PAUL SCHERRER INSTITUT

Abteilung Beschleuniger / Konzepte und Entwicklung (ABK)
Sektion Diagnostik, Gruppe Messtechnik (Dr. P.-A. Duperrex)

HOCHSCHULE FURTWANGEN UNIVERSITY HFU

Bachelor-Thesis

Digitale Signalverarbeitung zur Auswertung von Teilchen-Strahlen

Jonas Müller

14. Juli 2011

Betreuender Hochschuldozent:	Prof. Dr. Wolfgang Rülling
Zweitbetreuer der Hochschule:	Prof. Dr. Ekkehard Batzies
Betreuer am PSI:	Dipl. Ing. Ernst Johansen
Tag der Anmeldung:	27.01.2011
Tag der Abgabe:	14.07.2011

Inhaltsverzeichnis

Kurzfassung

Kurzfassung

In dieser Bachelorthesis wird mit Hilfe des Softwaretools Matlab Simulink ein Digital Down Converter (DDC) simuliert, mithilfe des Simulink HDL-Coder VHDL-Code erzeugt, dieser in einer Kosimulation mit Modelsim überprüft und schlussendlich mit Xilinx ISE für ein Virtex-6 FPGA synthetisiert. Der DDC soll dabei ein Signal mit einer Frequenz von etwa 9MHz, welches von einem 16-Bit Analog-Digital-Converter mit einer Sample-Frequenz von 46Msps generiert wird, demodulieren. Ziel ist es, Position und Phasenlage des Protonenstrahls am Protonen-Beschleuniger des PSI zu bestimmen.

Ausgehend vom bestehenden System, welches den ASIC ISL5216 von Intersil verwendet, werden die Funktionsblöcke des DDC Schritt für Schritt nachgebildet und schlussendlich der DDC als ganzes zusammengestellt. Dabei müssen die verwendeten Simulationsblöcke in Simulink HDL-Coder kompatibel sein.

Für die Platz- und Zeitoptimierung der entstehenden FPGA-Implementation wird die verfügbare Hardware berücksichtigt sowie die Möglichkeiten der Softwaretools ausgelotet. Damit wird erreicht, dass 8 DDC-Kanäle im vorgegebenen FPGA mit einer Frequenz von 92.4MHz beim langsamsten, bzw. 102.3MHz beim schnellsten Speedgrade lauffähig sind.

Abstract

In the following bachelor thesis a Digital Down Converter (DDC) is simulated with Matlab Simulink. Out of the simulation, VHDL code for the DDC has been generated with the help of the Simulink HDL-Coder. The automatically generated code could then be verified by a co-simulation in Modelsim and finally synthesized for a Virtex-6 FPGA by using the Xilinx ISE. The DDC serves the purpose of demodulating a 9MHz signal sampled at 46Msps from a 16-bit Analog Digital Converter (ADC). The Down Converter is part of a system which aims to determine the position and phase of a proton beam at the proton accelerator at Paul Scherrer Institut in Switzerland.

Based on the existing system, which uses the Intersil ASIC ISL5216, the functional blocks of the DDC were replicated step-by-step and put together to build the entire DDC in the end. Therefore it was only possible to use HDL-Coder compatible blocks in Simulink.

For timing and resource optimization in the FPGA the code was adapted to the resources of the Virtex-6 FPGA and the optimization possibilities of the software tools were evaluated. With these measures it was possible to make the 8-channel DDC run at a frequency of 92.4MHz (lowest speed grade), or 102.3MHz (highest speed grade).

Vorwort

An dieser Stelle möchte ich allen, welche mich während meiner Bachelorthesis unterstützt haben, meinen gebührenden Dank aussprechen.

Zuerst möchte ich meinem Betreuer am PSI, Dipl. Ing. Ernst Johansen, für seine Tatkräftige Unterstützung danken. Weiter danke ich auch meinem direkten Vorgesetzten Dr. Pierre-Andre Duperrex, welcher meine Arbeit am Institut ermöglicht hat.

Mein Dank geht ebenfalls an meine Betreuer der Hochschule Furtwangen: Prof. Dr. Wolfgang Rülling und Prof. Dr. Ekkehard Batzies.

Tatkräftig zur Seite gestanden und unterstützt haben mich auch meine Eltern, welche mir den Weg während der Bearbeitungszeit frei hielten so dass ich mich voll auf die Thesis konzentrieren konnte.

Nicht zuletzt möchte ich den Personen danken, welche das Korrekturlesen übernommen haben. Dies sind mein Betreuer Dipl- Ing. Ernst Johansen sowie mein Vater.

1 Einleitung

Mit dem Protonenbeschleuniger des Paul Scherrer Instituts PSI[1] werden Protonen auf etwa 236'000 km/s - was beinahe 80% der Lichtgeschwindigkeit entspricht - beschleunigt. Die Aufgabe der Diagnostik-Gruppe am PSI ist es unter anderem, die genaue Position und Phasenlage des Protonenstrahls zu bestimmen.

1.1 Aufgabenstellung

Die vorliegende Arbeit, welche als Bachelor-Thesis von der FH Furtwangen betreut und am Paul Scherrer Institut verfasst wurde, behandelt die Simulation eines Digital Down Converters (DDC) in Matlab Simulink. Dieser wird für die Bestimmung von Position und Phasenlage des gepulsten Teilchenstrahles benötigt und stellt eine Demodulation dieses Hochfrequenzsignales dar. Mit Hilfe des HDL-Coders, welcher MathWorks zur Verfügung stellt, soll aus der Simulation VHDL-Code erzeugt werden, um die Funktion des DDC in einem FPGA implementieren zu können [Mat11l, Mat11o, Mat11m]. Damit soll ein bereits bestehendes System, welches DDCs aus einem ASIC[2] verwendet, ersetzt werden, um eine höhere Flexibilität zu erreichen. Der neue DDC soll funtional mindestens gleich gut sein wie das bestehende System. Neben dem funktionalen Ziel geht es auch darum, aufzuzeigen, welche Möglichkeiten Simulink und der HDL-Coder bietet und wie effizient der damit erzeugte Code ist, konkret wie viel Platz er in einem FPGA benötigt und mit welcher Taktrate dieses lauffähig ist. Die Projektbeschreibung / Aufgabenstellung, welche vor Beginn der Thesis von Ernst Johansen erstellt wurde, ist in Anhang D ersichtlich.

1.2 Pflichtenheft

Aus der Aufgabenstellung und in Rücksprache mit Ernst Johansen wurde folgendes Pflichtenheft definiert. Es besteht aus Pflicht- und Wunschzielen:

Pflichtziele:

- Simulation des DDC in Simulink mit HDL Coder-konformen Blöcken ohne die Umwandlung von Kartesischen- in Polarkoordinaten
- Erzeugen von VHDL-Code mit Hilfe des HDL Workflow Advisers
- Co-Simulation des VHDL-Codes und der Simulink-Simulation zur Überprüfung des automatisch erzeugten Codes

[1] Das Paul Scherrer Institut PSI ist das grösste Forschungszentrum für Natur- und Ingenieurwissenschaften in der Schweiz `http://www.psi.ch`

[2] ISL5216 von Intersil, `http://www.intersil.com/data/fn/fn6013.pdf`

- Optimierung der Simulations- und Umwandlungsparameter so, dass der erzeugte VHDL Code möglichst klein und schnell ist
- Bewertung des Codes auf Realisierbarkeit

Wunschziele:

- Implementierung des CORDIC-Algorithmus zur Umwandlung von kartesischen- in Polarkoordinaten
- 8 DDC Kanäle nehmen maximal 60% der Logik des später verwendeten FPGA[3] in Anspruch und sind mit einer Frequenz von mehr als 100MHz lauffähig (dies, weil das Chip-Framework mit 100MHz läuft und damit eine direkte Implementierung möglich wäre)

Implementierung des CORDIC-Algorithmus bei der Koordinatenwandlung ist nur als Wunschziel aufgeführt, weil unklar ist, wie komplex die Umsetzung ist. Simulink verfügt über keinen Standard-Block, welcher die Umwandlung vom kartesischen ins Polarkoordinatensystem anhand des CORDIC-Algorithmus vornimmt. Laut Simulink Release Note der verwendeten Fixed-Point Toolbox Software Version 3.2 (R2010b) ist der CORDIC-Algorithums erst für Sinus, Cosinus und Exponentialfunktion integriert.

Ein weiteres Wunschziel ist die Auslastung des FPGA und die Taktrate. Da es schwierig ist abzuschätzen, wie effizient der automatisch generierte VHDL-Code sein wird, wurden die Vorstellungen an Platzbedarf und Taktrate nicht als Pflichtziele definiert.

Das später zur Verwendung vorgesehene FPGA-Board ist momentan noch bei einer externen Firma in Entwicklung und daher kann kein direkter Hardwaretest durchgeführt werden. Auch das Framework, welches bei derselben Firma in Entwicklung ist, kann noch nicht in die Synthese des FPGAs einbezogen werden. Daher sollte die Auslastung ohne Framework 60% nicht überschreiten - das Framework benötigt auch noch Platz - und die erreichte Taktrate sollte etwas über 100MHz sein, so dass auch bei einer Verlangsamung des Systems durch hinzufügen des Frameworks die 100MHz erreicht werden.

1.3 Zeitplan

Für eine bessere Übersicht ist der Zeitplan in Anhang A abgebildet. An dieser Stelle möchte ich jedoch einige Anmerkungen zum Zeitplan machen.

Manche Zeitfenster überlappen sich, was bedeutet dass der eine Prozess noch nicht fertig bearbeitet, mit einem anderen aber schon begonnen wurde. Nur bei längeren Bearbeitungspausen eines Punktes wurde dies in der Grafik berücksichtigt.

Die Einarbeitung ins Thema erfolgte vor allem durch das Studium der Beschreibung des momentan eingesetzten Systems mit dessen Dokumentation [Joh10] sowie des Datenblatts des darin eingesetzten DDC [Int07]. Daneben war auch Wissen zum Beschleuniger allgemein nötig, welches über die PSI-Homepage [PSI11c, PSI11b] und auf mündlichem Weg erlangt wurde.

Einarbeitung in Simulink und andere Software: Damit ist das kennenlernen des erstmals verwendeten Simulationstools und der Zusatztools wie HDL-Coder, ModelSim 6.0 und Xilinx ISE

[3] Xilinx Virtex6 xc6vlx130t-1ff1156

12.1 gemeint. Dabei war Ernst Johansen eine grosse Hilfe. Daneben wurden die entsprechenden Software-Dokumentationen und Hilfen sowie das Internet verwendet. [Mat11o, Mat11m]

Beim Modulieren des DDC wurden zuerst die Einzelfunktionen, angefangen mit dem numerischen Oszillator, simuliert und verifiziert. Mit den dadurch gewonnenen Erkenntnissen und Erfahrungen erfolgte danach die Simulation des Gesamtsystems. Die Kosimulation wurde jeweils bei den entsprechenden Schritten vorgenommen und wird nicht speziell aufgeführt.

1.4 Gliederung der Arbeit

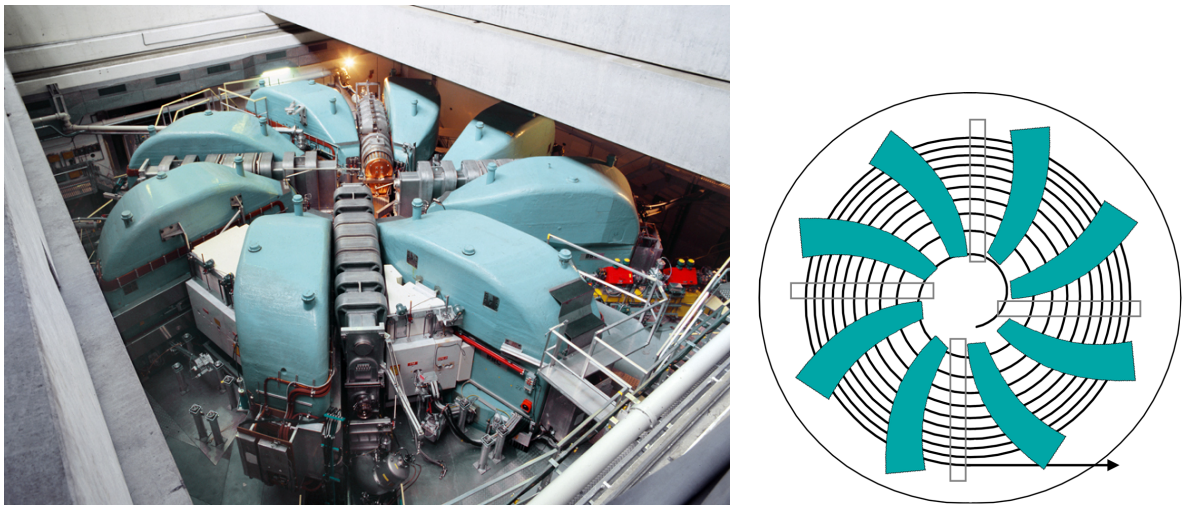
Als nächstes erfolgt im Kapitel Systemkonzept eine Übersicht über das Umfeld. Zuerst wird der Protonenbeschleuniger kurz vorgestellt. Danach erfolgt ein Überblick über das Positionsmessverfahren, wobei auch die Systemabgrenzung für die Thesis an dieser Stelle erfolgt. In Kapitel 3 wird dann das Funktionsprinzip des Digital Down Converters, spezifisch auf die Anwendung am PSI bezogen, erklärt. Zum besseren Verständnis der folgenden Kapitel folgt in Kapitel 4 eine kurze Einführung zu Simulink und wie man daraus VHDL-Code erzeugt und ins FPGA integriert (Workflow). Der folgende Hauptteil DDC in Simulink erklärt zuerst die Teilfunktionen des DDC anhand der verwendeten Simulink Blöcke und schliesst mit dem aus den Teilfunktionen zusammengesetzten Gesamtschaltbild. Wie die Simulation und damit der erzeugte VHDL-Code optimiert werden kann und welche Optionen man beim implementieren des Codes ins FPGA zur Optimierung hat wird in Kapitel 7 erklärt. Des weiteren wird im Kapitel Verifikation beschrieben, wie gut der optimierte Digital Down Converter funktioniert und ob die Resultate überhaupt sinnvoll sind. Im Kapitel Diskussion der Ergebnisse werden die optimierten Resultate kritisch betrachtet und die Performance mit dem momentan verwendeten ASIC ISL5216 verglichen. Zum Abschluss erfolgt eine Zusammenfassung und Ausblick der Ergebnisse sowie ein Ausblick auf die Weiterführung des Projektes.

2 Systemkonzept

In diesem Kapitel soll das Umfeld der Thesis beschrieben werden. Der Überblick soll das Prinzip des Protonenbeschleunigers und des Positionsmessverfahrens erklären, sowie das System abgrenzen und zeigen, was zur Thesis gehört und was nicht.

2.1 Protonenbeschleuniger am PSI

Der Protonenbeschleuniger am PSI ist die weltweit stärkste Anlage dieser Art in seiner Kategorie[1]. Die Protonen, welche in einer Quelle aus Wasserstoffatomen gewonnen werden, werden in drei Beschleunigern auf ihre Endgeschwindigkeit von etwa 236'000 km/s - was beinahe 80% der Lichtgeschwindigkeit entspricht - beschleunigt. Das Herzstück der Anlage ist der dritte, leistungsstärkste Beschleuniger, der die Protonen auf ihre Endgeschwindigkeit und somit auf eine Bewegungsenergie von 590 MeV beschleunigt.



(a) Der grosse Ringbeschleuniger am PSI **(b)** Beschleunigerprinzip

Abbildung 2.1: Protonenbeschleuniger

Dieser Ringbeschleuniger - auch genannt Ringzyklotron - ist in 2.1(a) zu sehen. Gut erkennbar sind die acht grossen, türkisfarbenen Umlenkmagnete sowie die vier grauen, länglichen Kavitäten[2], welche für die Beschleunigung verantwortlich sind. Die zu Strahlpaketen gebündelten Protonen fliegen von einem Umlenkmagnet zum nächsten und somit im Kreis herum. Da die Protonen - durch die Kavitäten beschleunigt - immer schneller werden, wird der Radius der

[1] Ein neuer Weltrekord, 2.4mA Strahlstrom, wurde am 20. Juni 2011 realisiert. Siehe dazu Anhang J

[2] Hohlraumresonator, siehe `http://de.wiktionary.org/wiki/Kavität`

Flugbahn wegen ihrer Masse und der Fliehkraft immer grösser. Die Flugbahn der Protonen beschreibt somit eine Spirale, wie in 2.1(b) dargestellt. Die Strahlpakete werden vom kleineren Zyklotron her in die Mitte des grossen Beschleunigers eingeschossen und verlassen diesen nach 186 Runden zuäusserst mit der genannten Endgeschwindigkeit. [PSI11b, PSI11c, Tru08]

2.2 Positionsmessverfahren

Der Protonenstrahl wird nach Austreten aus dem Beschleuniger an diverse Targets in der Experimentierhalle geleitet. Auf den verschiedenen Strahlwegen - zur Übersicht siehe Abbildung 2.2 (ev. in Anhang) - muss die Position des Strahles überprüft werden. Insbesondere geschieht dies nach dem Durchlaufen von Ablenkmagneten, um deren Funktion zu verifizieren.

