a}\right)$ aus Formel 5.31 eingegangen werden. Um diesen si-Effekt zu visualisieren, wurde das MATLAB-File E.1 geschrieben. Es wird ein Signalspektrum $S(f)$ betrachtet, welches von $-\frac{f_a}{2}$ bis $+\frac{f_a}{2}$ die Amplitude 1 und sonst 0 hat. Die Images aus Formel 5.31 sollen dabei vernachlässigt werden. Bild 5.12 zeigt das Signalspektrum $S(f)$ und die Filterfunktion $\mathrm{si}\left(\pi\frac{f}{f_a}\right)$.

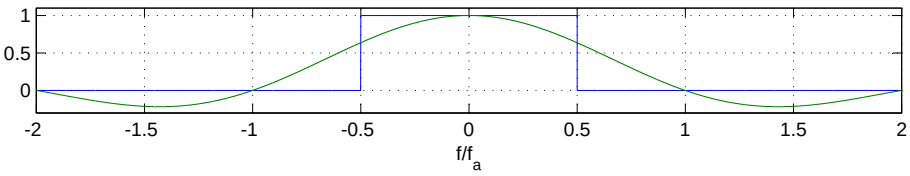


Bild 5.12.: Signalspektrum $S(f)$ und Filterfunktion $\mathrm{si}\left(\pi\frac{f}{f_a}\right)$

Durch Multiplikation des Signalspektrums $S(f)$ mit der Filterfunktion $\mathrm{si}\left(\pi\frac{f}{f_a}\right)$ verzerrt sich das Signalspektrum, wie Bild 5.13 zeigt.

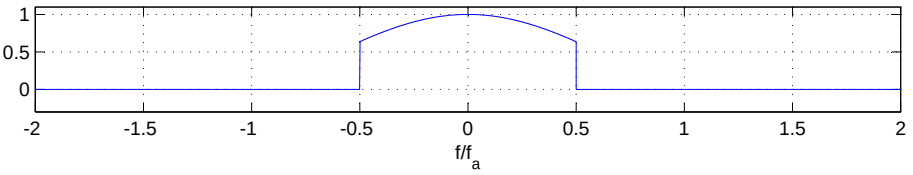


Bild 5.13.: Multiplikation vom Signalspektrum $S(f)$ mit Filterfunktion $\mathrm{si}\left(\pi \frac{f}{f_a}\right)$

Wie bereits erwähnt, wird das DAC-Ausgangssignal digital interpoliert und digital gefiltert. Dadurch verändert sich das spektrale Verhältnis vom Signalspektrum $S(f)$ zur Filterfunktion $\mathrm{si}\left(\pi \frac{f}{f_a}\right)$. Bild 5.14 verdeutlicht das bessere Verhältnis.

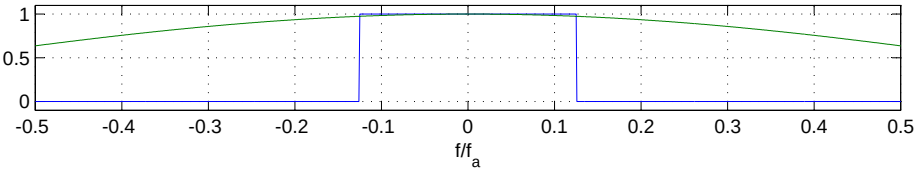


Bild 5.14.: Signalspektrum $S(f)$ und Filterfunktion $\mathrm{si}\left(\pi \frac{f}{f_a}\right)$ bei 4-fach-Interpolation

Bei 4-fach-Interpolation ergibt die Multiplikation des Signalsspektrums $S(f)$ mit der Filterfunktion $\mathrm{si}\left(\pi \frac{f}{f_a}\right)$ eine geringere Verzerrung, wie Bild 5.15 zeigt.

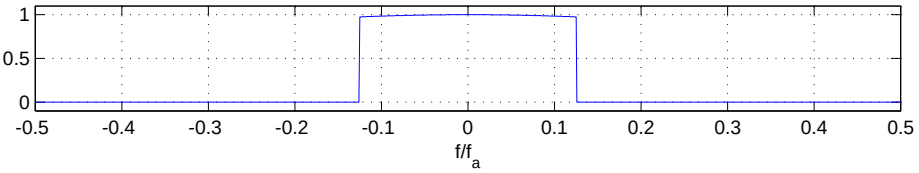


Bild 5.15.: Multiplikation vom Signalspektrum $S(f)$ mit Filterfunktion $\mathrm{si}\left(\pi \frac{f}{f_a}\right)$ bei 4-fach-Interpolation

Die Interpolation verringert damit eine Verzerrung des DAC-Ausgangssignals durch die Multiplikation mit der $\mathrm{si}\left(\pi \frac{f}{f_a}\right)$-Filterfunktion.

5.3.2. Theorie des Anti-Aliasing

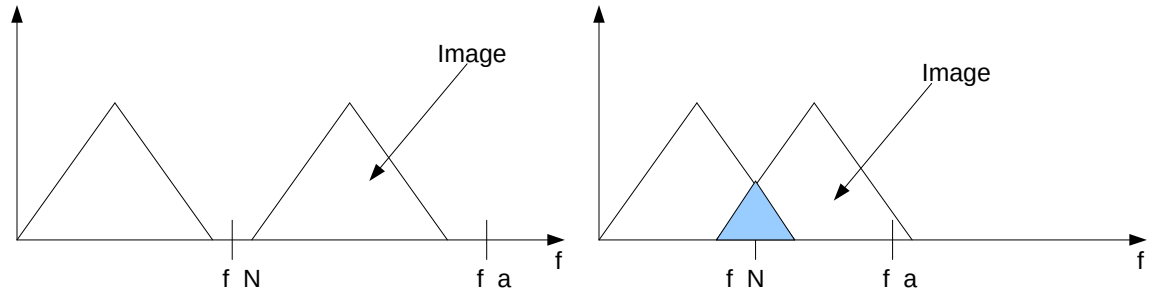
Formel 5.31 aus Kapitel 5.3.1 lässt auf das Spektrum $S_{\mathrm{ADC,in}}(f)$ eines abgetasteten analogen Signals mit dem Spektrum $S(f)$ schließen.

$$S_{\mathrm{ADC,in}}(f) = t_0 \cdot \mathrm{si}\,(\pi f t_0) \cdot 1/T \sum_{n=-\infty}^{\infty} S(f - nf_a) \tag{5.32}$$

Die Summe $\sum_{n=-\infty}^{\infty} S(f - nf_a)$ führt zu spektralen Wiederholungen des Eingangssignals $S(f)$. Entscheidend ist die Bandbreite des Eingangssignals $S(f)$. Ist die Bandbreite breiter als die halbe Abtastfrequenz, so überlappen sich die spektralen Wiederholungen. Die halbe Abtastfrequenz wird auch Nyquist-Frequenz genannt. Durch Ausschreiben der Summe für die Fälle $n = -1$ und $n = 0$ kann dies verdeutlicht werden.

$$S_{\mathrm{ADC,in}}(f) = t_0 \cdot \mathrm{si}\,(\pi f t_0) \cdot 1/T \cdot [\cdots + S(f + f_a) + S(f) + \ldots] \tag{5.33}$$

Bild 5.16(a) zeigt den Fall, dass das Signalspektrum $S(f)$ eine geringere Bandbreite, wie die Grenzen der Nyquist-Frequenz dies zulassen, hat. Bild 5.16(b) zeigt den Fall, dass das Signalspektrum $S(f)$ eine größere Bandbreite, als die Grenzen der Nyquist-Frequenz dies zulassen, hat. Das Spektrum in Bild 5.16(b) überlappt sich mit dem eigenen Image. Der



(a) Spektrum mit Frequenzanteilen, die bis zur Nyquist-Frequenz reichen

(b) Spektrum mit Frequenzanteilen, die die über die Nyquist-Frequenz reichen

Bild 5.16.: Visualisierung des Aliasing-Effekts

überlappte Bereich ist nicht rekonstruierbar. Dieses Phänomen wird Aliasing genannt. Ein Anti-Aliasing-Filter muss somit die Bandbreite eines zu digitalisierenden Signal auf die Grenzen eines Nyquist-Bandes begrenzen. Das erste Nyquist-Band geht von $-f_N = -\frac{f_a}{2}$ bis $+f_N = +\frac{f_a}{2}$.

5.3.3. Realisierung des Rekonstruktionsfilters

Zur Ermittlung der Filterspezifikation schauen wir uns das Ausgangssignal des DAC bei OOK und 4-fach Interpolation an. Es wird auf eine 8-fach-Interpolation verzichtet, da bei 4-fach-Interpolation DAC und ADC von einem Clockboard versorgt werden können. Bei 8-fach-Interpolation müsste das Clockboard einen Takt bei 240 MHz und einen Takt bei 640 MHz erzeugen. Dies kann das Clockboard nicht leisten [31]. Bild 5.17 zeigt das Spektrum des DAC-Ausgangs bei OOK und 4-fach-Interpolation.

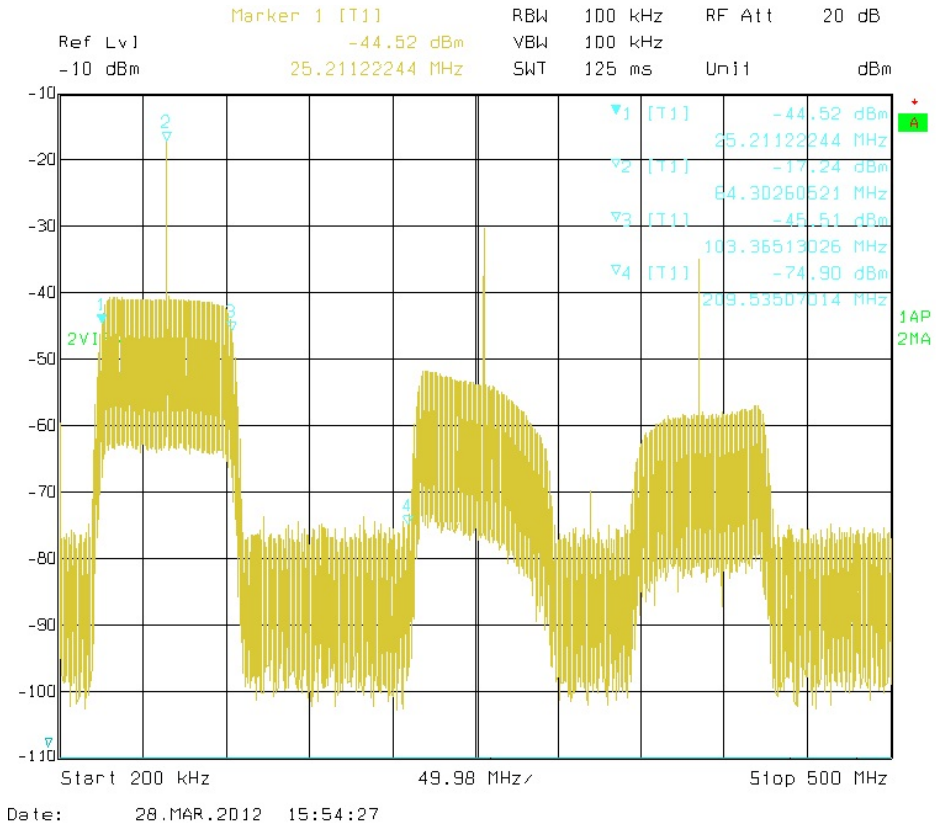
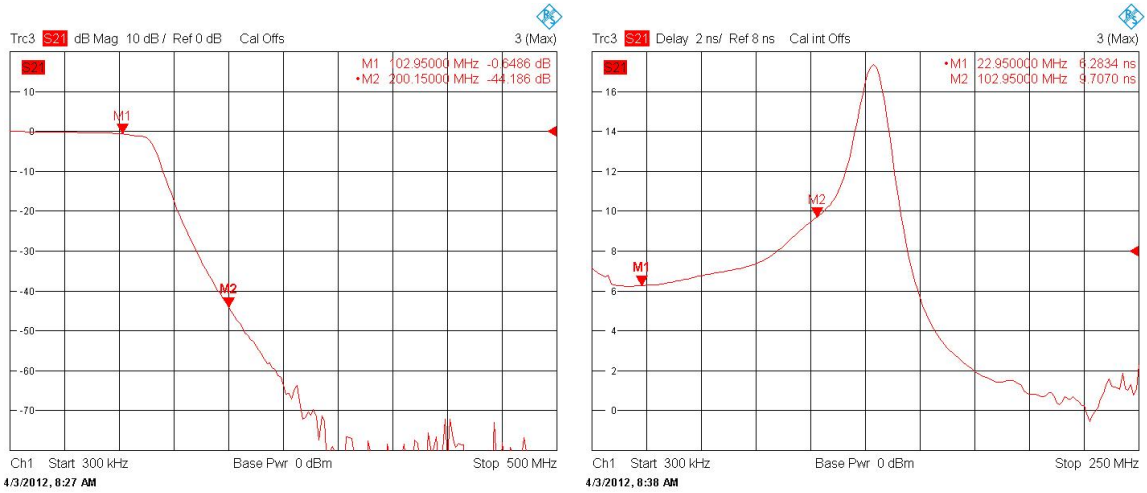


Bild 5.17.: Spektrum des DAC-Ausgangs bei OOK und 4-fach-Interpolation

Das Nutzsignal reicht von 25,2 MHz bis 103,4 MHz. Das erste Image fängt bei 209,5 MHz an. Das Rekonstruktionsfilter darf somit einen Übergangsbereich von 103,4 MHz bis 209,5 MHz aufweisen. Weitere Randbedingungen sind eine lineare Phase und einen möglichst geringen Ripple im Durchgangsbereich. Ein Bessel-Filter würde einen linearen Phasengang vorweisen, aber im Durchgangsbereich bis zur Grenzfrequenz kontinuierlich abfallen [23, Kapitel 7]. Außerdem wäre eine sehr hohe Filterordnung nötig, um eine akzeptable Dämpfung im Sperrbereich zu erreichen. Ein Chebychev-Filter erreicht sehr schnell

eine starke Dämpfung, jedoch ist die Phase um die Cut-off-Frequenz nicht linear [23, Kapitel 7]. Daher wird die Cut-off-Frequenz des Filters weiter in den Übergangsbereich nach $135\,\mathrm{MHz}$ verschoben. Der Durchgangsbereichsripple wird auf $0.1\,\mathrm{dB}$ festgelegt. Die Filterordnung wird auf 7 festgelegt. Bild 5.18 zeigt die Netzwerkanalyse des 1. Rekonstruktionsfilters. Der Amplitudengang in Bild 5.18(a) zeigt eine geringe Dämpfung an der Signal-



(a) Amplitudengang des 1. Rekonstruktionsfilters (b) Gruppenlaufzeiten des 1. Rekonstruktionsfilters

Bild 5.18.: Netzwerkanalyse des 1. Rekonstruktionsfilters

grenzfrequenz von $0{,}65\,\mathrm{dB}$ und eine ausreichende Dämpfung im Sperrbereich von $44{,}2\,\mathrm{dB}$. Die Gruppenlaufzeiten in Bild 5.18(b) zeigen einen deutlichen Zeitunterschied vom Anfang des Signalspektrums mit $6{,}28\,\mathrm{ns}$ bis zum Ende des Signalspektrums mit $9{,}71\,\mathrm{ns}$ auf. Die Differenz entspricht $9{,}71\,\mathrm{ns} - 6{,}28\,\mathrm{ns} = 3{,}43\,\mathrm{ns}$. Dies könnte zu Problemen wie ISI führen. Da das Rekonstruktionsfilter die Gesammtübertragungsfunktion von Impulsformungsfilter und Rekostruktionsfilter beeinflusst. Dennoch werden die produzierten Filter für einen ersten Aufbau herangezogen. Die Amplitudengänge und Gruppenlaufzeiten aller 4 Rekonstruktionsfilter sind im Anhang E.1 und E.2 abgebildet.

Das Spektrum des DAC-Ausgangs bei OOK und 4-fach-Interpolation hinter dem Rekonstruktionsfilter ist in Bild 5.19 abgebildet.

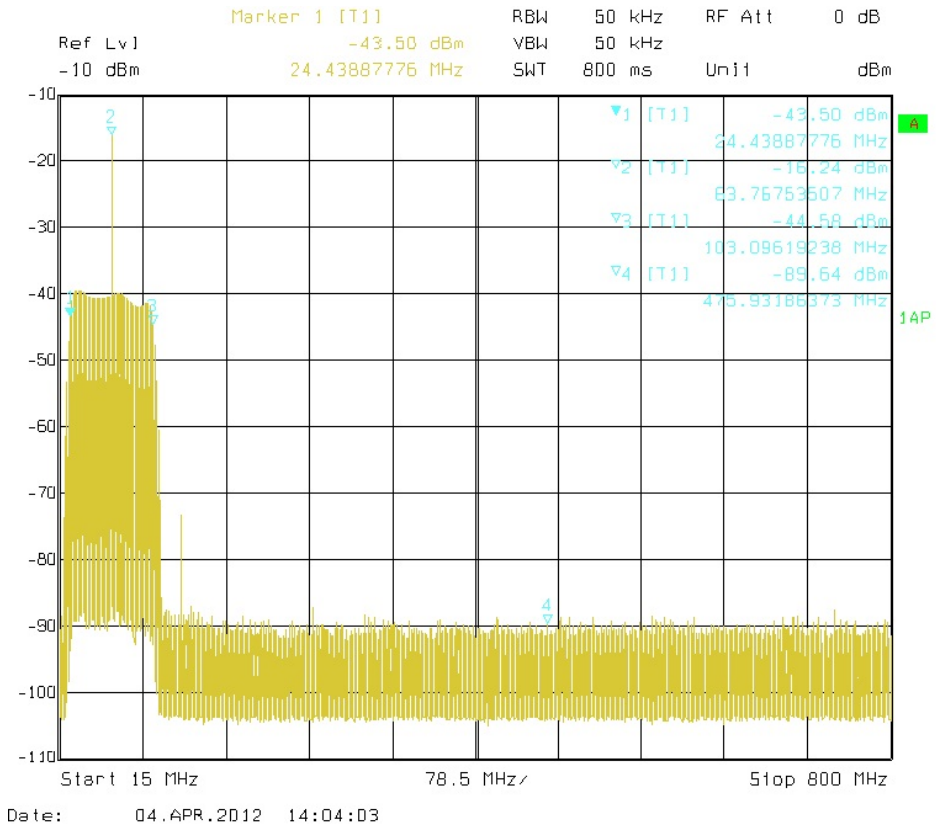


Bild 5.19.: Spektrum des DAC-Ausgangs bei OOK und 4-fach-Interpolation hinter dem Rekonstruktionsfilter

Das Image aus Bild 5.17 ist durch den Rekonstruktionsfilter komplett unterdrückt, wie dies Bild 5.19 zeigt.

Das Rekonstruktionsfilter ersetzt zusätzlich das Anti-Aliasing-Filter. Da auf der Übertragungsstrecke im demonstrativen Aufbau keine weiteren Signalanteile durch signifikantes Kanalrauschen oder Störsignale hinzukommen, kann auf das Anti-Aliasing-Filter verzichtet werden. Das Empfängersignal enthält keine Frequenzanteile die größer als die Nyquist-Frequenz sind.

5.4. Takt- und Datenrückgewinnung

Die Takt- und Datenrückgewinnung (engl.: Clock and Data Recovery (CDR)) gewinnt aus dem empfangenen Basisbandsignal den Symboltakt und mit Hilfe des zurückgewonnenen Symboltakts die Daten zurück. Dabei gibt es Konzepte, in denen diese beiden Schritte in

einem vollzogen werden. Es kann aber auch separat der Symboltakt zurückgewonnen werden und anschließend das Basisbandsignal synchron zum Symboltakt übernommen und per Schwellwertentscheidung einen Datenwert zugeordnet werden. Eine Taktrückgewinnung und eine Datenrückgewinnung kann dann in je einem eigenen Modul realisiert werden. Der Vorteil ist ein Taktrückgewinnungsmodul, das für verschiedene Modulationsarten verwendet werden kann.

Es gibt verschiedene Möglichkeiten der Taktrückgewinnung. Hier sollen die bekanntesten erwähnt werden. 1986 wurde von F. M. Gardner ein Algorithmus zur Taktregelung angegeben [12]. Heute bekannt unter dem Namen *Gardner Taktregelung*. Die Funktion wird ausgiebig in [21, Seite 349 ff] dargestellt. Die Gardner-Taktregelung bedingt zur Funktion einen bereits eingestellten adaptiven Entzerrer. Im Systemkonzept des aktuellen Demonstrators ist kein Entzerrer vorgesehen. Damit fällt die Gardner-Taktregung zur Taktrückgewinnung weg. Das gleiche Problem liegt bei der *Entscheidungsrückgekoppelten Taktregelung* vor[21, Seite 352 ff]. Die Taktrückgewinnung durch Gleichrichtung des Datensignals ist jedoch eine universelle und robuste Art der Taktrückgewinnung, die auch ohne eingestellten adaptiven Entzerrer funktioniert. Außerdem ist diese Art der Taktrückgewinnung für alle gängigen Modulationsarten anwendbar [21, Seite 348].

Durch Formung der Impulse mit Hilfe des Impulsformungsfilters werden die Spektrallinien des Symboltakt schon im Sender unterdrückt [21, Seite 344]. Der Symboltakt ist also nicht im empfangenen Datensignal enthalten. Durch nichtlineare Verzerrung kann das Signal so verfälscht werden, dass Spektrallinien beim Symboltakt oder bei vielfachen des Symboltaktes entstehen. Durch Quadratur des Datensignals entsteht eine Spektrallinie bei der doppelten Symbolfrequenz. Die Spektrallinie kann jedoch nicht mit der aktuell gewählten Abtastrate von $240\,\mathrm{MHz}$ dargestellt werden. Die doppelte Symbolfrequenz von $80\,\mathrm{MHz}$ entspricht $160\,\mathrm{MHz}$ und ist damit größer als die Nyquist-Frequenz von $120\,\mathrm{MHz}$. Somit ist diese Spektrallinie nicht im aktuellen System darstellbar. Daher kann das Quadrieren nicht als nichtlineare Verzerrung verwendet werden. Alternativ kann das Signal durch Differenzieren und anschließendem Gleichrichten so nichtlinear Verzerrt werden, dass eine Spektrallinie bei der Symbolfrequenz entsteht. Bild 5.20 verdeutlicht diesen Prozess.

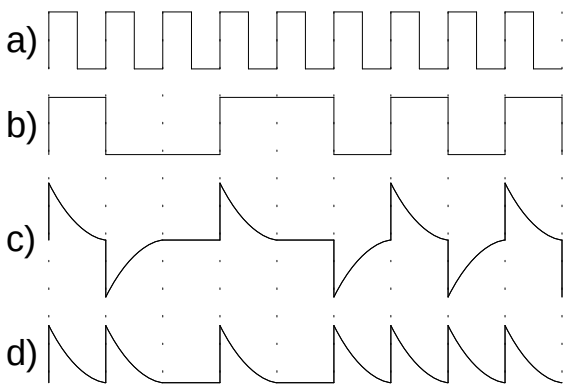


Bild 5.20.: Differenzieren und anschließendes Gleichrichten eines Datensignals

Die Kurve *b* zeigt ein Datensignal, welches synchron mit dem Symboltakt *a* erzeugt wurde. Kurve *c* zeigt das differenzierte Datensignal. Kurve *d* zeigt das differenzierte und anschließend gleichgerichtete Datensignal. Kurve *d* zeigt eine Periodizität auf. Diese Periodizität spiegelt sich im Signalspektrum als Spektrallinie bei der Symboltaktfrequenz wieder. Die Spektrallinie kann durch einen BP separiert werden und mittels anschließender Schwellwertentscheiung zu einem Taktsignal geformt werden. Blockschaltbild 5.21 zeigt die Signalverarbeitungsschritte auf.

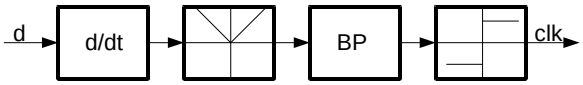


Bild 5.21.: Blockschaltbild einer Taktrückgewinnung mittels Differenzieren und anschließendem Gleichrichten

Das Datensignal d wird durch den Differenzierer $\frac{d}{dt}$ differenziert und anschließend durch einen Gleichrichter gleichgerichtet. Ein BP filtert die Symboltaktfrequenz aus dem gleichgerichteten Signal. Ein Schwellwertentscheider erzeugt aus dem gefilterten Signal ein Symboltakt clk.

Noch besser ist es, statt des BP eine Phasenregelschleife (engl.: PLL) zu verwenden. Da bei einer langen Folge von Nullen oder von Einsen das Signal hinter dem BP abklingt. Eine PLL läuft weiter, und wird bis zum nächsten Signalwechsel nicht synchronisiert. PLLs sollen folgend etwas genauer betrachtet werden.

5.4.1. Phasenregelschleifen

Eine Phasenregelschleife regelt, in einem geschlossenen Regelkreis, die Phasenlage eines regelbaren Oszillators möglichst konstant zu einem Referenzsignal. Das Signal des regelbaren Oszillators ist am Ausgang der PLL phasengleich zum Referenzsignal. Eine Möglichkeit ist die Multiplikation des Referenzsignals mit dem Oszillatorsignal. Dadurch entsteht eine Schwingung der doppelten Frequenz mit einem Gleichanteil der dem Phasenfehler zwischen den beiden Signalen entspricht. Sei $\alpha = 2\pi f_0 t + \theta_{\text{ref}}$ mit f_0 der eingeregelten Frequenz, θ_{ref} der Phasenverschiebung des Referenzsignals und $\beta = 2\pi f_0 t + \theta_{\text{O}}$ mit θ_{O} der Phasenverschiebung des Oszillators, so errechnet sich aus der Multiplikation der beiden Schwingungen $\sin\alpha$ und $\cos\beta$

$$A \sin\alpha \cos\beta = A\frac{1}{2}\left\{\sin\left(\alpha - \beta\right) + \sin\left(\alpha + \beta\right)\right\}. \tag{5.34}$$

A ist dabei die Amplitude des Referenzsignals. Durch Substitution erhalten wir

$$\begin{aligned} A \sin\alpha \cos\beta = A\frac{1}{2}[&\sin\left(2\pi f_0 t + \theta_{\text{ref}} - 2\pi f_0 t - \theta_{\text{O}}\right) + \\ &\sin\left(2\pi f_0 t + \theta_{\text{ref}} + 2\pi f_0 t + \theta_{\text{O}}\right)]. \end{aligned} \tag{5.35}$$

Durch Kürzen und Zusammenfassen erhalten wir

$$A \sin\alpha \cos\beta = A\frac{1}{2}[\sin\left(\theta_{\text{ref}} - \theta_{\text{O}}\right) + \sin\left(4\pi f_0 t + \theta_{\text{ref}} + \theta_{\text{O}}\right)]. \tag{5.36}$$

Durch Tiefpassfilterung kann der Term $A\frac{1}{2}\sin\left(\theta_{\text{ref}} - \theta_{\text{O}}\right)$ separiert werden. Ein solcher Filter wird Schleifenfilter genannt. Da in $A\frac{1}{2}\sin\left(\theta_{\text{ref}} - \theta_{\text{O}}\right)$ der Phasenfehler $\theta_{\text{e}} = \theta_{\text{ref}} - \theta_{\text{O}}$ enthalten ist, kann mit diesem Term der regelbare Oszillator gesteuert werden. Bild 5.22 zeigt ein System Generator-Modell.

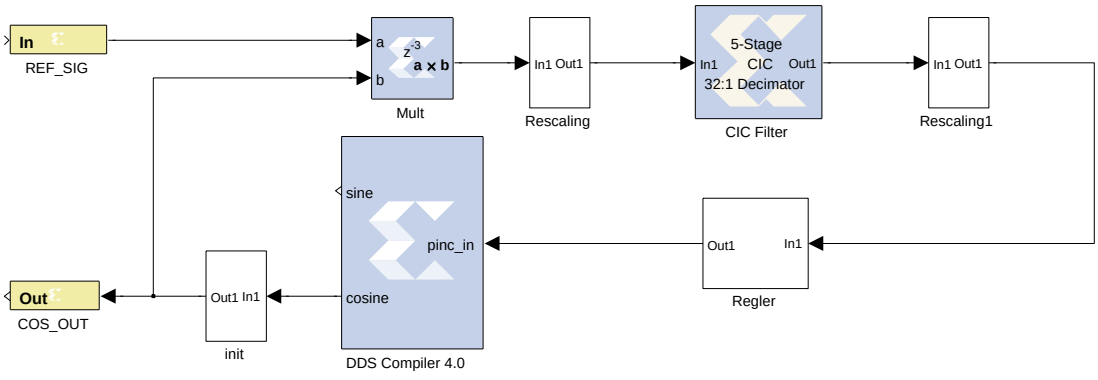


Bild 5.22.: System Generator-Modell einer einfachen PLL

Das Referenzsignal *REF_SIG* wird mit dem Oszillatorsignal multipliziert. Durch einen *CIC Filter* wird der oben beschriebene Term mit dem Phasenfehler separiert. Über einen Regler wird der Oszillator (*DDS Compiler 4.0*) mit dem gefilterten Signal gesteuert. Die Schwingung des Oszillators wird an den Ausgang *COS_OUT* ausgegeben. Die beiden *Rescaling*-Blöcke passen das Signal nach einer Operation an den Datenraum an. Der *init*-Block sorgt für eine korrekte Initialisierung. Der Regler wird weiter unten im Kapitel 5.4.2 beschrieben.

Diese Art von PLL hat zwei grundlegende Nachteile. Zum einen ist der Term $A\frac{1}{2}\sin(\theta_{\text{ref}} - \theta_{\text{O}})$, mit dem der Regler gespeist wird, abhängig von der Amplitude A des Referenzensignals. Zum anderen erfolgt aus $\sin(\theta_{\text{ref}} - \theta_{\text{O}})$ keine lineare Regelkurve. Bild 5.23 zeigt die Regelkurve von $-180°$ bis $+180°$ Phasenfehler.

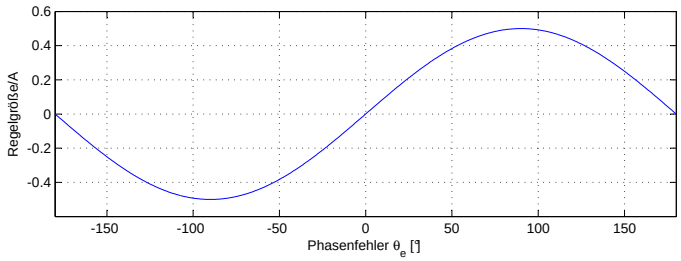


Bild 5.23.: PLL-Regelkurve

Von einen Phasenfehler von $0°$ bis $90°$ bzw. von $0°$ bis $-90°$ steigt der Betrag der Regelgröße an. Für Phasenfehler größer $|\pm 90°|$ nimmt der Betrag der Regelgröße wieder ab. Dies ist unerwünscht, denn auf einem großen Phasenfehler soll der Regler mit einer großen Stellgröße reagieren.

Durch Betrachtung der Costas-Loop [10, Seite 294] soll auf ein Verfahren geschlossen werden, welches eine lineare Regelkurve hat. Bild 5.24 zeigt ein System Generator-Modell der Costas-Loop.

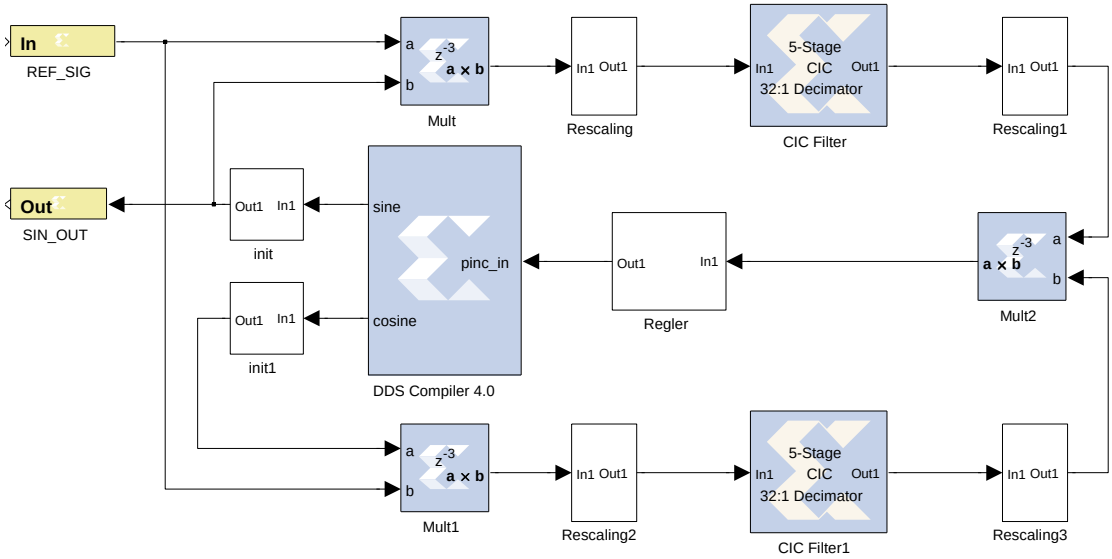


Bild 5.24.: System Generator-Modell einer Costas-Loop

Bei der Costas-Loop wird das Referenzsignal einmal mit der Cosinusschwingung des Oszillators und einmal mit der Sinusschwingung des Oszillators multipliziert. Anschließend werden beide Signale tiefpassgefiltert. Hinter dem unteren Filter erhält man den, wie bei der einfachen PLL gezeigten Term $A\frac{1}{2}\sin(\theta_{\text{ref}} - \theta_{\text{O}})$. Hinter dem oberen Filter erhält man Term $A\frac{1}{2}\cos(\theta_{\text{ref}} - \theta_{\text{O}})$. Hinter dem dritten Multiplikator errechnet sich

$$A\frac{1}{2}\sin(\theta_{\text{ref}} - \theta_{\text{O}}) \cdot A\frac{1}{2}\cos(\theta_{\text{ref}} - \theta_{\text{O}}) = A^2\frac{1}{8}\sin(2(\theta_{\text{ref}} - \theta_{\text{O}})) \tag{5.37}$$

Die Form der Regelkurve wird für die Runtermischung von BPSK-Signale verwendet. Bei der BPSK wird das Trägersignal mit 1 und -1 zu $0°$ und $180°$ multipliziert. Durch das Quadrieren der Amplitude ist die Regelkurve für $180°$ Sprünge unanfällig. Die PLL bleibt eingerastet. An dieser Stelle interessiert die Tatsache, dass durch die komplexe Multiplikation des Referenzsignals die Terme $A\frac{1}{2}\sin(\theta_{\text{ref}} - \theta_{\text{O}})$ und $A\frac{1}{2}\cos(\theta_{\text{ref}} - \theta_{\text{O}})$ auch als eine komplexwertige Zahl aufgefasst werden kann. Somit lässt sich mit

$$\theta_{\text{ref}} - \theta_{\text{O}} = \arg\left\{A\frac{1}{2}\cos(\theta_{\text{ref}} - \theta_{\text{O}}) + jA\frac{1}{2}\sin(\theta_{\text{ref}} - \theta_{\text{O}})\right\} \tag{5.38}$$

der Phasenfehler $\theta_{\text{ref}} - \theta_{\text{O}}$ berechnen. Der Coordinate Rotation Digital Computer (CORDIC)-Algorithmus bietet die Möglichkeit aus einer komplexen Zahl die Phase zu berechnen. Bild 5.25 zeigt ein System Generator-Modell mit komplexer Multiplikation und integriertem CORDIC-Algorithmus.

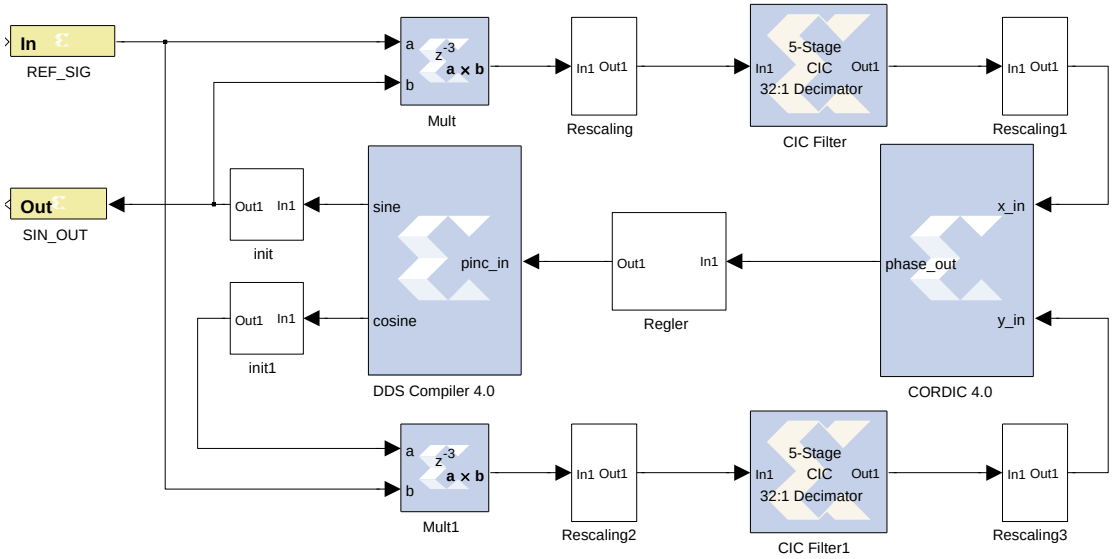


Bild 5.25.: System Generator-Modell einer PLL mit komplexer Mischung und CORDIC-Algorithmus

Die Regelkurve ist nun linear und nicht mehr von der Amplitude des Referenzsignals abhängig. Der Phasenfehler wird direkt an den Regler übergeben.

5.4.2. Regelung der PLL

Ein Regler stellt automatisch je nach Regelabweichung eine sogenannte Stellgröße zur Steuerung einer Regelstrecke ein. Durch einen Regler kann das Verhalten eines Systems beeinflusst werden. Ein System kann durch einen Regler schneller regeln, den Regelfehler verringern und stabil werden. Je nach Wahl des Reglers und Einstellung der Regelparameter können die genannten Eigenschaften erreicht werden.

Bild 5.26 zeigt einen Standard-Regelkreis.

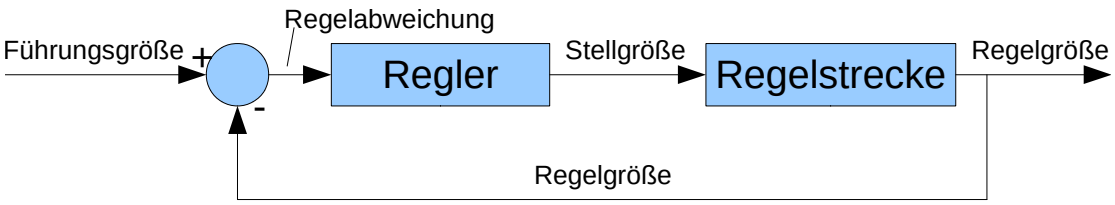


Bild 5.26.: Standard-Regelkreis

Der *Regler* errechnet aus einer Regelabweichung eine Stellgröße, mit der die *Regelstrecke* angesteuert wird. Die *Regelstrecke* gibt eine Regelgröße aus. Die Differenz aus Regelgröße und Führungsgröße ergibt wiederum die Regelabweichung.

Zur Regelung der PLL wurde ein PI-Regler eingesetzt. Ein PI-Regler lässt durch den I-Anteil keinen Regelfehler zu und durch den P-Anteil wird die Regelgeschwindigkeit beeinflusst. Bild 5.27 zeigt den mit System Generator realisierten PI-Regler.

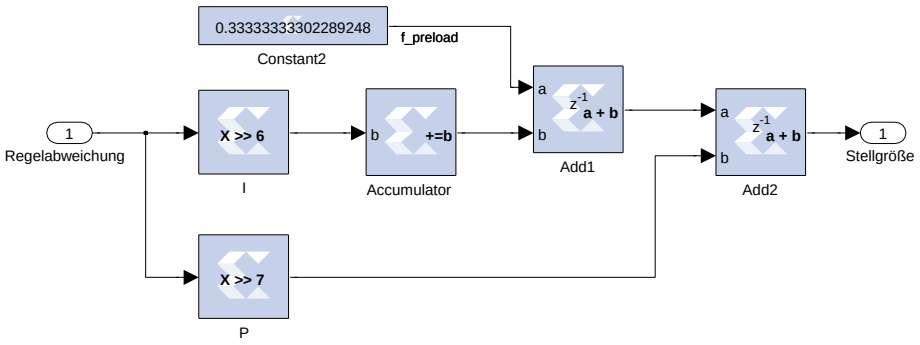
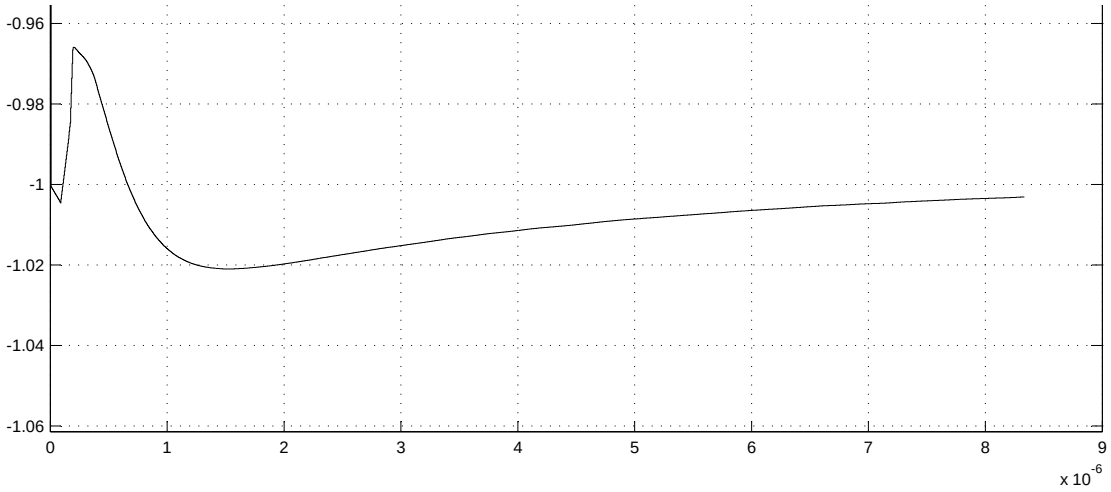


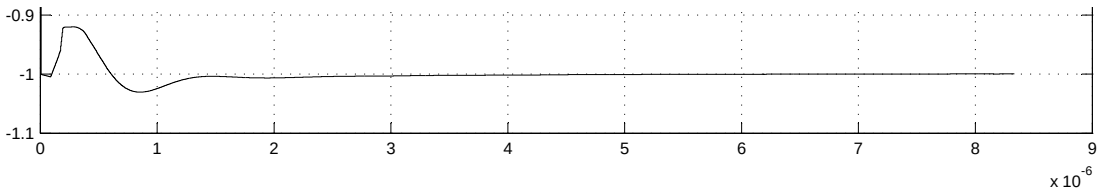
Bild 5.27.: System Generator-Modell eines PI-Reglers

Der P- und I-Anteil werden in dem System Generator-Modell durch je eine Bitverschiebung realisiert. In einem FPGA kostet eine Bitverschiebung keine zusätzliche Hardware, da eine Bitverschiebung durch Verdrahtung realisiert werden kann. Diese Maßnahme erspart zwei zusätzliche Multiplikationen. Wie im System Generator-Modell gezeigt, wird ein P-Regler durch eine proportionale Skalierung der Regelabweichung erreicht. Ein I-Regler wird durch Integrieren der Regelabweichung erreicht. Anschließend wird der P- und I-Ausgang zur Stellgröße aufaddiert. Es hat sich als sehr hilfreich herausgestellt, den Regler mit einem Wert vorzuinitialisieren. Daher wird die berechnete Stellgröße mit einem Initialwert verrechnet.

Die Regelparameter I und P können mathematisch berechnet werden. MATLAB/Simulink ermöglicht jedoch auch die optimale Regelparameterkombination simulativ zu ermitteln. Das im Anhang angehängte MATLAB-File F.1 startet das System Generator-Modell aus Bild 5.25 für verschiedene Regelparameterkombinationen. Da zum Start jeder Simulation die Phase des Referenzsignals nicht phasengleich zu dem Oszillatorsignal ist, kann der Verlauf der Regelabweichung als Sprungantwort aufgefasst werden. Alle Sprungantworten werden in einem Diagramm aufgenommen und können miteinander verglichen werden. Die beiden, in Bild 5.28 dargestellten Kurven, stellen zwei Sprungantworten des geregelten Systems mit unterschiedlichen Parameterkombinationen dar.



(a) Sprungantwort bei $P = 2^{-8}$ und $I = 2^{-16}$



(b) Sprungantwort bei $P = 2^{-7}$ und $I = 2^{-14}$

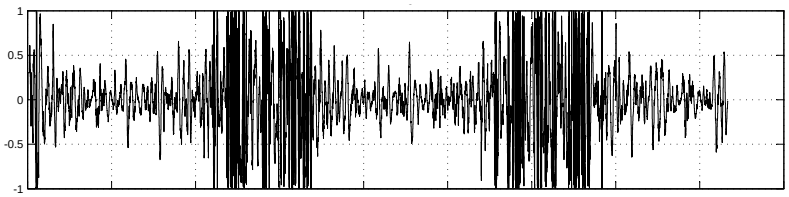
Bild 5.28.: Sprungantworten des geregelten Systems

Kurve 5.28(a) zeigt eine sehr starke Dämpfung auf. Die Regelgeschwindigkeit ist sehr gering. Dafür ist das System sehr stabil. Kurve 5.28(a) zeigt eine geringere Dämpfung auf. Die Regelgeschwindigkeit ist hier sehr viel höher. Die Dämpfung ist immer noch so stark, dass das System sehr stabil ist. Beide Regelparameterkombinationen sind für eine Regelung der PLL geeignet. Im folgenden Kapitel wird die gesamte PLL mit den beiden ermittelten Regelparametern getestet.

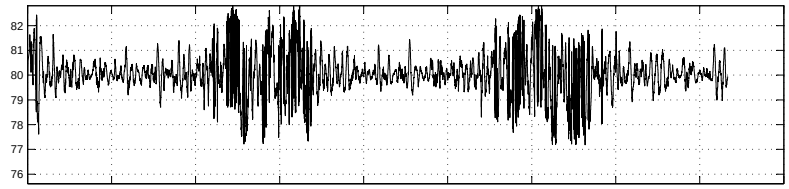
5.4.3. Test der PLL

Zum Test wurde ein simuliertes Testsignal erzeugt, mit dem die PLL gespeist wurde. Beide ermittelten Regelparameterkombinationen wurden getestet. Regelabweichung, Stellgröße sowie die normierte Taktabweichung wurden aufgenommen. Die normierte Taktabweichung zeigt die Abweichung von Referenztakt zum geregelten PLL-Ausgangstakt auf. Eine

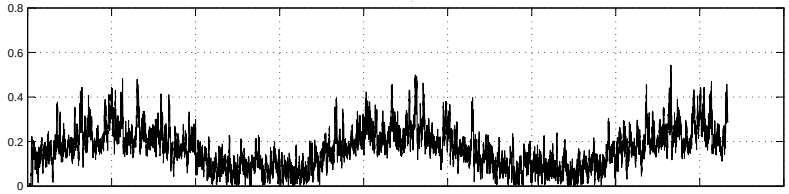
normierte Taktabweichung von 1 entspricht dabei einer vollen Takt-Abweichung vom Referenztakt zum PLL-Ausgangstakt. Da der Test nicht das gewünschte Ergebnis aufzeigte, wurde zusätzlich die Amplitude des Referenzsignals aufgenommen. Bild 5.29 zeigt die vier Kurven auf bei den Regelparametern $P = 2^{-7}$ und $I = 2^{-14}$.



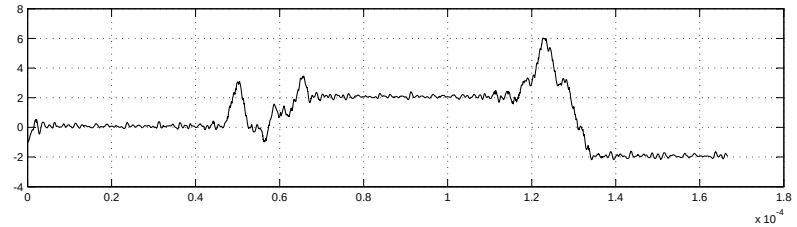
(a) Regelabweichung $[\theta_{\mathrm{ref}} - \theta_{\mathrm{O}}]$



(b) Stellgröße [MHz]



(c) Amplitude des Referenzsignals



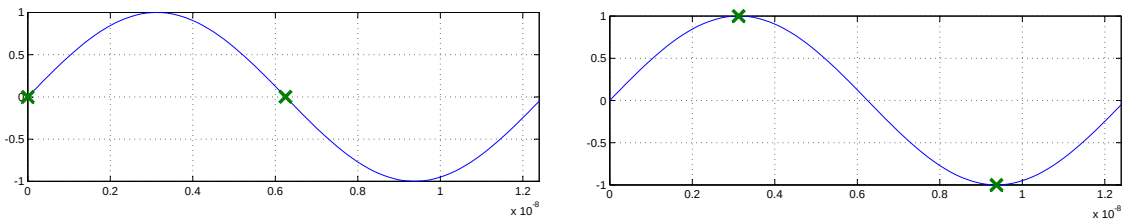
(d) Normierte Taktabweichung

Bild 5.29.: Test der PLL mit den Regelparametern $P = 2^{-7}$ und $I = 2^{-14}$

Die Kurve 5.29(a) der Regelabweichung zeigt große Schwankungen auf. Daraus resultiert eine stark schwankende Stellgröße, abgebildet in Kurve 5.29(b). Die starke Schwankung der

Stellgröße resultiert in starker Taktabweichung. Dies zeigt die normierte Taktabweichung in Kurve 5.29(d) auf. Die Taktabweichung zeigt sogar Abweichung ganzer Takte auf. Die PLL kann den Oszillator nicht genau genug für eine Taktrückgewinnung regeln. Um der Ursache auf den Grund zu gehen, wurde außerdem die Amplitude des Referenzsignals in Kurve 5.29(c) aufgenommen. Es fällt auf, dass die größte Regelabweichung zum Zeitpunkt geringer Referenzamplitude stattfindet. Daraus lässt sich schließen, dass das System den Phasenfehler $[\theta_{\text{ref}} - \theta_{\text{O}}]$ und damit die Regelabweichung bei geringer Amplitude nicht richtig messen kann. Bei geringer Amplitude wird eine falsche Regelabweichung ermittelt und daraus eine falsche Stellgröße propagiert. Das gleiche Phänomen entsteht bei den Regelparametern $P = 2^{-8}$ und $I = 2^{-16}$. Der Vollständigkeit halber werden die vier Kurven bei dieser Parametrisierung in Bild F.1 im Anhang dargestellt.

Die Amplitude des Referenzsignals schwankt nur bei inkohärentem Empfänger, wenn der zu regenerierende Takt nicht exakt ein Vielfaches der Signalabtastfrequenz entspricht. Daraus lässt sich schließen, dass zu bestimmten Zeitpunkten das Signal an Punkten abgetastet wird, bei denen der zurück zu gewinnende Takt geringe Signalwerte aufweist. Tastet man ein Sinussignal beispielsweise mit einer gering höheren Frequenz als die doppelte Signalfrequenz ab, so erhält man zu bestimmten Zeitpunkten, Signalwerte um den Nullpunkt und zu anderen Zeitpunkten, Signalwerte um das Maximum des Sinussignals. Bild 5.30 verdeutlicht diesen Effekt anhand einer Darstellung eines Sinussignals im Zeitbereich. In Kurve 5.30(a)



(a) Abtastung eines Sinussignals in der Nähe der Nulldurchgänge

(b) Abtastung eines Sinussignals in der Nähe der Maximalwerte

Bild 5.30.: Abtastung eines Sinussignals bei doppelter Signalfrequenz

wird das Sinussignal zum Nulldurchgang des Sinussignals abgetastet. Die daraus resultierende Amplitude des Sinussignals entspricht 0. In Kurve 5.30(b) wird das Sinussignal zum Maximalwert des Sinussignals abgetastet. Die daraus resultierende Amplitude des Sinussignals entspricht dem tatsächlichen Amplitudenwert des Ursprungssignals. Im konkreten System liegt die Abtastfrequenz bei $240\,\text{MHz}$ und die Frequenz des Referenzsignals bei $80\,\text{MHz}$. Jedoch wurde das Signal in der Zwischenfrequenz abgetastet und anschließend ins Basisband runtergemischt. In der Zwischenfrequenz lag der höchste Signalanteil bei etwa $100\,\text{MHz}$. Damit liegt die Abtastfrequenz sehr nahe an der doppelten Signalfrequenz.

Diese Tatsache wird als Ursache der Amplitudenschwankung des Referenzsignals vermutet.

Bei einer weiteren Recherche zum Thema Taktrückgewinnung stellte sich heraus, dass zum Betrieb einer PLL eine 8-fache Überabtastung gängig ist. Mindestens jedoch eine 4-fache Überabtastung zu gewährleisten ist [15, Seite 610 - 613], [29] und [13, Seite 135 - 140]. Die entwickelte Taktrückgewinnung kann somit bei der geringen Überabtastung schwerlich betrieben werden.

Der Demonstrator soll die Leistungsfähigkeit der Millimeterwellen-Frontends darstellen und zur Demonstration und zum Test dieser entwickelt werden. Ist ein Signalverarbeitungsglied, wie im konkreten Fall die Taktrückgewinnung, jedoch hauptsächlich für die Leistungsfähigkeit verantwortlich, so ist dieses Glied aus dem Demonstrator zu entfernen. Das Systemdesign wird dahingehend geändert, dass der Sender-FPGA und Empfänger-FPGA kohärent zueinander betrieben werden. Der Symboltakt leitet sich somit aus einem natürlichen Teiler des Empfänger-FPGA-Systemtaktes ab. Im konkreten System wird der Sender-FPGA mit $80\,\mathrm{MHz}$ und der Empfänger-FPGA mit $240\,\mathrm{MHz}$ betrieben. Der natürliche Teiler entspricht $\frac{240\,\mathrm{MHz}}{80\,\mathrm{MHz}} = 3$. Es sei an dieser Stelle noch zu erwähnen, dass die Taktrückgewinnung bei kohärenten Sender-FPGA- zum Empfänger-FPGA-Systemtakt ausgezeichnet funktioniert. Jedoch lässt sich bei einem kohärenten Symboltakt eine Taktrückgewinnung durch eine einfache Taktteilung realisieren, was auch zur Aufnahme der Testergebnisse realisiert wurde.

Ein weiteres Problem der geringen Abtastfrequenz wird bei der Realisierung der Datenrückgewinnung deutlich, wie in folgenden Kapitel dargestellt wird.

5.4.4. Datenrückgewinnung

Die Basisbanddaten werden zum rückgewonnenen Symboltakt übernommen und per Schwellwertentscheidung einem Datenwert zugeordnet. Bild 5.31 zeigt das System Generator-Modell der Datenrückgewinnung. Im System Generator-Modell wird erst per

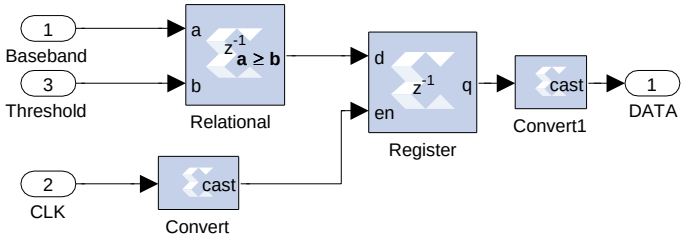


Bild 5.31.: System Generator-Modell der Datenrückgewinnung

Schwellwertentscheidung (*Relational*) den Basisbanddaten ein gültiger Datenwert zugeordnet und anschließend werden die Daten zum Symboltakt übernommen. Über den *Threshold*-Eingang lässt sich der Schwellwert extern einstellen. Das *CLK*-Signal ist das Symboltaktsignal und ist nicht mit dem FPGA-Takt zu verwechseln. Das Symboltaktsignal liegt als Freigabesignal vor und ist zum Übernehmen der Daten auf den Enable-Eingang des Registers gelegt. Die *cast*-Operatoren dienen der Datentypkonvertierung. Manche Blöcke benötigen bestimmte Datentypen wie Boolean oder unsigned-Fixed als Eingangssignale. Die Datentypkonvertierungen passen je nach Bedarf die Signale an die Blöcke an. Die Datentypkonvertierungen kosten keine zusätzliche Hardware, denn sie dienen nur zur Interpretation des Systems und werden nicht synthetisiert.

Zur näheren Erläuterung wird ein Basisbandsignal in Bild 5.32 als Augendiagramm dargestellt. Das Augendiagramm weist einen Zeitpunkt auf, an dem das Auge vertikal maximal

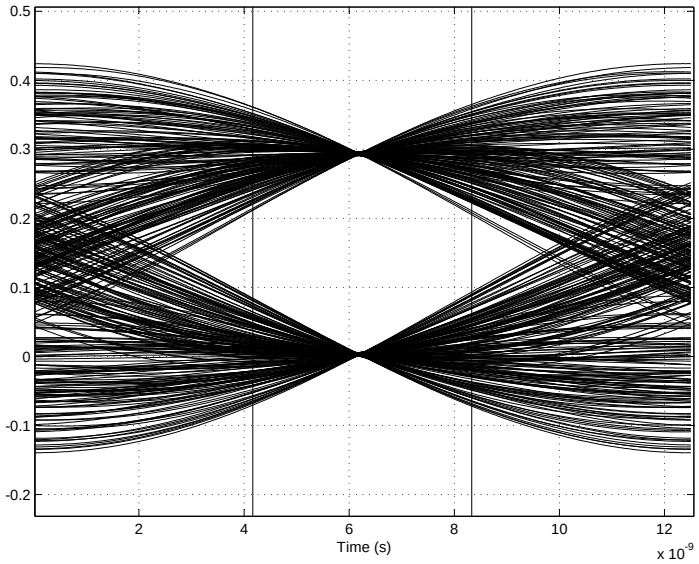


Bild 5.32.: Augendiagramm eines Basisbandsignals

geöffnet ist. Dies ist der optimale Zeitpunkt, an dem die Daten übernommen werden sollten. Zu diesem Zeitpunkt sollte das Freigabesignal *CLK* das Register zum Übernehmen der Daten freischalten. Da zum optimalen Zeitpunkt die Daten die Werte $0{,}3$ und 0 aufweisen, errechnet sich der optimale Schwellwert zu $\frac{0{,}3-0}{2} = 0{,}15$. Größer $0{,}15$ entspricht damit einer logischen 1 und kleiner $0{,}15$ einer logischen 0. Im aktuellen Demonstratoraufbau weist das Basisbandsignal drei Abtastwerte pro Symboltakt auf. Ist der Basisbandsignal inkohärent zum Symboltakt abgetastet worden, können die Abtastwerte zu beliebigen Zeitpunkten eines Symbolintervalles auftreten. Die in Bild 5.32 eingezeichneten zwei vertikalen Linien zeigen den schlechtesten Fall an. Liegt die Phasenlage des Abtasters zum Symboltakt gerade so, wie die vertikalen Linien dies anzeigen, verringert sich die tatsächliche Augenöffnung

des Augendiagramms auf die inneren Punkte an den vertikalen Linien zu $0{,}2 - 0{,}1 = 0{,}1$. Addiert sich zum Signal ein Rauschen, so ist das Auge schnell geschlossen und resultiert in einer hohen Fehlerrate. Ist jedoch der Abtaster zum Symboltakt kohärent und so eingestellt, dass einer der drei Abtastwerte genau zum optimalen Abtastzeitpunkt das Basisbandsignal abtastet, so ist das Auge des Augendiagramms immer maximal geöffnet. Dies ist ein weiterer Grund den Symboltakt kohärent zum Abtaster des Empfängers laufen zu lassen. Eine Alternative wäre eine Interpolation der optimalen Abtastwerte. Für den ersten Demonstratoraufbau ist dies aber nicht vorgesehen.

6. Implementierung der Bitfehlerratenmessung

Um die Leistungsfähigkeit einer Übertragungsstrecke oder einer Modulationsart zu messen, wird eine Bitfehlerratenmessung durchgeführt. Dazu werden die empfangenen Bits mit den erwarteten Bits verglichen. Die Fehlerrate BER wird im Verhältnis von fehlerhaften Bits e zu allen empfangenen Bits a angegeben, wie Formel 6.1 verdeutlicht.

$$\mathrm{BER} = \frac{e}{a} \tag{6.1}$$

Da dieses Verhältnis meist über mehrere Zehnerpotenzen variiert, wird das Verhältnis vorzugsweise als Exponent zu einer Basis von Zehn angegeben, wie Formel 6.2 zeigt.

$$\log_{10}(\mathrm{BER}) = \log_{10}\left(\frac{e}{a}\right) \tag{6.2}$$

6.1. Direkte Implementierung des Bitfehlerrate-Algorithmus

Zur Erfassung der empfangenen Bits a und der fehlerhaften Bits e werden zwei Zähler implementiert. Die Zählerbreite entspricht $64\,\mathrm{Bit}$. Mit $64\,\mathrm{Bit}$ lassen sich Fehlerraten von $2^{64} - 1 = 10^{-19.27}$ darstellen. Wird davon ausgegangen, dass eine Fehlerrate nach tausend Fehlern konvergiert, reduziert sich die maximal darstellbare Fehlerrate auf $10^{-16.27}/1000 = 10^{-16.27}$. Somit ist es gewährleistet auch sehr niedrige Fehlerraten messen zu können.

Der von System Generator zur Verfügung gestellte *Divider Generator* kann maximal $54\,\mathrm{Bit}$ Divisionen durchführen. Daher muss auf den CORDIC-Algorithmus zurückgegriffen werden. System Generator bietet sowohl einen CORDIC Divider, als auch einen CORDIC zur Berechnung von natürlichen Logarithmus. Bild 6.1 zeigt das System Generator Model einer Bitfehlerratenmessung nach Formel 6.2. Wobei der Logarithmus zur Basis Zehn durch den natürlichen Logarithmus ersetzt wurde. Hierzu wurde der Zusammenhang

$$\log_b r = \frac{\log_a r}{\log_a b} \tag{6.3}$$

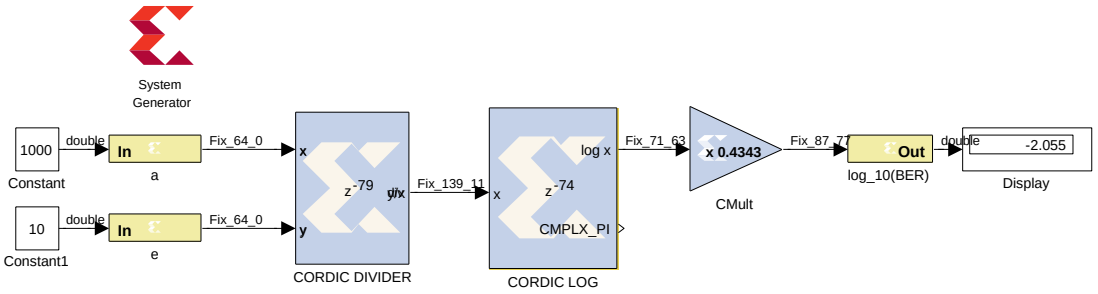


Bild 6.1.: Direkte Implementierung eines Bitfehlerrate-Algorithmus

verwendet. Im aktuellen Fall ergibt sich daraus

$$\log_{10} r = \frac{\ln r}{\ln 10}. \tag{6.4}$$

Der Faktor $\frac{1}{\ln 10} = 0.4343$ kann als konstante Multiplikation durchgeführt werden, wie dies in Bild 6.1 dargestellt ist.

Das Ergebnis der Berechnung ist ungenügend da die erstellte Hardware einen Wert von -2.055 berechnet. Richtig wäre ein Wert von $\log_{10}\left(\frac{10}{1000}\right) = -2$. Die Hardware ist somit trotz direkter Implementierung ungenau.

Nach Synthese wurden die benötigten Ressourcen ausgewertet. Tabelle 6.1 listet die benötigten Ressourcen auf. Auch die maximale Frequenz, mit der das Bitfehlerratenmessungs-

Benötigte Hardware	Anzahl	Anteil am Virtex-5 FX100T
FFs	16,481	25%
LUTs	16,201	25%
SLICEs	4,472	27%
DSP48As	20	7%
Block RAMs	0	0%

Tabelle 6.1.: Benötigte Ressourcen für die direkte Implementierung des Bitfehlerratenmessung Algorithmus

modul betrieben werden kann, sinkt auf 87 MHz. Damit ist diese Art der Implementierung sehr hardwarelastig. In Anbetracht, dass neben der Bitfehlerratenmessung noch Signalaufarbeitung und die Demodulation implementiert werden soll, ist diese Art der Implementierung nicht verwendbar.

6.2. Optimierung des Bitfehlerrate-Algorithmus

Da eine Division sehr rechenaufwändig ist und damit viele Hardware-Ressourcen benötigt werden, gilt es Divisionen in der Signalverarbeitung soweit als möglich zu vermeiden. Formel 6.2 lässt sich auf Grund des Logarithmus zu

$$\log_{10}(\mathrm{BER}) = \log_{10}(e) - \log_{10}(a) \tag{6.5}$$

umstellen. Somit ist eine Division vermieden worden. Da die Binäre Darstellung bereits in Zweierpotenzen codiert ist, lohnt es sich anscheinend den Logarithmus zur Basis Zehn durch einen Logarithmus zur Basis Zwei zu ersetzen. Durch Verwendung des Zusammenhanges

$$\log_b r = \frac{\log_a r}{\log_a b} \tag{6.6}$$

lässt sich Formel 6.2 zu

$$\log_{10}(\mathrm{BER}) = \{\log_2(e) - \log_2(a)\} \cdot \frac{1}{\log_2(10)} \tag{6.7}$$

umstellen. Bild 6.2 zeigt die Implementierung des Bitfehlerrate-Algorithmus nach Formel 6.7. Der Faktor $\frac{1}{\log_2(10)} = 0.30103$ ist als konstante Multiplikation mit den Faktor 0.3008 realisiert

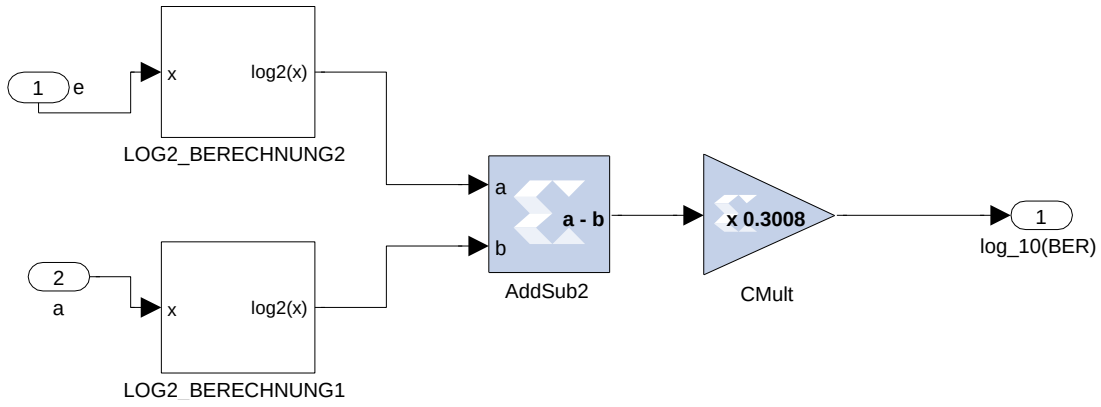


Bild 6.2.: Optimierte Implementierung eines Bitfehlerrate-Algorithmus

worden. Diese Ungenauigkeit ist der endlichen Genauigkeit der Festkommazahl geschuldet. Um die konstante Multiplikation möglichst gering vom Rechenaufwand zu halten, wurden für die Konstante 8 Bit mit einer Kommastelle am höchsten Bit gewählt. Der relative Fehler entspricht damit $\frac{0.3008}{0.30103} = 0.24\%$.

Die gezeigte optimierte Implementierung des Bitfehlerrate-Algorithmus lässt sich in einem zweiten Schritt weiter optimieren. Dazu wird die Darstellung einer binären Zahl betrachtet. Eine binäre Zahl stellt sich als eine Reihe von Zweierpotenzen, in der Form

$$Z = \sum_{i=0}^{m} z_i \cdot 2^i = z_0 \cdot 2^0 + z_1 \cdot 2^1 + \cdots + z_{m-1} \cdot 2^{m-1} + z_m \cdot 2^m \tag{6.8}$$

dar. Dabei gilt

$$(m \in \mathbb{N} \quad z_i \in \{0, 1\}) \,. \tag{6.9}$$

Wird eine Möglichkeit gefunden, das höchste gesetzte Bit x einer Zahl Z mit einem geschickten Hardwareaufbau zu finden, so lässt sich der Logarithmus zu Basis 2 in einem weiteren Schritt optimieren. Die Hardware zur Findung des höchste gesetzte Bit wird weiter unten ermittelt. Vorerst soll der Vorteil aus der gewonnen Information x dargestellt werden. Der Logarithmus zur Basis 2 lässt sich unter Kenntnis der Position x wie folgend berechnen.

$$\log_2 Z = \log_2 \left(z_x \cdot 2^x + z_{x-1} \cdot 2^{x-1} + z_{x-2} \cdot 2^{x-2} + z_{x-3} \cdot 2^{x-3} + z_{x-4} \cdot 2^{x-4} \right) \tag{6.10}$$

Die Beschränkung auf vier weitere Bits unter dem Bit x, erzeugt wieder eine kleine Ungenauigkeit der Berechnung. Weiter unten wird dargestellt, dass diese Ungenauigkeit sehr gering ist. Vorerst soll aber dargestellt werden, dass durch den Zusammenhang

$$\log_a (x + y) = \log_a x + \log_a \left(1 + \frac{y}{x} \right) \tag{6.11}$$

Formel 6.10 zu

$$\log_2 Z = \log_2 (z_x \cdot 2^x) + \log_2 \left(1 + z_{x-1} \cdot 2^{-1} + z_{x-2} \cdot 2^{-2} + z_{x-3} \cdot 2^{-3} + z_{x-4} \cdot 2^{-4} \right) \tag{6.12}$$

erweiteret werden kann. Der linke Term löst sich zu

$$\log_2 Z = x + \log_2 \left(1 + z_{x-1} \cdot 2^{-1} + z_{x-2} \cdot 2^{-2} + z_{x-3} \cdot 2^{-3} + z_{x-4} \cdot 2^{-4} \right) \tag{6.13}$$

auf. Das Besondere an dieser Form ist, dass der linke Term gleich der bereits gefundenen Information x entspricht. Die Position des höchst gesetzte Bit entspricht somit dem linken Term und muss nicht mehr berechnet werden. Der rechte Term hat kein x mehr in den Zweierpotenzen. Damit variiert der rechte Term je nachdem welche Bits unterhalb dem höchste gesetzte Bit gesetzt sind. Dazu gibt es bei vier weiteren Bits $2^4 = 16$ Möglichkeiten. Diese können einfach und ressourcensparend mit einem ROM realisiert werden.