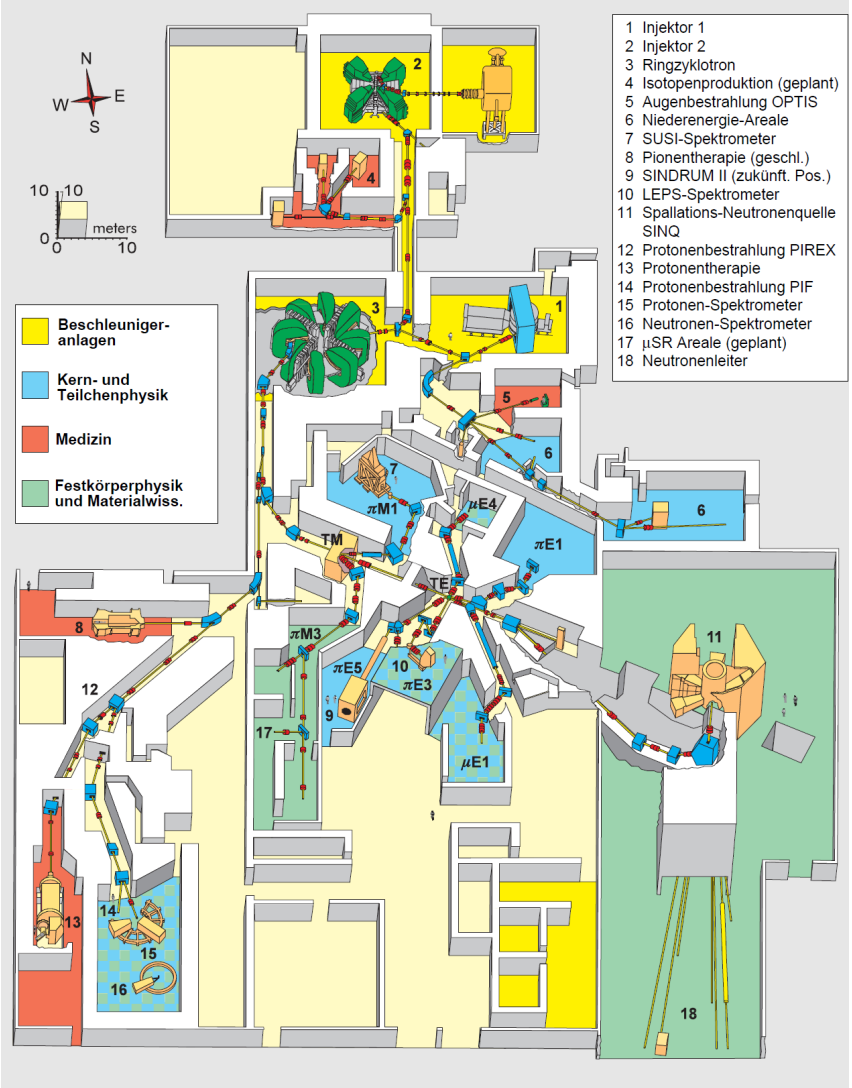
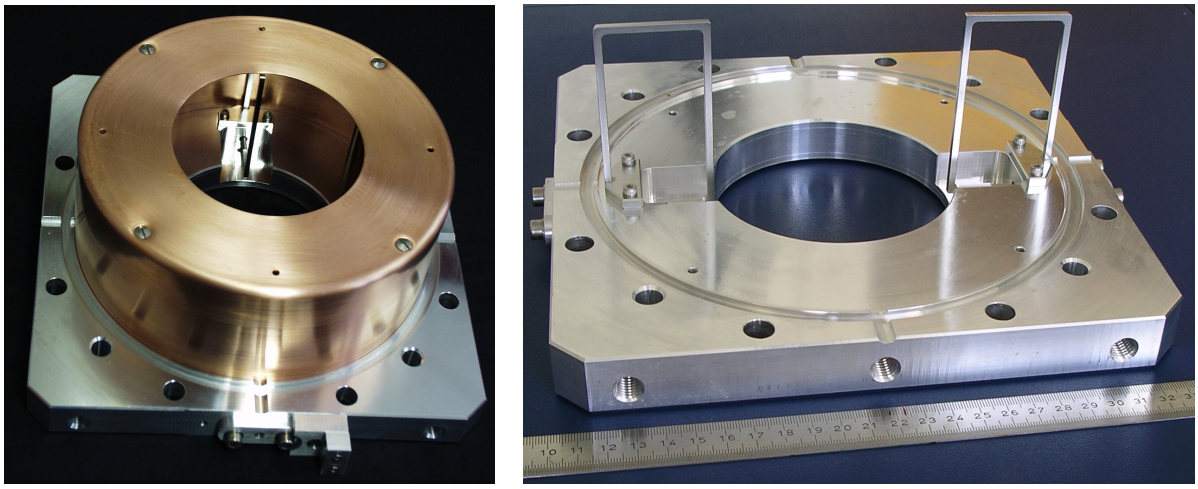


Abbildung 2.2: Übersicht über die Experimentierhalle

Zur Positionsmessung sind in der Anlage Luftspulen installiert, in welchen beim Vorbeiflug der positiv geladenen Protonen eine Spannung induziert wird. Diese Spannung hängt unter anderem

davon ab, in welcher Entfernung die Protonen vorbeigeflogen sind, womit sich die Position des Strahls bestimmen lässt.



(a) mit Kupferabdeckung **(b)** ohne Kupferabdeckung

Abbildung 2.3: Messpulen im Strahlweg

In Abbildung 2.3 ist die Anordnung der Spulen im Strahlweg dargestellt. Die Spulen bestehen aus nur einer Windung und sind aus einer Aluminiumlegierung hochpräzise gefräst, damit sie alle dieselben mechanischen Abmessungen und somit dieselben elektrischen Eigenschaften haben. Wie in 2.3(b) ersichtlich, befinden sich zwei solcher Messspulen auf einer Aluminiumplatte. Daraus folgt, dass zwei solche Platten hintereinander montiert werden müssen, um sowohl X- wie auch Y-Koordinate zu erhalten.

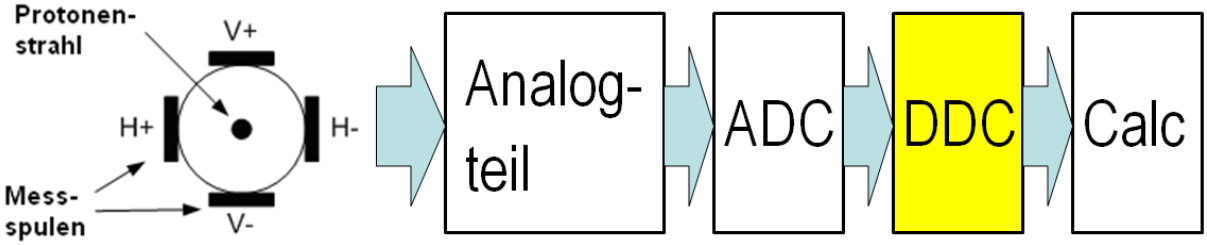


Abbildung 2.4: Übersicht über das eingesetzte Messsystem

Die Abbildung 2.4 zeigt eine Übersicht über das gesamte am PSI eingesetzte Messsystem zur Bestimmung der Position des Protonenstrahls. Die in den vier Messspulen aufgenommene Spannung - sie hat eine Frequenz von 101.2656MHz, was der ersten Oberwelle der Frequenz der Protonenpakete entspricht[3] - gelangt zuerst in einen Analogteil. In diesem wird zum Strahlsignal ein Referenzsignal hinzuaddiert, welches eine um 90kHz tiefere Frequenz hat, also 101.1756MHz. Das Pilotsignal dient als Referenz, um Verluste in Kabeln und Steckern wegrechnen zu können.

Das Signal wird dann in einem Variable Gain Amplifier (VGA) - also einem verstellbaren Verstärker - in Abhängigkeit der Strahlintensität unterschiedlich stark verstärkt. Dies, damit der Wan-

[3]Die Grundfrequenz wird darum nicht verwendet, weil die Speisung der Kavitäten mit einem Rechteck dieser Frequenz erfolgt und somit bei der Grundfrequenz und allen ungeraden harmonischen zu starke Störsignale auftreten.

delbereich des nachfolgenden Analog-Digital-Converter (ADC) möglichst gut ausgenutzt werden kann.

Die Analog-Digital-Wandlung erfolgt mit einer Samplerate von 46.000Msps, woraus schliesst, dass das Prinzip des undersampling angewendet wird. Das Nyquist-Shannon-Abtasttheorem wird somit nicht erfüllt [Wik11h]. Da die Frequenz des zu wandelnden Signals aber bekannt und das Frequenzband - unter anderem wegen der oben erwähnten analogen Filter - sehr begrenzt ist, entsteht trotzdem kein Informationsverlust. [Wik11k]

Das undersampling führt dazu, dass das digitale Strahlsignal eine Frequenz von 9.2656MHz hat (101.2656MHz – 2 x 46.000MHz). Das Referenzsignal weist digitalisiert demzufolge eine Frequenz von 9.1756MHz auf, was immer noch 90kHz tiefer ist als die Strahlsignalfrequenz.

(Eine detailliertere Erklärung des Prinzips ist in [Joh10] gegeben.)

Nach der Digitalisierung des Signals wird dieses im Digital Down Converter auf DC heruntergemischt. Damit ist es möglich, Phase und Amplitude des Signals zu bestimmen. Da vier Strahlsignale und vier Referenzsignale verarbeitet werden müssen, sind insgesamt acht DDC's nötig. Eine genauere Beschreibung des DDC erfolgt in Kapitel 3.

Im letzten Block „Calc“ der Abbildung 2.4 wird das Strahlsignal zuerst mithilfe der Referenzsignale normalisiert, worauf die abschliessende Positionsberechnung folgt.

Die Bestimmung der Phase ist nicht direkt nötig für die Bestimmung der Strahlposition. Die Phase wird allerdings gebraucht, um die in Abschnitt 2.1 beschriebenen Kavitäten steuern zu können, damit diese den Protonenpaketen immer im richtigen Moment den nächsten Schub versetzen.

2.3 Systemabgrenzung für die Thesis

Im Umfang dieser Thesis wird der in Abbildung 2.4 gelb markierte Teil „DDC“ behandelt. Das Eingangssignal ist dabei ein Signal bestehend aus einem 9.2656MHz- einem 9.1756MHz-Anteil und etwas Rauschen. Als Ausgangsgrösse sollen Amplitude und Phase vorliegen. Wie oben erwähnt werden insgesamt acht DDC gebraucht. Deshalb sollen im FPGA acht DDC parallel implementiert werden.

Die Positionsberechnung erfolgt nicht im FPGA, da hierfür eine Division nötig ist. Dies stellt für ein FPGA eine schwierige Aufgabe dar, weshalb dafür ein Mikrocontroller verwendet wird.

3 Digital Down Converter

In diesem Kapitel soll die Funktionsweise des Digital Down Converters oder kurz DDC erklärt werden. Dabei wird im speziellen auf den am PSI verwendeten DDC im ASIC ISL5216 von Intersil und dessen Konfiguration eingegangen.

Generell gibt es zwei Klassen von DDC's, Breitband und Schmalband. Da in der Anwendung am PSI, welche in Abbildung 3.1 ersichtlich ist und die es in Simulink nachzubilden gibt, die Ausgangssamplerate mit 50ksps 920mal tiefer ist als die Eingangssamplerate von 46Msps und somit die Dezimation 920 ist, handelt es sich hier um einen Schmalband-DDC. [Eng11]

Die Down Conversion beruht auf dem Prinzip, dass bei einer Multiplikation von zwei sinusförmigen Schwingungen jeweils Signale mit der Summe und der Differenz derer Frequenzen entstehen [Kar10, S. 220]. Dies ist am Beispiel in Gleichung 3.1 an der Multiplikation von zwei Cosinus ersichtlich. [Wik11c]

$$\cos x \cos y = \frac{1}{2}\Big(\cos(x-y) + \cos(x+y)\Big) \tag{3.1}$$

Dieses Prinzip ist auch aus der Amplitudenmodulation bekannt. [Mie11]

Um das Eingangssignal auf DC herunter zu transformieren, wird also eine sinusförmige Schwingung mit derselben Frequenz benötigt. Diese wird in einem Numerically Controlled Oscillator (NCO) generiert, bei welchem wie in Abbildung 3.1 ersichtlich die gewünschte Frequenz anhand einer Konstanten eingestellt werden kann. Mehr zum NCO in Abschnitt 6.1.

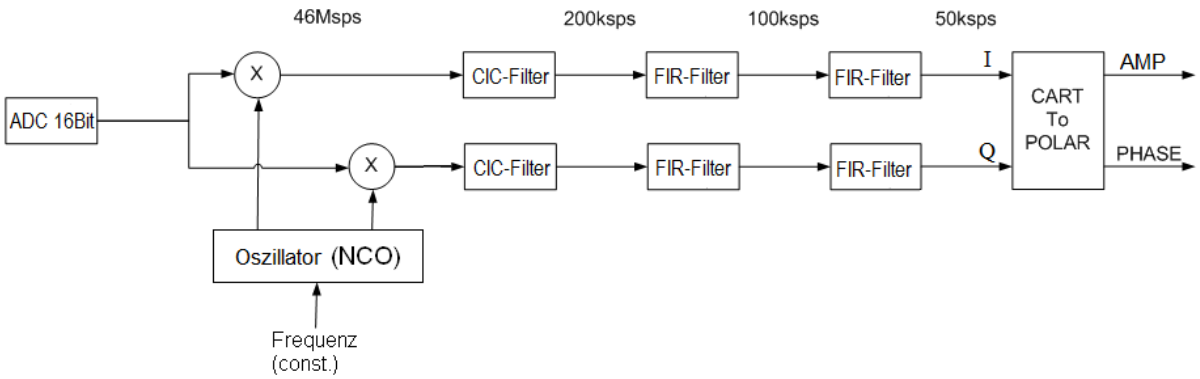


Abbildung 3.1: Prinzipschaltbild des Digital Down Converters

Da die Phasenlage des Eingangssignals nicht bekannt ist, erfolgt die Multiplikation komplex mit Cosinus und -Sinus, analog zum IQ-Verfahren. [Wik11d]

Da dabei auch die Summe der Frequenzen sowie Oberwellen entstehen und ausserdem die Rauschleistung so klein wie möglich sein sollte, erfolgt anschliessend an die Multiplikation eine Tiefpassfilterung sowohl für I- wie auch Q-Signal. Dazu eignet sich die Anordnung eines CIC-Filters

gefolgt von zwei FIR-Filtern besonders gut, um Hardwareeffizient und mit grosser Güte zu filtern, wobei die Filter wie oben erwähnt um den Faktor 920 dezimieren. Zum Schluss soll aus I- und Q-Signal, was kartesischen Koordinaten entspricht, die Amplitude und die Phase, sprich Polarkoordinaten berechnet werden.

4 Einführung Simulink

In diesem Kapitel sollen einige Worte zur Arbeit mit Simulink gesagt werden, damit die folgenden Kapitel besser verständlich sind. Die Einarbeitung in Simulink erfolgte mithilfe von Ernst Johanson, der Simulink-Hilfe und der Simulink Dokumentation wie beispielsweise [Mat11m].

Damit aus einer Simulation in Simulink überhaupt VHDL-Code erzeugt werden kann, sind einige Einstellungen notwendig:

- Im Menü *Simulation -> Configuration Parameters* muss unter *Diagnostics –> Data Validity -> Parameters -> Detect Overflow „warning“* eingestellt werden anstelle von *„error“*.
- Weiter muss bei *Simulation -> Configuration Parameters* der Solver auf *Type:fixed step / Solver: discrete (no continuous states)* ausgewählt werden. *Tasking mode for periodic sample times* muss *SingleTasking* sein.

4.1 Farbliche Markierungen

Jedes Signal in der Simulation muss eine Sample-Frequenz haben, da diskrete Werte und keine kontinuierlichen Signale nötig sind für die Realisierung im FPGA. Die Simulink Blöcke und Verbindungslinien lassen sich farblich so kennzeichnen, dass man direkt die entsprechende Samplefrequenz ablesen kann. Dies geschieht unter *Format -> Samplet Time Display -> Colors*. Die Farbcodierung wird damit auch gleich angezeigt. Ein Beispiel dafür ist in Abbildung 4.1 abgebildet.

4.2 Datentypen

Auch die Datentypen können angezeigt werden, so dass man an den Verbindungslinien den jeweiligen Datentyp ablesen kann. Hierzu wird unter *Format -> Port/Signal Displays -> Colors* das Häckchen bei *Port Data Types* gesetzt.

Neben den Datentypen „double“ (Double-precision floating point) und „int$X$“ bzw. „uint$X$“, Signed- bzw. Unsigned Integers mit X Bit Datenlänge werden vorallem das Format „sfix$X$“ bzw. „sfix$X$_EnY“ verwendet. Das sind Signed Fixed-Point Datentypen mit X Bits und Y Nachkommastellen.

4.3 Simulink Blockschaltbilder

Hier ein Beispiel, wie ein Simulink Blockschaltbild aussehen könnte. In Abbildung 4.1 ist die oberste Ebene der Simulation des DDC ersichtlich.

Auf der linken Seite wird das in Abschnitt 2.3 beschriebene benötigte Eingangssignal aus zwei Sinusgeneratoren und Rauschen zusammengestellt. Die Sample-Rate des Eingangssignal ist $2.1739e{-}008$, was in einer Frequenz von $\frac{1}{2.1739e-008} = 46\text{Msps}$ entspricht (hellgrüne Linien). Der Datentyp wird dann von „double“ gemäss zukünftigen ADC auf „int16“ gewandelt.

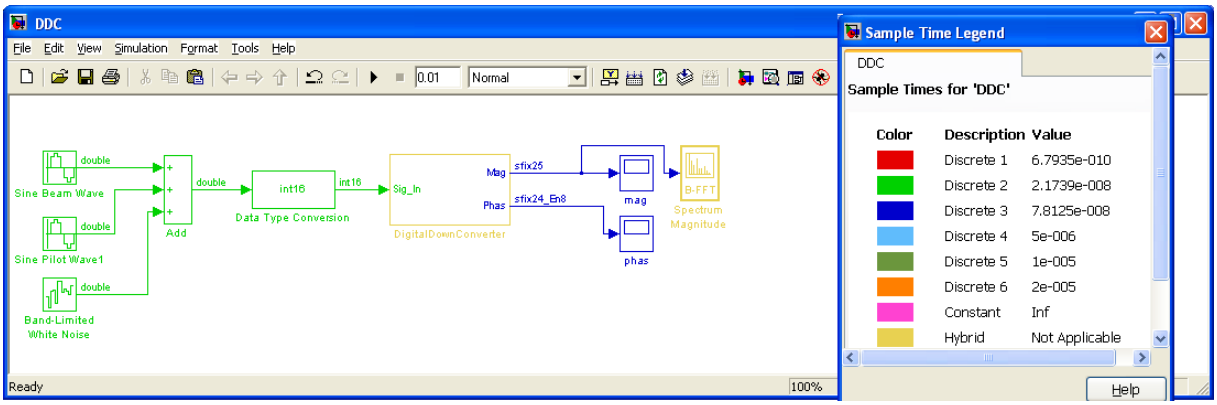


Abbildung 4.1: Blockschaltbild des DDC in Simulink mit Sample-Times und Datentypen

Der DDC selbst ist als ein Block dargestellt und ist senffarbig. Dies kommt daher, dass inerhalb des Blocks verschiedene Frequenzen vorkommen (In der Sample Time Legend durch „Hybrid“ gekennzeichnet).

Amplitude und Phase am Ausgang haben eine Sample-Frequenz von $\frac{1}{7.8125e-008} = 12.8\text{Msps}$ und eine Auflösung von 25 Bit bzw. 24 Bit mit 8 Kommastellen. Von beiden Signalen wird der Zeit- und von der Amplitude der Frequenzverlauf dargestellt, was mit den Blöcken „Scope“ bzw. „Spectrum Scope“ geschieht.

Ausserhalb des DDC-Blockes dürfen alle Simulink-Blöcke verwendet werden, da nur aus dem im DDC-Block befindlichen Blockschaltbild Code erzeugt wird. So sind Sinus- und Rauschgeneratoren sowie Zeit- und Frequenzverlauf nicht HDL-Coder-konform.

5 Workflow

In diesem Kapitel soll gezeigt werden, wie aus einer Simulation in Simulink VHDL-Code erzeugt werden und in das FPGA integriert werden kann. Dazu sind folgende Tools verwendet worden:

- MathWorks MATLAB Version 7.11.0.584 (R2010b) 32-bit (win32)
- Simulink Version 7.6 (R2010b)
- Simulink HDL Coder Version 2.0 (R2010b) und andere Toolboxen, siehe Anhang L
- Mentor Graphics Modelsim SE 6.6b
- Xilinx ISE Design Suite 12.1

Abbildung 5.1 zeigt eine Übersicht über die Arbeitsschritte welche nötig sind, um aus einem Simulink-Model VHDL-Code zu erzeugen und diesen in einem Xilinx-FPGA zu synthetisieren. Dabei wird aus Simulink der HDL Workflow Adviser gestartet, welcher aus dem Simulink-Model VHDL-Code sowie Modelsim-Simulationsfiles erzeugt. Diese können dann mit der Ko-Simulation verglichen werden. Falls das Resultat beider Simulationen identisch ist, kann der VHDL-Code mithilfe von Xilinx ISE für ein FPGA synthetisiert werden.