Bild 6.3 zeigt die Implementierung des Logarithmus zur Basis Zwei nach Formel 6.13. Ein Zähler zählt von 63 runter bis 0. Der Zählerstand wird als Select-Eingang für ein Multiplexer verwendet. Somit werden mit dem Zähler die einzelnen Bits von Eingang y von oben nach unten durchsucht. Trifft der Zähler dabei auf ein gesetztes Bit, so wird der Zählerstand an

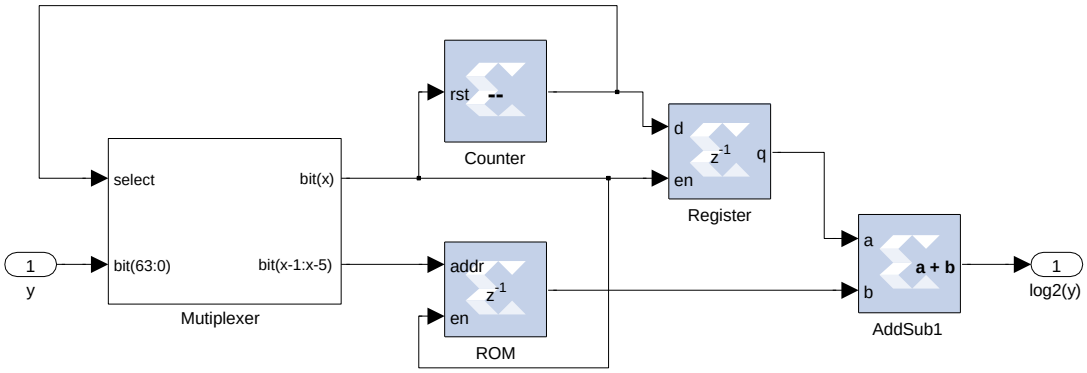


Bild 6.3.: Optimierte Implementierung des Logarithmus zur Basis Zwei

einem Register übernommen. Ein ROM, der mit den vier nächst tieferen Bits am Adressbus angesteuert wir, wird ausgelesen. Der im Register übernommene Zählerstand wird mit dem Ausgang des ROMs addiert. Der Ausgang des Addierers ist der $\log_2(y)$. Beim Übernehmen der Daten im Register und am ROM wird der Zähler zeitgleich gelöscht. Somit startet der Vorgang von Neuem.

Es stellt sich die Frage, mit welchen Werten der ROM gefüllt werden muss und mit welcher Genauigkeit diese Implementierung arbeitet. Durch Betrachten des rechten Terms in Formel 6.13 ist erkenntlich, dass nun die Exponenten frei von der Variablen x sind. Es handelt sich somit um Konstanten. Es wurden 4 weitere Bits ausgewertet. Aus den 4 Bits ergeben sich 16 verschiedene Werte, die mit dem MATLAB-Code

```
1 log2(1+(linspace(0,15,15)/16))
```

berechnet werden können und fest in einen ROM geschrieben werden.

Durch Berechnung des kleinsten gesetzten Bit ergibt sich eine Genauigkeit des Exponenten von

$$\text{Resolution}\left\{\log_{10}(\text{BER})\right\} = \frac{1}{2}\cdot\log_2\left(1+1\cdot 2^{-4}\right)\cdot 0.3008 = 0.0132. \tag{6.14}$$

Eine Genauigkeit von 0.0132 ist für eine Zehnerpotenz, an der statistisch festgestellt werden soll, wie fehlerfrei ein System arbeitet, ausreichend.

Nach Synthese wurden die benötigten Ressourcen ausgewertet. Tabelle 6.2 listet die benötigten Ressourcen auf.

Benötigte Hardware	Anzahl	Anteil am Virtex-5 FX100T
FFs	38	0.06%
LUTs	215	0.34%
SLICEs	93	0.58%
DSP48As	0	0%
Block RAMs	0	0%

Tabelle 6.2.: Benötigte Ressourcen für die optimierte Implementierung des Bitfehlerratenmessung Algorithmus

Die maximale Frequenz mit dem diese Komponente betrieben werden kann, steigt auf $250\,\mathrm{MHz}$. Damit ist die optimierte Art der Implementierung um den Faktor $\frac{250\,\mathrm{MHz}}{87\,\mathrm{MHz}} = 2.9$ schneller. Eine Aktualisierung der Bitfehlerrate im Sekundentakt wäre ausreichend. Jedoch erspart die Möglichkeit der hohen Taktung zusätzliche Taktteiler, welche auch wiederum Hardware benötigen würden. Die benötigten Ressourcen reduzieren sich um den Faktor 435 bei den FFs, 75 bei den LUTs und 48 bei den Slices. Im Gegensatz zur direkten Implementierung werden auch keine DSP48As benötigt.

7. Implementierung der Modulationsarten

In Kapitel 3.5 wurden die verschiedenen Basisbandmodulationsarten ausführlich besprochen und zwei Basisbandmodulationsarten, das OOK und ein 3-DPSK, zur Implementierung ausgewählt. Im aktuellen Kapitel soll auf die Implementierung eingegangen werden. Zusätzlich zeigt eine Systemsimulation die Funktion der Implementierung.

7.1. Implementierung des OOK

Die Implementierung des OOK gestaltet sich sehr simpel, da die zu übertragenden Bits direkt am Sender-FPGA ausgegeben werden können um im DAC interpoliert und mit dem Mischer auf den Träger moduliert zu werden. Das System Generator-Modell des OOK-Modulators ist in Bild 7.1 abgebildet.

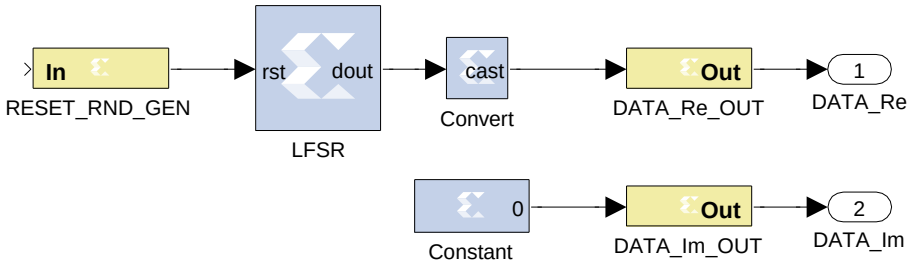


Bild 7.1.: System Generator-Modell des implementierten OOK-Modulators

Ein linear feedback shift register (LFSR), welches sich extern durch den *RESET_RND_GEN*-Dateneingang zurücksetzen lässt, gibt ein pseudozufälliges Signal an dem Datenausgang *DATA_Re_OUT* aus. In dem LFSR ist ein Pseudozufallspolynom aus [1] der Länge 8 implementiert worden. Damit wiederholt sich die Bitfolge nach $2^8 - 1 = 255\,\mathrm{Bit}$. Der *Convert*-Block passt den Datentyp des LFSR-Ausgangssignals an den notwendigen Datentyp des

DATA_Im_OUT-Datenausgangs an. Auf den *DATA_Im_OUT*-Datenausgang wird dauerhaft eine 0 angelegt.

Am Empfänger ist mehr Aufwand für die Demodulation notwendig. Zum einen kann das empfangene Signal auf der Übertragungsstrecke gedämpft worden sein. Somit ist der Pegel des Signals je nach Dämpfung unterschiedlich. Ein fester Schwellwert kann daher nicht implementiert werden. Des Weiteren ist das empfangene Signal durch die Impulsformung nur zu einem Zeitpunkt optimal zu unterscheiden. Als Drittes kann nicht vorausgesetzt werden, dass das empfangene Signal ausschließlich einen Realteil aufweist. Selbst bei einem sehr geringen CFO zwischen Sender und Empfänger wird sich das Konstellationsdiagramm des OOK drehen. Somit steckt zeitweise im Imaginärteil, zeitweise im Realteil und zeitweise in beiden Teilen das modulierte Signal. Das System Generator-Modell des OOK-Demodulators ist in Bild 7.2 abgebildet.

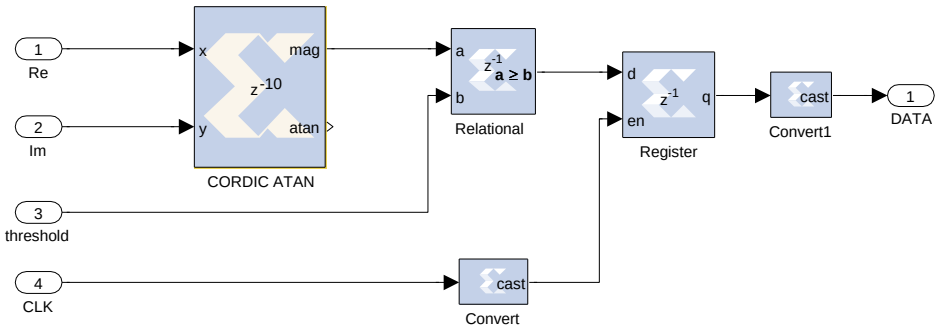


Bild 7.2.: System Generator-Modell des implementierten OOK-Demodulators

Der Real- und Imaginärteil des Basisbandsignals wird auf einem CORDIC-Block gegeben. Dieser rechnet aus dem Real- und Imaginärteil den Betrag des Basisbandsignals aus. Somit ist das Ausgangssignal des CORDIC-Blocks unabhängig von einem CFO. Extern lässt sich ein Schwellwert über den *threshold*-Port eingeben. Der *Relational*-Block entscheidet mit dem Schwellwert, welche Signalwerte einer logischen 0 und welche Signale einer logischen 1 zugewiesen werden. Über den *CLK*-Port wird der optimale Zeitpunkt zum Übernehmen der Daten übergeben. Das *CLK*-Signal ist nicht der FPGA-Systemtakt, sondern der Symboltakt der für eine FPGA-Systemtaktlänge *high* gehalten wird, wenn der optimale Zeitpunkt zum Übernehmen der Daten ist. Am Ausgangsport *DATA* wird die demodulierte Datenfolge ausgegeben.

In einer System Generator Simulation wurde das gesamte System mit Sender und Empfänger simuliert. Bild 7.3 zeigt das System Generator-Modell der Simulation.

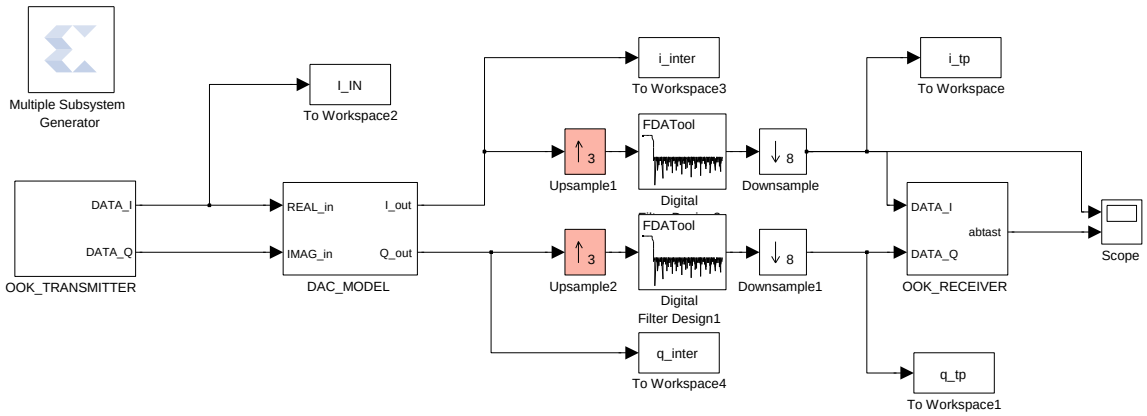


Bild 7.3.: System Generator Simulation des gesamten Systems

Der *OOK_TRANSMITTER* sendet das basisbandmodulierte Signal an ein *DAC_MODELL*. Ein Modell des DAC mit Interpolationsfilter und Mischer wurde zum Zweck dieser Simulation in MATLAB/Simulink implementiert. Mit den beiden digitalen Filtern und den Up- und Down-Samplern wird die Datenrate auf die Eingangsdatenrate des *OOK_RECEIVER* angepasst. Ein Gesamtmodell des Receivers ist dem Anhang G.1 angehängt. Die Simulation zeigte, dass die Datenübertragung fehlerfrei funktioniert. In Bild 7.4 ist das Simulationsergebnis aufgeführt.

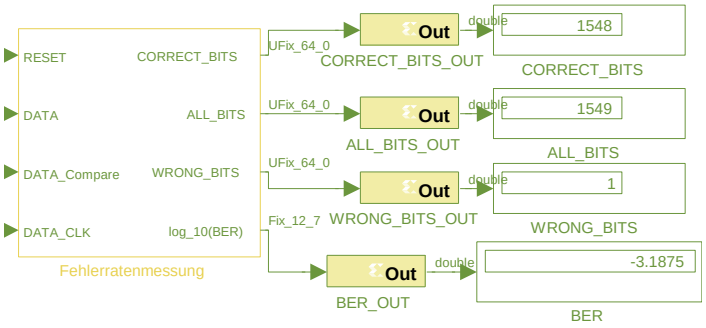


Bild 7.4.: System Generator OOK Simulationsergebnis

Es wurden 1548 Bit übertragen und 1548 Bit korrekt empfangen. Ein Fehlerbit wurde vorinitialisiert, damit die Bitfehlerratenberechnung einen gültigen Wert propagiert. Da die Simulation sehr rechenaufwändig ist, kann eine Fehlerrate erst mit dem endgültigen Aufbau ermittelt werden.

7.2. Implementierung des 3-DPSK

Die Implementierung des 3-DPSK erfordert sowohl für die Modulation, als auch für die Demodulation mehr Signalverarbeitungsschritte. Es ist sowohl eine Signalraumzuweisung, als auch ein Verarbeitungsschritt, mit dem die Eigenschaft eines *gedächnisbehafteten*-System erreicht wird, notwendig. Letztendlich muss die Phaseninformation in einen Real- und einen Imaginärteil umgerechnet werden. Bild 7.5 zeigt das System Generator-Modell des Modulators, welches in den Sender implementiert wurde.

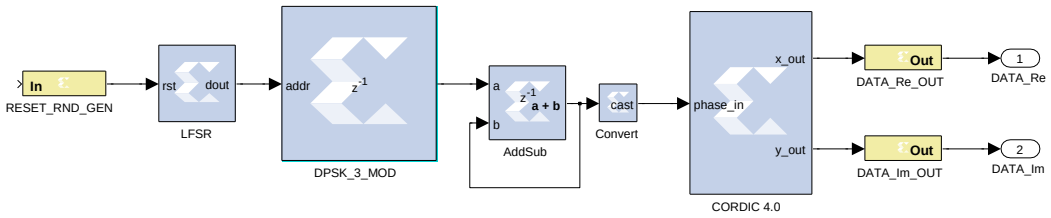


Bild 7.5.: System Generator-Modell des implementierten 3-DPSK-Modulators

Das pseudozufällige Signal wird auf dem gleichen Weg realisiert, wie oben bei der OOK-Modulation beschrieben. Die Daten aus dem LFSR werden mittels des ROMs, der in dem *DPSK_3_MOD*-Block realisiert ist, in Phasensprünge umgerechnet. Im gegebenen System ist die Phaseninformation auf π normiert. Ein Phasenwert von π bzw. von $180°$ entspricht somit dem Wert 1. Ein Phasenwert von $-\pi/2$ bzw. von $-90°$ entspricht somit dem Wert -0.5. Somit wird im *DPSK_3_MOD*-ROM eine logische 0 auf $+2/3$ und eine logische 1 auf $-2/3$ gemappt. Der nachgeschaltete *AddSub*-Block ist als ein vorzeichenbehafteter Addierer realisiert, dessen maximale Ausgangswerte von -1 bis $+1$ gehen. Überschreitet eine Addition die obere oder untere Grenze, so produziert der Addierer einen Überlauf. Diese Eigenschaft lässt sich durch den Parameter *wrap* einstellen. Der Addierer fungiert als Gedächtnis des Modulators, denn der Addierer sorgt dafür, dass ein propagierter Wert von allen vorangegangenen Werten abhängt. Die berechnete Phase wird mittels CORDIC-Block in einen Real- und einen Imaginärteil umgerechnet. An den Datenausgängen *DATA_Re_OUT* und *DATA_Im_OUT* werden die modulierten Daten ausgegeben.

Am Empfänger muss der Demodulator das empfangene Basisbandsignal erst wieder in eine Phase umrechnen. Ist dies geschehen, müssen die Phasensprünge detektiert werden. Dazu muss die Differenz zwischen zwei Symbolen berechnet werden. Ist die Differenz kleiner als π, so wurde eine logische 0 übertragen. Ist die Differenz größer als π, so wurde eine logische 1 übertragen. Der Demodulator ist unabhängig von Dämpfungen auf der Übertragungsstrecke, da die Information auf die Phase des Trägers moduliert wurde. Ein CFO wirkt sich jedoch als zusätzlichen Phasensprung aus. Bild 7.6 zeigt das System Generator-Modell des implementierten 3-DPSK-Demodulators.

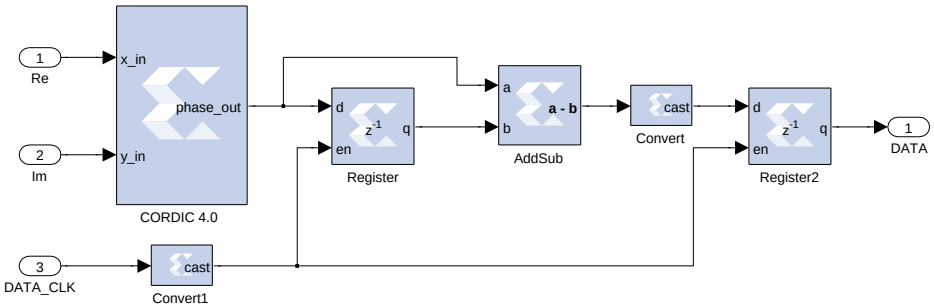


Bild 7.6.: System Generator-Modell des implementierten 3-DPSK-Demodulators

Der CORDIC-Block rechnet den Real- und Imaginärteil *Re_IN* und *Im_IN* in eine Phase um. Die berechnete Phase wird auf ein *AddSub*-Block, der als Subtrahierer konfiguriert ist, gegeben. Des Weiteren wird die berechnete Phase um einen Symboltakt mit dem *Register* verzögert. Aus der um einen Symboltakt verzögerten Phase und der aktuellen Phase wird mit dem *AddSub*-Block eine Differenz berechnet. Die Differenz entspricht einem Phasensprung. Mit dem *Convert*-Block wird der Phasensprung zu einem Datenwert gemappt. Am *Register2*-Block werden die Datenwerte übernommen. Über den *DATA_CLK*-Port wird der optimale Zeitpunkt zum Übernehmen der Phase und der Daten übergeben. Das *DATA_CLK*-Signal ist nicht der FPGA-Systemtakt, sondern der Symboltakt, der für eine FPGA-Systemtaktlänge *high* gehalten wird, wenn der optimale Zeitpunkt zum Übernehmen der Daten ist. Am Ausgangsport *DATA* wird die demodulierte Datenfolge ausgegeben.

In einer System Generator Simulation wurde das gesamte System mit Sender und Empfänger simuliert. Dazu wurde die gleiche Simulationsapplikation wie beim OOK verwendet. Oben in Bild 7.3 ist das System Generator-Modell der Simulation abgebildet und beschrieben worden. Ein Gesamtmodell des Receivers ist dem Anhang G.2 angehängt. Die Simulation zeigte, dass die Datenübertragung fehlerfrei funktioniert. In Bild 7.7 ist das Simulationsergebnis aufgeführt.

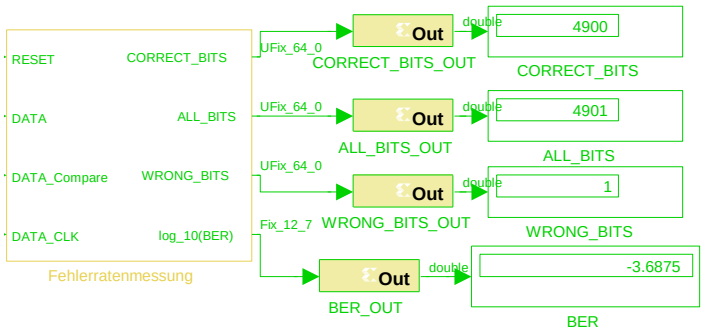


Bild 7.7.: System Generator 3-DPSK Simulationsergebnis

Es wurden 4900 Bit übertragen und 4900 Bit korrekt empfangen. Ein Fehlerbit wurde vorinitialisiert, damit die Bitfehlerratenberechnung eine gültigen Wert propagiert. Da die Simulation sehr rechenaufwändig ist, kann eine Fehlerrate erst mit dem endgültigen Aufbau ermittelt werden.

Teil III.

Test

In diesem letzten Teil der Masterarbeit soll der gesamte Aufbau mit allen implementierten Modulen getestet werden. In Kapitel 8 wird der Aufbau des Systems dokumentiert. In Kapitel 9 werden die beiden implementierten Modulationsarten getestet. Um Rückschlüsse auf die Phasenresistenz der Modulationsarten zu schließen, wurde in Kapitel 10 die Bitfehlerrate in Abhängigkeit eines CFO aufgenommen. Im letzten Kapitel 11 wird eine Zusammenfassung der Masterarbeit aufgeführt und ein Ausblick auf weitere Betätigungsfelder gegeben.

8. Systemaufbau

In Bild 8.1 ist ein Blockschaltbild des Hardwareaufbaus abgebildet. Folgend werden die einzelnen Elemente des Hardwareaufbaus dargestellt.

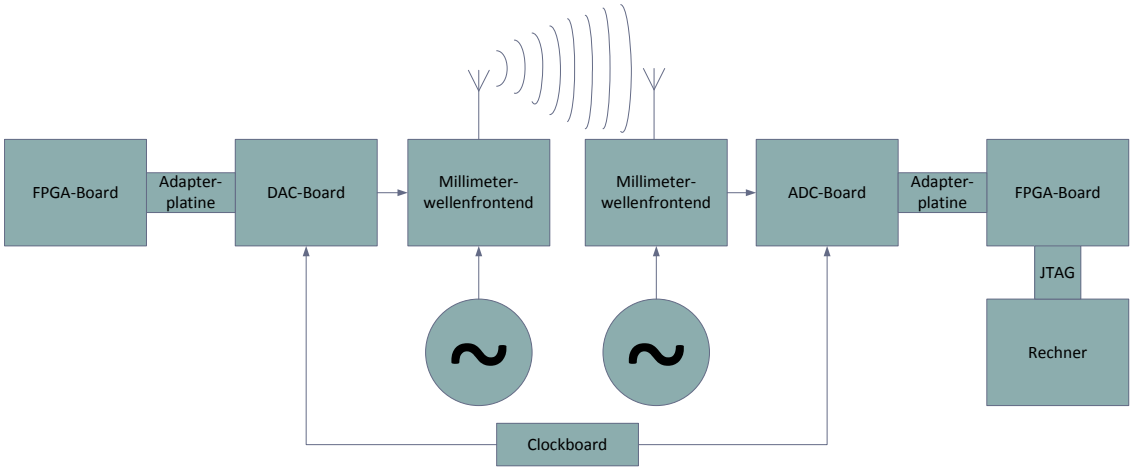


Bild 8.1.: Blockschaltbild des Hardwareaufbaus

Bild 8.2 zeigt den Sender-FPGA mit der in Kapitel 4.1.1 beschriebenen Adapterplatine, die das Sender-FPGA-Board mit dem DAC-Board verbindet.

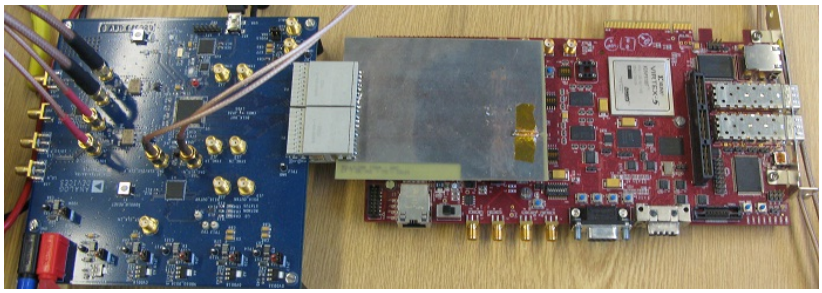


Bild 8.2.: Sender-FPGA mit DAC

Bild 8.3 zeigt das Clockboard, welches den ADC und den DAC mit einen Takt versorgt.

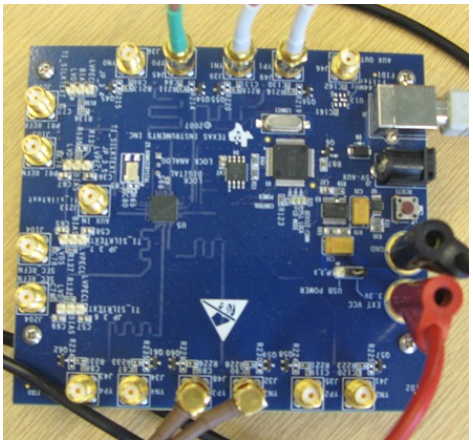


Bild 8.3.: Clockboard

Bild 8.4 zeigt die in Kapitel 5.3.3 beschriebenen Rekonstruktionsfilter.

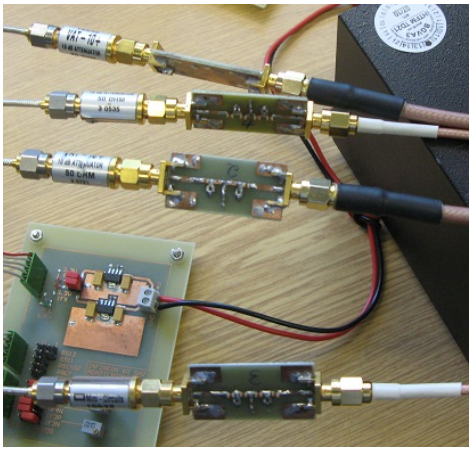


Bild 8.4.: Rekonstruktionsfilter

Hinter den Rekonstruktionsfiltern sind Dämpfungsglieder geschaltet, um die DAC-Ausgangspegel optimal an die Millimeterwellen-Frontend-Eingangspegel anzupassen. Bild 8.5 zeigt die Sender- und Empfänger-Millimeterwellen-Frontends verbunden mit Hohlleitern.

Bild 8.5.: Sender- und Empfänger-Millimeterwellen-Frontends verbunden mit Hohlleitern

In den Hohlleiter ist ein variables Dämpfungsglied eingebaut worden. Mit dem variablen Dämpfungsglied kann der Pegel des Hochfrequenzsignals optimal eingestellt werden. Die Millimeterwellen-Frontends sollen mit je einer, in Bild 8.6 dargestellten, Hornantenne betrieben werden.

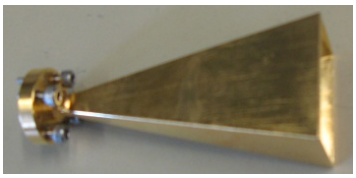


Bild 8.6.: 60 GHz-Hornantenne

Für den Testbetrieb und um Messreihen aufnehmen zu können, werden die Hohlleiter mit einem Dämpfungsglied verbunden. Dadurch lassen sich äußere Einflüsse eliminieren. Die Messergebnisse sind auch unabhängig von dem Abstand und dem Winkel der Millimeterwellen-Frontends zueinander. Dies sind wichtige Maßnahmen um die Messergebnisse reproduzierbar zu machen. In Bild 8.7 sind die Sinusquellen zur Generierung der Referenzfrequenzen zur Erzeugung der Trägersignale abgebildet, eine Sinusquelle zur Versorgung des Sender-Millimeterwellen-Frontends und eine Sinusquelle zur Versorgung des Empfänger-Millimeterwellen-Frontends.

Bild 8.7.: Sinusquellen zur Generierung der Trägersignale

Die beiden Sinusquellen lassen sich durch ein Referenzsignal miteinander synchronisieren. Somit lässt sich ein definierte Trägerfrequenzabweichung (CFO) der Quellen zueinander einstellen. Dies ist nötig, um die in Kapitel 10 gemachte Messreihe *BER in Abhängigkeit eines CFO* aufnehmen zu können. Bild 8.2 zeigt das ADC-Board mit der Adapterplatine, die das ADC-Board mit dem Empfänger-FPGA-Board verbindet.

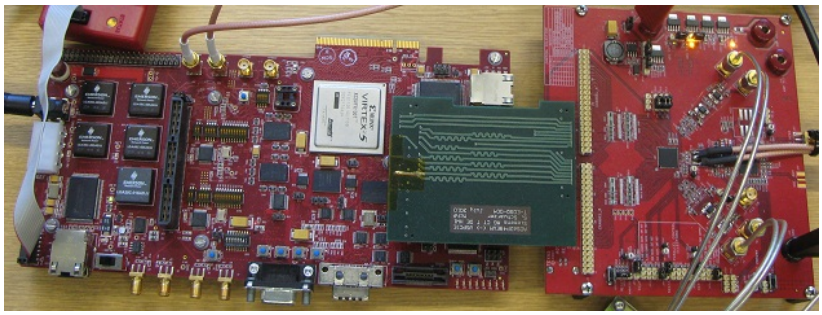


Bild 8.8.: Empfänger-FPGA mit ADC

Der Vollständigkeit halber ist in Bild 8.9 der Gesamtaufbau des Übertragungssystems noch einmal dokumentarisch dargestellt.

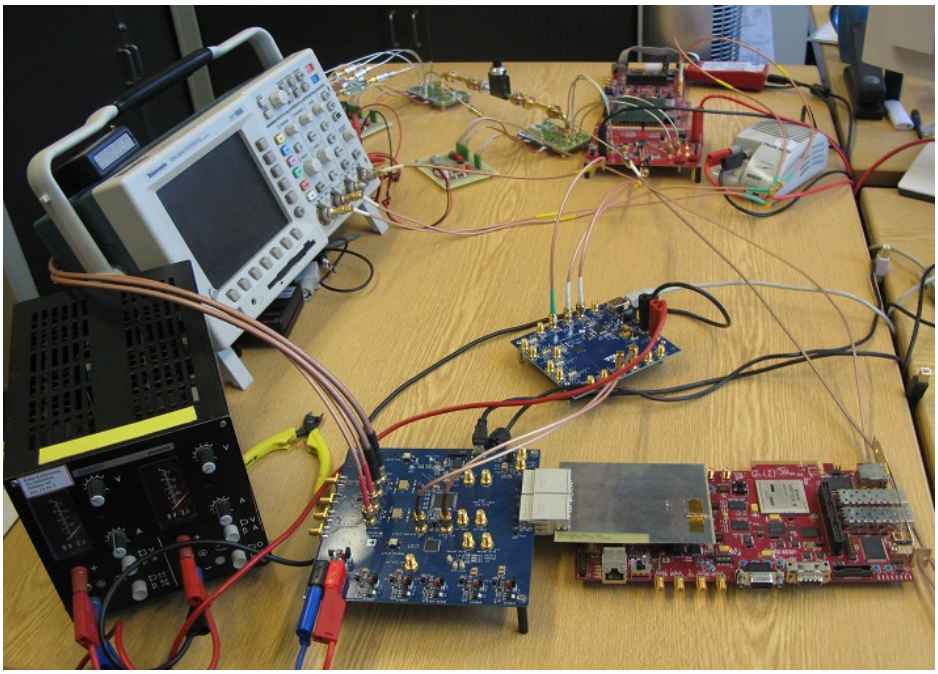


Bild 8.9.: Gesamtaufbau des Übertragungssystems

Um die Leistungsfähigkeit der beiden Modulationsarten ermitteln zu können, wurde in den Empfänger das in Kapitel 6 beschriebene Bitfehlerratenmessmodul integriert. Zur Visualisierung der Messergebnisse wurde der Empfänger-FPGA via JTAG-Schnittstelle mit einem Rechner verbunden. Außerdem wurden in den Empfänger-FPGA sogenannte ChipScope-Module integriert. ChipScope ist ein Debugging-Tool von Xilinx zum Testen von VHDL-Code im laufenden Betrieb. Mit ChipScope lassen sich einzelne Register in regelmäßigem Abstand auslesen oder sogar ganze Datensequenzen auf ein Trigger-Signal hin erfassen. Es können auch Register im laufenden Betrieb gesetzt werden. So wurde mit ChipScope via JTAG-Schnittstelle unter anderem die gemessene Bitfehlerrate aus dem Empfänger-FPGA ausgelesen.

Zur Realisierung einer modulierten Datenübertragung wurde die in Kapitel 7 erzeugten System Generator-Modelle in je ein Sender- und ein Empfänger-Top-Level-Design integriert. Neben den System Generator-Modellen sind in die Top-Level-Designs die ChipScope-Module und die Wandlerschnittstellen aus Kapitel 4.1.2 eingebunden. Im Anhang ist der Quellcode H.1 für den Sender-FPGA und der Quellcode H.2 für den Empfänger-FPGA angehängt. Die angehängten Quellcodes sind die Top-Level-Designs für das OOK. Die Top-Level-Designs für das 3-DPSK sind dem des OOK, bis auf die System Generator-Modell Integration, identisch.

9. Test der beiden Modulationsarten

In diesem Kapitel werden die beiden Modulationsarten OOK und 3-DPSK am Gesamtaufbau getestet.

9.1. OOK

Zuerst wurde mittels ChipScope ein Augendiagramm der empfangenen Daten aufgenommen. Bild 9.1 zeigt das aufgenommene Augendiagramm. Da der ADC das empfangene Si-

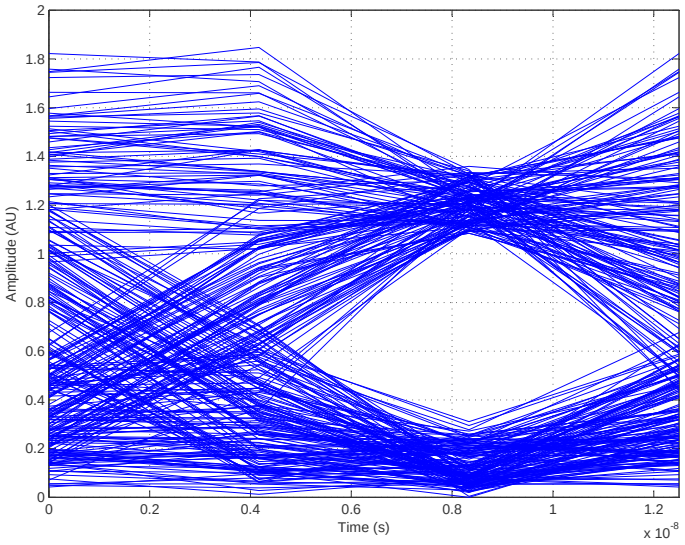


Bild 9.1.: Gemessenes Augendiagramm am Empfänger bei OOK

gnal mit drei Abtastpunkten pro Symbol abtastet, ist das Augendiagramm sehr eckig. Der Takt vom Clockboard, der den ADC versorgt, wurde so eingestellt, dass ein Abtastpunkt genau in der maximalen Öffnung des Augendiagramms liegt. Bei dieser Einstellung sind die besten Messergebnisse zu erwarten. Die Problematik, dass nur drei Abtastpunkte pro Symbol aufgenommen wurden, wurde bereits am Ende des Kapitels 5.4 besprochen. Dennoch

müsste nach den im Kapitel 5.1 simulierten Impulsformungsfilterkoeffizienten das Augendiagramm vertikal komplett geöffnet sein. Eine gesonderte Untersuchung dieser Erscheinung erfolgt im Unterkapitel 9.2. Dieses schon leicht geschlossene Auge führte zu einer Bitfehlerrate von 10^{-12}. Außerdem wurde ein Test ohne Dämpfungsglied und mit Hornantenne durchgeführt, um die Funktionsfähigkeit des Übertragungssystems bei Funkübertragung zu verifizieren. Die Fehlerrate betrug $10^{-11,7}$. Um die Messergebnisse mit den theoretischen Bitfehlerwahrscheinlichkeiten aus Kapitel 3.5 vergleichen zu können, müsste ein SNR bestimmt werden. Dies ist leider am aktuellen Stand des Demonstrators noch nicht möglich. Um eine SNR-Messung in den Demonstrator zu implementieren, müsste im Sender ein Sinussignal implementiert werden. Am Empfänger müssten die Daten mit ChipScope ausgelesen werden, um diese mit MATLAB auszuwerten.