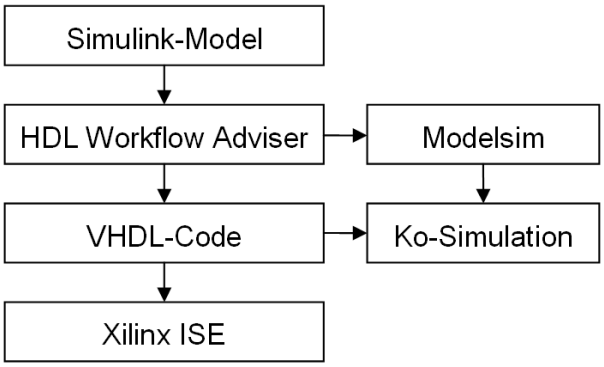


Abbildung 5.1: Workflow zur Einbindung von VHDL-Code aus Simulink-Models

5.1 HDL Workflow Adviser

Wichtig ist, dass das gesamte Blockschaltbild aus welchem VHDL-Code erzeugt werden soll in einem Subsystem ist. Dies wird durch anwählen der gewünschten Blöcke und Rechtsklick *create subsystem* erreicht. VHDL-Code wird erzeugt, indem bei Rechtsklick auf das Subsystem *HDL Coder -> HDL Workflow Adviser* gestartet wird. Der automatisch erzeugte VHDL-Code wird die Bezeichnungen der Simulink-Blöcke enthalten, daher ist es wichtig, diese sinnvoll zu benennen. Ansonsten wird der VHDL-Code sehr schwer lesbar.

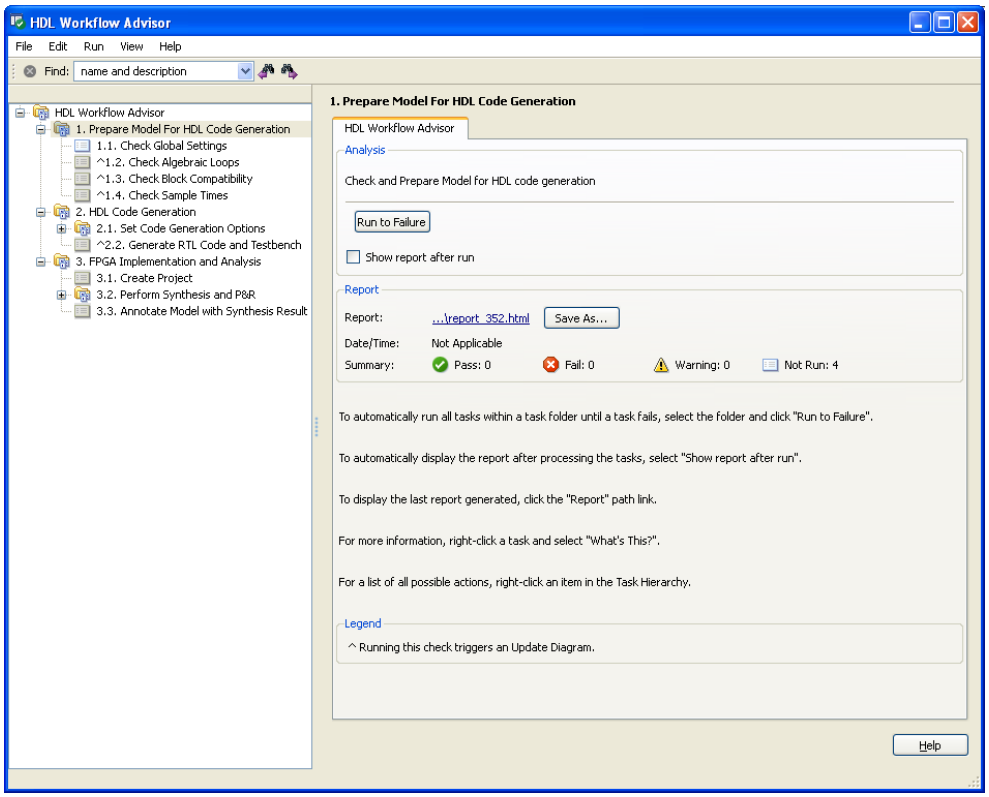


Abbildung 5.2: HDL Workflow Adviser

Die graphische Oberfläche des Workflow Advisers ist in Abbildung 5.2 abgebildet. Mit Click auf *1. Prepare Model For HDL Code Generation* und danach *Run to Failure* wird zuerst das Model überprüft, aus welchem Code erzeugt werden soll. Beim ersten Durchlauf kann der Punkt *1.1 Check Global Settings* nicht abgearbeitet werden. Ein Klick im entsprechenden Hängeregister auf *Modify All* löst das Problem normalerweise.

Grüne Häckchen auf der linken Seite zeigen, dass die entsprechenden Arbeitsschritte erfolgreich durchlaufen worden sind. Falls dies nicht der Fall ist, sind Änderungen im Model nötig. Die auftretenden Fehlermeldungen zeigen meist auf, wo das Problem liegt und warten oft auch mit Lösungsvorschlägen auf. Falls beispielsweise Blöcke im Subsystem verwendet wurden, welche nicht HDL-Coder kompatibel sind, so entsteht unter *1.3 Check Block Compatability* folgende Meldung: *Warning : Cannot find the implementation for block 'BLOCK'.*

Als nächstes wird der VHDL-Code erzeugt. Unter *2.1.1.* muss dafür VHDL gewählt werden. Daneben wird auch Verilog unterstützt. Falls eine Ko-Simulation stattfinden soll wie in unserem Fall, muss das entsprechende Häckchen, *Generate co-simulation model*, unter *2.2.* gesetzt werden. Danach kann die Code-Erzeugung unter *2. HDL Code Generation -> Run to Failure* gestartet werden.

5.2 Ko-Simulation

Nach der erfolgreichen Generierung des Codes startet automatisch ein neues Simulink-Fenster, welches die Co-Simulation ermöglicht. Als Beispiel ist ein solches in Abbildung 5.3 zu sehen.

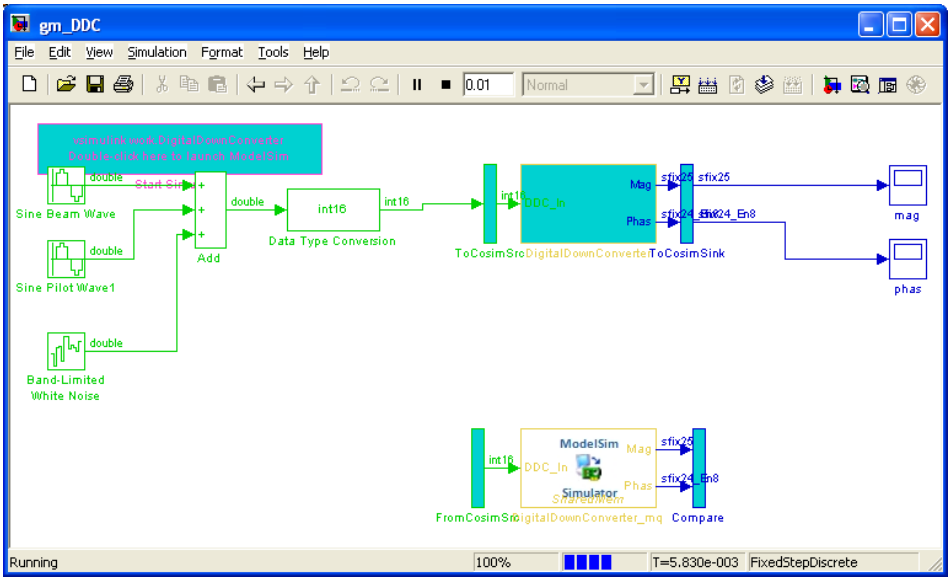


Abbildung 5.3: Ko-Simulation mit Simulink und Modelsim

Zuerst muss Modelsim mit einem Doppelklick auf das Fenster oben links gestartet werden.

Bei Starten der Simulation wird nun das Model in Simulink und der VHDL-Code in Modelsim simuliert und die Resultate werden verglichen. Der *Error* sollte wie im in Abbildung 5.4 abgebildeten Beispiel 0 sein. *cosim* steht dabei für Ko-Simulation, *dut ref* für *Device Under Test reference*, also die Simulink-Referenz. Zur Überprüfung kann der VHDL-Code auch noch von Hand etwas angeschaut werden. Bei Stichprobenartigem dursehen des Codes machte das gesehene Sinn und stimmte von der Struktur her mit dem Model in Simulink vollständig überein.

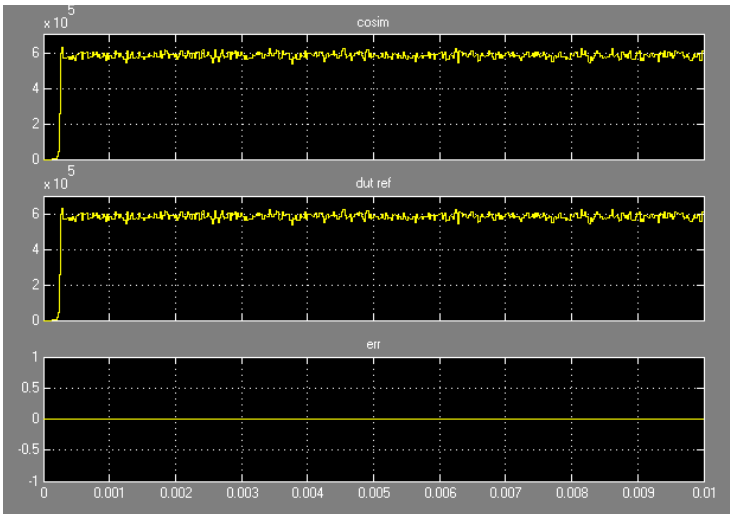


Abbildung 5.4: Vergleich der Simulationen

5.3 Xilinx ISE

Die Implementation des VHDL-Codes ins FPGA erfolgt in zwei Schritten: *Synthesize* und *Implement Design*. Zuerst muss aber ein neues Projekt angelegt werden (*File -> New Project...*), wobei das verwendete FPGA angegeben werden muss. In unserem Fall ist dies das Xilinx Virtex6 xc6vlx130t-1ff1156 mit Speedgrade 1.

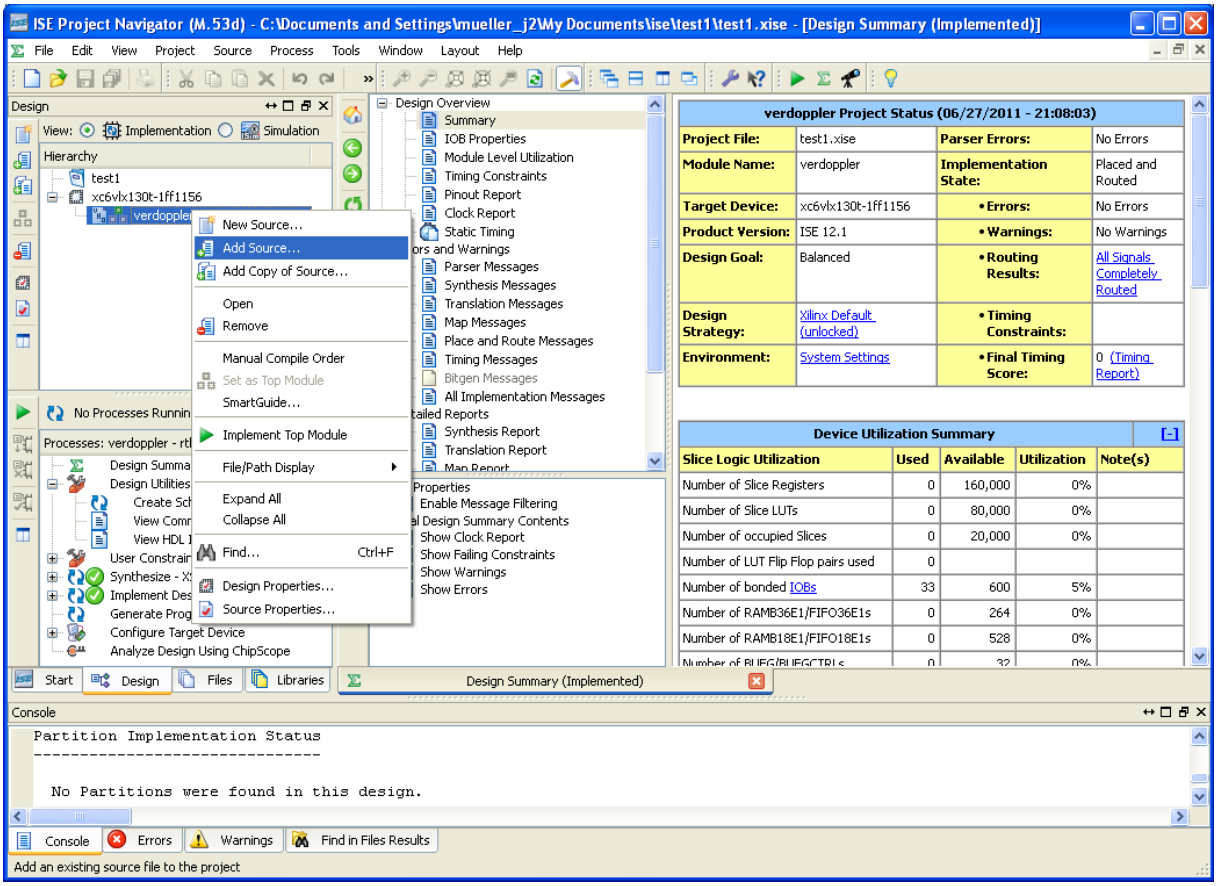


Abbildung 5.5: Übersicht über Xilinx ISE Project Navigator

Abbildung 5.5 zeigt eine Übersicht über den Projekt-Navigator von Xilinx ISE. Nach Anlegen des Projekts müssen mit Rechtsklick auf selbiges die benötigten, eben erzeugten Sourcefiles ins Projekt integriert werden. Danach können die beiden Schritte *Synthesize* und *Implement Design* ausgeführt werden. Läuft alles ohne Fehlermeldungen oder Warnings erscheinen danach zwei grüne Häckchen wie in der Abbildung.

Aus der Konsole unten im Projekt-Navigator kann die minimal benötigte Durchlaufzeit und somit die maximale Frequenz mit der der Code implementiert werden kann abgelesen werden. Bei Klick auf *Design Overview -> Summary* oben in der Mitte erscheint rechts oben der Projekt Status und unten die *Device Utilization Summary*. In letzterer kann abgelesen werden, wieviele Ressourcen für die Implementierung verwendet wurden. Wird nur der erste Schritt *Synthesize* ausgeführt, wird eine Abschätzung für minimale Durchlaufzeit und benötigte Ressourcen ausgegeben.

6 DDC in Simulink

In diesem Kapitel wird erläutert, wie der Digital Down Converter in Simulink realisiert wurde. Dazu werden die Teilfunktionen aus Abbildung 3.1 zuerst einzeln nachgebildet, wobei alle Teilfunktionen HDL-Coder-kompatibel sein müssen, da später VHDL-Code erzeugt werden soll. Bei der Simulation werden die Teilsysteme wie NCO, Filter und kartesisch- zu Polarkoordinatenwandlung einzeln getestet und überprüft. Schlussendlich wurde der DDC aus den Teilblöcken zusammengesetzt und die Gesamtfunktion überprüft (Kapitel 8).

Die Hilfe zur „DSP System Toolbox“ enthält viele nützliche Informationen zum Thema sowie Beispiele, welche die Einarbeitung in die Problematik vereinfachen [Mat11b, Mat11d, Mat11c]. Weiter wurde auch ein nur in der Matlabhilfe verfügbares DDC-Model als Beispiel herangezogen [Mat11g, Matlabhilfe: GSM Digital Down Converter]. Dieses lässt sich durch Eingabe von `dspddc` aus der Matlab-Konsole starten, wenn das benötigte „Signal Processing Blockset“-Paket installiert ist.

6.1 Numerically Controlled Oscialltor (NCO)

In diesem Kapitel wird der Block NCO näher beschrieben. Der Numerically Controlled Oscillator (NCO, engl. Numerisch gesteuerter Oszillator) wird im DDC dafür verwendet, auf digitalem Weg eine Sinusschwingung zu erzeugen. Ziel ist hier, mindestens die Qualität des NCO im ISL5216 zu erreichen. Dieser hat ein Signal-Rauschabstand von 115dB [Int07, S. 6, 11, 36-37].

6.1.1 Simulink NCO-Block

Simulink stellt in der „DSP System Toolbox„ einen Block „NCO“ zur Verfügung. Zuerst wurde mit der Matlabhilfe zum NCO berechnet, wie dieser Block eingesetzt werden könnte. [Mat11p] Dabei sind folgende Grössen herausgekommen:

- Anzahl Akkumulator-Bits: N = 32
- dies ergibt eine Frequenzauflösung von 22.12 mHz, was etwa dem NCO des momentan verwendeten DDC entspricht und somit für unsere Anforderungen genügt.
- um ein Signal-Rauschabstand von 115dB wie im bestehenden DDC zu erreichen, sind 18 „quantized accumulator bits“ nötig, wobei jedes weitere Bit zusätzliche 6dB einbringt und somit z.B. für 140dB 22 Bit notwendig sind.
- dither bits: diese Zahl entspricht der Anzahl „accumulator bit„ - “quantized accumulator bits„ und somit bei 115dB 32-18=14
- phase increment bei 9.2656MHz: 865120000

- phase offset: 0

Beim generieren von HDL-Code stürtzte der HDL Workflow Adviser allerdings jedesmal mit der Fehlermeldung "Out of memory„ ab, siehe Anhang H. Die empfohlenen Massnahmen zur Behebung des Fehlers nützten nichts. Daher wurde ein eigener NCO-Block realisiert und auch die Produktdemo [Mat11g] kann somit nicht verwendet werden und wurde deshalb nicht weiter in Betracht gezogen.

6.1.2 modularer NCO-Block

Der Block hat zwei Eingangsgrössen, und zwar Konstanten für die Frequenz und den Phasenoffset. Weiter kannn man den NCO in zwei Teile unterteilen: Phasenakkumulator und Sinuserzeugung. Das gesamte Blockschaltbild des NCO ist in Abbildung 6.1, sowie zur besseren Ansicht vergrössert in Anhang F zu sehen.

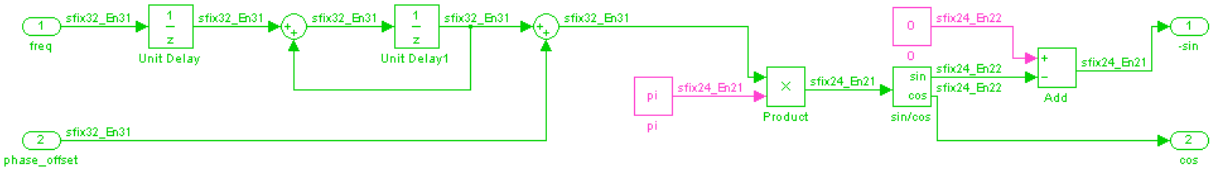


Abbildung 6.1: Blockschaltbild des NCO

6.1.3 Phasenakkumulator

In Abbildung 6.1 ist der Phasenakkumulator auf der linken Seite ersichtlich. Dieser besteht aus zwei Verzögerungsgliedern (Unit Delay), wobei der HDL-Coder daraus automatisch Speicherzellen erzeugt. Durch das Addierglied und die Rückführung des Signals aus dem zweiten Verzögerungsglied entsteht eine Treppenfunktion, wobei diese immer höher steigt, bis es zu einem Überlauf des Datentyps kommt und die Treppe von neuem beginnt. (Siehe Abbildung XY). Wichtig dabei: Der Datentyp wurde so gewählt, dass die Treppe die Werte zwischen -1 und 1 abdeckt. Damit bei einer sample-rate von 46Msps eine dem Beamsignal entsprechende Frequenz von 9.2656MHz entsteht, wurde die Konstante am Frequenzeingang folgendermassen berechnet: $\frac{9.2656 \cdot 10^6}{\frac{46 \cdot 10^6}{2}} = 0.4029$. Um Phase und Amplitude des Piltosignals zu bestimmen, muss diese Konstante ensprechend angepasst werden: $\frac{9.1756 \cdot 10^6}{\frac{46 \cdot 10^6}{2}} = 0.3989$

6.1.4 Erzeugung von Sinus und Cosinus

Zur Erzeugung von Sinus und Cosinus stellt Simulink verschiedene Blöcke zur Verfügung [Mat11k]. Um den Sinus in Hardware realisieren zu können, gibt es zwei Möglichkeiten. Einerseits ist dies die Verwendung von Lookup-Tables verschiedener Art, andererseits die Verwendung des CORDIC-Algorithmus, um die trigonometrischen Funktionen zu approximieren. Lookup-Tables haben den Vorteil, dass sie sehr schnell sind, meist schneller als Algorithmen zur Berechnung der Werte [Wik11a]. Der Nachteil der Lookup-Tables ist allerdings, dass sie viel Speicherplatz benötigen.

Daher soll der CORDIC-Algorithmus zur Berechnung des Sinus und Cosinus verwendet werden. Falls sich dieser als zu langsam herausstellt, könnte auf Lookup-Tables zurückgegriffen werden. Dies ist jedoch wie in Kapitel 7 ersichtlich nicht nötig, da der CORDIC-Algorithmus nicht den langsamsten Teil des DDCs darstellt.

Das Signal vom Phasenakkumulator wird zuerst mit Pi multipliziert, damit die Phasenwerte danach zwischen -Pi und Pi variieren. Danach wird anhand des „sin/cos“-Blocks von Simulink, welchen den CORDIC-Algorithmus unterstützt, aus diesen Werten ein Sinussignal erzeugt. Bei den Einstellungen des Blocks muss beim Feld „Approximation method“ entsprechend „CORDIC“ gewählt werden, denn sonst ist der Block nicht synthetisierbar. CORDIC steht für COordinate ROtation DIgital Computer wobei der Name schon einen Teil der Funktionsweise des Algorithmus erklärt. Dieser basiert auf der Rotation und verwendet dafür nur die Shift-Operation sowie die Addition, was ihn Hardware-Effizient macht. In iterativen Schritten wird nach und nach die gewünschte trigonometrische Funktion angenähert, wobei generell pro Iterationsschritt ein zusätzliches Bit an Genauigkeit entsteht. [And11] Der verlangte -sin wird erzeugt, indem das Sinussignal von der Konstante 0 subtrahiert wird.

6.1.5 Dither

Zur Verbesserung des Signal-Rausch-Abstandes werden in numerischen Oszialltoren Zufallsgeneratoren, auch genannt Dither, eingebaut [Mat11h, Wik11g]. Addieren eines kleinen, zufälligen Wertes zum zu wandelnden Zahlenwert bewirkt, dass nicht immer die exakt selben Werte in einen Sinus gewandelt werden. Dies würde heissen, dass immer dieselben Werte aus der abgespeicherten Tabelle verwendet werden. Da diese Werte auch nur eine bestimmte Genauigkeit haben, wäre die Folge davon, dass bei derselben Treppenstufe des Phasenakkumulator immer jeweils auf- oder abgerundet wird. Um das zu verhindern, wird dem Wert ein kleiner Zufallswert zugefügt oder abgezogen oder anders gesagt Rauschen hinzugefügt. Das zu erwartende Ergebnis ist wie gesagt ein besseres Signal-Rausch-Verhältnis oder Spurious Free Dynamic Range (SFDR), wobei insbesondere direkte Oberwellen der Grundfrequenz durch den Dither verschwinden sollten.

Zufallsgeneratoren gibt es viele verschiedene, wobei zwischen echten Zufallsgeneratoren und Pseudozufallsgeneratoren unterschieden wird. Echte Zufallsgeneratoren benötigen externe Hardware. So können beispielsweise das verstärkte Rauschen eines Bauteils, ungenaue Oszillatoren oder chaotische Systeme als Zufallsquelle dienen. Da in unserem Fall zusätzliche Hardware nicht erwünscht ist, sollte ein Pseudozufallsgenerator verwendet werden. Diese enthalten im Namen „Pseudo“, weil sie nicht eigentlich zufällig sind, sondern eine vordefinierte Folge von Zahlen, welche zufällig erscheinen mag, erzeugen. Dabei werden Pseudozufallsgeneratoren nach ihren kryptographischen Eigenschaften eingeteilt. Umso einfacher es ist, den hinter der Zufallsfolge stehenden Algorithmus nur Anhand der Zahlenfolge herauszufinden, umso schlechter die kryptographischen Eigenschaften [TLL03].

Pseudozufallsgeneratoren gibt es viele, hier wurde entschieden das verbreitetste, ein Linear Feedback Shift Register (LFSR), zu verwenden [New11, Wik11e, Wik11f]. Der Entscheid fiel auf das LFSR, weil dieses gute statistische Eigenschaften hat und sich effizient in Hardware umsetzen lässt. Der Nachteil, dass das dahinterstehende Polynom mithilfe des Berlekamp-Massey-Algorithmus entschlüsselt werden kann ist hier nicht relevant, da keine kryptographischen Eigenschaften nötig sind. Mithilfe der Xilinx-Application Note [Alf96] und der darin enthaltenen Tabelle idealer Polynome wurde das LFSR in Simulink umgesetzt.

Das beifügen eines Zufallswertes verbesserte allerdings in der Simulation das Singal-Rausch-Verhältnis nicht. Auch bei Verwendung anderer Polynome als des zuerst ausgewählten war dies der Fall. Da damit - trotz der grossen Verbreitung, was auch die extra dafür geschriebene Xilinx Application Note zeigt - Zweifel an der Eignung des LFSR als Zufallsgenerator für diese Anwendung aufkamen, wurden zwei andere Zufallsgeneratoren ausprobiert, zum einen der in [TLL03] beschriebene Blum Blum Shub und zum anderen der Simulink-Block „Random Number“ [Mat11j]. Der Blum Blum Shub wurde als alternative ausprobiert, weil dieser gute kryptographische Eigenschaften hat und sich damit vom LFSR unterscheidet. Der Simulink-Block wurde gewählt, weil der Block fertig vorliegt und somit mit kleinem Aufwand verwendet werden kann. Das Signal-Rausch-Verhältnis änderte sich allerdings auch bei Verwendung dieser Zufallsgeneratoren nicht. Der Simulink-Block „Random Number“ ist allerdings auch nicht synthetisierbar und könnte somit nicht verwendet werden. Schlussendlich wurde der Dither, wie in Unterabschnitt 7.1.2 genauer beschrieben, weggelassen.

6.2 Mischer (Multiplikation)

Das heruntermischen des Eingangssignals auf DC erfolgt nach Gleichung 3.1 auf Seite 9 mithilfe einer Multiplikation. Dazu wird in Simulink ein gewöhnlicher „Product“-Block verwendet.

6.3 Filterung

Die Filterung wurde dem momentan verwendeten DDC nachempfunden [Mül05]. Dieser enthält ein CIC-Dezimierungsfilter und zwei FIR-Dezimierungsfilter. Diese Kombination scheint sich bewährt zu haben, wurde sie doch auch in zahlreichen anderen DDC-Beispielen gefunden [Mat11g, Mat11d, Eng11]. Da dies der Standard zu sein scheint, wurde diese Kombination auch hier verwendet.

6.4 CIC-Filter

Zuerst wurde ein CIC-Filter aus Verzögerungsgliedern, Addierern und dem Downsample-Block realisiert wie in Abbildung 6.2 ersichtlich. Dieses wurde an das CIC-Filter im momentan mit dem ISL5216 verwendeten Design angepasst. Die Decimation beträgt 230, was bedeutet, dass aus 230 Samples am Eingang ein Wert am Ausgang entsteht. Die Taktrate wird dabei durch den Block „Downsample“ heruntergesetzt. Dadurch wird entsprechend die Ausgangs-Sample-Rate 230mal kleiner als diejenige am Eingang und beträgt noch $\frac{46\ \text{Msps}}{230} = 200$ ksps. Die im CIC-Filter integrierten Verzögerungsglieder sollen nur um einen Takt verzögern, daher wurden die „Integer Delay“-Blöcke entsprechend konfiguriert. Da im ISL5216 ein 4-stage-CIC Filter zur Anwendung kommt, ist hier die Anzahl Integrations- und Differentiationsstufen ebenfalls vier [Wik11b, Don11].

Simulink stellt allerdings auch einen Block names „CIC Decimation“ zur Verfügung, welcher etwas flexibler eingesetzt werden kann, beispielsweise beim Ändern der Anzahl Integrations- und Differentiationsstufen oder der Bestimmung der Datentypen [Mat11a]. Da dieser dieselbe Funktionali-

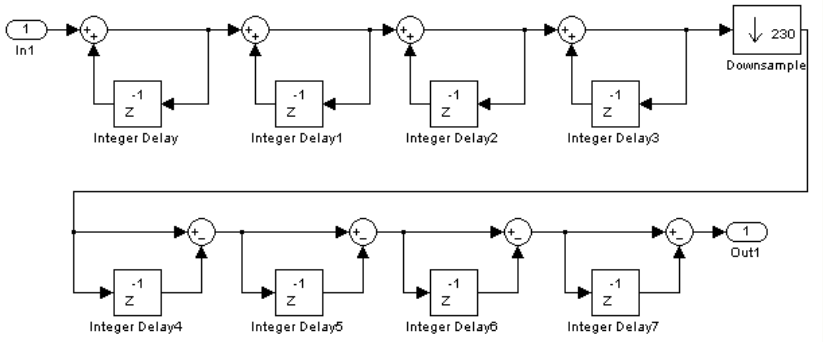


Abbildung 6.2: Struktur des CIC-Filters

tät bietet wurde schlussendlich dieser fertige Block verwendet. Dieser Block wurde ebenfalls dem im ISL5216 verwendeten CIC-Filter nachempfunden und hat somit folgende Parameter-Werte:

- Coefficient source: Dialog parameters
- Decimation factor (R): 230
- Differential delay (M): 1
- Number of sections (N): 4
- Data type specification mode: Minimum section word lengths
- Output word length: 32
- Rate options: Allow multirate processing

„Minimum section word lengths“ wurde gewählt, um die damit automatisch entstehende riesige Datenbreite von 56 Bits zu verhindern. Der damit erscheinende Dialog „Output word length“ wurde dann fürs erste auf der Standardeinstellung von 32 Bits belassen. „Allow multirate processing“ wurde daher gewählt, weil bei der alternativen Option „Enforce single-rating processing“ bereits bei der Simulation Fehlermeldungen auftauchen. Der Frequenzgang des damit entstandenen CIC-Filters ist in Abbildung 6.3 dargestellt.

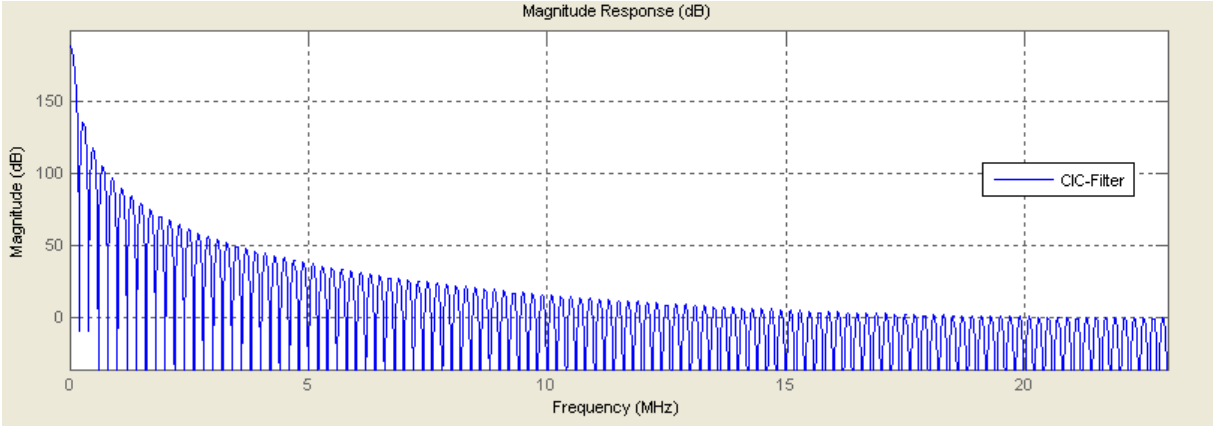


Abbildung 6.3: Frequenzgang des CIC-Filters

Nach dem CIC-Filter wurde ein Block „Extract Bits“ eingefügt. Ursprünglich diente dieser dazu, aus dem hochauflösenden Ausgang des CIC-Filters eine bestimmte Anzahl Most Signifikant Bits (MSB) auszuwählen, da die folgenden FIR-Filter die hohe Auflösung nicht verarbeiten konnten. Auch nach einstellen der gewünschten Auflösung am Ausgang des CIC-Filter darf dieser Block allerdings nicht weggelassen werden. Relevant ist wahrscheinlich die in dem Block vorgenommene Einstellung „Output scaling mode: Treat bit field as an integer“. Somit wird dieser Block im DDC verbleiben, wenn er womöglich auch durch einen Block ersetzt werden könnte, der Schlicht den Datentypen umwandelt.

6.5 FIR-Filter

Im Anschluss folgen in der Struktur des DDC pro Kanal zwei FIR-Decimation-Filter, wofür der gleichnamige Block in Simulink gewählt wurde [Mat11e]. Auch diese Filter wurden dem bestehenden Design nachempfunden. Die Filterkoeffizienten stammen somit aus dem ISL5216-Datenblatt (Seite 59) und zwar aus der Kolonne „DECIMATING HALFBAND #5 (DHBF #1, 23-TAP)“, da das ASIC für die bestehende Anwendung mit einem Filter vom Typ 5 konfiguriert wurde. Das Filter hat also 23 Verzögerungsglieder und entsprechend auch 23 Filterkoeffizienten. Diese sind in Tabelle 6.1 aufgelistet und in Abbildung 6.4 anhand der Impulsantwort dargestellt. Der Dezimierungsfaktor jedes FIR-Filters ist 2, was in der Halbierung der Samplefrequenz resultiert. Somit ist die Samplefrequenz am Ausgang der beiden FIR-Filter noch $\frac{200\ \text{ksps}}{2 \cdot 2} = 50$ ksps.

Koeff.:	Wert:	Koeff.:	Wert:	Koeff.:	Wert:
1	-0.000347137	9	-0.081981659	17	0.03055191
2	0	10	0	18	0
3	0.00251293	11	0.309417725	19	-0.010158539
4	0	12	0.5	20	0
5	-0.010158539	13	0.309417725	21	0.00251293
6	0	14	0	22	0
7	0.03055191	15	-0.081981659	23	-0.000347137
8	0	16	0		

Tabelle 6.1: FIR Filterkoeffizienten

Auffällig ist, dass fast die Hälfte der Koeffizienten den Wert 0 haben und die Koeffizienten ausserdem Symmetrisch sind (der erste Koeffizient entspricht dem letzten). Dies erstaunt allerdings in Bezug auf das Symmetrieprinzip nicht, da ein theoretisches, ideales Filter eine symmetrische Impulsantwort erzeugt. Mit diesen Koeffizienten entsteht für das erste FIR-Filter ein in Abbildung 6.5 abgebildeter Frequenzgang. Zu beachten ist, dass das nachfolgende zweite FIR-Filter dieselben Koeffizienten hat. Der Frequenzgang sieht ebenfalls gleich aus, wobei die X-Achsenwerte allerdings nur halb so gross sind. Dies folgt daraus, dass die Eingangssamplefrequenz nur halb so gross ist, also statt bis 100khz nur bis 50khz reicht. Der Frequenzgang stimmt optisch mit dem im ISL5216-Datenblatt auf Seite 55, FIGURE 16 abgebildeten Frequenzgang des HBF5-Filters überein und somit kann angenommen werden, dass das Filter in Simulink korrekt konfiguriert wurde.

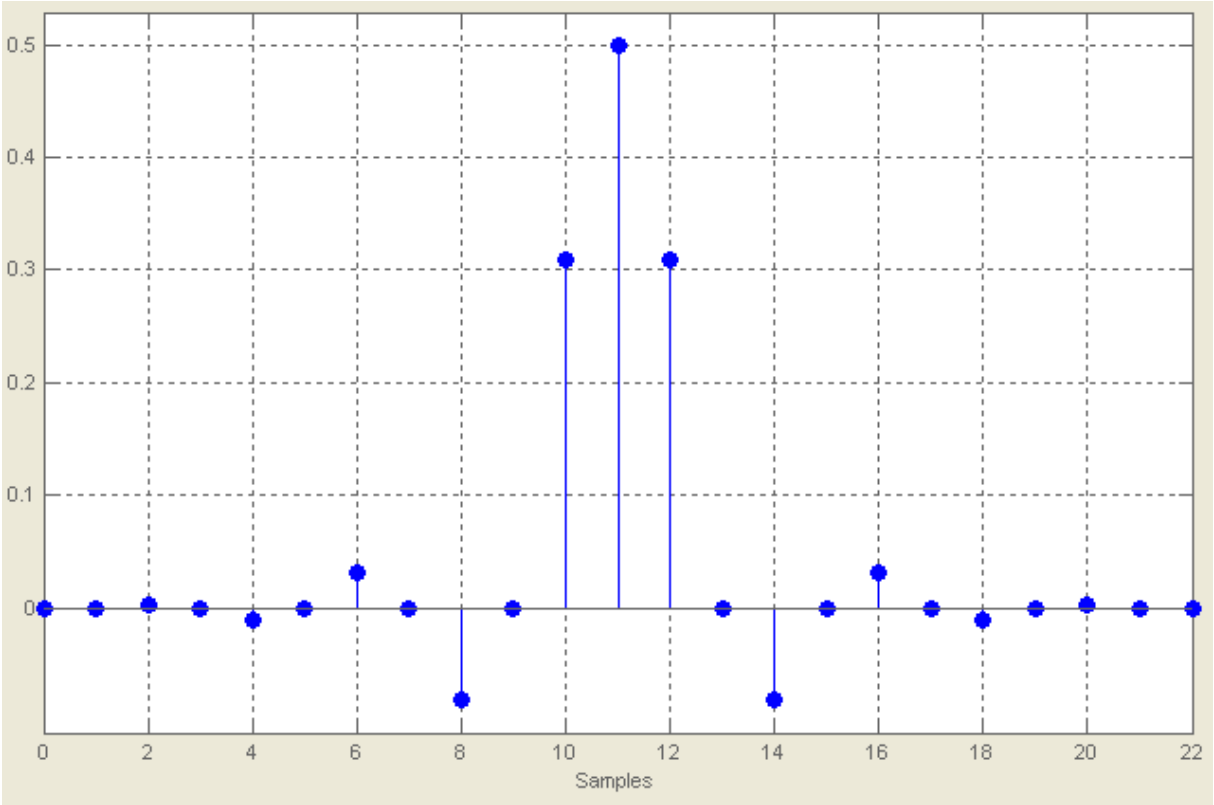


Abbildung 6.4: Impulsantwort des FIR-Filters

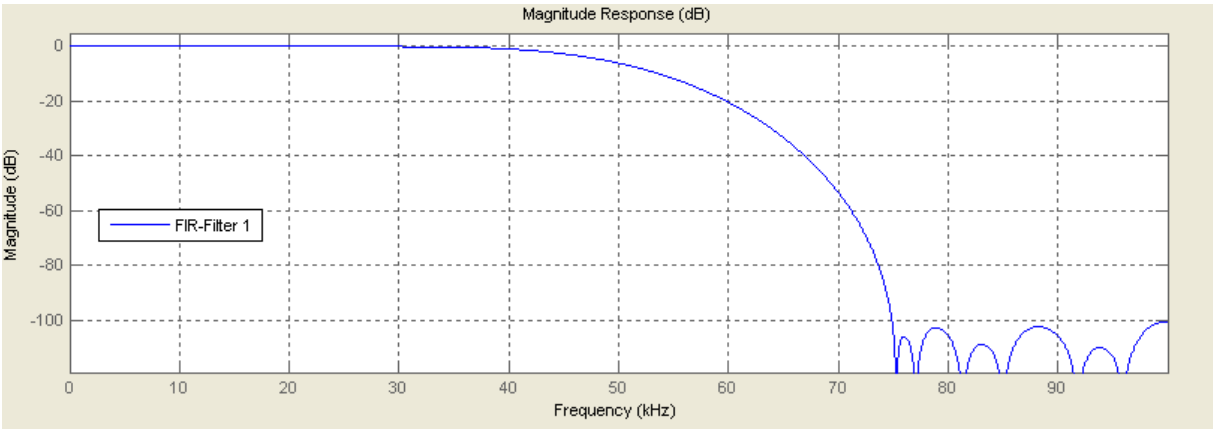


Abbildung 6.5: Frequenzgang des 1. FIR-Filters

6.6 Umwandlung der kartesischen in Polarkoordinaten

Für die Umwandlung von kartesischen in Polarkoordinaten wurde der „Cartesian2Polar“-Block aus den Simulink HDL-Coder Demos verwendet. [Mat11n]

Da der Block „Cartesian2Polar“ jeweils 256 Iterationen braucht, um einen Wert zu wandeln, wurde die Sample-Frequenz des Signals nach den FIR-Filtern von 50ksps mithilfe des „Rate Transition“-Blocks um 256 erhöht, so dass am Ausgang für jedes Sample aus den FIR-Filtern ein gewandelter Wert entsteht. Die Sample-Frequenz hätte dann wieder auf 50ksps zurückgestellt werden können, darauf wurde aber verzichtet, da dies keinen direkten Gewinn bringt.