9.2. Anmerkung zum Rekonstruktionsfilter

Beim ersten Test fiel auf, dass das Augendiagramm, abgebildet in Bild 9.1, vertikal nicht komplett geöffnet ist. Nach den im Kapitel 5.1 simulierten Impulsformungsfilterkoeffizienten sollte das Augendiagramm vertikal komplett geöffnet sein. An dieser Stelle soll untersucht werden, ob das realisierte Rekonstruktionsfilter die Gesamtübertragungsfunktion so sehr beeinflusst, dass ISI auftritt. Hierzu wurde ein MATLAB/Simulink-Modell, abgebildet in Bild 9.2, erzeugt. Ein Zufallsgenerator erzeugt ein Zufallssignal. Dieses Zufallssignal wird mit einem Vorzei-

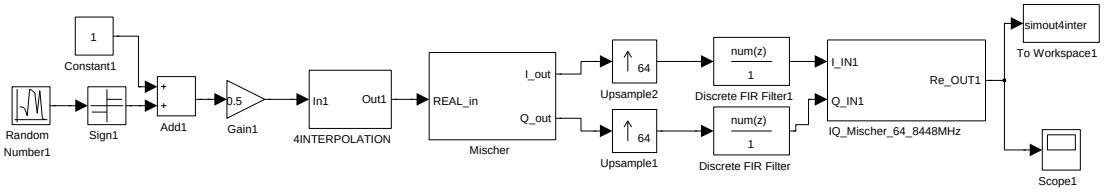


Bild 9.2.: MATLAB/Simulink-Modell zur Untersuchung der Gesamtübertragungsfunktion

chenoperator auf den Wert -1 und 1 gemappt. Durch Addition mit 1 und anschließender Multiplikation mit dem Faktor $0{,}5$ wird eine Zufallsfolge mit den Werten 0 und 1 erzeugt. Diese Zufallsfolge an das Interpolationsfilter/Impulsformungsfilter übergeben. Das Ausgangssignal der Interpolationsfilter mit einem Modell des DAC-Mischers in die Zwischenfrequenz gemischt. Das Signal teilt sich in einen In-Phase-Pfad und einen Quadratur-Phase-Pfad auf. Für eine bessere Augendiagrammauflösung wird die Abtastfrequenz um den Faktor 64 erhöht. Die digitalisierte Impulsantwort des Rekonstruktionsfilters wurde in den beiden FIR-Filtern nachgebildet. Schließlich wird das Signal, mit einem Modell des im Empfänger-FPGA realisierten Mischers, von der Zwischenfrequenz zurück ins Basisband gemischt und an den

MATLAB-Workspace übergeben. In MATLAB kann mit dem MATLAB-Code D.1 aus den Daten ein Augendiagramm erzeugt werden. Bild 9.3 zeigt das Augendiagramm der MATLAB/-Simulink Simulation.

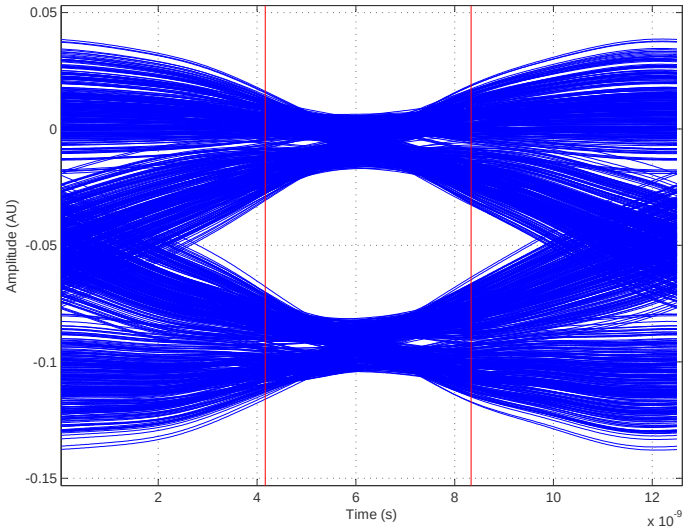


Bild 9.3.: MATLAB/Simulink simuliertes Augendiagramm zur Untersuchung der Gesamtübertragungsfunktion

Es ist klar erkennbar, dass das Rekonstruktionsfilter die Gesamtübertragungsfunktion so stark beeinflusst, dass ISI entsteht. Um ISI zu verhindern, muss ein neues Rekonstruktionsfilter entworfen werden. Dazu wäre es empfehlenswert das DAC-Ausgangssignal 8-fach zu interpolieren. Durch die höhere Interpolation schiebt sich das erste Image weiter vom Nutzsignal weg. Dadurch kann ein Rekonstruktionsfilter entworfen werden, welches einen breiteren Übergangsbereich hat. Ein Bessel-Filter könnte bei einem größeren Übergangsbereich die nötige Dämpfung im Sperrbereich erreichen. Eine Verzerrung des Spektrums durch einen nicht linearen Phasengang wie beim Chebyshev-Filter wäre damit ausgeschlossen.

9.3. 3-DPSK

Zuerst wurde mittels ChipScope ein Augendiagramm der empfangenen Daten aufgenommen. Bild 9.4 zeigt das aufgenommene Augendiagramm. Dabei handelt es sich um das Augendiagramm der Phasensprünge, die aus dem Real- und Imaginärteil des Basisbandsignals berechnet wurden. Es liegt die Vermutung nahe, dass dieses geschlossene Auge des Augendiagramms auf das Phasenrauschen der Millimeterwellen-Frontendszurück zu führen

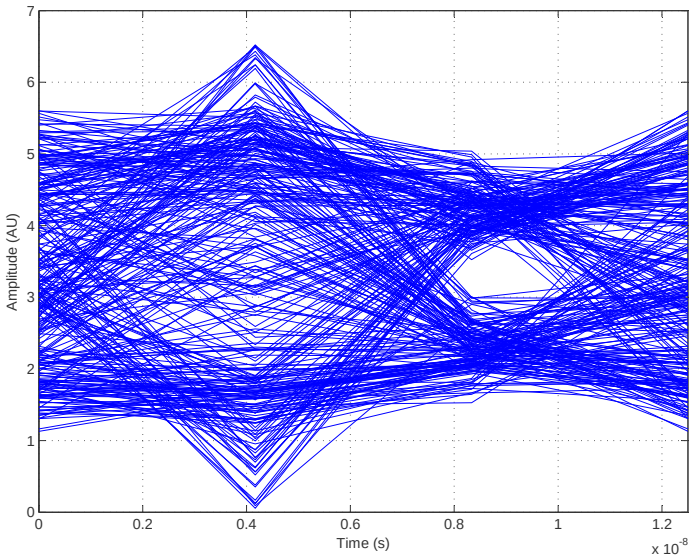


Bild 9.4.: Gemessenes Augendiagramm am Empfänger bei 3-DPSK

ist. In einem weiteren Versuch wurden die Millimeterwellen-Frontends aus der Übertragungsstrecke entfernt und die Zwischenfrequenz führenden Leitungen miteinander verbunden. Das Augendiagramm ohne Millimeterwellen-Frontendszeigte keine erkennbaren Unterschiede zum Augendiagramm mit Millimeterwellen-Frontends. Somit muss die Ursache des stark geschlossenen Auge eine andere Ursache haben als das Phasenrauschen. Eine schlüssige Erklärung des stark geschlossenen Auge ist, dass die in Kapitel 9.2 erwähnte ISI sich auf beide Kanäle, dem I-Kanal und Q-Kanal, auswirkt. Der ISI-Effekt addiert sich bei der 3-DPSK so sehr, dass das Auge im Augendiagramm sehr geschlossen ist. Es ist festzuhalten, dass bei einer Phasenmodulation ISI besonders zum Tragen kommt. Aus diesem Grund steigt die Bitfehlerrate bei der 3-DPSK auf $10^{-2,76}$ an.

10. BER in Abhängigkeit eines CFO

An dieser Stelle soll untersucht werden, wie resistent die beiden Modulationsarten gegenüber Phasenrauschen sind. Da keine Sinusquelle mit variablen Phasenrauschen zur Verfügung steht, wird eine Trägerfrequenzabweichung (engl.: Carrier Frequency Offset (CFO)) zwischen Sender- und Empfänger-Millimeterwellen-Frontends verwendet. Da Phasenrauschen, wie in Kapitel 2.2.1 ausführlich betrachtet, ein stochastischer Prozess ist und CFO ein rein deterministischer Prozess, lässt sich ein CFO nicht in ein Phasenrauschen umrechnen. Durch die folgende Messreihe lässt sich ausschließlich ein Trend erkennen.

10.1. 3-DPSK

In der Tabelle 10.1 ist die Bitfehlerrate in Abhängigkeit eines CFO bei 3-DPSK aufgenommen worden.

CFO	BER
0 MHz	$10^{-2,76}$
0,128 MHz	$10^{-2,62}$
0,256 MHz	$10^{-2,62}$
0,384 MHz	$10^{-2,37}$
0,512 MHz	$10^{-2,16}$
0,640 MHz	$10^{-2,00}$
0,768 MHz	$10^{-1,86}$
0,896 MHz	$10^{-1,77}$
1,024 MHz	$10^{-1,70}$
1,125 MHz	$10^{-1,63}$
1,280 MHz	$10^{-1,57}$
1,408 MHz	$10^{-1,52}$

Tabelle 10.1.: BER in Abhängigkeit eines CFO bei 3-DPSK

Wie erwartet steigt die Bitfehlerrate bei steigenden CFO. Bei einem CFO von $0{,}512\,\mathrm{MHz}$ steigt die Bitfehlerrate von $10^{-2,76}$ auf 10^{-2}. Ein CFO von $0{,}512\,\mathrm{MHz}$ entspricht bei Nor-

mierung auf Kanalbandbreite von $\approx 76{,}5\,\mathrm{MHz}$, entnommen aus dem Bild 5.5 aus Kapitel 5.2, $0{,}669\%$. Bei Normierung auf die Symbolrate von $80\,\mathrm{MHz}$ sogar nur $0{,}64\%$. Bild 10.1 zeigt die BER in Abhängigkeit eines CFO bei 3-DPSK in einem Plot. Es zeigt sich somit,

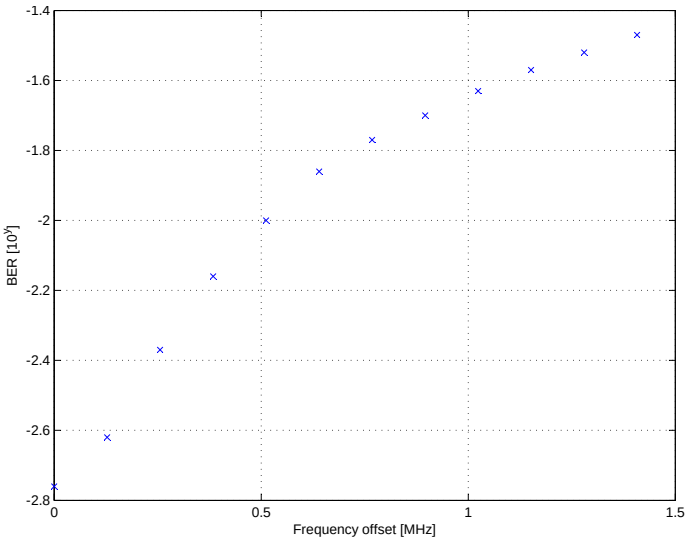


Bild 10.1.: BER in Abhängigkeit eines CFO bei 3-DPSK

dass eine Phasenmodulation sehr anfällig für ein CFO ist. Ein ähnliches Verhalten ist bei Auftreten eines Phasenrauschen zu erwarten.

10.2. OOK

In der Tabelle 10.2 ist die Bitfehlerrate in Abhängigkeit eines CFO bei OOK aufgenommen worden.

CFO	BER		CFO	BER
51,2 MHz	$10^{-10,30}$		192,0 MHz	$10^{-6,37}$
64,0 MHz	$10^{-9,53}$		204,8 MHz	$10^{-6,20}$
76,8 MHz	$10^{-8,61}$		217,6 MHz	$10^{-5,97}$
89,6 MHz	$10^{-7,83}$		230,4 MHz	$10^{-5,66}$
102,4 MHz	$10^{-7,42}$		243,2 MHz	$10^{-5,29}$
115,2 MHz	$10^{-7,03}$		256,0 MHz	$10^{-4,88}$
128,0 MHz	$10^{-6,73}$		268,8 MHz	$10^{-4,40}$
140,8 MHz	$10^{-6,61}$		281,6 MHz	$10^{-3,88}$
153,6 MHz	$10^{-6,55}$		294,4 MHz	$10^{-3,34}$
166,4 MHz	$10^{-6,52}$		307,2 MHz	$10^{-2,80}$
179,2 MHz	$10^{-6,47}$		320,0 MHz	$10^{-2,35}$

Tabelle 10.2.: BER in Abhängigkeit eines CFO bei OOK

Ein CFO macht sich erst bei sehr großem CFO bemerkbar. Es werden noch sehr niedrige Bitfehlerraten erreicht, obwohl das Basisband nicht mehr in das Nyquistband gemischt wird. So ist bei einem CFO von $128,0\,\mathrm{MHz}$ kein Signalanteil mehr im Nyquistband, welches von $0\,\mathrm{Hz}$ bis $120\,\mathrm{MHz}$ geht. Jedoch ist, wie in Kapitel 5.3 erwähnt wurde, kein Anti-Aliasing-Filter implementiert worden. Dadurch wird das Signal im zweiten Nyquistband von $120\,\mathrm{MHz}$ bis $240\,\mathrm{MHz}$ und bei noch größeren CFO im dritten Nyquistband von $240\,\mathrm{MHz}$ bis $360\,\mathrm{MHz}$ abgetastet. Bild 10.2 zeigt die BER in Abhängigkeit eines CFO bei OOK in einem Plot. Die

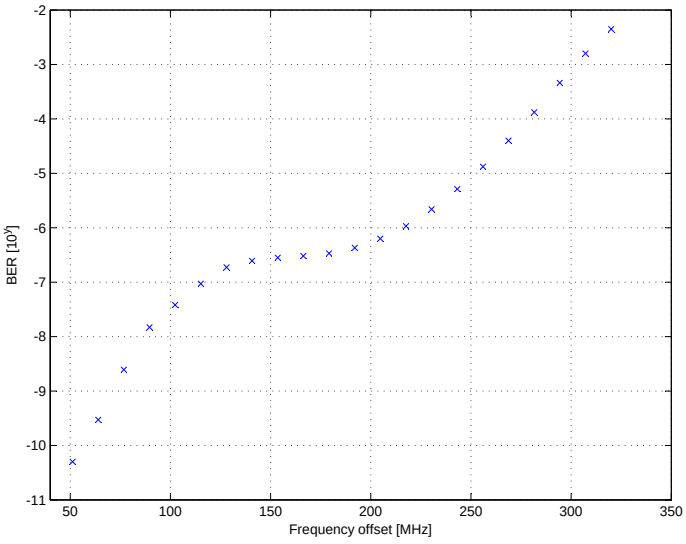


Bild 10.2.: BER in Abhängigkeit eines CFO bei OOK

Sattelstelle von $\approx 120\,\mathrm{MHz}$ bis $\approx 240\,\mathrm{MHz}$ im Plot lässt sich durch den Übergänge der Ny-

quistbänder erklären. An der Sattelstelle steigt die Bitfehlerrate nur sehr gering, da sich das Signalband genau im zweiten Nyquistband befindet. An den Übergängen der Nyquistbänder steigt die Bitfehlerrate sehr viel schneller an, da in beiden Bändern abgetastet wird. Erklären lässt sich das Steigen der Fehlerrate mit steigenden CFO durch die höhere Dämpfung des ADC-Einganges bei höheren Frequenzen. Es zeigt sich somit, dass eine Amplitudenmodulation sehr resistent gegen ein CFO ist. Ein ähnliches Verhalten ist bei Auftreten eines Phasenrauschens zu erwarten.

11. Zusammenfassung und Ausblick

Es wurde eine komplette Übertragungsstrecke mit Basisbandmodulation von OOK und 3-DPSK mit Impulsformung, Mischung in eine Zwischenfrequenz, Einbindung der Millimeterwellen-Frontends und Bitfehlerratenmessung erfolgreich implementiert und getestet. Eine Symboltaktrückgewinnung wurde ausführlich untersucht. Der enorme Implementierung- und Simulations-Vorteil, der durch das Xilinx-Tool System Generatorerreicht wurde, führte zu solch einem umfangreichen System in einer so kurzen Zeit. Ein Großteil der Arbeit beschäftigte sich mit der Theorie und Konzeption, sowie der Implementierung des Systems. In der Konzeption wurden die Einflüsse der Millimeterwellen-Frontends auf eine Datenübertragung umfangreich untersucht. Für alle gängigen Basisbandmodulationsverfahren wurden der theoretische Einfluss von Phasenrauschen und $\frac{E_b}{N_0}$ untersucht. Im Test wurde gezeigt, dass Amplitudenmodulationsverfahren weitgehend resistent gegenüber einem CFO sind. Phasenmodulationen dagegen sind sehr anfällig gegenüber einem CFO. Eine Skalierbarkeit des Systems kann zum jetzigen Zeitpunkt noch nicht erfolgen. Zuvor müssen Fehlerquellen, wie das Rekonstruktionsfilter, eliminiert werden. Sicher lassen sich dann mehr Stufen in die Amplitude einbauen. Eventuell lässt sich sogar Information in auf die Phase modulieren. Dabei ist die große Unbekannte, wie sich die 240 GHz-Millimeterwellen-Frontends verhalten werden.

Ein wichtiger nächster Schritt für weitere Untersuchungen, ist eine Möglichkeit zu schaffen, die Parameter des SNR und des Phasenrauschens zu bestimmen und bestenfalls variabel zu beeinflussen. Nur so lässt sich die Theorie aus Kapitel 3.5 überprüfen. Für die Bestimmung des SNR wurde bereits ein Verfahren in Kapitel 9 angegeben. Der SNR lässt sich variabel durch Dämpfung des Hochfrequenzsignals beeinflussen. Ein variables Dämpfungsglied ist bereits in den Demonstrator integriert. In [4] ist ein Verfahren angegeben, wonach sich aus dem Spektrum eines Oszillators das Oszillator-Phasenrauschen berechnen lässt. Dieses Verfahren kann auch auf ein empfangenes Sinussignal angewendet werden. Somit ließe sich ein Phasenrauschen des gesamten Übertragungssystems bestimmen. Durch Messen des Phasenrauschens mit Millimeterwellen-Frontends und ohne Millimeterwellen-Frontends würde sogar ein Verfahren entstehen, wonach sich das Phasenrauschen der Millimeterwellen-Frontendsbestimmen ließe, denn die Varianz des Phasenrauschens ist nach [17] kommutativ. Durch

$$\sigma^2_{\theta,\text{Frontend}} = \frac{\sigma^2_{\theta,\text{Sys. mit Frontends}} - \sigma^2_{\theta,\text{Sys. ohne Frontends}}}{2} \tag{11.1}$$

ließe sich die Varianz des Phasenrauschens der Millimeterwellen-Frontends bestimmen. Ein Verfahren, um ein Phasenrauschen variabel zu beeinflussen, ist noch nicht untersucht worden. Auch hier eröffnet sich ein interessantes Betätigungsfeld. Lässt sich das Phasenrauschen bestimmen, so können auch Rückschlüsse auf das Verhältnis von Phasenrauschen zu CFO geschlossen werden. Die beiden Eigenschaften lassen sich zwar mathematisch nicht ineinander überführen, wie bereits in Kapitel 2.2.1 und 10 erklärt wurde, jedoch ließen sich durch Gegenüberstellen von der Bitfehlerrate in Abhängigkeit eines Phasenrauschens mit der Bitfehlerrate in Abhängigkeit eines CFO bestimmen, wie proportional der Einfluss von Phasenrauschen zu CFO ist.

Verzeichnisse

Glossar

additive white gaussian noise
: ist weißes Rauschen welches sich additive auswirkt. Die Eigenschaften sind eine gaußverteilte Signalamplitude und eine konstante Rauschleistungsdichte. In der Nachrichtentechnik werden AWGN-Kanäle als Modellkanäle angenommen, da es sich um ein einfach zu berechnendes Modell handelt, welches der Realität sehr nahe kommt.

bidirektional
: Das Attribut bidirektional bedeutet, dass eine Datenübertragung in beide Richtungen Punkt zu Punkt stattfindet.

ChipScope
: Ist ein Debugging-Tool von Xilinx zum testen von VHDL-Code im laufenden Betrieb.

Field Programmable Gate Array
: Sind Weiterentwicklungen von maskenprogrammierbare Gate-Arrays. Dabei wurden die Transistoren bereits vorgefertigt und die individuellen Schaltungsfunktionalität wurden durch anwenderspezifische Masken für die Verdrahtungsebene erreicht...Dieses Konzept wurde durch Einführung der feldprogrammierbaren Gate-Arrays (FPGAs) dahingehend erweitert, dass der Anwender bei diesen Bausteinen, ..., seine spezifischen Anwendungen nun auch vor Ort (“im Feld”) konfigurieren kann [18].

Intersymbolinterferenz
: auch als Symbolübersprechen bezeichnet, beschreibt das zeitliche Übersprechen von Symbolimpulsfolgen. Die Ursache für Intersymbolinterferenz ist die Bandbegrenzung von Übertragungskanälen.

IP CORE Generator & Architecture Wizard
: ist ein von Xilinx entwickelter Wizard mit dem IP COREs und Hardwarearchitekturen erzeugt werden können.

Jitter
bezeichnet das zeitliche Taktzittern bei der Übertragung von Digitalsignalen, sowie eine Genauigkeitsschwankung im Übertragungstakt. Jitter wird auch im Zusammenhang mit reinen Sinusquellen als zeitliches Analog zum Phasenrauschen verwendet.

Joint Test Action Group
Joint Test Action Group bezeichnet den IEEE-Standard 1149.1, der eine Ansammlung von Verfahren zum Testen und Debuggen von elektronischer Hardware direkt in der Schaltung beschreibt.

monolithisch
aus einer einheitlichen, nicht trennbaren Einheit bestehend.

Reflexion
bezeichnet in der Physik das Zurückwerfen von Wellen (elektromagnetischen Wellen, Schallwellen, etc.) an einer Grenzfläche, an der sich der Wellenwiderstand oder der Brechungsindex des Mediums ändert.

Signalintegrität
ist die Eigenschaft eines Signals klar unterscheidbare Logikpegel zu fest definierten Zeitpunkten zu übertragen.

unidirektional
Unidirektional bedeutet “nur in eine Richtung”.

Index

Z

Tabellenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[1] ALFKE, Peter: *Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators*. Xilinx Application Note XAPP 052 (v1.1). 1996

[2] ALTERA: *Altera Product Catalog*. Product Catalog Version 11.1. 2012

[3] ANALOG DEVICES: *Dual 12-/14-/16-Bit 800 MSPS DAC with Low Power 32-Bit Complex NCO*. Data Sheet. 2008

[4] ANALOG DEVICES: *Converting Oszillator Phase Noise to Time Jitter*. TUTORIAL MT-008. 2009

[5] ANALOG DEVICES: *Correcting Imperfections in IQ Modulators to Improve RF Signal Fidelity*. Application Note AN-1039. 2009

[6] ANALOG DEVICES: *Microstrip and Stripline Design (MT-094)*. Application Notes. 2009

[7] ANALOG DEVICES: *Wireless Transmitter IQ Balance and Sideband Suppression*. Application Note AN-1100. 2010

[8] AVNET: *EXP Expansion Connector Specification*. User Guide Rev.1.4. 2007

[9] AVNET: *Xilinx Virtex-5 FXT PCI Express Development Kit User Guide*. User Guide Rev.1.1. 2011

[10] BEST, Roland E.: *Phase-Locked Loops (Design, Simulation and Applications)*. 5. Auflage. The McGraw-Hill Companies, 2003. – ISBN 0-07-141201-8

[11] FRAUNHOFER-INSTITUT FÜR ANGEWANDTE FESTKÖRPERPHYSIK (IAF): *Jahresbericht 2010*. Jahresbericht. 2011

[12] GARDNER, F. M.: Timing-Error Detectior for Sampled Receivers. In: *IEEE Trans. on Communications*, 1986

[13] GE, Ning ; LIU, Yuyu ; YANG, Huazhong ; WANG, Hui: Sigma-delta based clock recovery using on-chip PLL in FPGA. In: *2006 IEEE International Conference on Field Programmable Technology(2006)*, 2006, S. 135 – 140. – ISBN 0-7803-9728-2

[14] HALL, Stephen H. ; HALL, Garrett W. ; MCCALL, James A.: *High-speed digital system design*. 3. Auflage. John Wiley & Sons, Inc, 2000. – ISBN 0-471-36090-2

[15] HALLER, I. ; BARUCH, Z.F.: High-speed clock recovery for low-cost FPGAs. In: *Design, Automation & Test in Europe Conference & Exhibition*, 2010, S. 610 – 613. – ISBN 978-3-9810801-6-2

[16] HUFSCHMID, M.: *Mischer*. 2010

[17] IQBAL, Mahboob ; LEE, Jeongseon ; KIM, Kiseon: PERFORMANCE COMPARISON OF DIGITAL MODULATION SCHEMES WITH RESPECT TO PHASE NOISE SPECTRAL SHAPE. In: *IEEE Electrical and Computer Engineering, 2000 Canadian Conference*, 2000, S. 856 – 860. – ISBN 0-7803-5957

[18] JÜRGEN REICHARDT, Prof. D. rer. nat: *Lehrbuch Digitaltechnik*. 2. Auflage. Oldenbourg Wissenschaftsverlag GmbH, 2011. – ISBN 978-3-486-70680-2

[19] JÜRGEN REICHARDT, Prof. D. rer. nat ; SCHWARZ, Prof. Dr.-Ing. B.: *VHDL-Synthese*. 5. Auflage. Oldenbourg Wissenschaftsverlag GmbH, 2009. – ISBN 978-3-486-58987-0

[20] KALLFASS, Prof. Dr. Ing. I.: *MILLILINK Verbundantrag*. Verbundantrag. 2009

[21] KAMMEYER, Prof. Dr.-Ing. Karl-Dirk: *Nachrichtenübertragung*. 5. Auflage. Vieweg-Teubner Verlag, 1992. – ISBN 978-3-8348-0896-7

[22] KESTER, Walt: *High Speed System Applications*. 3. Auflage. Analog Devices, Inc., 2006. – ISBN 978-1-56619-909-4

[23] KIKKERT, J. C.: *RF Electronics*. AWR Corp. Application Notes. 2008

[24] LEHRSTUHL FÜR NACHRICHTENTECHNIK, TECHNISCHE UNIVERSITÄT MÜNCHEN: *Ein Lerntutorial für Nachrichtentechnik*. Mai 2012. – URL http://www.LNTwww.de

[25] NATIONAL SEMICONDUCTOR CORPORATION: *LVDS Owner's Manual*. Manuel. 2008

[26] ODYNIEC, Michal: *RF and microwave oscillator design*. 2. Auflage. ARTECH HOUSE, INC., 2002. – ISBN 1-58053-320-5

[27] PAWULA, R.F. ; S.O.RICE ; ROBERTS, J. H.: Distribution of the Phase Angle Between Two Vectors Perturbed by Gaussian Noise. In: *IEEE Transactions on Communications, Vol.Com-30, No.8*, 1982, S. 1828 – 1841. – ISBN 0090-6778/82/0800-1828

[28] ROHDE, Ulrich: *DIDITAL PLL FREQUENZY SYNTESIZERS Theory and Design*. 1. Auflage. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1983. – ISBN 0-13-214239-2

[29] SAWYER, Nick: *Data to Clock Phase Alignment*. Xilinx Application Note XAPP225 (v1.2). 2007

[30] SPINNLER, Bernhard ; HUBER, Johannes: Aufwandsgünstige inkohärente CPM-Übertragungsverfahren mit Vermeidung von Fehlerfortpflanzung durch Signalformung. In: *Kleinheubacher Tagung*, 1995

[31] TEXAS INSTRUMENTS: *Five/Ten Output Clock Generator/ Jitter Cleaner With Integrated Dual VCOs*. Data Sheet. 2008

[32] TEXAS INSTRUMENTS: *Dual Channel 14-/12-Bit, 250-/210-MSPS ADC With DDR LVDS and Parallel CMOS Outputs (SLAS635B)*. Data Sheet. 2009

[33] XILINX: *Using System Generator for Systematic HDL Design, Verification, and Validation*. White Paper WP283 v.1.0. 2008

[34] XILINX: *Virtex 5 FPGA Data Sheet DC and Switching Characteristics, User Guide 202*. Data Sheet. 2010

[35] XILINX: *Virtex-5 FPGA User Guide 190*. User Guide. 2010

[36] XILINX: *Extensible Processing Plattform*. Product Brief ZYNQ-7000 EPP. 2011

[37] XILINX: *LogiCORE IP DDS Compiler v4.0*. Product Specification DS558. 2011

[38] XILINX: *Virtex-5 Libraries Guide for HDL Designs, User Guide 621*. User Guide. 2011

[39] XILINX: *7 Series FPGAs Overview*. Advance Product Specification DS180 v1.11. 2012

Anhang

A. Berechnungen zur Auswahl der Modulationsarten

Listing A.1: mod_calc.m

```
%File: mod_calc.m
%Autor: Sönke Appel
%Date: 07.01.2012

%Symbolvectoren der veschiedenen Modulationsarten
cpm = [1.*exp(i*pi*2*1/3) 1.*exp(i*pi*2*2/3) 1.*exp(i*pi*2*3/3)];
ask4 = [1 1/3 -1/3 -1];
ask4_abs = [0 1/3 2/3 1];
psk8 = [1.*exp(i*pi*2*1/8) 1.*exp(i*pi*2*2/8) 1.*exp(i*pi*2*3/8)
    1.*exp(i*pi*2*4/8) 1.*exp(i*pi*2*5/8) 1.*exp(i*pi*2*6/8) 1.*exp
    (i*pi*2*7/8) 1.*exp(i*pi*2*8/8)];
ask_psk_16 = [1 -1 1i -1i 3/5 -3/5 3/5i -3/5i 3/5+3/5i -3/5+3/5i
    3/5-3/5i -3/5-3/5i 1/5+1/5i 1/5-1/5i -1/5+1/5i -1/5-1/5i];
qam16 =[1+1i 1-1i -1+1i -1-1i 1/3+1/3i 1/3-1/3i -1/3-1/3i -1/3+1/3
    i 1-1/3i 1+1/3i -1+1/3i -1-1/3i 1/3+1i -1/3+1i 1/3-1i -1/3-1i];
bpsk = [1 -1];
ook = [0 1];
qpsk_null = [-1 1 1i -1i];
qpsk_pi_viertel = [1.*exp(i*pi*2*1/8) 1.*exp(i*pi*2*3/8) 1.*exp(i*
    pi*2*5/8) 1.*exp(i*pi*2*7/8)];

%Hier muss die Modulationsart eingetragen werden,
%welche berechnet werden soll.
data_mod = psk8

%Berechnungd der Kenneigenschaften
PAPR = max(abs(data_mod).^2).*length(data_mod)./sum(abs(data_mod)
    .^2)
data_mod_2 = abs(data_mod);
data_mod_2(~data_mod_2) = nan;
PMPR = max(abs(data_mod).^2)./min(data_mod_2.^2)
clear space;
```

```
m=1;
for k=1:(length(data_mod))
    for l=k+1:length(data_mod)
        space(m) = data_mod(l)-data_mod(k);
        m = m+1;
    end
end
minimums_space = min(abs(space));
PMSPR = max(abs(data_mod).^2)./minimums_space.^2
```

Listing A.2: normalverteilung.m

```
%File: normalverteilung.m
%Autor: Sönke Appel
%Date: 07.01.2012
function y = normalverteilung(x, standardabweichung)

y = 1./standardabweichung./sqrt(2.*pi).*exp(-1./2.*(x./
   standardabweichung).^2);
```

Listing A.3: Q_fuction.m

```
%File: Q_fuction.m
%Autor: Sönke Appel
%Date: 09.01.2012
function y = Q_fuction(x)

y = 1./2.*erfc(x./sqrt(2));
```

Listing A.4: I_e.m

```
%File: I_e.m
%Autor: Sönke Appel
%Date: 09.01.2012
%Ref: Distribution of the Phase
%Angle Between Two Vectors Perturbed by
%Gaussian Noise, R. F. PAWULA, S. 0. RICE,
%AND J. H. ROBERTS
function y = I_e(k,x)
step = 0.001;
winkel = [0:0.001:pi];
y = 1./sqrt(1-k.^2)-1./pi.*sum(exp(-x.*(1-k.*cos(winkel))./(1-k.*
   cos(winkel))).*step;
```

Listing A.5: ber_calc.m

```
%File: ber_calc.m
%Autor: Sönke Appel
%Date: 09.01.2012

EbN0 = [0:0.01:100];

BER_BPSK = 0.5 .* erfc(sqrt(EbN0)); %Kammeyer seite374 QPSK Seite
    378
BER_PSK8 = 1/3.*erfc(sqrt(EbN0.*3).*sin(pi/8)).*(1-1/2.*erfc(sqrt(
    EbN0.*3).*sin(3*pi/8)))+1/3.*erfc(sqrt(EbN0.*3).*sin(3*pi/8));
    % Kammeyer Seite 380
BER_DBPSK = 2.*BER_BPSK; % Kammeyer Seite 385
BER_QAM16 = 2/log2(16).*(1-1./sqrt(16)).*erfc(sqrt(3.*log2(16).*
    EbN0/2/(16-1))); % Kammeyer Seite 387
M = 4;
BER_OOK = Q_fuction(sqrt(EbN0)); % www.LNTwww.de
BER_ASK_pos = Q_fuction(sqrt(EbN0./3)); % www.LNTwww.de
A=sqrt(3/2/(M-1));
BER_ASK4 = 2./log2(M).*1/sqrt(M).*((sqrt(M)-2).*erfc(sqrt(A^2.*
    EbN0))+1.*erfc(sqrt(A^2.*EbN0)));% Kammeyer Seite 386
x = 10.*log10(EbN0);
semilogy(x, BER_BPSK ,x,BER_PSK8,x,BER_DBPSK,x,BER_QAM16,x,
    BER_ASK4,x,BER_ASK_pos,x,BER_OOK)
grid on;
xlabel('E_b/N_0 [dB]');
ylabel('BER');
AXIS([0 20 10^-12 10^-0.0001])
legend('BPSK, QPSK', '8-PSK','DBPSK', '16-QAM', '4-ASK mit neg.
    Werten', '4-ASK ohne neg. Werten', 'OOK');
```

Listing A.6: phasenoise_ber.m

```
%File: phasenoise_ber.m
%Autor: Sönke Appel
%Date: 09.01.2012
%Ref.: PERFORMANCE COMPARISON OF DIGITAL
%MODULATION SCHEMES WITH RESPECT TO
%PHASE NOISE SPECTRAL SHAPE
%,Mahboob Iqbal, Jeongseon L,ee,  and Kiseon Kim

clear all;
EbN0_1_dB = 20;
EbN0_1 = 10^(EbN0_1_dB./10);
```

```
12 standardabweichung_bereich = logspace(-2,0.5);
13 l=1;
14 for standardabweichung = logspace(-2,0.5);
15 step = 0.0005; %bestimmt die genauigkeit der Berechnung
16 phsenoffset = [-10*pi:step:10*pi];
17 warscheinlichkeit_phasenoffset = normalverteilung(phsenoffset,
       standardabweichung);
18
19 %Fehlerwarscheinlichkeiten in Abhängigkeit der des Phasenoffsets
20 BER_BPSK_phaseoffset = Q_fuction(sqrt(2.*EbN0_1).*cos(phsenoffset)
       );
21 BER_QPSK_phaseoffset = 0.5.*(Q_fuction(sqrt(2.*EbN0_1).*cos(
       phsenoffset)+sin(phsenoffset))+ Q_fuction(sqrt(2.*EbN0_1).*cos(
       phsenoffset)-sin(phsenoffset)));
22 BER_PSK8_phaseoffset = 1./2./log2(8).*(erfc(sqrt(log2(8).*EbN0_1)
       .*sin((pi./8)-phsenoffset))+erfc(sqrt(log2(8).*EbN0_1).*sin((pi
       ./8)+phsenoffset)));
23
24 %Fehlerwarscheinlichkeiten in abhänigkeit des Phasenrauschens
25 BER_PSK8_var(l) = sum(warscheinlichkeit_phasenoffset.*
       BER_PSK8_phaseoffset.*step);
26 BER_QPSK_var(l) = sum(warscheinlichkeit_phasenoffset.*
       BER_QPSK_phaseoffset.*step);
27 BER_BPSK_var(l) = sum(warscheinlichkeit_phasenoffset.*
       BER_BPSK_phaseoffset.*step);
28
29 %gesonderte Berechnung der Fehlerwarscheinlichkeit von DBPSK
30 for o = 1:length(phsenoffset)
31     if abs(phsenoffset(o))<(pi/2)
32         BER_DPSK2_phaseoffset(o) = 0.5.*(1-cos(phsenoffset(o)).*
               I_e(sin(abs(phsenoffset(o))),EbN0_1));
33     elseif ((phsenoffset(o)>(pi/2)) && (phsenoffset(o)<pi))
34         BER_DPSK2_phaseoffset(o) = -0.5.*(1-cos( pi - phsenoffset(
               o) ).*I_e(sin(abs(pi-phsenoffset(o))),EbN0_1));
35     elseif ((phsenoffset(o)>(-pi)) && (phsenoffset(o)<(-pi/2)))
36         BER_DPSK2_phaseoffset(o) = -0.5.*(1-cos( pi + phsenoffset(
               o) ).*I_e(sin(abs(pi+phsenoffset(o))),EbN0_1));
37     else
38         BER_DPSK2_phaseoffset(o) = 0;
39     end
40 end
41 prob_pi_gross = erfc(sqrt((pi.^2)./8./(standardabweichung.^2)));
42 BER_DPSK2_var(l) = sum(warscheinlichkeit_phasenoffset.*
       BER_DPSK2_phaseoffset.*step)+prob_pi_gross;
```

```
43
44 l=l+1;
45 end
46
47 %Fehlerwarscheinlichkeiten der Phasenresistenten Modulationsarten
48 BER_OOK = Q_fuction(sqrt(EbN0_1));
49 BER_OOK_var(1:length(standardabweichung_bereich)) = BER_OOK;
50 BER_ASK_pos = Q_fuction(sqrt(EbN0_1./3));
51 BER_ASK_pos_var(1:length(standardabweichung_bereich)) =
       BER_ASK_pos;
52
53 varianz = standardabweichung_bereich.^2;
54 loglog(varianz,BER_BPSK_var,varianz,BER_QPSK_var,varianz,
       BER_OOK_var,varianz,BER_ASK_pos_var,varianz,BER_PSK8_var,
       varianz,BER_DPSK2_var)
55 grid on;
56 xlabel('Varianz \sigma^2');
57 ylabel('BER');
58 title(['E_b/N_0 = ',num2str(EbN0_1_dB)]);
59 legend('BPSK', 'QPSK', 'OOK','4-ASK ohne neg. Werten','8-PSK','
       DBPSK');
```

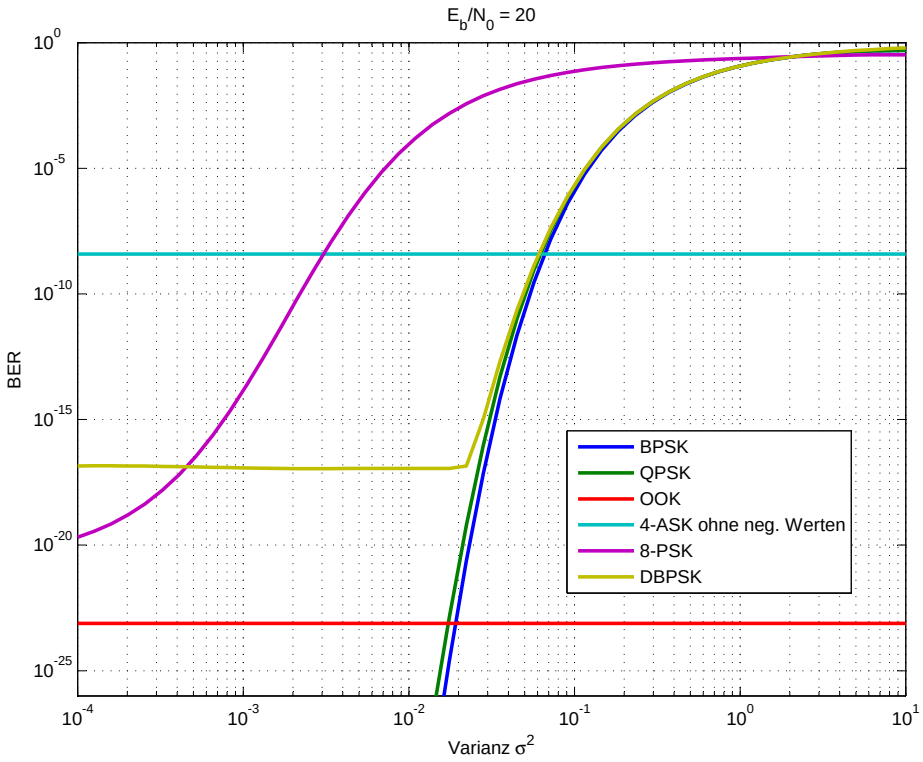


Bild A.1.: Bitfehlerwahrscheinlichkeit in Abhängigkeit der Phasenfehlervarianz bei $E_b/N_0 = 20\,\mathrm{dB}$

B. DAC-Interface

Listing B.1: DAC_INTERFACE.vhd

```
1  ----------------------------------------------
2  -- Company: Siemens AG
3  -- Engineer: Sönke Appel
4  --
5  -- Create Date:    12:44:57 11/17/2011
6  -- Design Name: DAC_INTERFACE
7  -- Module Name:    DAC_INTERFACE - Behavioral
8  -- Project Name: MILLILINK
9  ----------------------------------------------
10 library IEEE;
11 use IEEE.STD_LOGIC_1164.ALL;
12
13 library UNISIM;
14 use UNISIM.vcomponents.all;
15 use work.DELAY_VALUE.all;
16
17
18 entity DAC_INTERFACE is
19    generic (DATA_1_DELAY: type_delay_value:= (others => 0);
20             DATA_2_DELAY: type_delay_value:= (others => 0);
21             DATA_CLK_DELAY: integer range 0 to 63:=0);
22     Port ( DATA_1_OUT : out  STD_LOGIC_VECTOR (15 downto 0);
23            DATA_2_OUT : out  STD_LOGIC_VECTOR (15 downto 0);
24            DATA_CLK_P_IN : in  STD_LOGIC;
25            DATA_CLK_N_IN : in  STD_LOGIC;
26            SYSCLK_IN : in  STD_LOGIC;
27            SYSCLK_OUT : out STD_LOGIC;
28            RESET_IN : in  STD_LOGIC;
29            DATA_1_IN : in STD_LOGIC_VECTOR (15 downto 0);
30            DATA_2_IN : in STD_LOGIC_VECTOR (15 downto 0);
31            DATA_CLK_OUT : out STD_LOGIC;
32            SYS_CLK_DCM_LOCKED_OUT : out STD_LOGIC;
33            DATA_CLK_DCM_LOCKED_OUT : out STD_LOGIC);
34
```

```
35 end DAC_INTERFACE;
36
37 architecture Behavioral of DAC_INTERFACE is
38
39 signal DATA_1_OBUF_SIG : STD_LOGIC_VECTOR (15 downto 0);
40 signal DATA_2_OBUF_SIG : STD_LOGIC_VECTOR (15 downto 0);
41 signal DATA_1_ODDR_SIG : STD_LOGIC_VECTOR (15 downto 0);
42 signal DATA_2_ODDR_SIG : STD_LOGIC_VECTOR (15 downto 0);
43
44 component DOUBLE_CLK
45    port ( CLKIN_IN        : in    std_logic;
46           RST_IN          : in    std_logic;
47           CLK0_OUT        : out   std_logic;
48           CLK2X_OUT       : out   std_logic;
49           LOCKED_OUT      : out   std_logic);
50 end component;
51
52 signal RESET_DOUBLE_CLK_1_SIG: std_logic;
53 signal LOCKED_DOUBLE_CLK_1_SIG: std_logic;
54 signal CLK_200_SIG: std_logic;
55 signal CLK_100_SIG: std_logic;
56 signal IODELAY_RESET_1_SIG: std_logic;
57 signal IODELAY_RESET_2_SIG: std_logic;
58 signal IODELAY_RESET_SIG: std_logic;
59 signal IDELAYCTRL_RESET_SIG: std_logic;
60
61 signal SYSCLK_SIG: std_logic;
62
63 signal DATA_CLK_IBUFG_SIG: std_logic;
64 signal DATA_CLK_IODELAY_SIG: std_logic;
65
66 COMPONENT DAC_CLK_DCM
67    PORT(
68       CLKIN_IN : IN std_logic;
69       RST_IN : IN std_logic;
70       CLK0_OUT : OUT std_logic;
71       LOCKED_OUT : OUT std_logic
72       );
73 END COMPONENT;
74
75 signal DATA_CLK_DCM_SIG: std_logic;
76 signal RESET_DAC_CLK_DCM_SIG: std_logic;
77 signal LOCKED_DAC_CLK_DCM_SIG: std_logic;
78
```

```
79  signal RESET_WAIT_1_SIG: std_logic_vector(12 downto 0);
80  signal RESET_ODDR_SIG: std_logic;
81
82
83  begin
84
85  P_RESET_ROUTINE: process(RESET_IN, CLK_100_SIG,
        LOCKED_DOUBLE_CLK_1_SIG, RESET_WAIT_1_SIG)
86  begin
87
88  --Reset-Routine realisiert durch ein Schieberegister
89  if RESET_IN = '1' then
90     RESET_DOUBLE_CLK_1_SIG <= '1';
91     RESET_WAIT_1_SIG <= "1111" & "1111" & "1111" & '1';
92  else
93     RESET_DOUBLE_CLK_1_SIG <= '0';
94     if LOCKED_DOUBLE_CLK_1_SIG = '1' then
95        if CLK_100_SIG = '1' and CLK_100_SIG'event then
96           RESET_WAIT_1_SIG(12 downto 0) <= RESET_WAIT_1_SIG(11
                downto 0) & '0';
97        end if;
98     end if;
99  end if;
100
101 IDELAYCTRL_RESET_SIG <= RESET_WAIT_1_SIG(5);
102 RESET_DAC_CLK_DCM_SIG <= RESET_WAIT_1_SIG(11);
103 RESET_ODDR_SIG <= RESET_WAIT_1_SIG(12);
104
105 end Process P_RESET_ROUTINE;
106
107 --Instanzierung der Primitives. Nähere Informationen zu den
        einzelnen
108 -- Primitives im Xilinx UG621
109 IBUFGDS_DATA_CLK: IBUFGDS
110 generic map (
111    DIFF_TERM =>  TRUE,
112    IOSTANDARD => "LVDS_25")
113 port map (
114    O =>  DATA_CLK_IBUFG_SIG,
115    I =>  DATA_CLK_P_IN,
116    IB => DATA_CLK_N_IN);
117
118
119
```

```
120 IODELAY_DATA_CLK: IODELAY
121    generic map(
122       DELAY_SRC => "I",
123       HIGH_PERFORMANCE_MODE => TRUE,
124       IDELAY_TYPE => "FIXED",
125       IDELAY_VALUE => DATA_CLK_DELAY,
126       ODELAY_VALUE => 0,
127       REFCLK_FREQUENCY => 200.0,
128       SIGNAL_PATTERN => "CLOCK")
129    port map (
130       DATAOUT => DATA_CLK_IODELAY_SIG,
131       C => '0',
132       CE => '0',
133       DATAIN => '0',
134       IDATAIN => DATA_CLK_IBUFG_SIG,
135       INC => '0',
136       ODATAIN => '0',
137       RST => IODELAY_RESET_SIG,
138       T => '0');
139
140
141
142
143 DAC_CLK_DCM_1: DAC_CLK_DCM PORT MAP(
144       CLKIN_IN => DATA_CLK_IODELAY_SIG,
145       RST_IN => RESET_DAC_CLK_DCM_SIG,
146       CLK0_OUT => DATA_CLK_DCM_SIG,
147       LOCKED_OUT => LOCKED_DAC_CLK_DCM_SIG);
148
149 DATA_CLK_OUT <= DATA_CLK_DCM_SIG;
150 DATA_CLK_DCM_LOCKED_OUT <= LOCKED_DAC_CLK_DCM_SIG;
151
152
153 IBUFG_SYS_CLK: IBUFG
154 generic map (
155    IOSTANDARD => "DEFAULT")
156 port map (
157    O => SYSCLK_SIG,
158    I => SYSCLK_IN);
159
160
161 DOUBLE_CLK_1: DOUBLE_CLK port map(
162           CLKIN_IN        => SYSCLK_SIG,
163           RST_IN          => RESET_DOUBLE_CLK_1_SIG,
```

```
164             CLK0_OUT         => CLK_100_SIG,
165             CLK2X_OUT        => CLK_200_SIG,
166             LOCKED_OUT       => LOCKED_DOUBLE_CLK_1_SIG);
167 SYSCLK_OUT <=  CLK_100_SIG;
168 SYS_CLK_DCM_LOCKED_OUT <= LOCKED_DOUBLE_CLK_1_SIG;
169
170
171 OBUF_GEN: for N in 15 downto 0 generate
172    OBUF_DATA_1: OBUF
173    generic map (
174       DRIVE => 12,
175       IOSTANDARD => "LVCMOS25",
176       SLEW => "SLOW")
177    port map (
178       O => DATA_1_OUT(N),
179       I => DATA_1_OBUF_SIG(N));
180
181    OBUF_DATA_2: OBUF
182    generic map (
183       DRIVE => 12,
184       IOSTANDARD => "LVCMOS25",
185       SLEW => "SLOW")
186    port map (
187       O => DATA_2_OUT(N),
188       I => DATA_2_OBUF_SIG(N));
189 end generate;
190
191 IODELAY_GEN: for N in 15 downto 0 generate
192    IODELAY_DATA_1: IODELAY
193    generic map(
194       DELAY_SRC => "O",
195       HIGH_PERFORMANCE_MODE => FALSE,
196       IDELAY_TYPE => "FIXED",
197       IDELAY_VALUE => 0,
198       ODELAY_VALUE => DATA_1_DELAY(N),
199       REFCLK_FREQUENCY => 200.0,
200       SIGNAL_PATTERN => "DATA")
201    port map (
202       DATAOUT => DATA_1_OBUF_SIG(N),
203       C => '0',
204       CE => '0',
205       DATAIN => '0',
206       IDATAIN => '0',
207       INC => '0',
```

```
208         ODATAIN => DATA_1_ODDR_SIG(N),
209         RST => IODELAY_RESET_SIG,
210         T => '0');
211
212     IODELAY_DATA_2: IODELAY
213     generic map(
214         DELAY_SRC => "O",
215         HIGH_PERFORMANCE_MODE => FALSE,
216         IDELAY_TYPE => "FIXED",
217         IDELAY_VALUE => 0,
218         ODELAY_VALUE => DATA_2_DELAY(N),
219         REFCLK_FREQUENCY => 200.0,
220         SIGNAL_PATTERN => "DATA")
221     port map (
222         DATAOUT => DATA_2_OBUF_SIG(N),
223         C => '0',
224         CE => '0',
225         DATAIN => '0',
226         IDATAIN => '0',
227         INC => '0',
228         ODATAIN => DATA_2_ODDR_SIG(N),
229         RST => IODELAY_RESET_SIG,
230         T => '0');
231 end generate;
232
233 ODDR_GEN: for N in 15 downto 0 generate
234     ODDR_DATA_1: ODDR
235         generic map(
236             DDR_CLK_EDGE => "SAME_EDGE",
237             INIT => '0',
238             SRTYPE => "ASYNC")
239         port map(
240             Q => DATA_1_ODDR_SIG(N),
241             C => DATA_CLK_DCM_SIG,
242             CE => '1',
243             D1 => DATA_1_IN(N),
244             D2 => DATA_1_IN(N),
245             R => RESET_ODDR_SIG,
246             S => '0');
247     ODDR_DATA_2: ODDR
248         generic map(
249             DDR_CLK_EDGE => "SAME_EDGE",
250             INIT => '0',
251             SRTYPE => "ASYNC")
```

```
252         port map(
253             Q => DATA_2_ODDR_SIG(N),
254             C => DATA_CLK_DCM_SIG,
255             CE => '1',
256             D1 => DATA_2_IN(N),
257             D2 => DATA_2_IN(N),
258             R => RESET_ODDR_SIG,
259             S => '0');
260 end generate;
261
262
263 IDELAYCTRL_1: IDELAYCTRL
264    port map(
265    RDY => IODELAY_RESET_1_SIG,
266    REFCLK => CLK_200_SIG,
267    RST => IDELAYCTRL_RESET_SIG);
268
269 IDELAYCTRL_2: IDELAYCTRL
270    port map(
271    RDY => IODELAY_RESET_2_SIG,
272    REFCLK => CLK_200_SIG,
273    RST => IDELAYCTRL_RESET_SIG);
274
275 IODELAY_RESET_SIG <= IODELAY_RESET_1_SIG and IODELAY_RESET_2_SIG;
276
277 end Behavioral;
```

B.1. Berechnung von Delay Taps für IODELAY-Primitives

Listing B.2: ODELAY_CALC.m

```
1  %-----------------------------------------------
2  % Autor:       Sönke Appel
3  % Company:     Siemens AG
4  % File:        ODELAY_CALC.m
5  % Projekt:     MILLILINK
6  %-----------------------------------------------
7
8  % Konstanten Deklaration
9  c=299792458; %m/s
10 Delay_Tap = 78; %ps
11 Er=4.3;
```

```
12
13 %Deklaration des Instanzenpfad
14 INSTANT_PFAD = 'Inst_DAC_INTERFACE/IODELAY_GEN';
15 INSTANT_NAME_1 = 'IODELAY_DATA_1';
16 INSTANT_NAME_2 = 'IODELAY_DATA_2';
17
18 %Deklaration der Signalpfadlängen P1 (mm)
19 P1(1)=80;
20 P1(2)=80;
21 P1(3)=80;
22 P1(4)=80;
23 P1(5)=80;
24 P1(6)=80;
25 P1(7)=80;
26 P1(8)=79;
27 P1(9)=143;
28 P1(10)=154;
29 P1(11)=159;
30 P1(12)=164;
31 P1(13)=168;
32 P1(14)=172;
33 P1(15)=177;
34 P1(16)=183;
35
36 %Deklaration der Signalpfadlängen P2 (mm)
37 P2(1)=76;
38 P2(2)=76;
39 P2(3)=76;
40 P2(4)=76;
41 P2(5)=76;
42 P2(6)=76;
43 P2(7)=76;
44 P2(8)=74;
45 P2(9)=136;
46 P2(10)=147;
47 P2(11)=150;
48 P2(12)=156;
49 P2(13)=160;
50 P2(14)=165;
51 P2(15)=169;
52 P2(16)=174;
53
54 %Ermittlung der längsten Datenpfadlänge (mm)
55 maximum_lenght = max(max(P1),max(P2));
```

```
56
57 %Berechnung der relativen Datenpfadlängen
58 %zur längsten Datenpfadläng (m)
59 PP1 = (maximum_lenght-P1)/1000;
60 PP2 = (maximum_lenght-P2)/1000;
61
62 %Berechnung der relativen Zeitdifferenzen
63 %aller Datenpfade (s)
64 T1=PP1*sqrt(Er)/c;
65 T2=PP2*sqrt(Er)/c;
66
67 %Umrechnung in (ps)
68 TT1 = T1*1000*1000*1000*1000;
69 TT2 = T2*1000*1000*1000*1000;
70
71 %Berechnung der Anzahl der Delay Taps
72 DELAY_1 = round(TT1/Delay_Tap);
73 DELAY_2 = round(TT2/Delay_Tap);
74
75 %Erzeugung des UCF-Files
76 fileID = fopen('IODELAY.ucf', 'w');
77 for k=1:length(DELAY_1)
78
79     fprintf(fileID, 'INST␣"%s<%d>.%s"␣ODELAY_VALUE␣=␣%d;␣#␣\n',
          INSTANT_PFAD, k-1, INSTANT_NAME_1, DELAY_1(k) );
80 end
81 fprintf(fileID,'\n');
82 for k=1:length(DELAY_1)
83
84     fprintf(fileID, 'INST␣"%s<%d>.%s"␣ODELAY_VALUE␣=␣%d;␣#␣\n',
          INSTANT_PFAD, k-1, INSTANT_NAME_2, DELAY_2(k) );
85 end
86
87 fclose(fileID);
```

B.2. Sender-FPGA UCF-File

Listing B.3: TRANSMITTER_FX100T.ucf

```
1 ##################################################
2 # Autor    Sönke Appel
3 # Company  Siemens AG
```

```
# Projekt  Millilink
# File     TRANSMITTER.ucf
####################################################

# Board clock 100 MHz
NET "SYSCLK_IN"         LOC=G15   | IOSTANDARD=LVCMOS25;
NET "SYSCLK_IN"         TNM_NET = SYSCLK_IN;
TIMESPEC TS_sysclk_in= PERIOD "SYSCLK_IN" 100 Mhz HIGH 50%;

#On-Board-Schalter
#NET "SWITCH_IN<0>" LOC=C20; #
#NET "SWITCH_IN<1>" LOC=B20; #
#NET "SWITCH_IN<2>" LOC=B21; #
#NET "SWITCH_IN<3>" LOC=A21; #
#NET "SWITCH_IN<4>" LOC=C19; #
#NET "SWITCH_IN<5>" LOC=C18; #
#NET "SWITCH_IN<6>" LOC=C22; #
#NET "SWITCH_IN<7>" LOC=B22; #

#On-Board-LED
NET "LED_OUT<0>" LOC=B16;
NET "LED_OUT<1>" LOC=B15;
#NET "LED_OUT<2>" LOC=AN34;
#NET "LED_OUT<3>" LOC=AN33;
#NET "LED_OUT<4>" LOC=AN32;
#NET "LED_OUT<5>" LOC=AP32;
#NET "LED_OUT<6>" LOC=AG15;
#NET "LED_OUT<7>" LOC=AG16;

#On-Board-Push-Button
NET "PB_1_IN" LOC=D14; #RESET_IN

#DAC Daten-Clock
NET "DATA_CLK_P_IN" LOC=AG21;
NET "DATA_CLK_N_IN" LOC=AG20;
NET "DATA_CLK_P_IN"       TNM_NET = DATA_CLK_P_IN;
TIMESPEC TS_DATA_CLK_P_IN= PERIOD "DATA_CLK_P_IN" 80 Mhz HIGH 50%;
NET "DATA_CLK_N_IN"       TNM_NET = DATA_CLK_N_IN;
TIMESPEC TS_DATA_CLK_N_IN= PERIOD "DATA_CLK_N_IN" 80 Mhz HIGH 50%;

#Datenleitungen
NET "DATA_1_OUT<0>"  LOC=AJ15 | Drive = 6;
NET "DATA_1_OUT<1>"  LOC=AK14 | Drive = 6;
NET "DATA_1_OUT<2>"  LOC=AK12 | Drive = 6;
```

```
48 NET "DATA_1_OUT<3>"  LOC=AL13 | Drive = 6;
49 NET "DATA_1_OUT<4>"  LOC=AL15 | Drive = 6;
50 NET "DATA_1_OUT<5>"  LOC=AM15 | Drive = 6;
51 NET "DATA_1_OUT<6>"  LOC=AM16 | Drive = 6;
52 NET "DATA_1_OUT<7>"  LOC=AK16 | Drive = 6;
53 NET "DATA_1_OUT<8>"  LOC=AK18 | Drive = 6;
54 NET "DATA_1_OUT<9>"  LOC=AL20 | Drive = 6;
55 NET "DATA_1_OUT<10>" LOC=AJ19 | Drive = 6;
56 NET "DATA_1_OUT<11>" LOC=AK21 | Drive = 6;
57 NET "DATA_1_OUT<12>" LOC=AJ22 | Drive = 6;
58 NET "DATA_1_OUT<13>" LOC=AK23 | Drive = 6;
59 NET "DATA_1_OUT<14>" LOC=AH23 | Drive = 6;
60 NET "DATA_1_OUT<15>" LOC=AH24 | Drive = 6;
61
62
63 NET "DATA_2_OUT<0>"  LOC=AJ12 | Drive = 6;
64 NET "DATA_2_OUT<1>"  LOC=AJ14 | Drive = 6;
65 NET "DATA_2_OUT<2>"  LOC=AK13 | Drive = 6;
66 NET "DATA_2_OUT<3>"  LOC=AL14 | Drive = 6;
67 NET "DATA_2_OUT<4>"  LOC=AJ17 | Drive = 6;
68 NET "DATA_2_OUT<5>"  LOC=AL16 | Drive = 6;
69 NET "DATA_2_OUT<6>"  LOC=AK17 | Drive = 6;
70 NET "DATA_2_OUT<7>"  LOC=AJ16 | Drive = 6;
71 NET "DATA_2_OUT<8>"  LOC=AL18 | Drive = 6;
72 NET "DATA_2_OUT<9>"  LOC=AL19 | Drive = 6;
73 NET "DATA_2_OUT<10>" LOC=AK19 | Drive = 6;
74 NET "DATA_2_OUT<11>" LOC=AL21 | Drive = 6;
75 NET "DATA_2_OUT<12>" LOC=AK22 | Drive = 6;
76 NET "DATA_2_OUT<13>" LOC=AL23 | Drive = 6;
77 NET "DATA_2_OUT<14>" LOC=AK24 | Drive = 6;
78 NET "DATA_2_OUT<15>" LOC=AJ24 | Drive = 6;
79
80 #On-Board-SMA-Connector
81 #NET "SMA_CONNECTOR_J24" LOC=E4;
82 #NET "SMA_CONNECTOR_J25" LOC=D4;
83 NET "SMA_CONNECTOR_J26" LOC=AF18;
84 NET "SMA_CONNECTOR_J27" LOC=AE18;
85
86 #Festlegung der Positionen Zeitkritischer Primitives
87 INST "Inst_DAC_INTERFACE/IDELAYCTRL_1"                    LOC = "
      IDELAYCTRL_X1Y0";
88 INST "Inst_DAC_INTERFACE/IDELAYCTRL_2"                    LOC = "
      IDELAYCTRL_X1Y2";
```

```
89  INST "Inst_DAC_INTERFACE/DOUBLE_CLK_1/CLK0_BUFG_INST"        LOC =
       "BUFGCTRL_X0Y30";
90  INST "Inst_DAC_INTERFACE/DOUBLE_CLK_1/CLK2X_BUFG_INST"       LOC =
       "BUFGCTRL_X0Y31";
91  INST "Inst_DAC_INTERFACE/DOUBLE_CLK_1/DCM_ADV_INST"          LOC =
       "DCM_ADV_X0Y11";
92  INST "Inst_DAC_INTERFACE/DAC_CLK_DCM_1/CLK0_BUFG_INST"          LOC
        = "BUFGCTRL_X0Y0";
93  INST "Inst_DAC_INTERFACE/DAC_CLK_DCM_1/DCM_ADV_INST"            LOC
        = "DCM_ADV_X0Y0";
94
95  #Eistellungen der ODELAY Taps der IODELAY-Primitives
96  INST "Inst_DAC_INTERFACE/IODELAY_GEN<0>.IODELAY_DATA_1"
       ODELAY_VALUE = 9; #
97  INST "Inst_DAC_INTERFACE/IODELAY_GEN<1>.IODELAY_DATA_1"
       ODELAY_VALUE = 9; #
98  INST "Inst_DAC_INTERFACE/IODELAY_GEN<2>.IODELAY_DATA_1"
       ODELAY_VALUE = 9; #
99  INST "Inst_DAC_INTERFACE/IODELAY_GEN<3>.IODELAY_DATA_1"
       ODELAY_VALUE = 9; #
100 INST "Inst_DAC_INTERFACE/IODELAY_GEN<4>.IODELAY_DATA_1"
       ODELAY_VALUE = 9; #
101 INST "Inst_DAC_INTERFACE/IODELAY_GEN<5>.IODELAY_DATA_1"
       ODELAY_VALUE = 9; #
102 INST "Inst_DAC_INTERFACE/IODELAY_GEN<6>.IODELAY_DATA_1"
       ODELAY_VALUE = 9; #
103 INST "Inst_DAC_INTERFACE/IODELAY_GEN<7>.IODELAY_DATA_1"
       ODELAY_VALUE = 9; #
104 INST "Inst_DAC_INTERFACE/IODELAY_GEN<8>.IODELAY_DATA_1"
       ODELAY_VALUE = 4; #
105 INST "Inst_DAC_INTERFACE/IODELAY_GEN<9>.IODELAY_DATA_1"
       ODELAY_VALUE = 3; #
106 INST "Inst_DAC_INTERFACE/IODELAY_GEN<10>.IODELAY_DATA_1"
       ODELAY_VALUE = 2; #
107 INST "Inst_DAC_INTERFACE/IODELAY_GEN<11>.IODELAY_DATA_1"
       ODELAY_VALUE = 2; #
108 INST "Inst_DAC_INTERFACE/IODELAY_GEN<12>.IODELAY_DATA_1"
       ODELAY_VALUE = 1; #
109 INST "Inst_DAC_INTERFACE/IODELAY_GEN<13>.IODELAY_DATA_1"
       ODELAY_VALUE = 1; #
110 INST "Inst_DAC_INTERFACE/IODELAY_GEN<14>.IODELAY_DATA_1"
       ODELAY_VALUE = 1; #
111 INST "Inst_DAC_INTERFACE/IODELAY_GEN<15>.IODELAY_DATA_1"
       ODELAY_VALUE = 0; #
```

```
112
113 INST "Inst_DAC_INTERFACE/IODELAY_GEN<0>.IODELAY_DATA_2"
       ODELAY_VALUE = 9; #
114 INST "Inst_DAC_INTERFACE/IODELAY_GEN<1>.IODELAY_DATA_2"
       ODELAY_VALUE = 9; #
115 INST "Inst_DAC_INTERFACE/IODELAY_GEN<2>.IODELAY_DATA_2"
       ODELAY_VALUE = 9; #
116 INST "Inst_DAC_INTERFACE/IODELAY_GEN<3>.IODELAY_DATA_2"
       ODELAY_VALUE = 9; #
117 INST "Inst_DAC_INTERFACE/IODELAY_GEN<4>.IODELAY_DATA_2"
       ODELAY_VALUE = 9; #
118 INST "Inst_DAC_INTERFACE/IODELAY_GEN<5>.IODELAY_DATA_2"
       ODELAY_VALUE = 9; #
119 INST "Inst_DAC_INTERFACE/IODELAY_GEN<6>.IODELAY_DATA_2"
       ODELAY_VALUE = 9; #
120 INST "Inst_DAC_INTERFACE/IODELAY_GEN<7>.IODELAY_DATA_2"
       ODELAY_VALUE = 10; #
121 INST "Inst_DAC_INTERFACE/IODELAY_GEN<8>.IODELAY_DATA_2"
       ODELAY_VALUE = 4; #
122 INST "Inst_DAC_INTERFACE/IODELAY_GEN<9>.IODELAY_DATA_2"
       ODELAY_VALUE = 3; #
123 INST "Inst_DAC_INTERFACE/IODELAY_GEN<10>.IODELAY_DATA_2"
       ODELAY_VALUE = 3; #
124 INST "Inst_DAC_INTERFACE/IODELAY_GEN<11>.IODELAY_DATA_2"
       ODELAY_VALUE = 2; #
125 INST "Inst_DAC_INTERFACE/IODELAY_GEN<12>.IODELAY_DATA_2"
       ODELAY_VALUE = 2; #
126 INST "Inst_DAC_INTERFACE/IODELAY_GEN<13>.IODELAY_DATA_2"
       ODELAY_VALUE = 2; #
127 INST "Inst_DAC_INTERFACE/IODELAY_GEN<14>.IODELAY_DATA_2"
       ODELAY_VALUE = 1; #
128 INST "Inst_DAC_INTERFACE/IODELAY_GEN<15>.IODELAY_DATA_2"
       ODELAY_VALUE = 1; #
129
130 #Eistellungen der ODELAY Taps der IODELAY-Primitives des Daten-
       Clocks
131 INST "Inst_DAC_INTERFACE/IODELAY_DATA_CLK"  IDELAY_VALUE = 61; #
```