Bei der Verwendung des „Cartesian2Polar“ Blocks wurde durch ausprobieren herausgefunden, dass die Variable „rate“ im Beispiel an 3 verschiedenen Orten in Simulink auf jeweils denselben Wert eingestellt werden muss und zwar unter:

- Rechtsklick auf den Block, dann unter *Edit Mask -> Initialization* sowie *Parameters*
- *File -> Model Properties -> Callbacks -> InitFcn*

Letzteres ist für die Verwendung im DDC allerdings nicht notwendig sondern wurde nur im Testfile gebraucht. Die für den DDC eingestellte rate beträgt $256 \cdot 50$ ksps = 12.8 Msps.

Folgende weitere Änderung musste mithilfe von Rechtsklick auf den Block *Look Under Mask* für den Gebrauch des Blocks vorgenommen werden: Im Block *Controller -> Enabled Counter -> Switch* Threshold auf *rate - 1* einstellen anstelle von 7[1]

Die Genauigkeit des Algorithmus wurde getestet, indem zwei sinusförmige Signale auf I und Q Eingänge des Blocks gegeben wurden, parallel mit dem nicht HDL-fähigen „Cartesian to Polar“-Block von Simulink gewandelt und die Ausgangssignale verglichen wurden. Das dafür verwendete Blockschaltbild ist in Anhang G ersichtlich. Dieses lag aus der Simulink HDL-Coder Demo vor, wurde jedoch wie oben erwähnt abgeändert. Ausserdem wurde das Testsignal auf 50ksps erhöht um die spätere Anwendung spezifisch zu testen. Der Block „Cartesian2Polar“ entspricht dem im DDC eingesetzten Block, welcher den CORDIC-Algorithmus verwendet. „Reference“ enthält den Simulink-Block „Cartesian2Polar“, welcher die Quadratwurzel und den Arcus-Tangens für die Umwandlung verwendet. Die beiden Umwandlungsresultate werden im Block „Error“ verglichen, indem der Absolutwert der Differenz der Signale ausgerechnet wird. Der damit simulierte Fehler für die Umwandlung mit dem CORDIC-Algorithmus gegenüber der Referenz, welche mit Quadratwurzel und Arcus-Tangens arbeitet, lag unter 1 Prozent.

6.7 Simulink-Schema des DDC

Das Simulink-Schema, welches aus der Zusammenstellung der oben aufgeführten Blöcke für den DDC entstanden ist, ist in Abbildung 6.6 als Übersicht dargestellt. Zur besseren Lesbarkeit ist es ausserdem in Anhang E etwas grösser abgebildet. Beim Zusammenstellen des DDC aus den Teilfunktionen enstanden auch Probleme. Die zwei wichtigsten sind, dass einerseits die FIR-Filter nicht beliebig grosse Eingangsdatenbreiten verarbeiten können und daher der Block „Extract Bits“ in jedem Kanal hinzugefügt wurde und andererseits die Sample-Rate für den „Cartesian2Polar“-Block um 256 heraufgesetzt werden musste, da nur bei jedem 256. Zyklus ein

[1] Im Beispiel war rate = 8, somit entsprach 7 da auch rate - 1

Ausgabewert entsteht und somit die Eingangsgrösse während 256 Zyklen konstant sein sollte, was mit dieser Sample-Rate Erhöhung erzielt wird.

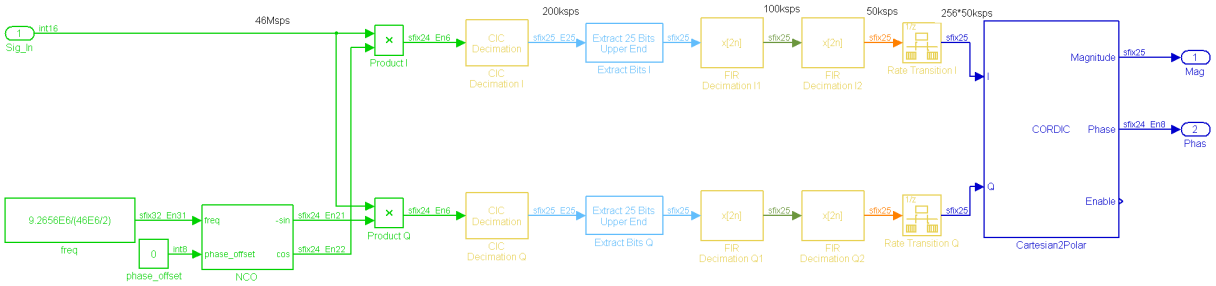
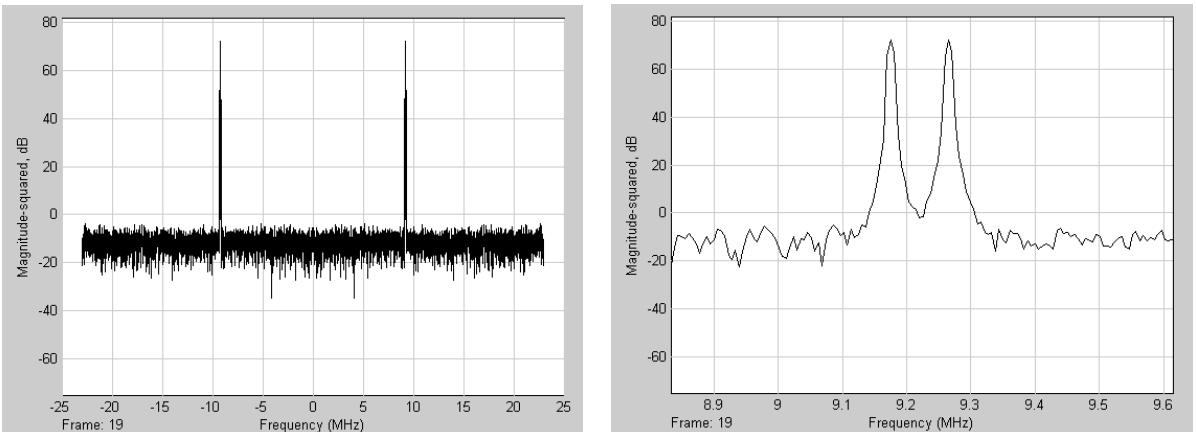


Abbildung 6.6: Blockschaltbild des Digital Down Converters

Das Schema sieht dem DDC-Schema in Kapitel 3 äusserst ähnlich. Unterschied ist, dass man hier die Konstanten sieht, welche für die Bestimmung der Frequenz und des Phasenoffset im NCO zuständig sind und die genannten Hilfsblöcke zwischen den Filtern bzw. vor der Berechnung der Polarkoordinaten eingefügt wurden. Wie im Schema ersichtlich, hat der DDC nur einen Signaleingang. Das Eingangssignal kommt im späteren Einsatz vom Analog-Digital Wandler und hat eine Datenbreite von 16Bit[2]. Diese wurde so gewählt, weil zukünftig eingesetze ADCs diese Datenbreite haben könnten. Das Eingangssignal ist wie bereits gesagt aus dem Strahlsignal, dem Pilotsignal und dem unvermeidbaren Rauschen zusammengesetzt. Ein solches Signal wurde auch in der Simulation erstellt und ist im Frequenzspektrum in Abbildung 6.7 zu sehen.



(a) Gesamtspektrum **(b)** Spektrum um 9 MHz

Abbildung 6.7: Frequenzspektrum des Eingangssignals

In Abbildung 6.7(a) sieht man das gesamte Spektrum, welches aufgrund des Symmetrieprinzips [Kar10, S. 127ff] in der Simulation von -23 ... 23 MHz reicht. Dies ergibt eine Frequenzbreite von 46MHz, was der Sample-Frequenz des Eingangssignals entspricht. Deutlich sieht man das Rauschen über die ganze Bandbreite. Daneben gibt es Signalspitzen in der Nähe von 9MHz. Bei 9 MHz liegen das Beam-Signal (9.2656MHz) sowie 90kHz darunter das Pilotsignal (9.1756MHz)[3].

[2]Die Eingangsdatenbreite kann einfach geändert werden. Dazu muss im Block „Data Type Conversion“, welcher vor dem Eingang des DDC-Blocks liegt, der „Output Data Type“ verändert werden.

[3]siehe Kapitel 2 oder [Joh10]

Abbildung 6.7(b) zeigt einen Ausschnitt des Frequenzspektrums, um Strahlsignal und Pilotsignal - die beiden höchsten peaks - deutlich erkennen zu können.

7 Optimierung

In diesem Kapitel wird erklärt, wie die Simulation und alle Prozesse im Workflow in der Hinsicht optimiert wurden, dass der damit erzeugte Code auf dem FPGA schneller läuft und weniger Platz in Anspruch nimmt. Ziel dabei ist es, dass das vorgesehene FPGA Xilinx Virtex6 xc6vlx130t-1ff1156 bei implementieren von 8 DDC-Kanälen mit einer Taktfrequenz von 100MHz lauffähig ist und die Auslastung der Logik unter 60% liegt. Dies entspricht den Richtlinien aus dem Pflichtenheft. 100MHz sollte erreicht werden, damit eine Implementation zusammen mit dem später hinzukommenden Framework einfach möglich ist. Da das Framework auch Logik beansprucht, wurde für 8 DDC-Kanale ohne Framework eine Auslastung von maximal 60% angestrebt. In einigen Fällen musste auch zwischen Platz- oder Zeitoptimierung entschieden werden, wobei wie sich in folgenden Abschnitten zeigt meistens die Zeitoptimierung Vorrang hatte, weil die 100MHz schwieriger zu erreichen waren als die 60% Logikauslastung. Das Kapitel soll einige der verwendeten Konzepte zur Optimierung erklären, wenn auch manche Optimierungen durch ausprobieren vorgenommen wurden. Um Verbesserungspotential ausfindig zu machen wurde der erzeugte VHDL-Code unter anderem auch mit Xilinx ISE 12.1 für das FPGA implementiert und dort der langsamste Signalpfad angeschaut.

7.1 Optimierung der Simulation

In Simulink wurden einerseits Simulationsparameter, vor allem die Datenbreiten, angepasst und andererseits konnten bei Simulink-Blöcken Einstellungen in Bezug auf den HDL-Coder vorgenommen werden. Wenn in diesem Kapitel von Frequenz gesprochen wird, ist die mögliche Frequenz im FPGA zur Codeausführung gemeint.

7.1.1 Multiplikatoren

Eine der wichtigsten Anpassungen war die Reduzierung der Datenbreite bei den Multiplikationen. Die Virtex-6 Familie verfügt über DSP48E1-Slices welche einen 25 x 18 Bit Multiplikatoren enthalten. [Xil11e, S. 2] Folgende Teile der Schaltung waren von dieser Änderung betroffen:

- Multiplikator im NCO: 25 Bit vom Akkumulator, 18 Bit Konstante Pi
- Mischer: 16 Bit Inputsignal, 24 Bit vom NCO
- Eingang FIR-Filter 25 Bit wegen interner Multiplikationsblöcken sowie bei *Function Block Parameteres -> Datatypes -> Coefficients: fixdt(1,18,17) , Product output: Inherit: Same as input, Accumulator: fixdt(1,25,0)* und *Output: Inherit: Same as accumulator*

Das sehr hochauflösende Signal der CIC-Filter wurde dabei mit dem „Extract Bits“-Block auf 25 Bit verkleinert. Diese Massnahmen erbrachten einen Geschwindigkeitsgewinn, vorallem aber eine Reduktion der benötigten Ressourcen, da dadurch keine Kaskadierung von Multiplikator-Blöcken mehr verwendet werden muss. Die Reduktion der Datenbreite hat allerdings auch eine Minderung der Genauigkeit zur Folge. Verglichen mit dem momentan eingesetzten DDC, also dem ISL5216, welcher eine interne Auflösung von 24Bit hat, ist 25Bit immer noch genauer.

7.1.2 Pipelining

Eine grosse Verschnellerung des Systems wurde mit dem Einfügen von Pipelining erreicht. Bei Rechtsklicken auf einen Block und der Auswahl *HDL Coder -> HDL Block Properties...* können Input- und Output-Pipelines eingestellt werden. Diese bewirken, dass beim erzeugen des VHDL-Codes zusätzliche Speicherregister eingefügt werden. Dadurch kann der Code im FPGA schneller ausgeführt werden, da die Laufzeit der Logikbausteine zwischen zwei Flipflops für die Maximalfrequenz massgebend ist. Pipelining betrifft folgende Blöcke:

- CIC-Filter: *AddPipelineRegisters: on*
- FIR-Filter: *AddPipelineRegisters: on*
- Sin/Cos-Erzeugung im NCO: *InputPipeline = 1, OutputPipeline = 1*
- Multiplikator bei Mischer-Multiplikatoren: *InputPipeline = 1, OutputPipeline = 1*

Nach einfügen der Pipelines in den CIC- und FIR-Filtern sprang die mögliche Frequenz im FPGA von 45.466MHz auf 53MHz. Es stellte sich dann heraus, dass der Dither-Block mit dem Zufallsgenerator ein Nadelöhr darstellte. Da dieser wie in Unterabschnitt 6.1.5 erklärt keinen Einfluss auf die Qualität des NCO-Signals hatte, wurde dieser weggelassen. Die mögliche Frequenz im FPGA sprang dann für einen DDC von 53MHz auf 80.684MHz. Als die Pipelines beim Sin/Cos-Block hinzugefügt wurden, konnte erneut ein Anstieg der Frequenz festgestellt werden, nämlich auf 97.907MHz. Was allerdings keine Geschwindigkeitsverbesserung brachte, war die Geauigkeit des Cordic-Algorithmus von 24 Iterationen auf 20 zu reduzieren. Da pro Iteration ein Bit Genauigkeit erwartet werden darf und der Ausgang eine Auflösung von 24 Bit hat, machen 24 Iterationen Sinn. [And11] Als nächstes wurde Pipelining bei den Mischern aktiviert, wodurch die Frequenz auf 106.947MHz stieg. Beim Multiplikator im NCO brachten Pipelines keine Geschwindigkeitsverbesserung.

7.1.3 SerialPartition bei FIR-Filtern

Bei den FIR-Filtern wurde zuerst Pipelining auch bei den Multiplikatoren aktiviert, allerdings später teilweise wieder ausgeschaltet. Da die FIR-Filter viele Multiplikatoren benötigen und nur eine begrenzte Anzahl, nämlich 480 DSP48E1-Slices vorhanden sind, wurde bei den HDL-Properties SerialPartition auf 6 anstelle von -1 eingestellt. Dies bewirkt, dass Multiplikatoren mehrmals verwendet werden können, allerdings ist dann Pipelining nur noch begrenzt möglich. „AddPipelineRegisters„ kann nach wie vor auf “on„ sein[1], “InputPipeline„ und “OutputPipeline„ müssen allerdings den Wert 0 haben. Die mehrmahlige Verwendung ist daher möglich, weil die

[1] “AddPipelineRegisters„ fügt bei Addierern mit Baumstruktur zwischen jedem Level ein Register zu und ein Register wird nach den Produkten hinzugefügt [Mat11f].

Sample-Frequenz am Eingang der FIR-Filter nur 200ksps beträgt, was ein Bruchteil der FPGA-Taktfrequenz ist. Eine vom HDL-Workflow-Adviser gegebene Übersicht über die verschiedenen Möglichkeiten bei SerialPartition ist in Abbildung 7.1 sowie in Anhang H ersichtlich. Die Grafik zeigt, dass bei einem Folding Factor von 6, was SerialPartition von 6 entspricht, nur noch ein Multiplizierer pro FIR-Filter benötigt wird. Dies rührt daher, dass von den 23 Filterkoeffizienten 10 den Wert 0 haben, einer 0.5 ist und die Multiplikation somit mit einem Shift-Befehl ausgeführt werden kann und die restlichen 12 Koeffizienten Symmetrisch sind, das heisst jeder Koeffizient kommt zweimal vor. Beim Ausprobieren zeigte sich, dass beim optimalen SerialPartition = 6 am meisten Multiplikatoren gespart werden können ohne Geschwindigkeits-Verluste hinnehmen zu müssen.

The following table is the summary of various filter lengths:

Coefficients	Total	Zeros	^2s	A/Symm	Effective
Polyphase # 1	12	0	0	6	6
Polyphase # 2	12	11	1	0	0

Effective polyphase filter length for SerialPartition value is 6.

Table of 'SerialPartition' values with corresponding values of folding factor and number of multipliers for the given filter:

Folding Factor	Multipliers	SerialPartition
1	6	[1 1 1 1 1 1]
2	3	[2 2 2]
3	2	[3 3]
4	2	[4 2]
5	2	[5 1]
6	1	[6]

Abbildung 7.1: Optionen bei SerialPartition

Pro DDC waren am Schluss statt 54 noch 10 DSPs nötig, was bei 8 Kanälen ein Total von 80 DSPs oder 16% Auslastung ergibt anstelle von 432 DSPs oder 90% Auslastung ohne Serial Partition. Obwohl Serial Partition die maximal mögliche Taktfrequenz des FPGA's nicht heruntersetzte, sind bis zum Schluss die FIR-Filter die langsamsten, geschwidigkeitsbegrenzenden Glieder geblieben.

7.2 Optimierung in Xilinx ISE

Durch die Option *Synthesize XST -> Register Balancing* werden die Flipflops bei der Einbettung in das FPGA zeitoptimiert verteilt. Dies erbrachte eine Steigerung der ausführbaren Frequenz von 106.947MHz auf 108.951MHz. Werden 2 DDCs parallel realisiert, sinkt die Ausführbare Frequenz von 108.951MHz auf 107.101MHz. Dies ist der Fall, weil bei höherer Auslastung des FPGA's weniger ideale Signalpfade zur Verfügung stehen. Bei 8 parallel genutzten DDCs sinkt die Frequenz gar auf 92.4MHz und ist damit unter den gewünschten 100MHz.

Eine Übersicht der bei 8 DDC Kanälen benötigten Ressourcen ist im nachfolgenden Abbildung 7.2 zu sehen. Die benötigte Logik („Number of Slice LUTs“) beträgt 57% und ist somit kleiner als die gewünschten 60%. Die Erwartungen diesbezüglich sind also erfüllt und auf weitere Platz-Optimierungen kann somit verzichtet werden.