C. ADC-Programmiermodul

C.1. Detailliertes Blockschaltbid für das ADC-Programmiermodul

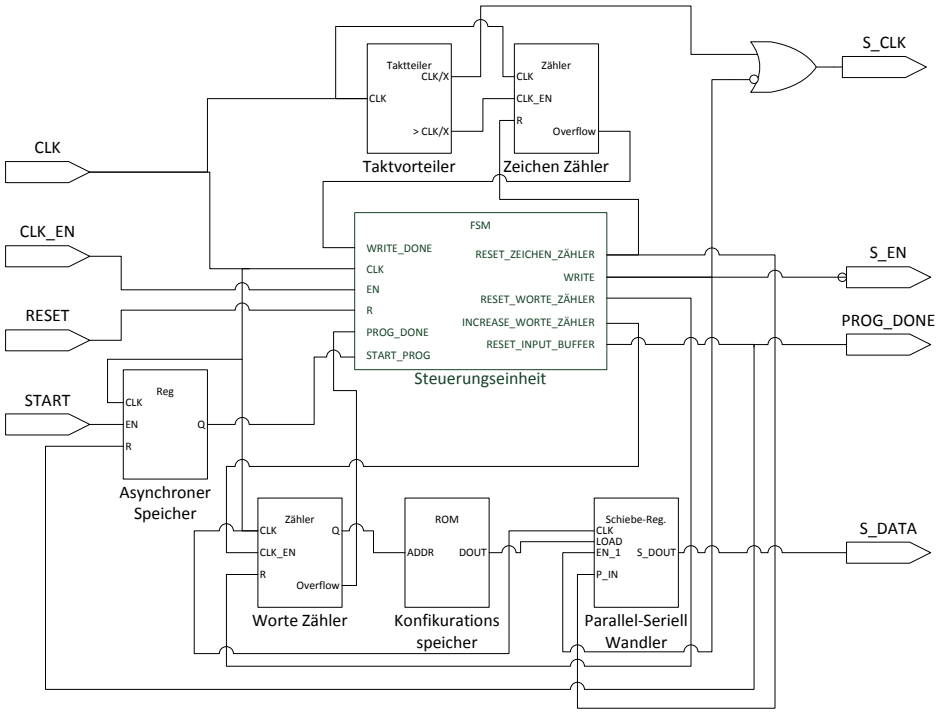


Bild C.1.: Detailliertes Blockschaltbild für das ADC-Programmiermodul

C.2. Taktvorteiler

Listing C.1: COUNT_SCLK.vhd

```
1 ----------------------------------------------
2 -- Company:        Siemens AG
```

```
3  -- Engineer:      Sönke Appel
4  --
5  -- Create Date:    14:57:34 10/28/2011
6  -- Design Name:
7  -- Module Name:    COUNT_SCLK - Behavioral
8  -- Project Name:  Millilink
9  -- Revision 0.01 - File Created
10 ----------------------------------------------
11 library IEEE;
12 use IEEE.STD_LOGIC_1164.ALL;
13 use IEEE.numeric_std.all;
14
15
16 entity COUNT_SCLK is
17     Port ( I_CLK : in  bit;
18            O_SCLK_TAKT_START_SIGNAL : out  bit;
19            O_SCLK : out  bit);
20 end COUNT_SCLK;
21
22 architecture Behavioral of COUNT_SCLK is
23 signal FLANKENDETEKT: bit;
24 signal FLANKENDETEKT2: bit;
25 signal SCLK: bit;
26 signal ZAEHLER_SCLK_GEN: unsigned(6 downto 0):=
27 ( --Initialisierung
28 "0000000"
29 );
30
31 begin
32
33 -- Zähler zur Taktteilen
34 P_ZAEHLER_SCLK_GEN: process(I_CLK)
35 begin
36    if I_CLK = '1' and I_CLK'event then
37       ZAEHLER_SCLK_GEN <= ZAEHLER_SCLK_GEN + 1 after 2 ns;
38    end if;
39 end process P_ZAEHLER_SCLK_GEN;
40 -- Nebenläufig Taktgenerierung
41 SCLK <= '1' when ZAEHLER_SCLK_GEN > "0111111" else '0' after 2 ns;
42 O_SCLK <= SCLK;
43
44 -- Nebenläufig Periodenerkennung realisiert durch eine
      Flankendetektion
45 P_FLANKENDETEKT: process(I_CLK)
```

```
46 begin
47    if I_CLK = '1' and I_CLK'event then
48       FLANKENDETEKT  <= SCLK after 2 ns;
49       FLANKENDETEKT2 <= FLANKENDETEKT after 2 ns;
50    end if;
51 end process P_FLANKENDETEKT;
52 O_SCLK_TAKT_START_SIGNAL <= FLANKENDETEKT and not FLANKENDETEKT2
      after 1 ns;
53
54 end Behavioral;
```

Listing C.2: SCLK_COUNTER.do

```
1 force -freeze sim:/count_sclk/I_CLK 1 0, 0 {5000 ps} -r 10000
2 run 3 us
```

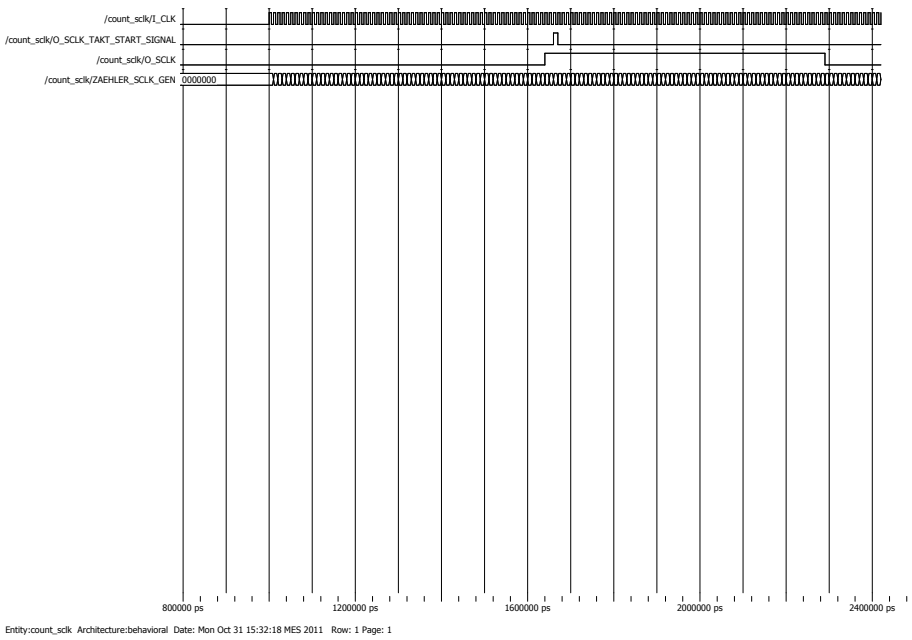


Bild C.2.: Simulationsergebnis des Taktvorteilers

Bild C.2 zeigt das Simulationsergebnis von VHDL-Quellcode des Taktvorteilers C.1 und der Stimuli C.2. In der ersten Zeile ist der Eingangstakt (I_CLK) aufgeführt. In der dritten Zeile ist der runtergeteilte Ausgangstakt (O_CLK) aufgeführt. Dieser hat eine $\frac{1}{128}$ Periode des Eingangstaktes. In der zweiten Zeile ist das vom Taktvorteiler erzeugte Ein-Systemtakt lange Steuerungssignal (O_CLK_TAKT_START_SIGNAL) aufgeführt. Dieses erfolgt nach einer kurzen Verzögerung zur Taktflanke des Ausgangstaktes.

C.3. Zeichenzähler

Listing C.3: COUNT_WORD.vhd

```
1  ----------------------------------------------
2  -- Company:        Siemens AG
3  -- Engineer:       Sönke Appel
4  --
5  -- Create Date:     16:00:00 10/28/2011
6  -- Module Name:     COUNT_WORD - Behavioral
7  -- Project Name:   Millilink
8  -- Revision 0.01 - File Created
9  -- Additional Comments:
10 ----------------------------------------------
11 library IEEE;
12 use IEEE.STD_LOGIC_1164.ALL;
13 use IEEE.STD_LOGIC_UNSIGNED.all;
14
15
16 entity COUNT_WORD is
17     Port ( CLK : in  bit;
18            RESET : in  bit;
19            CLK_EN : in  bit;
20            WORD_DONE : out  bit);
21 end COUNT_WORD;
22
23 architecture Behavioral of COUNT_WORD is
24 signal S_WORD_DONE: bit;
25 signal FLANKENDETEKT: bit;
26 signal FLANKENDETEKT2: bit;
27 signal ZAEHLER_WORT: std_logic_vector(3 downto 0):=
28 ( --Initialisierung
29 "0000"
30 );
31 begin
32
33 -- Zähler der Takte Zählt, bis ein Wort komplett ist,
34 P_ZAEHLER_WORT: process(CLK,RESET)
35 begin
36    if RESET = '1' then
37       ZAEHLER_WORT <= (others => '0') after 2 ns;
38    elsif CLK = '1' and CLK'event then
39       if CLK_EN = '1' then
40          ZAEHLER_WORT <= ZAEHLER_WORT + 1 after 2 ns;
```

```
41        end if;
42     end if;
43  end process P_ZAEHLER_WORT;
44  S_WORD_DONE <= '1' when ZAEHLER_WORT = "0000" else '0' after 2 ns;
45
46  -- Ein ein SystemTakt langes Ausgangssignal wird erzeugt
47  P_FLANKENDETEKT: process(CLK)
48  begin
49     if CLK = '1' and CLK'event then
50        FLANKENDETEKT <= S_WORD_DONE after 2 ns;
51        FLANKENDETEKT2 <= FLANKENDETEKT after 2 ns;
52     end if;
53  end process P_FLANKENDETEKT;
54  WORD_DONE <= FLANKENDETEKT and not FLANKENDETEKT2 after 1 ns;
55
56  end Behavioral;
```

Listing C.4: WORD_COUNT.do

```
1  force -freeze sim:/count_word/CLK 1 0, 0 {5000 ps} -r 10000
2  force -freeze sim:/count_word/CLK_EN 1 0, 0 {10000 ps} -r 100000
3  run 2 us
4  force -freeze sim:/count_word/RESET 1 0
5  run 10 ns
6  force -freeze sim:/count_word/RESET 0 0
7  run 40 ns
```

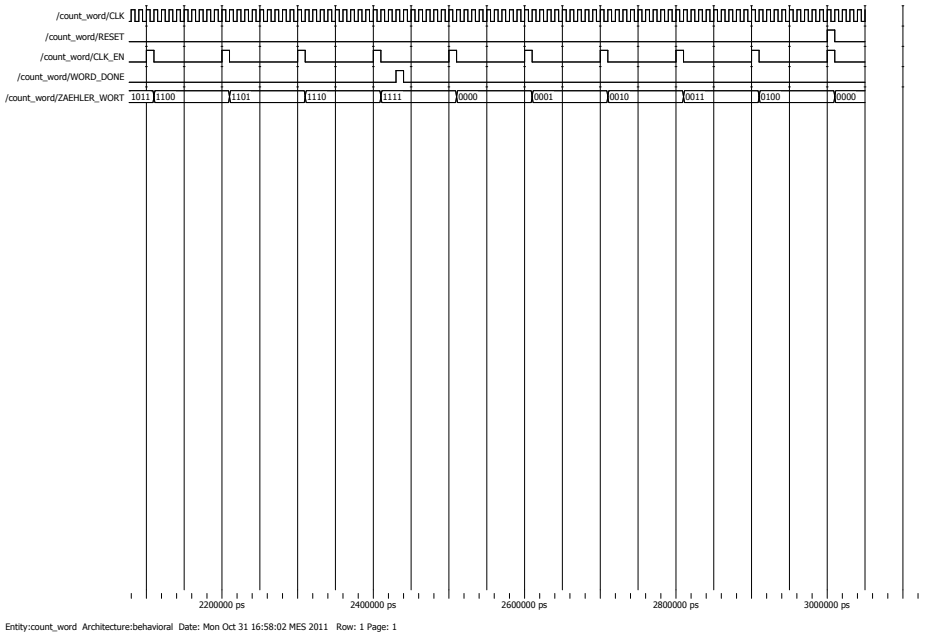


Bild C.3.: Simulationsergebnis des Zeichenzähler

Bild C.3 zeigt das Simulationsergebnis von VHDL-Quellcode des Zeichenzählers C.3 und der Stimuli C.4. Mit jedem Steuerungssignal (CLK_EN), aufgeführt in der dritten Zeile, wird der Zählerstand (ZAEHLER_WORD), aufgeführt in der fünften Zeile, um eins erhöht. Läuft der Zähler über, wird ein Ein-Systemtakt langes Steuerungssignal (WORD_DONE), aufgeführt in der vierten Zeile, gesendet. Über das Reset-Signal (RESET), aufgeführt in der zweiten Zeile, wird der Zählerstand zurückgesetzt.

C.4. Wortezähler

Listing C.5: COUNT_WORDS.vhd

```
1 ----------------------------------------------
2 -- Company:        Siemens AG
3 -- Engineer:       Sönke Appel
4 --
5 -- Create Date:     16:03:56 10/28/2011
6 -- Design Name:    COUNT_WORDS.vhd
7 -- Module Name:     COUNT_WORDS - Behavioral
8 -- Project Name:   Millilink
9
```

```
10 -- Revision 0.01 - File Created
11 ----------------------------------------------
12 library IEEE;
13 use IEEE.STD_LOGIC_1164.ALL;
14 use IEEE.STD_LOGIC_UNSIGNED.all;
15
16 entity COUNT_WORDS is
17     Port ( CLK : in  bit;
18            CLK_EN : in  bit;
19            RESET : in  bit;
20            PROG_DONE : out  bit;
21            ZAEHLER_STAND: out std_logic_vector(4 downto 0));
22 end COUNT_WORDS;
23
24 architecture Behavioral of COUNT_WORDS is
25 signal ZAEHLER_WORTE: std_logic_vector(4 downto 0):=
26 ( --Initialisierung
27 "00000"
28 );
29 begin
30 -- Zähler der Worte Zählt, bis die Programmierung komplett ist.
31 P_ZAEHLER_WORTE: process(CLK,RESET)
32 begin
33    if RESET = '1' then
34       ZAEHLER_WORTE <= (others => '0') after 2 ns;
35    elsif CLK = '1' and CLK'event then
36       if CLK_EN = '1' then
37          ZAEHLER_WORTE <= ZAEHLER_WORTE + 1 after 2 ns;
38       end if;
39    end if;
40 end process P_ZAEHLER_WORTE;
41
42 -- Ein Ein-SystemTakt langes Ausgangssignal wird erzeugt.
43 -- nebenlaeufige Zuweisung
44 ZAEHLER_STAND <= ZAEHLER_WORTE;
45 PROG_DONE <= '1' when ZAEHLER_WORTE = 19 else '0' after 2 ns;
46
47 end Behavioral;
```

Listing C.6: WORDS_COUNT.do

```
1 force -freeze sim:/count_words/CLK 1 0, 0 {5000 ps} -r 10000
2 force -freeze sim:/count_words/CLK_EN 1 0, 0 {10000 ps} -r 100000
3 run 2 us
4 force -freeze sim:/count_words/RESET 1 0
```

```
5 run 10 ns
6 force -freeze sim:/count_words/RESET 0 0
7 run 40 ns
```

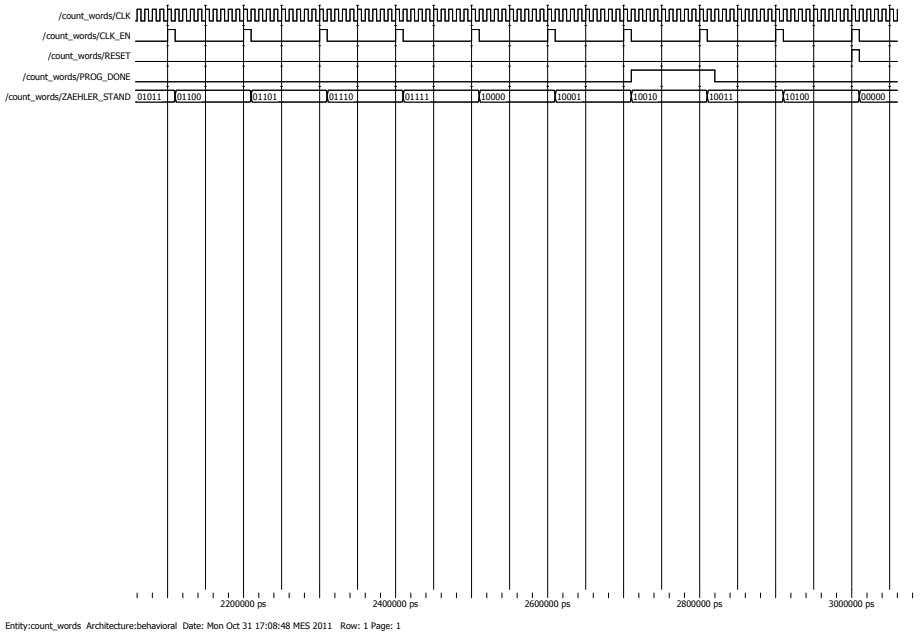


Bild C.4.: Simulationsergebnis des Wortezählers

Bild C.4 zeigt das Simulationsergebnis von VHDL-Quellcode des Wortezählers C.5 und der Stimuli C.6. Mit jedem Steuerungssignal (CLK_EN), aufgeführt in der zweiten Zeile, wird der Zählerstand (ZAEHLER_STAND), aufgeführt in der fünften Zeile, um eins erhöht. Läuft der Zähler über, wird ein Ein-Systemtakt langes Steuerungssignal (PROG_DONE), aufgeführt in der vierten Zeile, gesendet. Über das Reset-Signal (RESET), aufgeführt in der dritten Zeile, wird der Zählerstand zurückgesetzt.

C.5. Konfigurationsspeicher

Listing C.7: PROG_ROM.vhd

```
1 ----------------------------------------------------
2 -- Company:        Siemens AG
3 -- Engineer:       Sönke Appel
4 --
```

```
5  -- Create Date:    15:17:33 10/28/2011
6  -- Design Name:   PROG_ROM.vhd
7  -- Module Name:    PROG_ROM - Behavioral
8  -- Project Name:  Millilink
9
10 -- Revision 0.01 - File Created
11 ----------------------------------------------------
12 library IEEE;
13 use IEEE.STD_LOGIC_1164.ALL;
14 use IEEE.STD_LOGIC_UNSIGNED.all;
15
16 entity PROG_ROM is
17     Port ( --CLK : in bit;
18            --CLK_EN : bit;
19            I_ADDR : in  STD_LOGIC_VECTOR (4 downto 0);
20            O_DOUT : out  STD_LOGIC_VECTOR (15 downto 0));
21 end PROG_ROM;
22
23 architecture Behavioral of PROG_ROM is
24 --Signale für den ROM
25 type INSTR_ROM_TYPE is array (0 to 31) of std_logic_vector(7
       downto 0);
26 constant ADDR_ROM: INSTR_ROM_TYPE :=
27 ( --Diese Werte wurden mit dem MATLAB-FILE ADC_DATA_and_ADDR.m
       erzeugt.
28 x"00",x"20",x"3F",x"40",
29 x"41",x"44",x"50",x"51",
30 x"52",x"53",x"55",x"57",
31 x"62",x"63",x"66",x"68",
32 x"6A",x"75",x"76", others => (x"00")
33 );
34 constant DATA_ROM: INSTR_ROM_TYPE :=
35 ( --Diese Werte wurden mit dem MATLAB-FILE ADC_DATA_and_ADDR.m
       erzeugt.
36 "10000000","00000000","00000000","00001000",
37 "10000000","00000000","00000100","10101010",
38 "00101010","00000000","00000000","00000000",
39 "00000000","00000000","00000000","00000000",
40 "00000000","00000000","00000000", others => ("00000000")
41 );
42
43 signal ADDR_DOUT: std_logic_vector(7 downto 0);
44 signal DATA_DOUT: std_logic_vector(7 downto 0);
45
```

```
46 begin
47
48 -- 19x8 bit ROM für die Adressenerzeugung
49 P_ADDR_ROM: process(I_ADDR)
50 begin
51    ADDR_DOUT <= ADDR_ROM(conv_integer(I_ADDR)) after 1 ns;
52 end process P_ADDR_ROM;
53
54 -- 19x8 bit ROM für die Datenerzeugung
55 P_DATA_ROM: process(I_ADDR)
56 begin
57    DATA_DOUT <= DATA_ROM(conv_integer(I_ADDR)) after 1 ns;
58 end process P_DATA_ROM;
59
60 -- nebenläufig DOUT-Bus zusammenschustern
61 O_DOUT(15 downto 8) <= ADDR_DOUT;
62 O_DOUT(7 downto 0) <= DATA_DOUT;
63
64 end Behavioral;
```

Listing C.8: ADC_DATA_and_ADDR.m

```
1  %-----------------------------------------------
2  % Autor:       Sönke Appel
3  % Company:     Siemens AG
4  % File:        ODELAY_CALC.m
5  % Projekt:     MILLILINK
6  % Beschreibung: MATLAB-Skript zur Erzeugung der
   Konfigurationsdaten für den ADC ADS62P49 von Texas Instruments
7  %-----------------------------------------------
8  function ADC_DATA_and_ADDR()
9
10 %Adresszuweisung
11 ADD0 = hex2dec('00');
12 ADD1 = hex2dec('20');
13 ADD2 = hex2dec('3F');
14 ADD3 = hex2dec('40');
15 ADD4 = hex2dec('41');
16 ADD5 = hex2dec('44');
17 ADD6 = hex2dec('50');
18 ADD7 = hex2dec('51');
19 ADD8 = hex2dec('52');
20 ADD9 = hex2dec('53');
21 ADD10 = hex2dec('55');
22 ADD11 = hex2dec('57');
```

```
ADD12 = hex2dec('62');
ADD13 = hex2dec('63');
ADD14 = hex2dec('66');
ADD15 = hex2dec('68');
ADD16 = hex2dec('6A');
ADD17 = hex2dec('75');
ADD18 = hex2dec('76');

%Adressen zusammen setzen
ADC_ADDR = [ADD0,ADD1,ADD2,ADD3,ADD4,ADD5,ADD6,ADD7,...
            ADD8,ADD9,ADD10,ADD11,ADD12,ADD13,ADD14,ADD15,...
            ADD16,ADD17,ADD18];

%Ausgabe auf der Matlab Console
assignin('base', 'ADC_ADDR', ADC_ADDR);

%Kofiguration des ADCs
%Die hier gewählten Namen sind identisch mit den Namen im User
   Guide
SOFTWARE_RESET = '1';
SERIAL_READOUT = '0';
ENABLE_LOW_SPEED_MODE = '0';
INTERNAL_REF = '00';
STANDNY = '0';
POWER_DOWN_MODES = '1000';
LVDS_DDR = '1';
CLOCK_EDGE_CONTROL = ['000','000'];
INDEPENDENT_CHANNEL_CONTROL = '0';
DATA_FORMAT = '10';
COSTOM_PATTERN_LOW = ['1010', '1010'];
COSTOM_PATTERN_HIGH = ['1010','10'];
ENABLE_OFFSET_CORRECTION_CH_A = '0';
GAIN_COMMON_CH_A = '0000';
OFFSET_CORR_TIME_CONSTANT_CH_A = '0000';
FINE_GAIN_ADJUST_CH_A = ['0000','000'];
TEST_PATTERN_CH_A = '000';
OFFSET_PEDESTAL_CH_A = ['0000','00'];
ENABLE_OFFSET_CORRECTION_CH_B = '0';
GAIN_CH_B = '0000';
OFFSET_CORR_TIME_CONSTANT_CH_B = '0000';
FINE_GAIN_ADJUST_CH_B = ['0000','000'];
TEST_PATTERN_CH_B = '000';
OFFSET_PEDESTAL_CH_B = ['0000','00'];

```

```
%Zusammensetzen der einzelnen Konfigurationsdaten
DATA0 = bin2dec([SOFTWARE_RESET,'000','000',SERIAL_READOUT]);
DATA1 = bin2dec(['0000','0',ENABLE_LOW_SPEED_MODE,'00']);
DATA2 = bin2dec(['0',INTERNAL_REF,'0','00',STANDNY,'0']);
DATA3 = bin2dec(['0000',POWER_DOWN_MODES]);
DATA4 = bin2dec([LVDS_DDR,'000','0000']);
DATA5 = bin2dec([CLOCK_EDGE_CONTROL,'00']);
DATA6 = bin2dec(['0',INDEPENDENT_CHANNEL_CONTROL,'00','0',
   DATA_FORMAT,'0']);
DATA7 = bin2dec(COSTOM_PATTERN_LOW);
DATA8 = bin2dec(['00',COSTOM_PATTERN_HIGH]);
DATA9 = bin2dec(['0',ENABLE_OFFSET_CORRECTION_CH_A,'00','0000']);
DATA10 = bin2dec([GAIN_COMMON_CH_A,OFFSET_CORR_TIME_CONSTANT_CH_A
   ]);
DATA11 = bin2dec(['0',FINE_GAIN_ADJUST_CH_A]);
DATA12 = bin2dec(['0000','0',TEST_PATTERN_CH_A]);
DATA13 = bin2dec(['00',OFFSET_PEDESTAL_CH_A]);
DATA14 = bin2dec(['0',ENABLE_OFFSET_CORRECTION_CH_B,'00','0000']);
DATA15 = bin2dec([GAIN_CH_B,OFFSET_CORR_TIME_CONSTANT_CH_B]);
DATA16 = bin2dec(['0',FINE_GAIN_ADJUST_CH_B]);
DATA17 = bin2dec(['0000','0',TEST_PATTERN_CH_B]);
DATA18 = bin2dec(['00',OFFSET_PEDESTAL_CH_B]);

%Daten zusammensetzen
ADC_DATA = [DATA0,DATA1,DATA2,DATA3,DATA4,DATA5,DATA6,DATA7,...
           DATA8,DATA9,DATA10,DATA11,DATA12,DATA13,DATA14,DATA15
              ,...
           DATA16,DATA17,DATA18];

%Ausgabe auf der Matlab Console
assignin('base', 'ADC_DATA', ADC_DATA);

%TXT-Datei erzeugen zur Ausgabe der Konfigurationen
fileID = fopen('PROG_DATEN.txt', 'w');
fprintf(fileID, 'ADC_ADDR = ');
for k=1:length(ADC_ADDR)
    fprintf(fileID, 'x"%s",', dec2hex(ADC_ADDR(k),2));
end
fprintf(fileID, '\nADC_ADDR = ');
for k=1:length(ADC_DATA)
    fprintf(fileID, '"%s",', dec2bin(ADC_DATA(k),8));
end
fclose(fileID);
```

Listing C.9: SPEICHER_DISTRIBUTED.do

```
#force -freeze sim:/prog_rom/CLK 1 0, 0 {5000 ps} -r {10 ns}
force -freeze sim:/prog_rom/I_ADDR 00000 0
run 20 ns
force -freeze sim:/prog_rom/I_ADDR 00001 0
run 20 ns
force -freeze sim:/prog_rom/I_ADDR 00010 0
run 20 ns
force -freeze sim:/prog_rom/I_ADDR 00011 0
run 20 ns
force -freeze sim:/prog_rom/I_ADDR 00100 0
run 20 ns
force -freeze sim:/prog_rom/I_ADDR 00101 0
run 20 ns
force -freeze sim:/prog_rom/I_ADDR 00110 0
run 20 ns
force -freeze sim:/prog_rom/I_ADDR 00111 0
run 20 ns
force -freeze sim:/prog_rom/I_ADDR 01000 0
run 20 ns
force -freeze sim:/prog_rom/I_ADDR 01001 0
run 20 ns
force -freeze sim:/prog_rom/I_ADDR 01010 0
run 20 ns
force -freeze sim:/prog_rom/I_ADDR 01011 0
run 20 ns
force -freeze sim:/prog_rom/I_ADDR 01100 0
run 20 ns
force -freeze sim:/prog_rom/I_ADDR 01101 0
run 20 ns
force -freeze sim:/prog_rom/I_ADDR 01110 0
run 20 ns
force -freeze sim:/prog_rom/I_ADDR 01111 0
run 20 ns
force -freeze sim:/prog_rom/I_ADDR 10000 0
run 20 ns
force -freeze sim:/prog_rom/I_ADDR 10001 0
run 20 ns
force -freeze sim:/prog_rom/I_ADDR 10010 0
run 20 ns
force -freeze sim:/prog_rom/I_ADDR 10011 0
run 20 ns
```

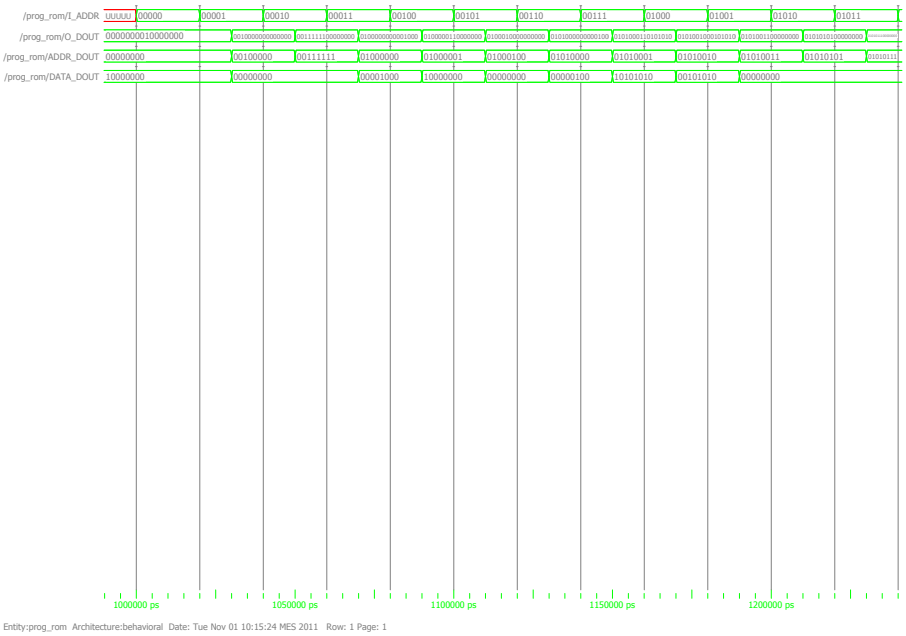


Bild C.5.: Simulationsergebnis des Konfigurationsspeichers

Bild C.5 zeigt das Simulationsergebnis von VHDL-Quellcode des Konfigurationsspeichers C.7 und der Stimuli C.9. Durch Anlegen der Adresse (I_ADDR), aufgeführt in der ersten Zeile, an den Konfigurationsspeicher werden die Programmierdaten (O_DOUT), aufgeführt in der zweiten Zeile, ausgegeben. Die Programmierdaten bestehen aus den Konfigurationsadressen (ADDR_DOUT) und den Konfigurationsdaten (DATA_DOUT), aufgeführt in Zeile Drei und Vier.

C.6. Parallel-Seriell-Wandler

Listing C.10: SHIFT_REG.vhd

```
1 ----------------------------------------------------
2 -- Company:        Siemens AG
3 -- Engineer:       Sönke Appel
4 --
5 -- Create Date:     15:42:13 10/28/2011
6 -- Design Name:    SHIFT_REG.vhd
7 -- Module Name:     SHIFT_REG - Behavioral
8 -- Project Name:   Millilink
9
```

```
10 -- Revision 0.01 - File Created
11 ----------------------------------------------------
12 library IEEE;
13 use IEEE.STD_LOGIC_1164.ALL;
14
15 entity SHIFT_REG is
16     Port ( CLK : in  bit;
17            LOAD_SHIFT_REG : in  bit;
18            CLK_EN_1 : in  bit;
19            CLK_EN_2 : in  bit;
20            I_PDATA : in  STD_LOGIC_VECTOR (15 downto 0);
21            O_SDATA : out  STD_LOGIC);
22 end SHIFT_REG;
23
24 architecture Behavioral of SHIFT_REG is
25 signal SHIFT_VALUES: std_logic_vector(15 downto 0);
26 signal CLK_EN: bit;
27 begin
28
29 -- Verknüpfung zweier Enable Signale
30 CLK_EN <= CLK_EN_1 and CLK_EN_2;
31
32 -- Schieberegister
33 P_SHIFT_REG: process(CLK)
34
35 begin
36    if CLK = '1' and CLK'event then
37       if LOAD_SHIFT_REG = '1' then
38          SHIFT_VALUES <= I_PDATA after 2 ns;
39       elsif CLK_EN = '1' then
40          SHIFT_VALUES <= SHIFT_VALUES(14 downto 0) & '0' after 2
               ns;
41       end if;
42    end if;
43 end process P_SHIFT_REG;
44 O_SDATA <= SHIFT_VALUES(15);
45
46
47 end Behavioral;
```

Listing C.11: SHIFT_REG.do

```
1 force -freeze sim:/shift_reg/CLK 1 0, 0 {5000 ps} -r {10 ns}
2 force -freeze sim:/shift_reg/I_PDATA 1100110011001101 0
3 run 10 ns
```

```
force -freeze sim:/shift_reg/LOAD_SHIFT_REG 1 0
run 10 ns
force -freeze sim:/shift_reg/LOAD_SHIFT_REG 0 0
force -freeze sim:/shift_reg/CLK_EN_1 1 0
force -freeze sim:/shift_reg/CLK_EN_2 1 0
run 170 ns
```

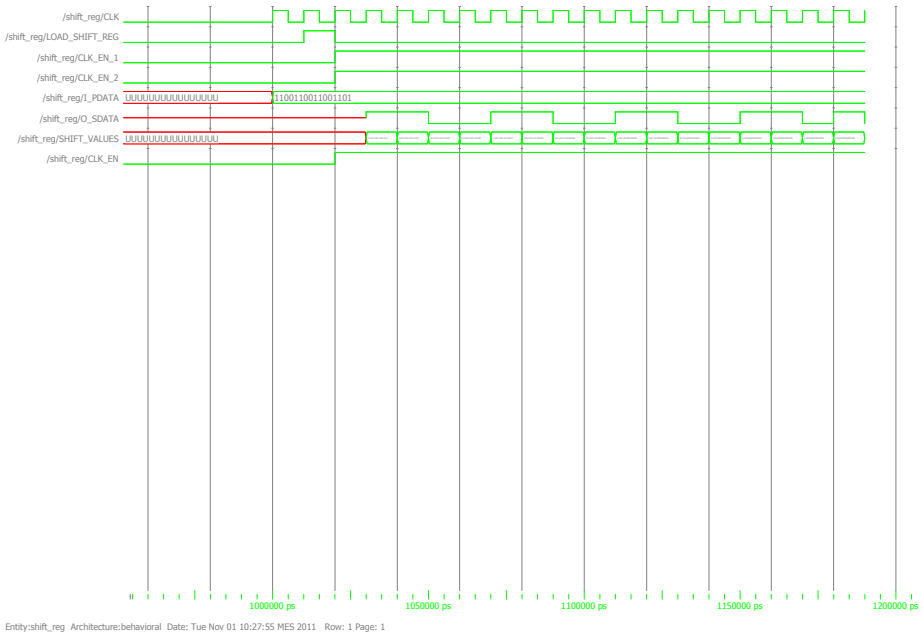


Bild C.6.: Simulationsergebnis des Parallel-Seriell-Wandlers

Bild C.6 zeigt das Simulationsergebnis von VHDL-Quellcode des Parallel-Seriell-Wandlers C.10 und der Stimuli C.11. Mit dem Signal (LOAD_SHIFT_REG) werden die Eingangsdaten (I_PDATA) auf die Schieberegister (SHIFT_VALUES) übernommen. Mit jedem Takt (CLK) wird ein Bit an (O_SDATA) ausgegeben.

C.7. Asynchroner Speicher

Listing C.12: EXT_MEM.vhd

```
----------------------------------------------------
-- Company:        Siemens AG
-- Engineer:       Sönke Appel
```

```
4  --
5  -- Create Date:    17:27:33 10/29/2011
6  -- Design Name:   EXT_MEM.vhd
7  -- Module Name:    EXT_MEM - Behavioral
8  -- Project Name:  Millilink
9  -- Revision 0.01 - File Created
10 ----------------------------------------------
11 library IEEE;
12 use IEEE.STD_LOGIC_1164.ALL;
13
14 entity EXT_MEM is
15     Port ( CLK_EN : in  bit;
16            RESET : in  bit;
17            CLK : in  bit;
18            O_Q : out  bit);
19 end EXT_MEM;
20
21 architecture Behavioral of EXT_MEM is
22
23 -- Realisierung eines Registers
24 begin
25 P_EXT_MEM: process(CLK,RESET)
26 begin
27
28    if RESET = '1' then
29       O_Q <= '0' after 2 ns;
30    elsif CLK = '1' and CLK'event then
31       if CLK_EN = '1' then
32          O_Q <= '1' after 3 ns;
33       end if;
34    end if;
35
36
37 end process P_EXT_MEM;
38
39 end Behavioral;
```

Listing C.13: EXT_SPEICHER.do

```
1 force -freeze sim:/ext_mem/CLK 1 0, 0 {5000 ps} -r {10 ns}
2 run 40 ns
3 force -freeze sim:/ext_mem/CLK_EN 1 0
4 run 40 ns
5 force -freeze sim:/ext_mem/CLK_EN 0 0
6 run 40 ns
```

```
7  force -freeze sim:/ext_mem/RESET 1 0
8  run 40 ns
9  force -freeze sim:/ext_mem/RESET 0 0
10 run 40 ns
```

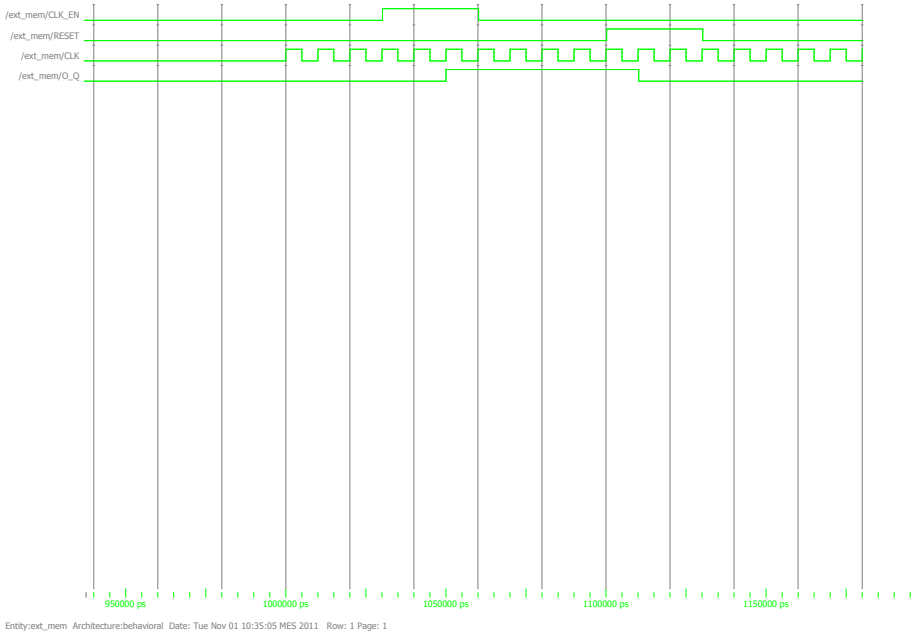


Bild C.7.: Simulationsergebnis des asynchronen Speichers

Bild C.7 zeigt das Simulationsergebnis von VHDL-Quellcode des asynchronen Speichers C.12 und der Stimuli C.13. Durch Setzen des (CLK_EN) Signals wird taktsynchron mit dem Takt (CLK) das Register (O_Q) auf *high* gesetzt. Mit dem (RESET)-Signal wird das Register zurück auf *low* gesetzt.

C.8. Steuerungseinheit

Listing C.14: CONTROL_FSM.vhd

```
1  ----------------------------------------------
2  -- Company:        Siemens AG
3  -- Engineer:       Sönke Appel
4  --
5  -- Create Date:     15:30:18 10/25/2011
```

```
6  -- Design Name:   CONTROL_FSM.vhd
7  -- Module Name:    CONTROL_FSM - Behavioral
8  -- Project Name:  Millilink
9  -- Revision 0.01 - File Created
10 ----------------------------------------------
11 library IEEE;
12 use IEEE.STD_LOGIC_1164.ALL;
13
14 entity CONTROL_FSM is
15     Port ( CLK : in  bit;
16            CLK_EN : in bit;
17            RESET : in bit;
18            I_WRITE_DONE : in  bit;
19            I_PROG_DONE : in  bit;
20            I_START_PROG : in  bit;
21            I_PROG_START_SIG : in bit;
22            O_WRITE : out  bit;
23            O_RST_COUNT_WORD : out  bit;
24            O_RST_COUNT_WORDS : out  bit;
25            O_INCREASE_WORDS_COUNT : out bit;
26            O_RESET_INPUT_BUF : out  bit);
27 end CONTROL_FSM;
28
29 architecture Behavioral of CONTROL_FSM is
30 type ZUSTAENDE is (IDLE, PROG_START, WRITE_A_COMMAND,
      INCREASE_WORDS, PROG_BREAK, PROG_DONE);
31 attribute ENUM_ENCODING: String;
32 attribute ENUM_ENCODING of ZUSTAENDE: type is "100000␣010000␣
      001000␣000100␣000010␣000001"; -- ONE HOT CODIERUNG
33
34 signal ZUSTAND, FOLGE_Z: ZUSTAENDE ;
35
36
37 -- Zustandsspeicher
38 begin
39    Z_SPEICHER: process(CLK, RESET)
40    begin
41       if RESET = '1' then ZUSTAND <= IDLE after 2 ns;
42       elsif CLK = '1' and CLK'event then
43          if CLK_EN = '1' then
44             ZUSTAND <= FOLGE_Z after 2 ns;
45          end if;
46       end if;
47    end process Z_SPEICHER;
```

```
48
49  -- Schaltnetz (Übergangs- und Ausgangsschaltnetz)
50     UE_AUS_SN: process(I_WRITE_DONE, I_PROG_DONE, I_START_PROG,
           I_PROG_START_SIG, ZUSTAND)
51     begin
52
53     --Festlegung der Defaults
54     O_WRITE <= '0' after 1 ns;
55     O_RST_COUNT_WORD <= '1' after 1 ns;
56     O_RST_COUNT_WORDS <= '1' after 1 ns;
57     O_RESET_INPUT_BUF <= '0' after 1 ns;
58     O_INCREASE_WORDS_COUNT <= '0' after 1 ns;
59
60        case ZUSTAND is
61
62           when IDLE =>                  if I_START_PROG = '1' then
63                                         FOLGE_Z <= WRITE_A_COMMAND
                                                after 2 ns;
64                                     else
65                                         FOLGE_Z <= IDLE after 3 ns;
66                                     end if;
67
68           when WRITE_A_COMMAND =>           O_WRITE <= '1' after 2 ns;
69                                     O_RST_COUNT_WORD <= '0' after 2
                                            ns;
70                                     O_RST_COUNT_WORDS <= '0' after 2
                                            ns;
71                                     if I_WRITE_DONE = '1' then
72                                         FOLGE_Z <= INCREASE_WORDS
                                                after 2 ns;
73                                     else
74                                         FOLGE_Z <= WRITE_A_COMMAND
                                                after 3 ns;
75                                     end if;
76
77           when INCREASE_WORDS =>            O_RST_COUNT_WORDS <= '0'
                after 2 ns;
78                                     O_INCREASE_WORDS_COUNT <= '1'
                                            after 2 ns;
79                                     if I_PROG_DONE = '1' then
80                                         FOLGE_Z <= PROG_DONE after 2
                                                ns;
81                                     else
```

```
82                                      FOLGE_Z <= PROG_BREAK after 3
                                           ns;
83                                    end if;
84
85          when PROG_BREAK =>            O_RST_COUNT_WORDS <= '0'
               after 2 ns;
86                                    if I_PROG_START_SIG = '1' then
87                                       FOLGE_Z <= WRITE_A_COMMAND
                                            after 2 ns;
88                                    else
89                                       FOLGE_Z <= PROG_BREAK after 3
                                           ns;
90                                    end if;
91
92
93          when PROG_DONE =>             O_RESET_INPUT_BUF <= '1' after
               2 ns;
94                                    if I_START_PROG = '0' then
95                                       FOLGE_Z <= IDLE after 2 ns;
96                                    else
97                                       FOLGE_Z <= PROG_DONE after 3
                                           ns;
98                                    end if;
99
100         when others =>                FOLGE_Z <= IDLE after 2 ns;
101
102      end case;
103   end process UE_AUS_SN;
104 end Behavioral;
```

Listing C.15: ADC_PROG.do

```
1  vlib work
2  vcom -explicit  -93 "SHIFT_REG.vhd"
3  vcom -explicit  -93 "PROG_ROM.vhd"
4  vcom -explicit  -93 "EXT_MEM.vhd"
5  vcom -explicit  -93 "COUNT_WORDS.vhd"
6  vcom -explicit  -93 "COUNT_WORD.vhd"
7  vcom -explicit  -93 "COUNT_SCLK.vhd"
8  vcom -explicit  -93 "CONTROL_FSM.vhd"
9  vcom -explicit  -93 "ADC_PROG.vhd"
10 vsim -voptargs="+acc" -t 1ps  -lib work work.ADC_PROG
11
12 add wave *
13
```

```
14 view wave
15 view structure
16 view signals
17
18 run 1000ns
19
20 force -freeze sim:/adc_prog/CLK 1 0, 0 {5000 ps} -r {10 ns}
21 force -freeze sim:/adc_prog/CLK_EN 1 0
22 run 50 us
23
24 force -freeze sim:/adc_prog/I_START_PB 1 0
25 run 56 ns
26
27 force -freeze sim:/adc_prog/I_START_PB 0 0
28 run 500 us
```

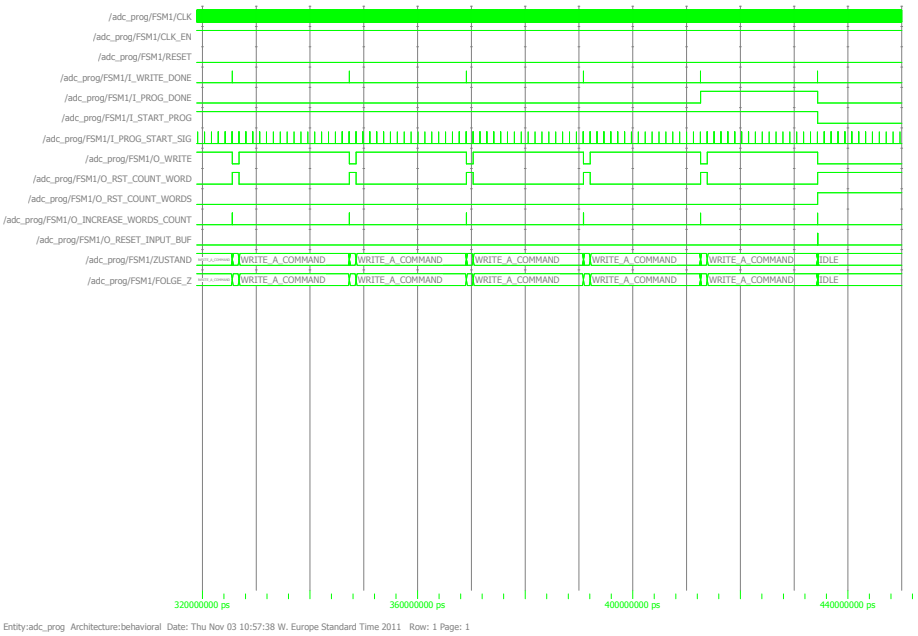


Bild C.8.: Simulationsergebnis der Steuerungseinheit

Bild C.8 zeigt das Simulationsergebnis von VHDL-Quellcode der Steuerungseinheit C.14 und der Stimuli C.15. Jeweils nach dem Empfangen eines (I_WRITE_DONE) wird eine Ein-Programmiertakt lange Pause im (O_WRITE) Signal gesendet. Dabei wird der Zeichenzähler mit dem (O_RST_COUNT_WORD)-Signal gelöscht. Im gleichen Zuge wird der Wortezähler mit dem (O_INCREASE_WORDS_COUNT)-Signal um einen Wert erhöht. Empfängt die

Steuerungseinheit ein (I_PROG_DONE)-Signal, springt die Steuerungseinheit zurück in den IDLE-Zustand.

C.9. Programmiermodul (Top Level Design)

Listing C.16: ADC_PROG.vhd

```
1  ----------------------------------------------------
2  -- Company:        Siemens AG
3  -- Engineer:       Sönke Appel
4  --
5  -- Create Date:     13:40:45 10/26/2011
6  -- Design Name:    ADC_PROG.vhd
7  -- Module Name:     ADC_PROG - Behavioral
8  -- Project Name:   Millilink
9  -- Revision 0.01 - File Created
10 ----------------------------------------------------
11 library IEEE;
12 use IEEE.STD_LOGIC_1164.ALL;
13 use IEEE.STD_LOGIC_UNSIGNED.all;
14
15
16 entity ADC_PROG is
17     Port ( CLK : in  bit;
18            CLK_EN : in  bit;
19            I_START_PB : in  bit;
20            O_SEN : out  std_ulogic;
21            O_SDATA : out  std_logic;
22            O_SCLK : out  std_ulogic;
23            RESET : in  bit);
24 end ADC_PROG;
25
26 architecture Behavioral of ADC_PROG is
27
28 signal TAKTVERZOEGERUNG: bit_vector(5 downto 0);
29
30 -- Komponentendeklaration der FSM
31 component CONTROL_FSM
32     Port ( CLK : in  bit;
33            CLK_EN : in bit;
34            RESET : in bit;
35            I_WRITE_DONE : in  bit;
36            I_PROG_DONE : in  bit;
```

```
37              I_START_PROG : in  bit;
38              I_PROG_START_SIG : in bit;
39              O_WRITE : out  bit;
40              O_RST_COUNT_WORD : out  bit;
41              O_RST_COUNT_WORDS : out  bit;
42              O_INCREASE_WORDS_COUNT : out bit;
43              O_RESET_INPUT_BUF : out  bit);
44  end component;
45  -- Signaldeklaration für die FSM
46  signal WRITE_A_WORD: bit;
47  signal WORT_FERTIG: bit;
48  signal PROG_DONE: bit;
49  signal RESET_WORD_COUNT, RESET_WORDS_COUNT: bit;
50  signal INCREASE_WORDS_COUNT: bit;
51
52
53  -- Komponentendeklaration des Zählers zum SCLK erzeugen 100MHz/128
54  component COUNT_SCLK
55      Port ( I_CLK : in  bit;
56             O_SCLK_TAKT_START_SIGNAL : out  bit;
57             O_SCLK : out  bit);
58  end component;
59  -- Signaldeklaration für den Zähler zum SCLK erzeugen 100MHz/128
60  signal SCLK_TAKT_START_SIGNAL: bit;
61  signal S_SCLK: bit;
62
63  -- Komponentendeklaration des Zählers zum zählen der Takte bis ein
        Word komplett ist
64  component COUNT_WORD
65      Port ( CLK : in  bit;
66             RESET : in  bit;
67             CLK_EN : in  bit;
68             WORD_DONE : out  bit);
69  end component;
70
71  -- Komponentendeklaration des Zählers zum zählen der Worte bis die
        Programmierung komplett ist
72  component COUNT_WORDS
73      Port ( CLK : in  bit;
74             CLK_EN : in  bit;
75             RESET : in  bit;
76             PROG_DONE : out  bit;
77             ZAEHLER_STAND: out std_logic_vector(4 downto 0));
78  end component;
```

```
79
80  -- Komponentendeklaration des ROMs in dem die Programmierdaten
        gespeichert sind
81  component PROG_ROM
82      Port ( --CLK : in bit;
83             I_ADDR : in  STD_LOGIC_VECTOR (4 downto 0);
84             O_DOUT : out  STD_LOGIC_VECTOR (15 downto 0));
85  end component;
86  -- Signaldeklaration für die
87  signal ADDR: STD_LOGIC_VECTOR (4 downto 0);
88
89  -- Komponentendeklaration des Schieberegisters zum schieben der
        Paralellen Daten auf den Seriellen Datenbus
90  component SHIFT_REG
91      Port ( CLK : in  bit;
92             LOAD_SHIFT_REG : in  bit;
93             CLK_EN_1 : in  bit;
94             CLK_EN_2 : in  bit;
95             I_PDATA : in  STD_LOGIC_VECTOR (15 downto 0);
96             O_SDATA : out  STD_LOGIC);
97  end component;
98  -- Signaldeklaration für das Schieberegister
99  signal LOAD_SHIFT_REG: bit;
100 signal PARALELL_DATA: STD_LOGIC_VECTOR (15 downto 0);
101
102
103 -- Komponentendeklaration des Zwischenspeichers um externe Signale
        zu erfassen
104 component EXT_MEM
105     Port ( CLK_EN : in  bit;
106            RESET : in  bit;
107            CLK : in  bit;
108            O_Q : out  bit);
109 end component;
110 -- Signaldeklaration für den Zwischenspeichers um externe Signale
        zu erfassen
111 signal RESET_INPUT_BUF: bit;
112 signal INPUT_MEM: bit;
113
114 -- Konfiguration zur Auswahl von Modellarchitekturen
115 for COUNT1: COUNT_SCLK use entity WORK.COUNT_SCLK(Behavioral);
116 for COUNT2: COUNT_WORD use entity WORK.COUNT_WORD(Behavioral);
117 for COUNT3: COUNT_WORDS use entity WORK.COUNT_WORDS(Behavioral);
118 for SPEICHER1: PROG_ROM use entity WORK.PROG_ROM(Behavioral);
```

```
119 for REG1: SHIFT_REG use entity WORK.SHIFT_REG(Behavioral);
120 for FSM1: CONTROL_FSM use entity WORK.CONTROL_FSM(Behavioral);
121 for EX_MEM1: EXT_MEM use entity WORK.EXT_MEM(Behavioral);
122
123
124 begin
125 -- Komponenteninstanziierung des Zählers zum SCLK erzeugen 100MHz
       /128
126 COUNT1: COUNT_SCLK port map(
127             I_CLK => CLK,
128             O_SCLK_TAKT_START_SIGNAL => SCLK_TAKT_START_SIGNAL,
129             O_SCLK => S_SCLK);
130
131 -- Prozess zur Takverzögerung um Daten und Takt in ein optimales
       Verhältnis zu bringen
132 P_TAKTVERZOEGERUNG: process(CLK)
133 begin
134 if CLK = '1' and CLK'event then
135    TAKTVERZOEGERUNG <= TAKTVERZOEGERUNG(4 downto 0) & (S_SCLK or
          not WRITE_A_WORD);
136 end if;
137 end process P_TAKTVERZOEGERUNG;
138 -- Casting von bit zu std_logic
139 O_SCLK <= To_StdULogic (TAKTVERZOEGERUNG(5));
140
141 -- Komponenteninstanziierung des Zählers zum zählen der Takte bis
       ein Word komplett ist
142 COUNT2: COUNT_WORD port map(
143             CLK => CLK,
144             RESET => RESET_WORD_COUNT,
145             CLK_EN => SCLK_TAKT_START_SIGNAL,
146             WORD_DONE => WORT_FERTIG);
147
148 -- Komponenteninstanziierung des Zählers zum zählen der Worte bis
       die Programmierung komplett ist
149 COUNT3: COUNT_WORDS port map(
150             CLK => CLK,
151             CLK_EN => INCREASE_WORDS_COUNT,
152             RESET => RESET_WORDS_COUNT,
153             PROG_DONE => PROG_DONE,
154             ZAEHLER_STAND=> ADDR);
155
156 -- Komponenteninstanziierung des ROMs in dem die Programmierdaten
       gespeichert sind
```

```
157 SPEICHER1: PROG_ROM port map(
158             --CLK => CLK,
159             I_ADDR => ADDR,
160             O_DOUT => PARALELL_DATA);
161
162 -- Komponenteninstanziierung des Schieberegisters zum schieben der
        Paralellen Daten auf den Seriellen Datenbus
163 REG1: SHIFT_REG port map(
164             CLK => CLK,
165             LOAD_SHIFT_REG => LOAD_SHIFT_REG,
166             CLK_EN_1 => WRITE_A_WORD,
167             CLK_EN_2 => SCLK_TAKT_START_SIGNAL,
168             I_PDATA => PARALELL_DATA,
169             O_SDATA => O_SDATA);
170
171 -- Komponenteninstanziierung der FSM
172 FSM1: CONTROL_FSM port map(CLK => CLK,
173                            CLK_EN => CLK_EN,
174                            RESET => RESET,
175                            I_WRITE_DONE => WORT_FERTIG,
176                            I_PROG_DONE => PROG_DONE,
177                            I_START_PROG => INPUT_MEM,
178                            I_PROG_START_SIG =>
                                  SCLK_TAKT_START_SIGNAL,
179                            O_WRITE => WRITE_A_WORD,
180                            O_RST_COUNT_WORD => RESET_WORD_COUNT,
181                            O_RST_COUNT_WORDS => RESET_WORDS_COUNT,
182                            O_INCREASE_WORDS_COUNT =>
                                  INCREASE_WORDS_COUNT,
183                            O_RESET_INPUT_BUF => RESET_INPUT_BUF);
184 -- casting von zu std_logic
185 O_SEN <= To_StdULogic (not WRITE_A_WORD) after 2 ns;
186 -- Verbing einen Signal mit einem Ausgang
187 LOAD_SHIFT_REG <= RESET_WORD_COUNT;
188
189 -- Komponenteninstanziierung des Zwischenspeichers um externe
       Signale zu erfassen
190 EX_MEM1: EXT_MEM    port map(
191                CLK_EN => I_START_PB,
192                RESET => RESET_INPUT_BUF,
193                CLK => CLK,
194                O_Q => INPUT_MEM);
195
196
```

```
197 end Behavioral;
```

Als Stimuli dient dieselbe Stimuli, welche bereits zur Simulation der FSM verwendet worden ist.

Listing C.17: ADC_PROG.do

```
1  vlib work
2  vcom -explicit  -93 "SHIFT_REG.vhd"
3  vcom -explicit  -93 "PROG_ROM.vhd"
4  vcom -explicit  -93 "EXT_MEM.vhd"
5  vcom -explicit  -93 "COUNT_WORDS.vhd"
6  vcom -explicit  -93 "COUNT_WORD.vhd"
7  vcom -explicit  -93 "COUNT_SCLK.vhd"
8  vcom -explicit  -93 "CONTROL_FSM.vhd"
9  vcom -explicit  -93 "ADC_PROG.vhd"
10 vsim -voptargs="+acc" -t 1ps  -lib work work.ADC_PROG
11
12 add wave *
13
14 view wave
15 view structure
16 view signals
17
18 run 1000ns
19
20 force -freeze sim:/adc_prog/CLK 1 0, 0 {5000 ps} -r {10 ns}
21 force -freeze sim:/adc_prog/CLK_EN 1 0
22 run 50 us
23
24 force -freeze sim:/adc_prog/I_START_PB 1 0
25 run 56 ns
26
27 force -freeze sim:/adc_prog/I_START_PB 0 0
28 run 500 us
```

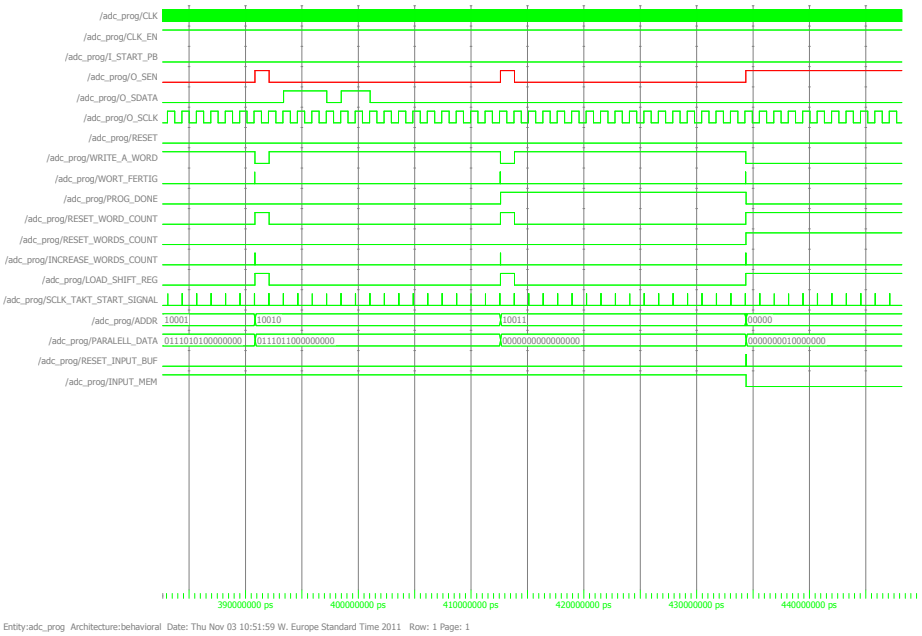


Bild C.9.: Simulationsergebnis des Programmiermoduls (Top Level Design)

Bild C.9 zeigt das Simulationsergebnis von VHDL-Quellcode C.16 des Programmiermoduls und der Stimuli C.15. Genauer betrachtet werden sollen die drei seriellen Programmiersignale, aufgeführt in Zeile 4 bis 6, mit denen der ADC programmiert wird. Das Enable-Signal (O_SEN) erfährt den Pegel *low* sobald Daten gesendet werden. Das Daten-Signal (O_SDATA) überträgt die Daten. Diese sind identisch mit den parallelen Daten (PARALLEL_DATA), aufgeführt in der dritten Zeile von unten. Das Programmiertakt-Signal (O_SCLK) übergibt die Daten am Ausgang des FPGA mit der steigenden Flanke. Im ADC werden die Daten mit der fallenden Flanke übernommen [32, Seite 17].

D. Impulsformung

Listing D.1: Eyediagramm.m

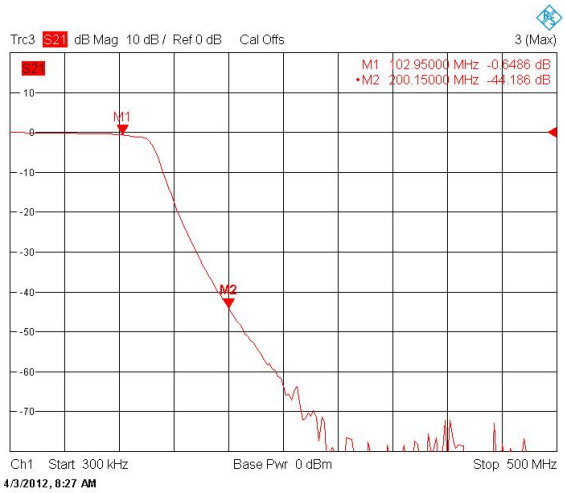
```
%file: Eyediagramm.m
%Autor: Sönke Appel

function Eyediagramm(x)
% Create an eye diagram scope object
h = commscope.eyediagram('SamplingFrequency', 64*4*80e6, ...
                         'SamplesPerSymbol', 64*4, ...
                         'SymbolsPerTrace', 1, ...
                         'MinimumAmplitude', -0.5, ...
                         'MaximumAmplitude', 1, ...
                         'MeasurementDelay', 0, ...
                         'PlotType', '2D Line', ...
                         'PlotTimeOffset', 0.7/80e6, ...
                         'ColorScale', 'linear', ...
                         'NumberOfStoredTraces', 1000, ...
                         'RefreshPlot', 'on');
% Update the eye diagram
update(h, x);
% Display the eye diagram figure
plot(h)
l = h.SamplesProcessed
h
hold on
t1 = [1/3/80e6 1/3/80e6];
t2 = [2/3/80e6 2/3/80e6];
vline = [-1 2];
plot(t1,vline,'red',t2,vline,'red')
```

E. Rekonstruktions-Filter und Anti-Aliasing-Filter

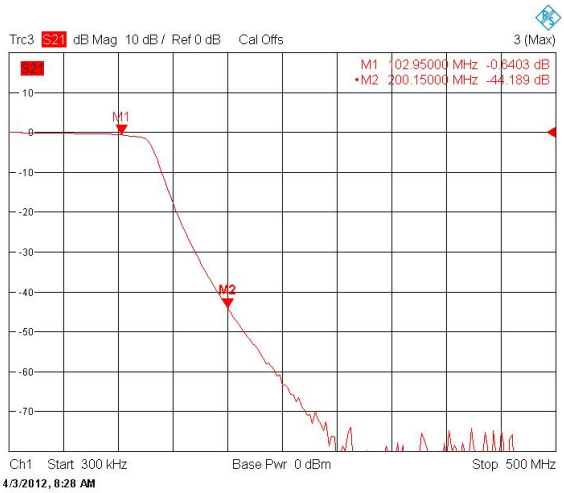
Listing E.1: sinc_effect.m

```
%titel: sinc_effect.m
%autor: Sönke Appel
%Date: 16.12.2011

clear all;
fa = 1;
f = [-2*fa:0.001:2*fa]; %Frequenzvektor

%Signalvektor
for k=1:length(f)
    if (f(k)<-fa/2)
        sig(k)=0;
    elseif (f(k)<= fa/2)
        sig(k)=1;
    else
        sig(k)=0;
    end
end
%Sinc-Vector
sinc_sig = sinc(f./fa);
subplot(4,1,1)
plot(f,sig,f,sinc_sig)
xlabel('f/f_a')
grid on;
AXIS([-2 2 -0.3 1.1])

subplot(4,1,2)
plot(f,sig.*sinc_sig)
grid on;
xlabel('f/f_a')
AXIS([-2 2 -0.3 1.1])

```

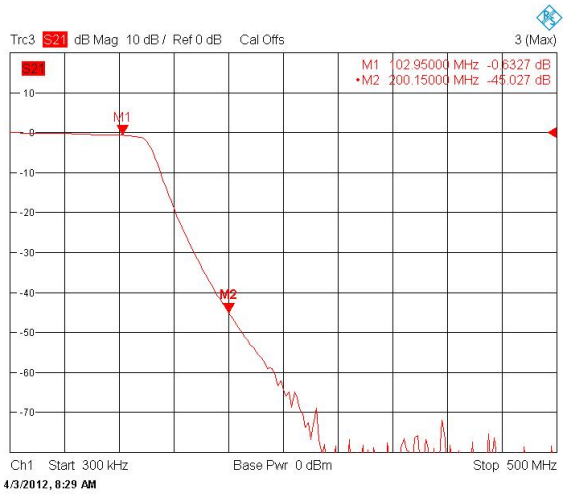
```
33 for k=1:length(f)
34     if (f(k)<-fa/2/4)
35         sig2(k)=0;
36     elseif (f(k)<= fa/2/4)
37         sig2(k)=1;
38     else
39         sig2(k)=0;
40     end
41 end
42
43 subplot(4,1,3)
44 plot(f,sig2,f,sinc_sig)
45 xlabel('f/f_a')
46 grid on;
47 AXIS([-2/4 2/4 -0.1 1.1])
48
49 subplot(4,1,4)
50 plot(f,sig2.*sinc_sig)
51 grid on;
52 xlabel('f/f_a')
53 AXIS([-2/4 2/4 -0.1 1.1])
```



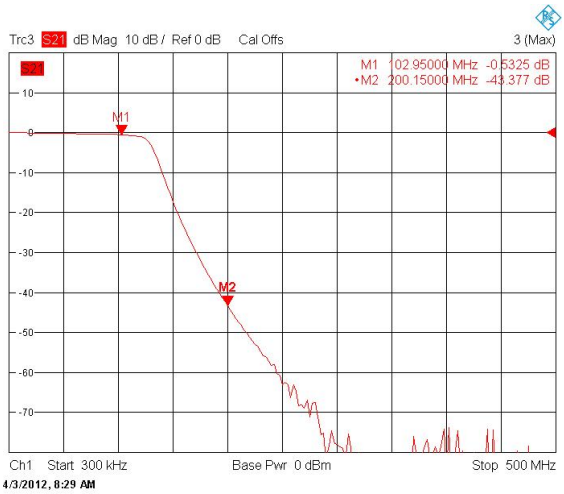
(a) Amplitudengang des 1. Rekonstruktionsfilter



(b) Amplitudengang des 2. Rekonstruktionsfilter

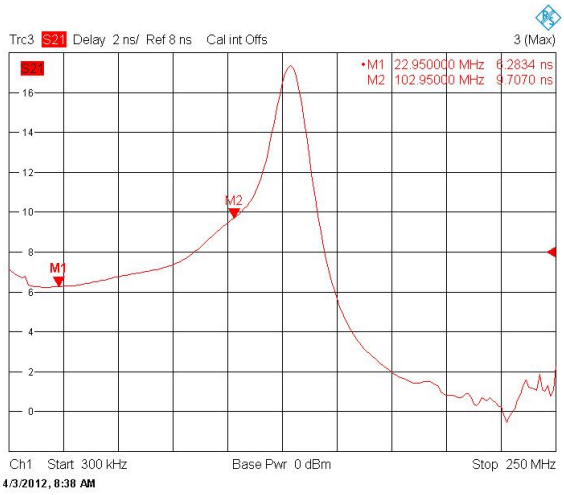


(c) Amplitudengang des 3. Rekonstruktionsfilter

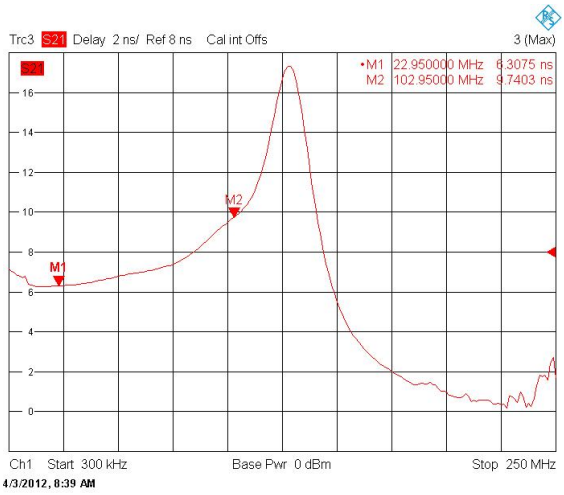


(d) Amplitudengang des 4. Rekonstruktionsfilter

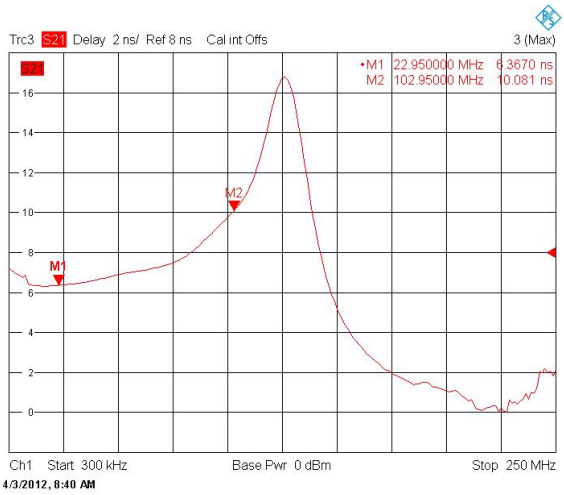
Bild E.1.: Amplitudengänge der Rekonstruktionsfilter