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	43,367	160,000	27%	
Number used as Flip Flops	43,367			
Number used as Latches	0			
Number used as Latch-thrus	0			
Number used as AND/OR logics	0			
Number of Slice LUTs	45,774	80,000	57%	
Number used as logic	38,579	80,000	48%	
Number using O6 output only	34,785			
Number using O5 output only	1,050			
Number using O5 and O6	2,744			
Number used as ROM	0			
Number used as Memory	832	27,840	2%	
Number used as Dual Port RAM	0			
Number used as Single Port RAM	0			
Number used as Shift Register	832			
Number using O6 output only	832			
Number using O5 output only	0			
Number using O5 and O6	0			
Number used exclusively as route-thrus	6,363			
Number with same-slice register load	6,287			
Number with same-slice carry load	76			
Number with other load	0			
Number of occupied Slices	12,693	20,000	63%	
Number of LUT Flip Flop pairs used	46,158			
Number with an unused Flip Flop	9,923	46,158	21%	
Number with an unused LUT	384	46,158	1%	
Number of fully used LUT-FF pairs	35,851	46,158	77%	
Number of unique control sets	154			
Number of slice register sites lost to control set restrictions	553	160,000	1%	
Number of bonded IOBs	524	600	87%	
IOB Flip Flops	520			
Number of RAMB36E1/FIFO36E1s	0	264	0%	
Number of RAMB18E1/FIFO18E1s	0	528	0%	
Number of BUFG/BUFGCTRLs	2	32	6%	
Number used as BUFGs	2			
Number used as BUFGCTRLs	0			
Number of ILOGICE1/ISERDESE1s	128	600	21%	
Number used as ILOGICE1s	128			
Number used as ISERDESE1s	0			
Number of OLOGICE1/OSERDESE1s	392	600	65%	
Number used as OLOGICE1s	392			
Number used as OSERDESE1s	0			
Number of BSCANs	0	4	0%	
Number of BUFHCEs	0	120	0%	
Number of BUFOs	0	30	0%	
Number of BUFIODQSs	0	60	0%	
Number of BUFRs	0	30	0%	
Number of CAPTUREs	0	1	0%	
Number of DSP48E1s	80	480	16%	
Number of EFUSE_USRs	0	1	0%	
Number of GTXE1s	0	20	0%	
Number of IBUFDS_GTXE1s	0	10	0%	
Number of ICAPs	0	2	0%	
Number of IDELAYCTRLs	0	15	0%	
Number of IODELAYE1s	0	600	0%	
Number of MMCM_ADVs	0	10	0%	
Number of PCIE_2_0s	0	2	0%	
Number of STARTUPs	1	1	100%	
Number of SYSMONs	0	1	0%	
Number of TEMAC_SINGLEs	0	4	0%	
Average Fanout of Non-Clock Nets	3.90			

Abbildung 7.2: Device Utilization Summary nach der Implementierung von 8 DDC-Kanälen

7.3 Übersicht

Zum Schluss sollen die relevanten Massnahmen zur Verbesserung des Designs hinsichtlich Taktfrequenzen und Hardwareauslastung in Tabelle 7.1 als Übersicht gezeigt werden. Dabei sind die aufgeführten Werte bei einem implementierten DDC-Kanal gültig.

Massnahme:	**Verbesserung:**
Reduktion Datenbreite Multiplikatoren	nicht getestet
Pipelines CIC/FIR-Filter	Takrate 53MHz (statt anfänglich 45.466MHz)
Weglassen Dither	Taktrate 80.684MHz
Pipelines Sin/Cos-Block NCO	Taktrate 97.907MHz
Pipelines Mischer	Taktrate 106.947MHz
Serial Partition FIR-Filter = 6	Benötigte DSPs sinkt von 54 auf 10, Taktrate bleibt
Register Balancing beim Synthetisieren	Taktrate 108.951MHz

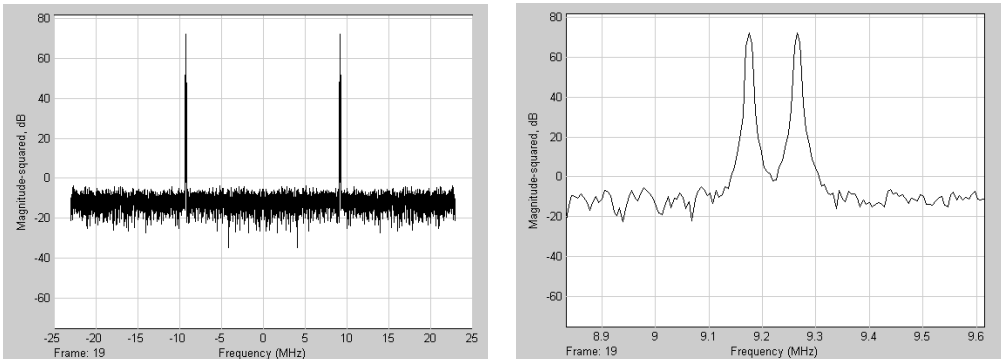
Tabelle 7.1: Übersicht der Optimierungen

8 Verifikation

In diesem Kapitel soll das erreichte Simulationsergebnis kontrolliert werden. Dazu wird eine generelle Aussage zur Signalverarbeitung mit dem Process Gain gemacht, das Eingangssignal verändert, um verschiedene Szenarios zu testen sowie die Qualität des NCOs bewertet. Zum Schluss wird das Ergebnis der Kosimulation gezeigt sowie die aufgetauchten Fehlermeldungen und Warnings ausgewiesen.

8.1 Process Gain

Das Testsignal wurde, wie bereits in Kapitel 6 erklärt, aus Beamsignal, Pilotsignal und Rauschen zusammengestellt. Abbildung 8.1 zeigt nochmals das Frequenzspektrum dieses Signals.



(a) Gesamtspektrum **(b)** Spektrum um 9MHz

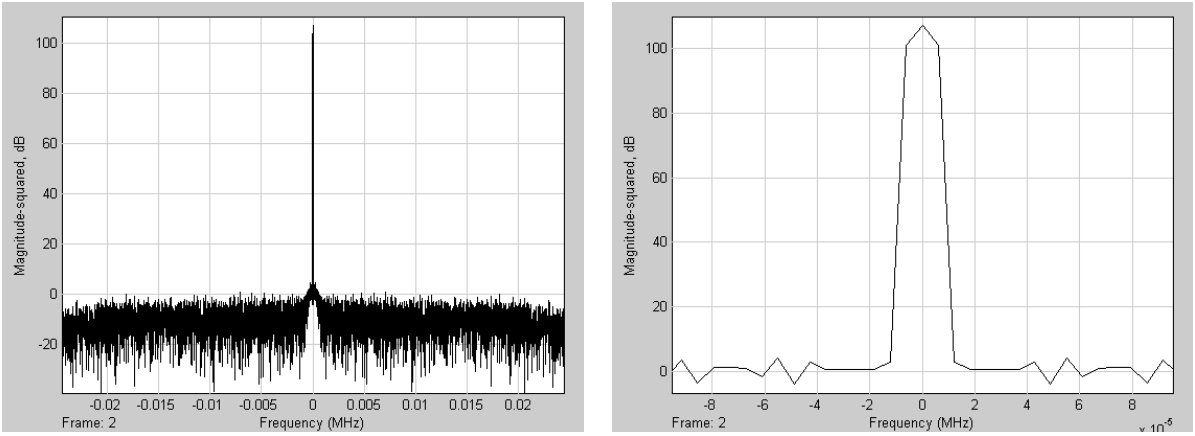
Abbildung 8.1: Frequenzspektrum des Eingangssignals

Wie in der Grafik ersichtlich, beträgt der Signal-Rausch-Abstand hier etwa 80dB. Als Eingangssignal war dabei die Amplitude des Beam- sowie des Pilotsignals auf $\frac{2^{15}-1}{3}$ eingestellt. Bei 16Bit wäre $2^{15}-1$ das maximal mögliche Signal ohne Überlauf, denn der Bereich reicht von -2^{15} bis $2^{15}-1$. Da das Signal zweimal vorkommt, darf dieses aber maximal halbsogross sein, ohne einen Überlauf zu bewirken. Schlussendlich wurde wegen des ebenfalls hinzuaddierten Rauschens entschieden, das maximal-Signal durch 3 zu teilen. Um das Rauschen etwa 80dB tiefer zu halten, wurde im Block „Band-Limited White Noise“ ein Noise-Power von 0.00001 eingestellt.

Ein Kriterium zur Überprüfung der Funktionstüchtigkeit ist der Process Gain. [Wik11i] Dieser ist als Bandbreite am Eingang geteilt durch Bandbreite am Ausgang definiert. In unserem Fall ist dies die Hälfte der Eingangssamplefrequenz geteilt durch die Ausgangsbandbreite BW siehe Gleichung 8.1.

$$\text{ProcessGain[dB]} = 10 \cdot \log\left(\frac{fs}{2 \cdot BW}\right) = 10 \cdot \log\left(\frac{46\text{Msps}}{2 \cdot 25\text{ksps}}\right) = 29.6379\text{dB} \tag{8.1}$$

Der Process Gain, der erwartet wird, ist somit 29.6379dB, also etwa 30dB. Da es keinen Sinn macht, das Spektrum der Amplitude zu berechnen - diese sollte relativ Konstant sein - ist das in Abbildung 8.2 dargestellte Signal das Frequenzspektrum des Ausgangssignals des zweiten FIR-Filters. (Die Funktionstüchtigkeit der Umwandlung von kartesischen in Polarkoordinaten wurde bereits in Abschnitt 6.6 gezeigt).



(a) Gesamtspektrum am Ausgang **(b)** Ausgangsspektrum um 0 Hz

Abbildung 8.2: Frequenzspektrum des Ausgangssignals

Das Signal ist etwa 110dB höher als das Rauschen. Da der Unterschied zwischen Signal und Rauschen am Eingang 80dB war, entspricht dies genau den Erwartungen.

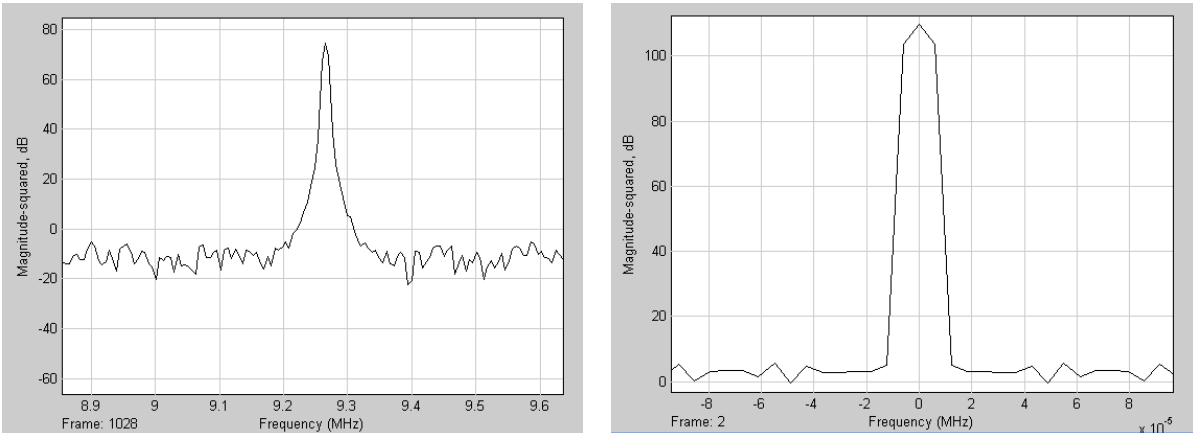
8.2 Variierung des Eingangssignals

In diesem Abschnitt soll gezeigt werden, dass der DDC auch bei Variieren des Eingangssignals korrekt funktioniert. In den nachfolgenden Abbildungen ist jeweils nur noch das gezoomte Spektrum in der Interessenszone dargestellt.

Bei weglassen des Pilotsignals wie in Abbildung 8.3 sieht das Spektrum praktisch identisch aus wie mit Pilotsignal in 8.1(b) auf der vorherigen Seite und 8.2(b). Allerdings ist sowohl am Eingang wie auch am Ausgang die Signalintensität ein klein wenig höher, sowohl beim Beamsignal wie auch beim Rauschen. Dies ist hat sich bei der Simulation so ergeben. Da das Singal to Noise Ratio (SNR) aber identisch ist kann gefolgert werden, dass die Kanalseparation korrekt funktioniert.

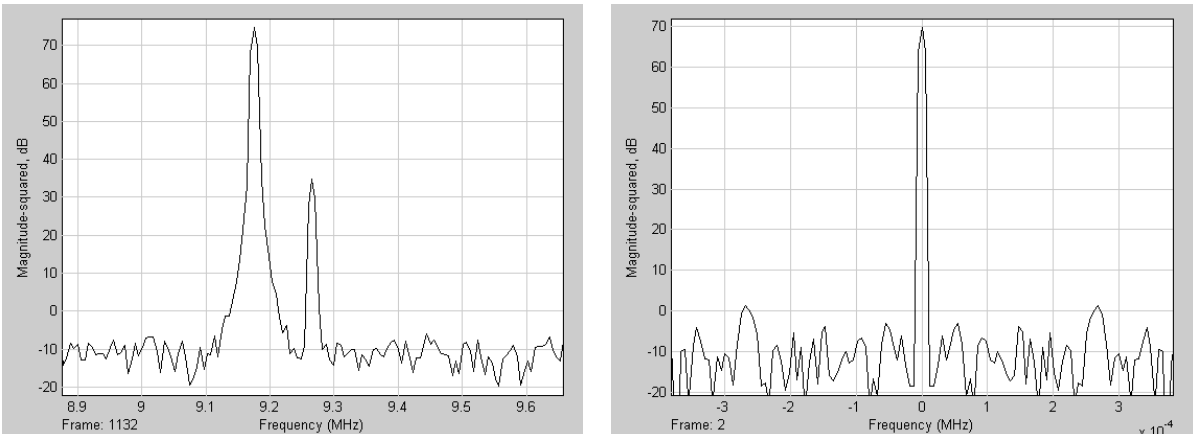
In Abbildung 8.4 ist die Amplitude des Beamsignals (rechts) um 40dB oder 100mal kleiner als die des Pilotsignals. Der Process Gain von 30dB bleibt dabei wie man sieht bestehen. Somit hat das Pilotsignal keinen Einfluss auf den Prozess, da es auch hier vollständig herausgefiltert wurde.

Ist das Beamsignal so klein, dass es gerade nicht im Rauschen untergeht, so ist das Ausgangssignal gerade 30dB über dem Rauschen, was dem Process Gain entspricht. Das heisst, dass der DDC auch bei kleinen numerischen Werten gut funktioniert und keine numerischen Fehler auftreten. Dargestellt ist diese Situation in Abbildung 8.5.



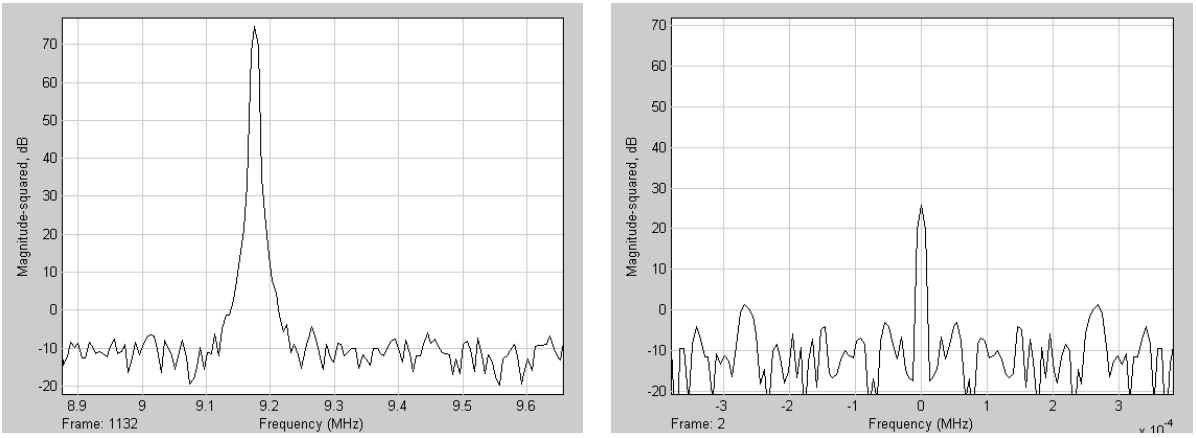
(a) Eingangsspektrum **(b)** Ausgangsspektrum

Abbildung 8.3: Spektren bei weglassen des Pilotsignals



(a) Eingangsspektrum **(b)** Ausgangsspektrum

Abbildung 8.4: Spektren bei Amplitudenverhältnis Beamsignal = 1 % Pilotsignal



(a) Eingangsspektrum **(b)** Ausgangsspektrum

Abbildung 8.5: Spektren wenn die Beamamplitude gerade nicht im Rauschen untergeht

8.3 Qualität NCO

Um die Qualität des NCO zu bestimmen, wird der Spurious Free Dynamic Range (SFDR) angeschaut. [Wik11j] Dieser besagt, wie gross der Abstand zwischen der gewünschten Frequenz des Generators zum höchsten Störsignal beträgt. Wie in Abbildung 8.6 zu sehen ist, beträgt der SFDR für den NCO mehr als 140dB. Dies ist im Vergleich zum bestehenden Design um 25dB besser. (Siehe Abschnitt 9.1)

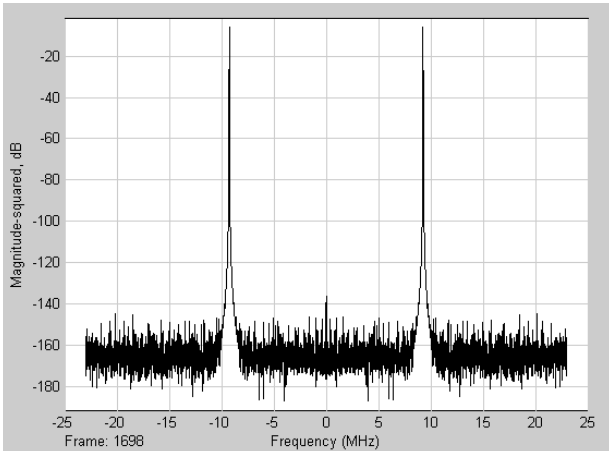


Abbildung 8.6: Ausgangsspektrum des NCO

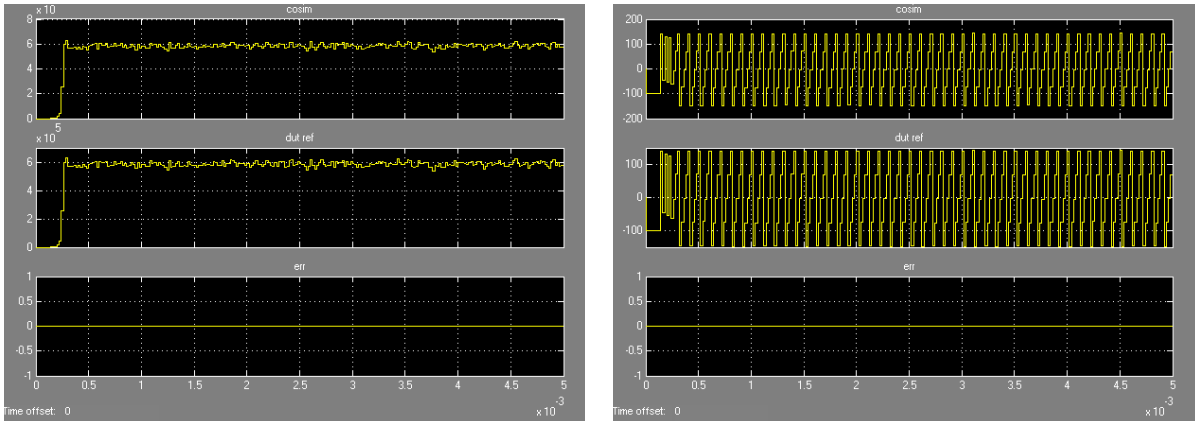
8.4 Ko-Simulation

In den vorherigen Abschnitten wurde gezeigt, dass die Simulink-Simulation des DDCs richtig funktioniert. In diesem Abschnitt soll gezeigt werden, dass der damit erzeugte VHDL-Code genau der Simulation entspricht und somit ebenfalls verifiziert werden kann. Dies geschieht wie in Kapitel 5 erklärt durch eine Simulation des VHDL-Codes in Modelsim und dem Vergleich der Ergebnisse mit denen der Simulink-Simulation. Abbildung 8.7 zeigt, dass sowohl Amplitude wie auch Phase identisch sind und die Abweichung der beiden - dargestellt im jeweils untersten Koordinatensystem der Abbildung - null beträgt. Somit kann gefolgert werden, dass der VHDL-Code korrekt funktioniert. Der VHDL-Code wurde zur Überprüfung der Funktionsfähigkeit nicht angeschaut. Der Vollständigkeit halber ist jedoch ein Code-Ausschnitt in Anhang I zu sehen.