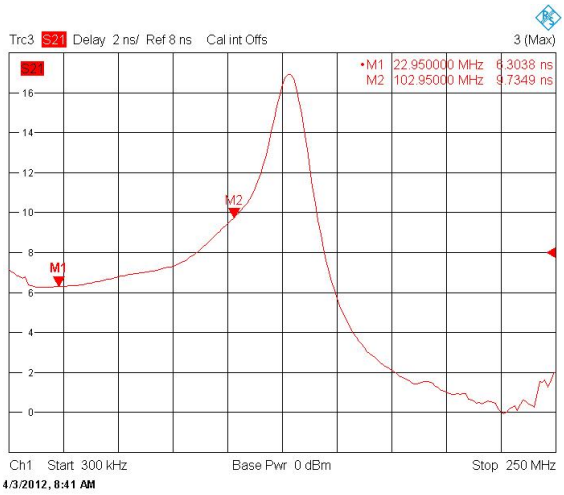
(a) Gruppenlaufzeiten des 1. Rekonstruktionsfilter



(b) Gruppenlaufzeiten des 2. Rekonstruktionsfilter



(c) Gruppenlaufzeiten des 3. Rekonstruktionsfilter



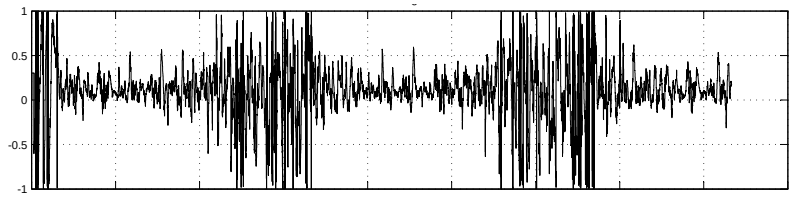
(d) Gruppenlaufzeiten des 4. Rekonstruktionsfilter

Bild E.2.: Gruppenlaufzeiten der Rekonstruktionsfilter

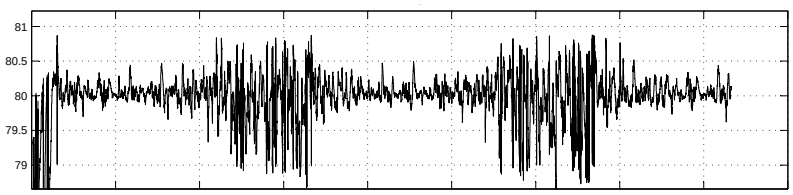
F. Takt- und Datenrückgewinnung

Listing F.1: pll_sim.m

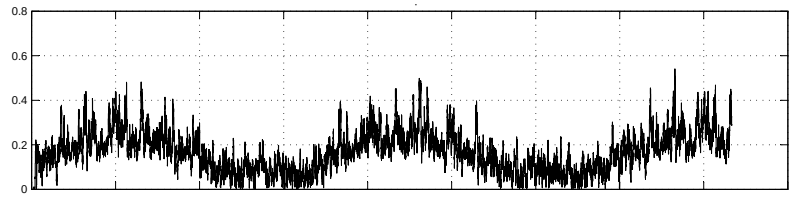
```
% File: pll_sim.m
% Autor: Sönke Appel
% Date: 21.03.2012

clear all;
lauf = 1;
for P=6:9
    for I=13:16
        [t phase freq] = sim('pll_complex_einstellung', [0
            2000/240e6]);
        y1(:,I-12,P-5) = phase;
        zwischen = freq(:,2);
        y2(:,I-12,P-5) = zwischen;
        beschriftung(lauf) =  cellstr(strcat('P=',int2str(P),'␣I='
            ,int2str(I)));
        lauf = lauf+1;
    end
end
figure(1)
plot(t,y1(:,:,1),t,y1(:,:,2),t,y1(:,:,3),t,y1(:,:,4))
legend(beschriftung)
grid on;
figure(2)
plot(t,y2(:,:,1),t,y2(:,:,2),t,y2(:,:,3),t,y2(:,:,4))
legend(beschriftung)
grid on;
```



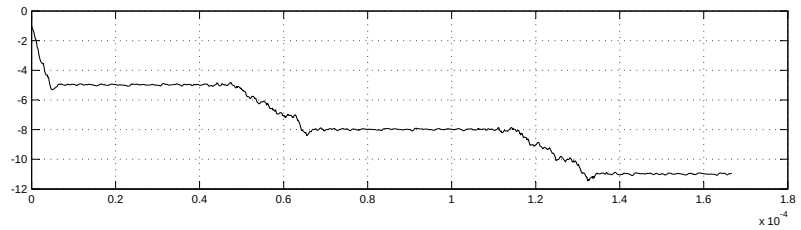
(a) Regelabweichung $[\theta_{\mathrm{ref}} - \theta_{\mathrm{O}}]$



(b) Stellgröße [MHz]



(c) Amplitude des Referenzsignals



(d) Normierte Taktabweichung

Bild F.1.: Test der PLL mit den Regelparametern $P = 2^{-8}$ und $I = 2^{-16}$

G. Implementierung der Modulationsarten

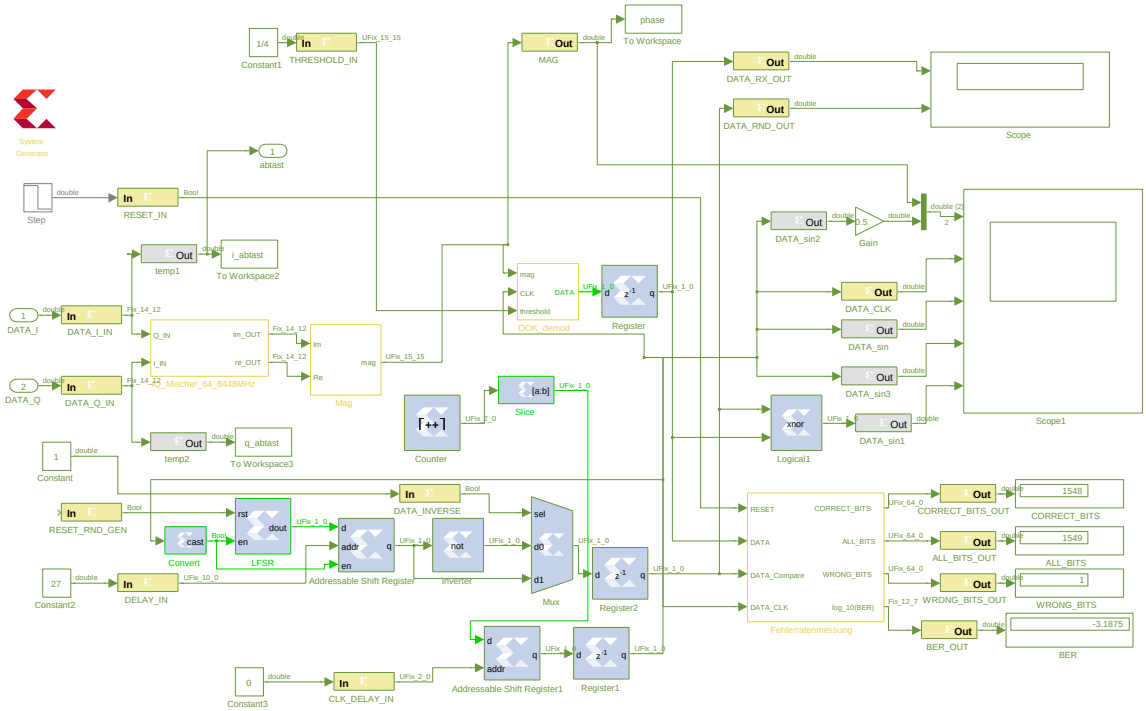


Bild G.1.: System Generator-Modell des OOK-Receivers

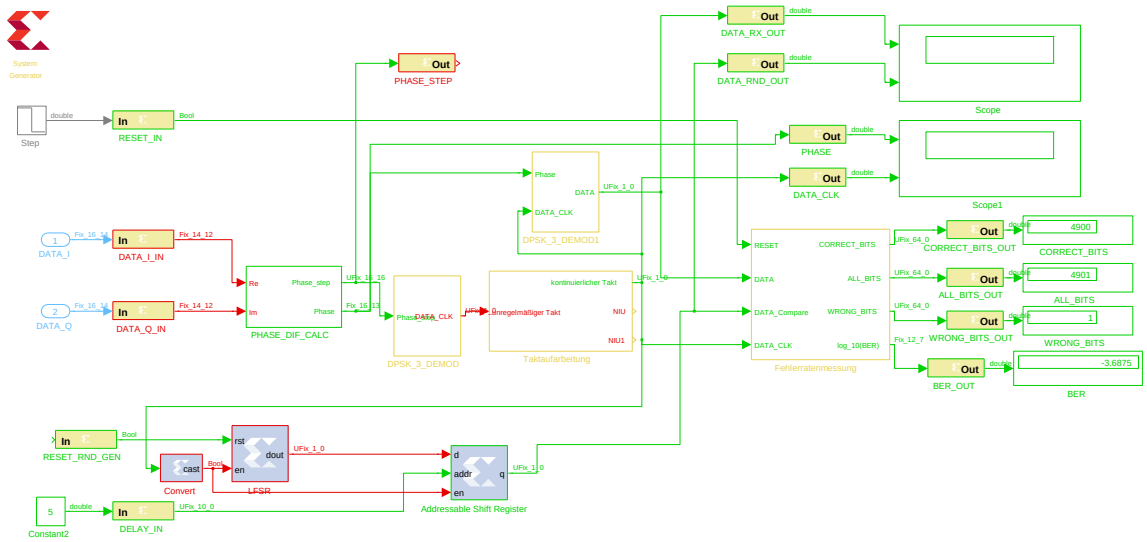


Bild G.2.: System Generator-Modell des DPSK-Receivers

H. Top-Level-Designs

Listing H.1: TRANSMITTER.vhd

```
1  --
       ----------------------------------------------------------------------------------
2  -- Company:     Siemens AG
3  -- Engineer:    Sönke Appel
4  --
5  -- Create Date:     13:17:32 11/21/2011
6  -- Design Name:    TRANSMITTER.vhd
7  -- Module Name:     TRANSMITTER - Behavioral
8  -- Project Name:   Millilink
9  --
       ----------------------------------------------------------------------------------
10 library IEEE;
11 use IEEE.STD_LOGIC_1164.ALL;
12 library UNISIM;
13 use UNISIM.vcomponents.all;
14 use work.DELAY_VALUE.all;
15
16 entity TRANSMITTER is
17     Port ( DATA_1_OUT : out  STD_LOGIC_VECTOR (15 downto 0);
18            DATA_2_OUT : out  STD_LOGIC_VECTOR (15 downto 0);
19            DATA_CLK_P_IN : in  STD_LOGIC;
20            DATA_CLK_N_IN : in  STD_LOGIC;
21            SYSCLK_IN : in  STD_LOGIC;
22            LED_OUT: out STD_LOGIC_VECTOR(1 downto 0);
23            PB_1_IN: in STD_LOGIC;
24           -- SMA_CONNECTOR_J24 : out STD_LOGIC;
25            SMA_CONNECTOR_J26 : out STD_LOGIC;
26            SMA_CONNECTOR_J27 : out STD_LOGIC);
27
28 end TRANSMITTER;
29
30 architecture Behavioral of TRANSMITTER is
```

```
31
32 --Komponenten Deklaration
33 COMPONENT DAC_INTERFACE
34    generic (
35       DATA_1_DELAY: type_delay_value;
36       DATA_2_DELAY: type_delay_value;
37       DATA_CLK_DELAY: integer range 0 to 63);
38    PORT(
39       DATA_CLK_P_IN : IN std_logic;
40       DATA_CLK_N_IN : IN std_logic;
41       SYSCLK_IN : IN std_logic;
42       RESET_IN : IN std_logic;
43       DATA_1_IN : IN std_logic_vector(15 downto 0);
44       DATA_2_IN : IN std_logic_vector(15 downto 0);
45       DATA_1_OUT : OUT std_logic_vector(15 downto 0);
46       DATA_2_OUT : OUT std_logic_vector(15 downto 0);
47       SYSCLK_OUT : OUT std_logic;
48       DATA_CLK_OUT : OUT std_logic;
49       SYS_CLK_DCM_LOCKED_OUT : OUT std_logic;
50       DATA_CLK_DCM_LOCKED_OUT : OUT std_logic
51       );
52 END COMPONENT;
53
54 signal             SYS_CLK_DCM_LOCKED_SIG : STD_LOGIC;
55 signal             DATA_CLK_DCM_LOCKED_SIG : STD_LOGIC;
56
57
58 signal           DATA_1_SIG : STD_LOGIC_VECTOR (15 downto 0);
59 signal           DATA_2_SIG :  STD_LOGIC_VECTOR (15 downto 0);
60 signal           DATA_CLK_SIG : STD_LOGIC;
61 signal           SYSCLK_SIG : STD_LOGIC;
62 signal           RANDOM_REG_SIG : STD_LOGIC_VECTOR (7 downto 0);
63
64
65 signal TOGGLE_SIG: std_logic_vector(15 downto 0);
66 signal GLOBAL_RESET: std_logic;
67 signal RANDOM_REG_RESET_SIG: std_logic;
68
69
70 --COMPONENT transmitter_mcw
71 -- PORT(
72 --    clk_1 : IN std_logic;
73 --    reset_rnd_gen : IN std_logic;
74 --    data_i_out : OUT std_logic_vector(15 downto 0);
```

```
75  --    data_q_out : OUT std_logic_vector(15 downto 0)
76  --    );
77  --END COMPONENT;
78
79     COMPONENT ook_transmitter_mcw
80     PORT(
81        clk_1 : IN std_logic;
82        reset_rnd_gen : IN std_logic;
83        data_i_out : OUT std_logic_vector(15 downto 0);
84        data_q_out : OUT std_logic_vector(15 downto 0)
85        );
86     END COMPONENT;
87
88
89
90  begin
91  GLOBAL_RESET <= PB_1_IN;
92
93  --Instanziierung der Komponenten
94
95  --DPSK_3_inst: transmitter_mcw PORT MAP(
96  --    clk_1 => DATA_CLK_SIG,
97  --    reset_rnd_gen => RANDOM_REG_RESET_SIG,
98  --    data_i_out => DATA_1_SIG,
99  --    data_q_out => DATA_2_SIG
100 -- );
101
102 Inst_ook_transmitter_mcw: ook_transmitter_mcw PORT MAP(
103       clk_1 => DATA_CLK_SIG,
104       reset_rnd_gen => RANDOM_REG_RESET_SIG,
105       data_i_out => DATA_1_SIG,
106       data_q_out => DATA_2_SIG
107    );
108
109
110 RANDOM_REG_RESET_SIG <= not DATA_CLK_DCM_LOCKED_SIG;
111
112
113
114 OBUF_SYSCLK: OBUF
115    generic map (
116       DRIVE => 12,
117       IOSTANDARD => "DEFAULT",
118       SLEW => "FAST")
```

```
119     port map (
120        O => SMA_CONNECTOR_J26,
121        I => SYSCLK_SIG);
122
123
124 OBUF_DATA_CLK: OBUF
125     generic map (
126        DRIVE => 12,
127        IOSTANDARD => "DEFAULT",
128        SLEW => "FAST")
129     port map (
130        O => SMA_CONNECTOR_J27,
131        I => DATA_CLK_SIG);
132
133
134 --Inst_SC_TESTING_MODULE: SC_TESTING_MODULE PORT MAP(
135 --     DATA_1_OUT => DATA_1_SIG,
136 --     DATA_2_OUT => DATA_2_SIG,
137 --     SYS_CLK_DCM_LOCKED_IN => SYS_CLK_DCM_LOCKED_SIG,
138 --     DATA_CLK_DCM_LOCKED_IN => DATA_CLK_DCM_LOCKED_SIG
139 -- );
140
141
142
143
144 Inst_DAC_INTERFACE: DAC_INTERFACE
145     generic map(
146        DATA_1_DELAY  => (others => 0),
147        DATA_2_DELAY  => (others => 0),
148        DATA_CLK_DELAY  => 0)
149     PORT MAP(
150        DATA_1_OUT => DATA_1_OUT,
151        DATA_2_OUT => DATA_2_OUT,
152        DATA_CLK_P_IN => DATA_CLK_P_IN,
153        DATA_CLK_N_IN => DATA_CLK_N_IN,
154        SYSCLK_IN => SYSCLK_IN,
155        SYSCLK_OUT => SYSCLK_SIG,
156        RESET_IN => GLOBAL_RESET,
157        DATA_1_IN => DATA_1_SIG,
158        DATA_2_IN => DATA_2_SIG,
159        DATA_CLK_OUT => DATA_CLK_SIG,
160        SYS_CLK_DCM_LOCKED_OUT => SYS_CLK_DCM_LOCKED_SIG,
161        DATA_CLK_DCM_LOCKED_OUT => DATA_CLK_DCM_LOCKED_SIG);
162
```

```
163
164 LED_OUT(1) <= DATA_CLK_DCM_LOCKED_SIG;
165 LED_OUT(0) <= SYS_CLK_DCM_LOCKED_SIG;
166
167 end Behavioral;
```

Listing H.2: RECEIVER.vhd

```
1  -- Company:     Siemens AG
2  -- Engineer:    Sönke Appel
3  --
4  -- Create Date:     12:35:08 11/14/2011
5  -- Design Name:    RECEIVER.vhd
6  -- Module Name:     RECEIVER - Behavioral
7  -- Project Name:   Millilink
8  --
   ----------------------------------------------------------------------------

9  library IEEE;
10 use IEEE.STD_LOGIC_1164.ALL;
11
12 entity RECEIVER is
13     Port ( SEN : out  STD_ULOGIC;
14            SCLK : out  STD_ULOGIC;
15            SDATA : out  STD_LOGIC;
16            I_START_PB : in bit;
17            I_START_PB_TRIG : in STD_LOGIC_VECTOR ( 0 downto 0 );
18            DATA_B_IN_P : in  STD_LOGIC_VECTOR (6 downto 0);
19            DATA_B_IN_N : in  STD_LOGIC_VECTOR (6 downto 0);
20            DATA_A_IN_P : in  STD_LOGIC_VECTOR (6 downto 0);
21            DATA_A_IN_N : in  STD_LOGIC_VECTOR (6 downto 0);
22            CLK_IN_P : in  STD_LOGIC;
23            CLK_IN_N : in  STD_LOGIC;
24            SYSCLK_IN : in  STD_LOGIC;
25            RESET_IN : in bit;
26            RESET_BER_IN : in STD_LOGIC;
27            DATA_RX_OUT : out STD_LOGIC;
28            DATA_RND_OUT : out STD_LOGIC;
29            SYSCLK_IN_EN : in bit;
30            LED_OUT: out STD_LOGIC_VECTOR (7 downto 0));
31 end RECEIVER;
32
33 architecture Behavioral of RECEIVER is
34
35 --Komponeneten Deklaration
```

```
36 component ANALYSER
37   PORT (
38     CONTROL : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0);
39     CLK : IN STD_LOGIC;
40     DATA : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
41     TRIG0 : IN STD_LOGIC_VECTOR(0 TO 0));
42 end component;
43
44 component ANALYSER2
45   PORT (
46     CONTROL : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0);
47     CLK : IN STD_LOGIC;
48     DATA : IN STD_LOGIC_VECTOR(13 DOWNTO 0);
49     TRIG0 : IN STD_LOGIC_VECTOR(0 TO 0));
50 end component;
51
52 component ANALYSER3
53   PORT (
54     CONTROL : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0);
55     CLK : IN STD_LOGIC;
56     DATA : IN STD_LOGIC_VECTOR(0 TO 0);
57     TRIG0 : IN STD_LOGIC_VECTOR(0 TO 0));
58
59 end component;
60
61 component ANALYSER_15_BIT
62   PORT (
63     CONTROL : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0);
64     CLK : IN STD_LOGIC;
65     DATA : IN STD_LOGIC_VECTOR(14 DOWNTO 0);
66     TRIG0 : IN STD_LOGIC_VECTOR(0 TO 0));
67
68 end component;
69
70
71 component ADC_INTERFACE
72     Port ( SEN : out  STD_ULOGIC;
73            SCLK : out  STD_ULOGIC;
74            SDATA : out  STD_LOGIC;
75            I_START_PB : in bit;
76            DATA_B_IN_P : in  STD_LOGIC_VECTOR (6 downto 0);
77            DATA_B_IN_N : in  STD_LOGIC_VECTOR (6 downto 0);
78            DATA_A_IN_P : in  STD_LOGIC_VECTOR (6 downto 0);
79            DATA_A_IN_N : in  STD_LOGIC_VECTOR (6 downto 0);
```

```
80              CLK_IN_P : in  STD_LOGIC;
81              CLK_IN_N : in  STD_LOGIC;
82              SYSCLK_IN : in  STD_LOGIC;
83              RESET_IN : in bit;
84              SYSCLK_IN_EN : in bit;
85              LED_OUT: out STD_LOGIC_VECTOR (2 downto 0);
86              DATA_A_OUT    : out std_logic_vector(13 downto 0);
87              DATA_B_OUT    : out std_logic_vector(13 downto 0);
88              CLK_DATA_OUT : out std_logic);
89  end component;
90
91
92  signal            DATA_A_SIG    :  std_logic_vector(13 downto 0);
93  signal            DATA_B_SIG    :  std_logic_vector(13 downto 0);
94  signal            CLK_DATA_SIG :  std_logic;
95
96  component ICON_CS
97    PORT (
98      CONTROL0 : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0);
99      CONTROL1 : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0);
100     CONTROL2 : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0);
101     CONTROL3 : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0);
102     CONTROL4 : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0);
103     CONTROL5 : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0);
104     CONTROL6 : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0);
105     CONTROL7 : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0);
106     CONTROL8 : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0);
107     CONTROL9 : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0);
108     CONTROL10 : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0);
109     CONTROL11 : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0));
110 end component;
111
112 for ADC_INTERFACE_1: ADC_INTERFACE use entity WORK.ADC_INTERFACE(
       Behavioral);
113
114 --Instanziierung der Komponenten
115
116 -- COMPONENT dpsk_3_demod_mcw
117 -- PORT(
118 --    clk_1 : IN std_logic;
119 --    data_i_in : IN std_logic_vector(13 downto 0);
120 --    data_q_in : IN std_logic_vector(13 downto 0);
121 --    delay_in : IN std_logic_vector(9 downto 0);
122 --    reset_in : IN std_logic;
```

```
123 --    reset_rnd_gen : IN std_logic;
124 --    all_bits_out : OUT std_logic_vector(63 downto 0);
125 --    ber_out : OUT std_logic_vector(11 downto 0);
126 --    correct_bits_out : OUT std_logic_vector(63 downto 0);
127 --    data_clk : OUT std_logic;
128 --    data_rnd_out : OUT std_logic;
129 --    data_rx_out : OUT std_logic;
130 --    phase_step : OUT std_logic_vector(15 downto 0);
131 --    phase : OUT std_logic_vector(15 downto 0);
132 --    wrong_bits_out : OUT std_logic_vector(63 downto 0)
133 --    );
134 -- END COMPONENT;
135
136 COMPONENT ook_receiver_mcw
137    PORT(
138       clk_1 : IN std_logic;
139       clk_delay_in : IN std_logic_vector(1 downto 0);
140       data_i_in : IN std_logic_vector(13 downto 0);
141       data_inverse : IN std_logic;
142       data_q_in : IN std_logic_vector(13 downto 0);
143       delay_in : IN std_logic_vector(9 downto 0);
144       reset_in : IN std_logic;
145       reset_rnd_gen : IN std_logic;
146       threshold_in : IN std_logic_vector(14 downto 0);
147       all_bits_out : OUT std_logic_vector(63 downto 0);
148       ber_out : OUT std_logic_vector(11 downto 0);
149       correct_bits_out : OUT std_logic_vector(63 downto 0);
150       data_clk : OUT std_logic;
151       data_rnd_out : OUT std_logic;
152       data_rx_out : OUT std_logic;
153       mag : OUT std_logic_vector(14 downto 0);
154       wrong_bits_out : OUT std_logic_vector(63 downto 0)
155       );
156    END COMPONENT;
157
158
159 component BER_VIO
160   PORT (
161     CONTROL : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0);
162     ASYNC_IN : IN STD_LOGIC_VECTOR(11 DOWNTO 0));
163
164 end component;
165
166
```

```
167 component DISPLAY_VIO
168   PORT (
169     CONTROL : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0);
170     ASYNC_IN : IN STD_LOGIC_VECTOR(63 DOWNTO 0));
171
172 end component;
173
174 component SHIFT_REGISTER_ADDR_VIO
175   PORT (
176     CONTROL : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0);
177     ASYNC_OUT : OUT STD_LOGIC_VECTOR(9 DOWNTO 0));
178
179 end component;
180
181 component VIO_I_1_BIT
182   PORT (
183     CONTROL : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0);
184     ASYNC_OUT : OUT STD_LOGIC_VECTOR(0 TO 0));
185
186 end component;
187
188 component VIO_I_15_BIT
189   PORT (
190     CONTROL : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0);
191     ASYNC_OUT : OUT STD_LOGIC_VECTOR(14 DOWNTO 0));
192
193 end component;
194
195 component VIO_I_2_BIT
196   PORT (
197     CONTROL : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0);
198     ASYNC_OUT : OUT STD_LOGIC_VECTOR(1 DOWNTO 0));
199
200 end component;
201
202
203 signal SHIFT_ADDR_LFDS_SIG : STD_LOGIC_VECTOR(9 DOWNTO 0);
204 signal BER_SIG : STD_LOGIC_VECTOR(11 DOWNTO 0);
205 signal CORRECT_BITS_SIG : STD_LOGIC_VECTOR(63 DOWNTO 0);
206 signal ALL_BITS_SIG : STD_LOGIC_VECTOR(63 DOWNTO 0);
207 signal WROG_BITS_SIG : STD_LOGIC_VECTOR(63 DOWNTO 0);
208
209
```

```
210 signal   CONTROL_SHIFT_REG_ADDR_VIO_1_SIG   :  std_logic_vector(35
        downto 0);
211 signal   CONTROL_BER_DISPLAY_SIG   :  std_logic_vector(35 downto
       0);
212 signal   CONTROL_CORRECT_BITS_DISPLAY_SIG   :  std_logic_vector(35
        downto 0);
213 signal   CONTROL_ALL_BITS_DISPLAY_SIG   :  std_logic_vector(35
       downto 0);
214 signal   CONTROL_WRONG_BITS_SIG   :  std_logic_vector(35 downto 0)
       ;
215 signal   CONTROL_ANALYSER_MAG   :  std_logic_vector(35 downto 0);
216 --signal CONTROL_ANALYSER_PHASE   :  std_logic_vector(35 downto 0)
       ;
217 signal   CONTROL_VIO_I_THRESHOLD   :  std_logic_vector(35 downto
       0);
218 signal   CONTROL_VIO_I_DATA_INVERSE   :  std_logic_vector(35 downto
        0);
219 signal   CONTROL_ANALYSER_DATA_A   :  std_logic_vector(35 downto
       0);
220 signal   CONTROL_ANALYSER_DATA_B   :  std_logic_vector(35 downto
       0);
221 signal   CONTROL_ANALYSER_DATA_RX_CLK   :  std_logic_vector(35
       downto 0);
222 signal   CONTROL_VIO_I_CLK_DELAY   :  std_logic_vector(35 downto 0)
       ;
223
224
225 signal   DATA_ANALYSER_MAG   :  std_logic_vector(14 downto 0);
226 --signal DATA_ANALYSER_PHASE   :  std_logic_vector(15 downto 0);
227 signal DATA_INVERSE_SIG   :  std_logic_vector(0 downto 0);
228 signal THRESHOLD_SIG :  std_logic_vector(14 downto 0);
229
230 signal   DATA_RX_CLK   :  std_logic_vector(0 downto 0);
231 signal   LED_SIG   :  std_logic_vector(2 downto 0);
232 signal   CLK_DELAY_SIG   :  std_logic_vector(1 downto 0);
233 signal   RESET_RND_GEN_SIG   :  std_logic;
234
235 begin
236
237 SHIFT_REG_ADDR_VIO_1_inst : SHIFT_REGISTER_ADDR_VIO
238   port map (
239     CONTROL => CONTROL_SHIFT_REG_ADDR_VIO_1_SIG,
240     ASYNC_OUT => SHIFT_ADDR_LFDS_SIG);
241
```

```
242 BER_DISPLAY_inst : BER_VIO
243   port map (
244     CONTROL => CONTROL_BER_DISPLAY_SIG,
245     ASYNC_IN => BER_SIG);
246
247 CORRECT_BITS_DISPLAY_inst : DISPLAY_VIO
248   port map (
249     CONTROL => CONTROL_CORRECT_BITS_DISPLAY_SIG,
250     ASYNC_IN => CORRECT_BITS_SIG);
251
252 ALL_BITS_DISPLAY_inst : DISPLAY_VIO
253   port map (
254     CONTROL => CONTROL_ALL_BITS_DISPLAY_SIG,
255     ASYNC_IN => ALL_BITS_SIG);
256
257 WRONG_BITS_inst : DISPLAY_VIO
258   port map (
259     CONTROL => CONTROL_WRONG_BITS_SIG,
260     ASYNC_IN => WROG_BITS_SIG);
261
262 INVERSE_DATA_inst : VIO_I_1_BIT
263   port map (
264     CONTROL => CONTROL_VIO_I_DATA_INVERSE,
265     ASYNC_OUT => DATA_INVERSE_SIG);
266
267 THRESHOLD_inst : VIO_I_15_BIT
268   port map (
269     CONTROL => CONTROL_VIO_I_THRESHOLD,
270     ASYNC_OUT => THRESHOLD_SIG);
271
272 CLK_DELAY_inst : VIO_I_2_BIT
273   port map (
274     CONTROL => CONTROL_VIO_I_CLK_DELAY,
275     ASYNC_OUT => CLK_DELAY_SIG);
276
277
278 ICON_CS_1 : ICON_CS
279   port map (
280     CONTROL0 => CONTROL_SHIFT_REG_ADDR_VIO_1_SIG,
281     CONTROL1 => CONTROL_BER_DISPLAY_SIG,
282     CONTROL2 => CONTROL_CORRECT_BITS_DISPLAY_SIG,
283     CONTROL3 => CONTROL_ALL_BITS_DISPLAY_SIG,
284     CONTROL4 => CONTROL_WRONG_BITS_SIG,
285     CONTROL5 => CONTROL_ANALYSER_MAG,
```

```
286     CONTROL6 => CONTROL_VIO_I_THRESHOLD,
287     CONTROL7 => CONTROL_ANALYSER_DATA_A,
288     CONTROL8 => CONTROL_ANALYSER_DATA_B,
289     CONTROL9 => CONTROL_VIO_I_DATA_INVERSE,
290     CONTROL10 => CONTROL_VIO_I_CLK_DELAY,
291     CONTROL11 => CONTROL_ANALYSER_DATA_RX_CLK);
292
293
294 --DPSK_3_DEMOD_inst: dpsk_3_demod_mcw PORT MAP(
295 --     clk_1 => CLK_DATA_SIG,
296 --     data_i_in => DATA_A_SIG,
297 --     data_q_in => DATA_B_SIG,
298 --     delay_in => SHIFT_ADDR_LFDS_SIG,
299 --     reset_in => RESET_BER_IN,
300 --     reset_rnd_gen => RESET_RND_GEN_SIG,
301 --     all_bits_out => ALL_BITS_SIG,
302 --     ber_out => BER_SIG,
303 --     correct_bits_out => CORRECT_BITS_SIG,
304 --     data_clk => DATA_RX_CLK(0),
305 --     data_rnd_out => DATA_RND_OUT,
306 --     data_rx_out => DATA_RX_OUT,
307 --     phase => DATA_ANALYSER_PHASE,
308 --     phase_step => DATA_ANALYSER_PHASE_STEP,
309 --     wrong_bits_out => WROG_BITS_SIG
310 -- );
311
312    Inst_ook_receiver_mcw: ook_receiver_mcw PORT MAP(
313       clk_1 => CLK_DATA_SIG,
314       clk_delay_in => CLK_DELAY_SIG,
315       data_i_in => DATA_A_SIG,
316       data_inverse => DATA_INVERSE_SIG(0), --VIO_in_1bit
317       data_q_in => DATA_B_SIG,
318       delay_in => SHIFT_ADDR_LFDS_SIG,
319       reset_in => RESET_BER_IN,
320       reset_rnd_gen => RESET_RND_GEN_SIG,
321       threshold_in => THRESHOLD_SIG, --VIO_in_15bit
322       all_bits_out => ALL_BITS_SIG,
323       ber_out => BER_SIG,
324       correct_bits_out => CORRECT_BITS_SIG,
325       data_clk => DATA_RX_CLK(0),
326       data_rnd_out => DATA_RND_OUT,
327       data_rx_out => DATA_RX_OUT,
328       mag => DATA_ANALYSER_MAG,
329       wrong_bits_out => WROG_BITS_SIG
```

```
330     );
331
332 RESET_RND_GEN_SIG <= not LED_SIG(1);
333
334 LED_OUT(3) <= '0';
335 LED_OUT(4) <= I_START_PB_TRIG(0);
336 LED_OUT(5) <= To_StdULogic(I_START_PB);
337 LED_OUT(6) <= To_StdULogic (SYSCLK_IN_EN);
338 LED_OUT(7) <= To_StdULogic (RESET_IN);
339
340
341 ADC_INTERFACE_1: ADC_INTERFACE port map(
342             SEN => SEN,
343             SCLK => SCLK,
344             SDATA => SDATA,
345             I_START_PB => I_START_PB,
346             DATA_B_IN_P => DATA_B_IN_P,
347             DATA_B_IN_N => DATA_B_IN_N,
348             DATA_A_IN_P => DATA_A_IN_P,
349             DATA_A_IN_N => DATA_A_IN_N,
350             CLK_IN_P => CLK_IN_P,
351             CLK_IN_N => CLK_IN_N,
352             SYSCLK_IN => SYSCLK_IN,
353             RESET_IN => RESET_IN,
354             SYSCLK_IN_EN => SYSCLK_IN_EN,
355             LED_OUT => LED_SIG,
356             DATA_A_OUT => DATA_A_SIG,
357             DATA_B_OUT => DATA_B_SIG,
358             CLK_DATA_OUT => CLK_DATA_SIG);
359
360 LED_OUT(2 downto 0) <= LED_SIG;
361
362 ANALYSER_MAG : ANALYSER_15_BIT
363   port map (
364     CONTROL => CONTROL_ANALYSER_MAG,
365     CLK => CLK_DATA_SIG,
366     DATA => DATA_ANALYSER_MAG,
367     TRIG0 => I_START_PB_TRIG);
368
369 --ANALYSER_PHASE : ANALYSER
370 --   port map (
371 --     CONTROL => CONTROL_ANALYSER_PHASE,
372 --     CLK => CLK_DATA_SIG,
373 --     DATA => DATA_ANALYSER_PHASE,
```

```
374 --    TRIG0 => I_START_PB_TRIG);
375
376 ANALYSER_DATA_A : ANALYSER2
377   port map (
378     CONTROL => CONTROL_ANALYSER_DATA_A,
379     CLK => CLK_DATA_SIG,
380     DATA => DATA_A_SIG,
381     TRIG0 => I_START_PB_TRIG);
382
383 ANALYSER_DATA_B : ANALYSER2
384   port map (
385     CONTROL => CONTROL_ANALYSER_DATA_B,
386     CLK => CLK_DATA_SIG,
387     DATA => DATA_B_SIG,
388     TRIG0 => I_START_PB_TRIG);
389
390 ANALYSER_DATA_RX_CLK : ANALYSER3
391   port map (
392     CONTROL => CONTROL_ANALYSER_DATA_RX_CLK,
393     CLK => CLK_DATA_SIG,
394     DATA => DATA_RX_CLK,
395     TRIG0 => I_START_PB_TRIG);
396
397 end Behavioral;
```

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 24. Mai 2012	
Ort, Datum	Unterschrift