8.5 Fehlermeldungen und Warnings

Obwohl die Funktionsfähigkeit des DDC nachgewiesen werden konnte traten doch Fehlermeldungen auf, welche teilweise bis zum Schluss nicht beseitigt werden konnten.

Ein häufig auftretendes Problem bei der Simulation war der „memory allocation error“. Diese Fehlermeldung trat bei umfangreichen Simulationen und langen Simulationszeiten auf. Eine von MathWorks angegebene Lösung war nicht erfolgreich. Meist half auch das Neustarten des Computers nichts und die einzige Möglichkeit, die Error-Meldung zu umgehen, war die Simulationszeit herunterzusetzen [Mat11q].



(a) Amplituden und deren Abweichung

(b) Phasen und deren Abweichung

Abbildung 8.7: Ergebnisse der Ko-Simulation

Die meisten in Matlab auftretenden „Warnings“ konnten während der Entstehung des DDC eliminiert werden. Beim Ausführen des schlussendlich verwendeten DDC-Models treten allerdings immer noch einige Warnungen auf, welche nicht beseitigt werden konnten. Die vollständige Liste der in der Matlab-Konsole auftretenden Rückmeldungen ist in Anhang M abgedruckt. Da die Funktionsfähigkeit des DDC nachgewiesen werden konnte, wurde auf diese Warnungen nicht weiter eingegangen.

Im HDL Workflow Adviser traten ebenfalls Meldungen auf. Diese sind in Anhang H beschrieben. Die Warnung in Abbildung H.1 zu den Lookup Tables konnte nicht beseitigt werden. Da die Kosimulation erfolgreich war, wurde diese Warnung ignoriert.

Der Error in Abbildung H.2 zeigt, dass der NCO-Block von Simulink nicht verwendet werden konnte, obwohl alle Schritte bis auf die Generierung des VHDL-Codes erfolgreich waren. Auch hier nützte weder die vorgeschlagene Lösung noch ein Neustart des Computers etwas.

9 Diskussion der Ergebnisse

In diesem Kapitel sollen die Ergebnisse kritisch betrachtet und mit den Anforderungen Verglichen werden. Weiter sollen wo nötig potenzielle Verbesserungsmöglichkeiten aufgezeigt und somit weitere Schritte nahegelegt werden.

9.1 Vergleich mit momentan eingesetzter Lösung

Wie in Kapitel 8 aufgezeigt, ist die Funktionsfähigkeit des DDC gegeben. Der Process Gain entspricht den theoretisch erwarteten 30dB und die Kanalseparation zwischen Beamsignal und Pilotsignal funktioniert wie gewünscht. Einzig dass der Dither im NCO keine Verbesserung gebracht hat, war dabei erstaunlich.

Um die Qualität des erzeugten DDC zu bewerten soll die neu entstandene Lösung mit dem bisher eingesetzten System verglichen werden. In Abbildung 9.1 sind dazu die wichtigsten Eckdaten zum Vergleich zusammengestellt.

	Bisher:	Neu:
Hardware:	2 ISL5216	1 FPGA
NCO SFDR:	115dB	140dB
Filter:	CIC + 2 FIR	CIC + 2 FIR
Dezimation:	920	920
Eingangsdatenbreite:	14 Bit	16 Bit
Interne Datenbreite:	24 Bit	25 Bit
Sample-Rate (max):	95Msps	92.4Msps

Abbildung 9.1: Vergleich der bisherigen Lösung mit der neu entstandenen Lösung

Anstelle von zwei ISL5216 ASIC's à je 4 Kanälen wird neu nur noch ein Xilinx Virtex-6-FPGA (XC6VLX130T-1FF1156) benötigt. Dies macht die Anwendung flexibler, da Änderungen und Erweiterungen leicht vorgenommen werden können. Desweitern könnte der VHDL-Code, welcher entstanden ist, in Zukunft auch auf anderen FPGAs als des bisher vorgesehenen eingesetzt werden (wobei aber die Optimierungsmöglichkeiten wieder neu betrachtet werden sollten). Ausserdem ist die Hardware, in welche die 8 DDCs implementiert werden sollen auch für andere Zwecke verwendbar und damit ebenfalls flexibler einsetzbar.

Der wichtigste Kennwert des numerischen Oszillators, nämlich der Spurious Free Dynamic Range, also der Abstand zwischen gewünschtem Signal und Rauschen, ist im Vergleich zum bisher eingesetzten ISL5216 um 25dB auf 140dB gestiegen.

Die verwendete Filterstruktur ist hingegen gleich geblieben. So wurde auch im neuen Design eine Kombination von einem CIC- und zwei FIR-Filtern eingesetzt und auch die Dezimierung ist mit 920 unverändert geblieben. Diese zu ändern wurde auch nie in Betracht gezogen, da das Ausgangssignal dieselbe Taktrate haben soll wie bisher.

Der bisher eingesetzte ADC hat eine Datenbreite von 14 Bit. Der hier entstandene DDC wurde auf einen ADC von 16Bit ausgelegt, dessen Verwendung allerdings auch mit dem ISL5216 möglich wäre.

Ein kleiner, aber tatsächlicher Zuwachs an Datenbreite ist jedoch die neue interne Datenbreite von 25 Bit gegenüber den bisherigen 24 Bit. Wie im Kapitel 7 gezeigt, macht die Verwendung einer kleineren internen Datenbreite keinen Sinn, da trotzdem genausoviele Multiplikatoren verwendet werden müssten.

Einziger Nachteil und somit Verbesserungspotenzial der entstandenen Lösung ist die maximal mögliche Sample-Rate bei 8 implementierten DDC Kanälen. Diese beträgt bei einem FPGA mit Speedgrade 1 nur 92.4Msps. Bei diesem Test wurden alle DDCs und deren NCOs leicht unterschiedlich implementiert um ansonsten auftretende Optimierungen zu verhindern und somit die implementation von 8 unabhängigen Kanälen zu gewährleisten. Die 92.4Msps erfüllen die Erwartungen nicht, da wie schon in der Aufgabenstellung erwähnt ein Framework eingesetzt werden soll, welches mit einer Taktrate von 100MHz läuft.

Somit werden alle Anforderungen bis auf die maximal ausführbare Geschwindigkeit erfüllt. Zu beachten ist, dass alle Frequenzangaben in vorherigen Kapiteln sich auf ein FPGA mit Speegrade 1, also dem langsamsten Speedgrade beziehen. Nun besteht die Möglichkeit, ein FPGA mit einem schnelleren Speedgrade zu verwenden. Die folgende Übersicht zeigt, wie viel Geschwindigkeitsgewinn dies bringen würde:

- Speedgrade 1: 92.4MHz (1013.75$)
- Speedgrade 2: 100.2MHz (1267.50$)
- Speedgrade 3: 102.3MHz (1773.75$)

Durch gezieltes setzen von Timing-constraints kann die Geschwindigkeit bei der Implementation oft weiter verbessert werden, da das Tool mit Optimieren aufhört, wenn diese eingehalten wurden. Allerdings hat ein Erhöhen des Timing-contraints von 100MHz auf 105MHz für Speedgrade 1 keinen Geschwindigkeitsgewinn gebracht und daher kann angenommen werden, dass mit 102.3MHz das Maximum oder beinahe das Maximum erreicht wurde. Selbst die 102.3MHz mit Speedgrade 3 befindet sich an der unteren Grenze, wenn man beachtet dass das dazukommende Framework die Ausführungsgeschwindigkeit weiter reduzieren könnte. Es wäre daher besser, noch etwas mehr Puffer zu haben. Ausserdem sind die FPGAs mit Speedgrade 3 wie oben ersichtlich massiv teurer als Speedgrade 1 oder 2 (Preise von AVNET, Stand Juni 2011) [AVN11]. Daher sollte weiterhin die Verwendung von Speedgrade 1 angestrebt werden.

Wenn die demodulierten Signale nicht wie in der momentanen Simulation alle einzeln an Pin-Ausgänge geführt werden müssen, wird das Design womöglich noch etwas schneller. Dieser Geschwindigkeitsgewinn dürfte allerdings durch den Code vom Framework, welches noch hinzukommt, wieder zunichte gemacht werden. Wahrscheinlich wird das Design durch diese beiden Änderungen eher langsamer als schneller.

Als Nadelöhr, das langsamste Glied, konnten wie in Kapitel 7 beschrieben die FIR-Filter ausfindig gemacht werden. Die weitere Optimierung anderer Teile der Schaltung wurde daher nicht verfolgt. Da wie in Abbildung 7.2 auf Seite 30 ersichtlich weniger als 60% der im FPGA verfügbaren Logik verwendet wurde, konnte auch diesbezüglich auf weiteres Optimieren verzichtet werden. Mit den erzielten Optimierungen in den FIR-Filtern wurden das Potenzial der Simulink-Blöcke und des HDL-Coders bestmöglich ausgenützt, daher scheint eine weitere Verbesserung nur mit Simulink und HDL-Coder nicht möglich. Im folgenden wurde daher nach weiteren Optionen gesucht, welche vielleicht zu einer Verbesserung führen könnten.

9.2 Xilinx DDC Core

Xilinx stellt schon seit längerem einen fertigen DDC-Core zur Verfügung, welcher zwischenzeitlich nicht mehr weiter unterstützt wurde [Xil11a]. Mit der neuen Version von Xilinx ISE kam Ende 2010 der „LogiCORE IP DUC/DDC Compiler v1.1“ auf den Markt [Xil11b]. Die Performance dieser DDC in einem Virtex6 ist in der Dokumentation auf Seite 31/32 ersichtlich. Da aber nur Wireless-Standards bis 20MHz unterstützt werden, scheint die Verwendung dieses Compilers für unsere Anwendung nicht günstig. Des weiteren scheinen auch die Konfigurationsmöglichkeiten relativ eingeschränkt zu sein. Weiter Abklärungen könnten aber sinnvoll sein. Im Blogeintrag [Xil11a] wurde allerdings der viel flexiblere Xilinx System Generator erwähnt, welcher als nächstes vorgestellt wird.

9.3 Xilinx System Generator

Eine Möglichkeit, um die Ausführungsgeschwindigkeit noch zu beschleunigen, ist die Verwendung des Xilinx System Generators [Xil11c]. Xilinx stellt mit diesem Tool Blöcke in Simulink zur Verfügung, welche speziell auf die Hardware von Xilinx angepasst und optimiert sind. Sowohl MathWorks wie auch Xilinx stellen Anleitungen zum System Generator zur Verfügung [Mat11i, Xil11d]. Anhand der Dokumentation von Xilinx fand eine Einarbeitung in dieses Thema statt, der Computer wurde für die Xilinx-Blöcke konfiguriert und einige der aufgeführten Übungsbeispiele wurden ausprobiert [Xil11d]. Der DDC konnte jedoch aus Zeitgründen noch nicht mit Xilinx-Blöcken realisiert werden. Eine Verschnellerung des Designs ist aber unter Verwendung der Xilinx-spezifischen Blocks sehr wohl denkbar. Daher sollte diese Option weiter verfolgt werden.

9.4 Fazit

Die 8 DDC-Kanäle mit aus Simulink generiertem VHDL-Code erreichen die gewünschte Geschwindigkeit knapp nicht. Verbesserungen können wahrscheinlich mit dem Xilinx System-Generator

und den Xilinx-spezifischen Blöcken erreicht werden. Ansonsten muss ein höherer Speedgrade eingesetzt werden, wobei auch dann die Realisierbarkeit mit 100MHz fraglich ist. Andernfalls muss die Implementierung mit einer von der Framework-Frequenz unterschiedlichen Sample-Rate erfolgen. Dies würde jedoch im Gegensatz zur Implementation mit der Framework-Frequenz von 100MHz einen grossen Mehraufwand bedeuten.

10 Zusammenfassung und Ausblick

Im Rahmen dieser Thesis wurde ein Digital Down Converter anhand des in Kapitel 5 beschriebenen Workflows mit HDL-Coder-fähigen Simulink-Blöcken realisiert, VHDL-Code generiert, überprüft und in einem FPGA implementiert. Zweck des DDC ist die in Kapitel 2 beschriebene Bestimmung von Position und Phase des Protonenstrahls am Protonenbeschleuniger des PSI. Die einzelnen Funktionsblöcke wurden dabei wie in Kapitel 6 beschrieben entsprechend dem bereits bestehenden System einzeln simuliert und getestet und schlussendlich Anhand der in Kapitel 7 beschriebenen Konzepte optimiert. Damit konnte erreicht werden, dass 8 DDC-Kanäle im vorgegebenen Virtex-6 FPGA mit einer Frequenz von 92.4MHz beim langsamsten, bzw. 102.3MHz beim schnellsten Speedgrade lauffähig sind. Es konnte gezeigt werden, dass die Erzeugung von VHDL-Code aus Matlab Simulink und nachfolgende Implementierung des Designs ins FPGA funktioniert. Das Kapitel 8 zeigt weiter, dass die Systemeigenschaften mit der Theorie grösstenteils übereinstimmen. Einzig eine Verbesserung des NCO durch hinzufügen eines Dither konnte nicht nachgewiesen werden. Dieser wird aber nicht zwingend benötigt und konnte aufgrund der bereits sehr hohen Qualität des Oszillators weggelassen werden.

Da die gewünschte FPGA-Taktrate von 100MHz nur bedingt erreicht und selbst beim schnellsten Speedgrade nur knapp überschritten wurde, sollte in weiterführenden Arbeiten untersucht werden, ob die Verwendung des Xilinx System Generators und damit die Verwendung von Hardware-optimierten Blöcken zu besseren Resultaten führt. Desweiteren müssen die 8 DDC-Kanäle zusammen mit dem bei einer externen Firma in Entwicklung stehenden Framework in das FPGA implementiert werden und Hardware-Tests durchgeführt werden, sobald das Framework und das ebenfalls in Entwicklung befindliche Board verfügbar sind.

A Zeitplan

#	Aufgabenname	Anfang	Abschluss	Dauer
1	Einarbeiten ins Thema	28.02.2011	07.03.2011	6t
2	Theorie Anhand des Buches „Signale Prozesse Systeme“ [Kar10]	02.03.2011	14.03.2011	9t
3	Einarbeiten in Simulink und andere Software	15.03.2011	18.03.2011	4t
4	Modulieren Digital Down Converter	23.03.2011	29.04.2011	28t
5	Theorie Anhand des Buches „Signale Prozesse Systeme“ [Kar10]	28.03.2011	21.04.2011	19t
6	*Militär*	*02.05.2011*	*20.05.2011*	*15t*
7	Optimierung Digital Down Converter	23.05.2011	03.06.2011	10t
8	Modulieren mit Xilinx-Blöcken	13.06.2011	16.06.2011	4t
9	Erstellen der Dokumentation & Präsentation	28.02.2011	14.07.2011	99t
10	Präsentation	30.06.2011	30.06.2011	1t

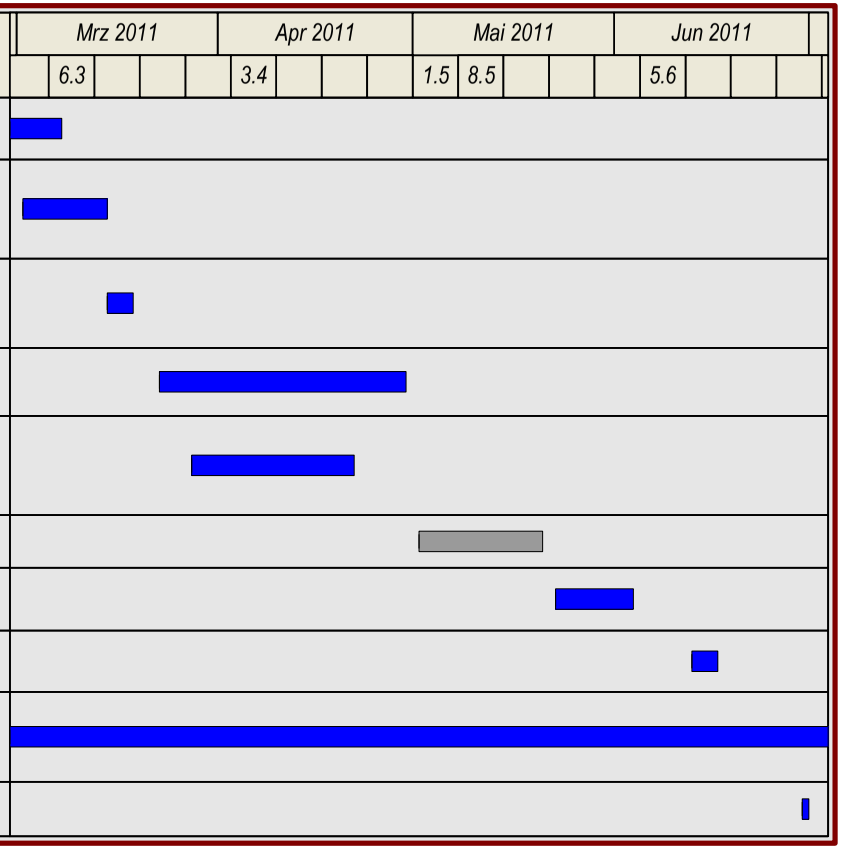


Abbildung A.1: Zeitplan

B Nomenklatur

B.1 Abkürzungen

ADC	Analog Digital Converter
ASIC	Application Specific Integrated Circuit
DC	Direct Current, Gleichstrom
DDC	Digital Down Converter
DSP	Digital Signal Processor
DUT	Device Under Test
FPGA	Field Programmable Gate Array
LFSR	Linear Feedback Shift Register
MSB	Most Significant Bit
NCO	Numerically Controlled Oscillator
PSI	Paul Scherrer Institut
SFDR	Spurious Free Dynamic Range
SNR	Signal to Noise Ratio
VHDL	Very high speed integrated circuit Hardware Description Language

C Inhaltsverzeichnis der CD

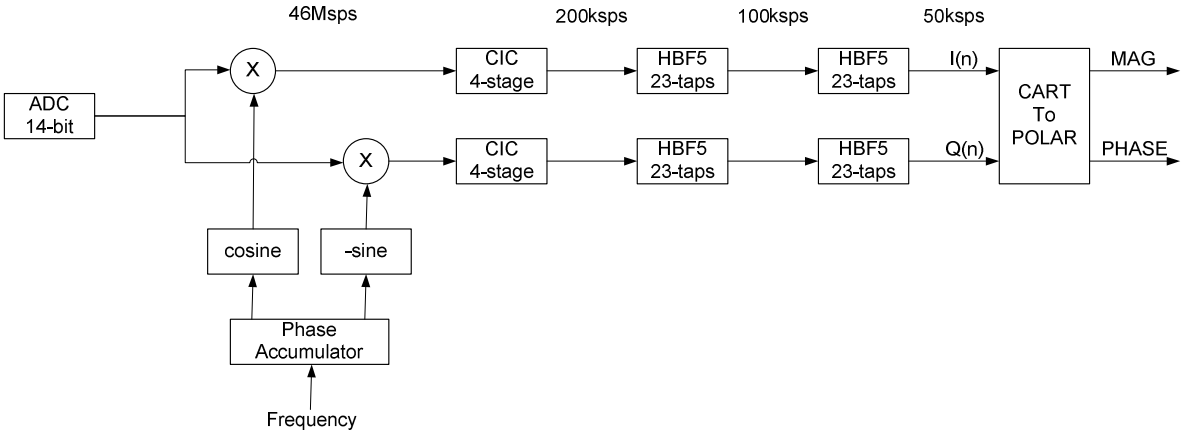
Folgende Files sind auf der beigelegten CD enthalten:

- Thesis Ausarbeitung Jonas Müller FS2011.pdf
- Thesis Poster Jonas Müller FS2011.pdf
- Thesis Präsentation Jonas Müller FS2011.pdf
- Ordner Simulink mit folgendem Inhalt:
    1. DDC.mdl
    2. filter_cic.mdl
    3. filter_model.mdl
    4. Cartesian2Polar.mdl
    5. lfsr.mdl
    6. nco_block.mdl
    7. nco_modular_dither.mdl

D Aufgabenstellung von E. Johansen

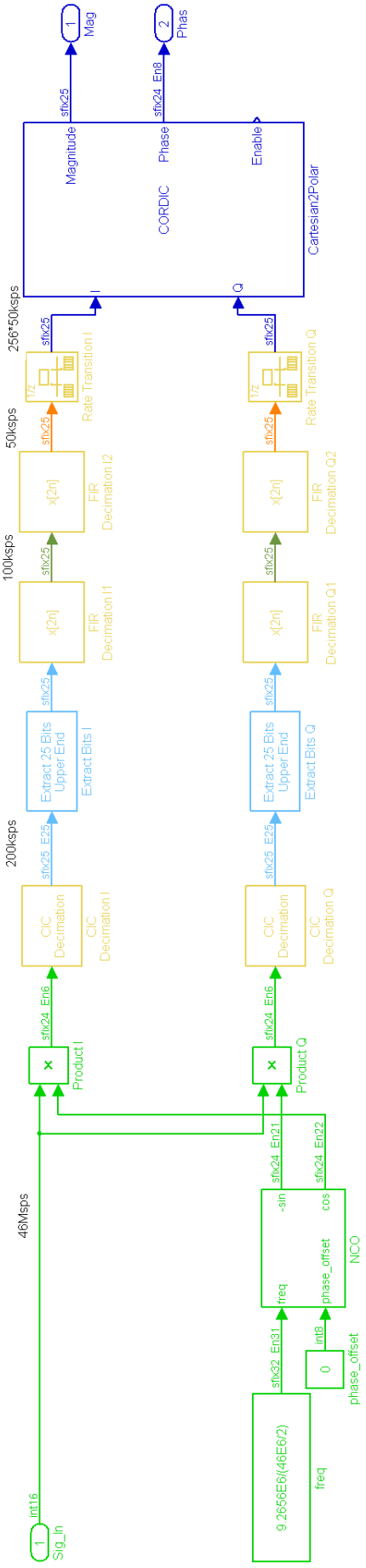
Digitale Signalverarbeitung zur Auswertung von Teilchen-Strahlen.

In Teilchenbeschleunigern werden geladene Teilchen auf grosse Geschwindigkeiten beschleunigt. Dabei entstehen gepulste Teilchenstrahlen, die sich durch ihre elektromagnetische Felder messen lassen. Die daraus erzeugten Hochfrequenzsignale werden mit Signalverarbeitung ausgewertet um wichtige Parameter des Strahls zu bestimmen, wie Position oder Phasenlage.
Gemeinsam für viele diese Messsysteme kommen Demodulationsverfahren zum Einsatz, die ausgewählte Spektralkomponenten des Signals in Intensität und Phasenlage messen. Es handelt sich hier um Algorithmen die man aus der Radiotechnik kennt.
In der Vergangenheit wurden diese Systeme in analoger Technik realisiert, später mit Hilfe von ASICs. Um diese Messsysteme flexibler und leistungsfähiger zu gestalten, wird es untersucht, ob digitaler Radio Verfahren sich mit Hilfe von Matlab/Simulink und modernen FPGAs implementieren lässt. Der Workflow gibt es seit einigen Jahren, aber die Ergebnisse waren in der Vergangenheit nicht befriedigend. Jetzt stehen verbesserte Werkzeuge zur Verfügung, die versprechen handkodierte Code ersetzten zu können. Dabei werden die neuen verbesserten Eigenschaften des Simulink HDL Coder untersucht.
Die Arbeit besteht darin zuerst das Demodulationsverfahren funktionell in Matlab/Simulink zu modellieren und durch eine Simulation zu verifizieren. Dabei werden die Funktionen eines „Digital-Down-Converters“ nachgebildet.

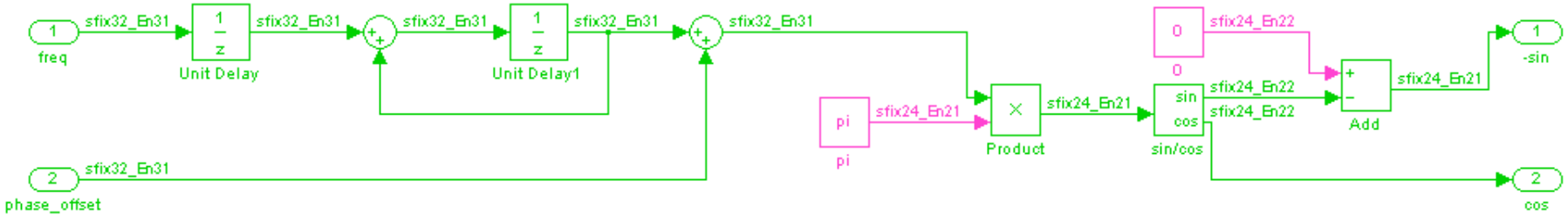


Anschliessend wird aus dem funktionellen Modell ein bitgenaues Matlab/Simulink Modell erzeugt. Dieses Modell wird so gestaltet, dass es sich für die Umwandlung in VHDL eignet. Hier kommen Filter-Design Werkzeuge von Matlab/Simulink zum Einsatz.
Es wird untersucht welche Teile der Schaltung sich automatisiert in VHDL umsetzten lässt. Funktionen die sich nicht umsetzten lassen werden für die weitere Analyse nicht berücksichtigt.
Das Modell wird mit Hilfe eines Simulators auf seine Funktion überprüft. Dabei werden Vergleichmessung zwischen das funktionale Modell und das bitgenaue Modell durchgeführt. Es werden Parameter wie Bitauflösung und Pipelining variiert und den Einfluss auf die Simulationsergebnisse ausgewertet.
Das bitgenaue Matlab/Simulink Modell wird anschliessend mit Hilfe des Simulink HDL Coders von Simulink Code in VHDL Code umgewandelt. Nachdem das Modell in VHDL vorliegt wird FPGA Synthese durchgeführt. Der VHDL Compiler setzt dabei die Schaltung in FPGA Gatter um. Es werden erneut Parameter in dem Simulink Modell verändert und die Auswirkung auf die synthetisierte Schaltung dokumentiert. Die Ergebnisse der FPGA Synthese werden auf Flächenbedarf und Taktrate untersucht um die Qualität des Workflows zu beurteilen.
Die Ergebnisse werden mit den Ressourcen eines neuen Rechners verglichen, der zurzeit in Entwicklung ist, und eine Aussage über die praktische Umsetzung des Verfahrens gemacht.

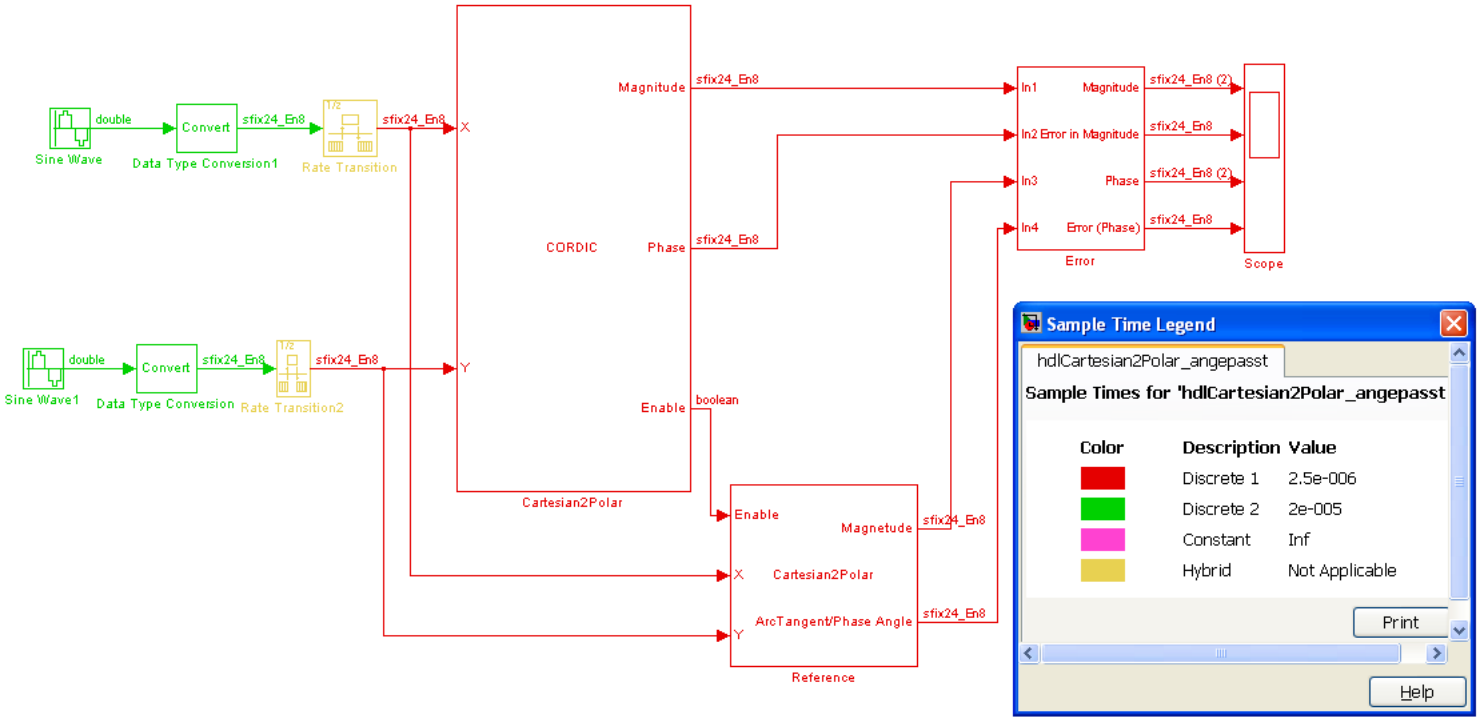
E Blockschaltbild des Digital Down Converters



F Blockschaltbild des Numerically Controller Oscillator (NCO)



G Blockschaltbild zur Überprüfung des Koordinatenwandlung



H HDL-Coder Fehlermeldungen und Warnings

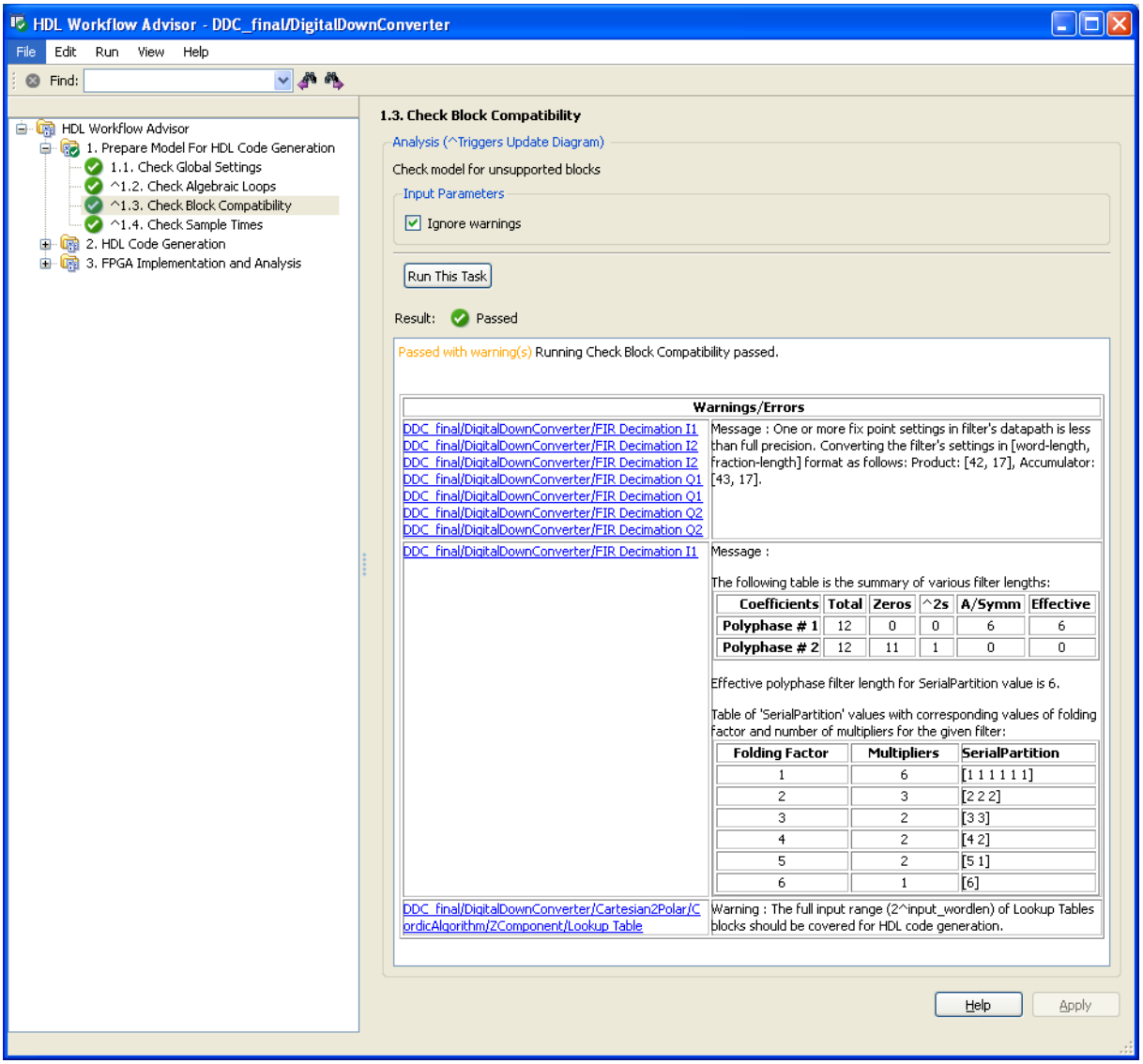


Abbildung H.1: Warning des HDL Coder bezüglich Block Compatibility

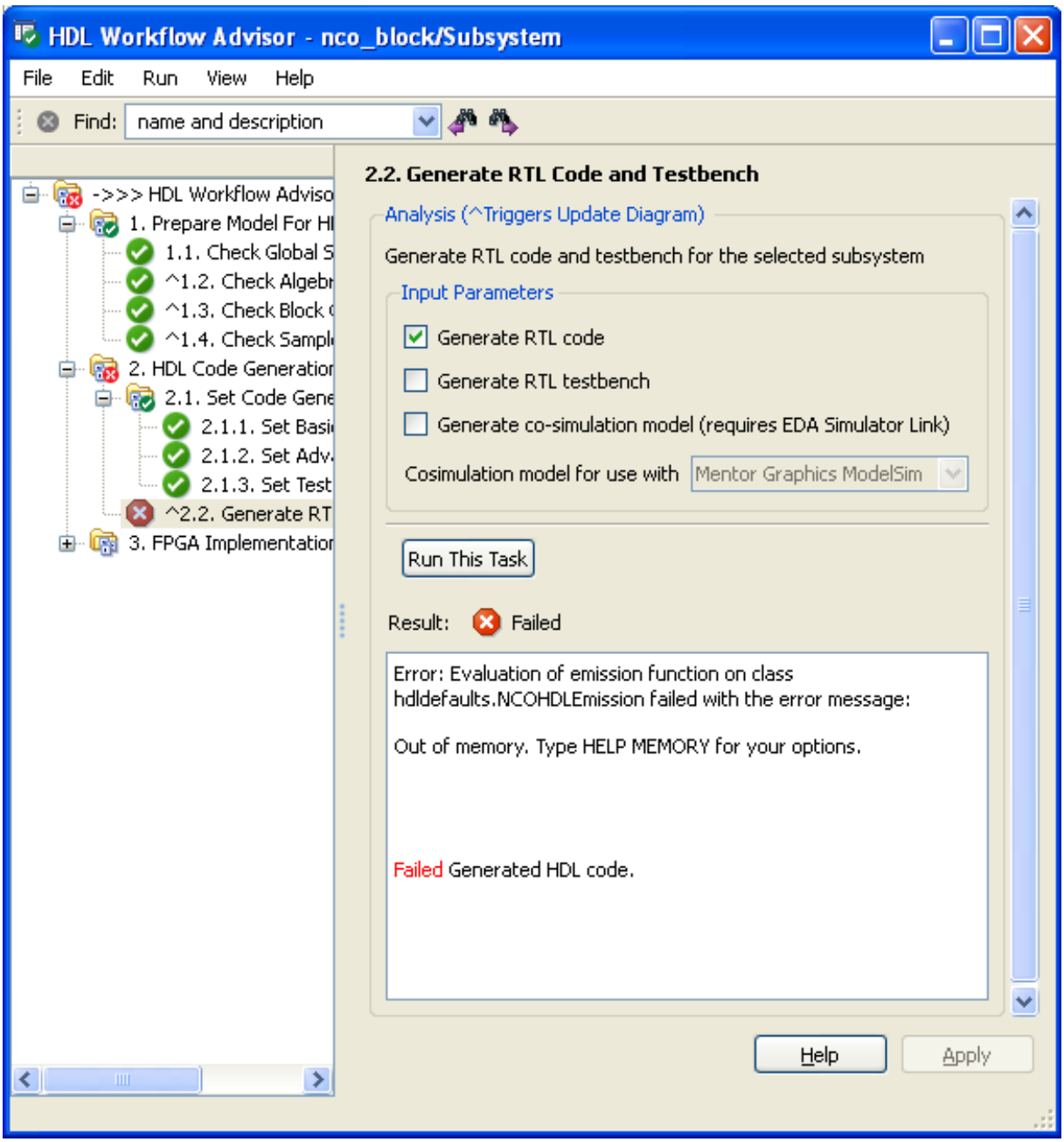


Abbildung H.2: Fehlermeldung NCO Block

I VHDL-Code-Ausschnitt als Beispiel

Als Ausschnitt wurde hier der Block NCO gewählt. Entsprechend heisst das dafür erzeugte File „nco.vhd“. Der Modulname und wie man zu diesem Modul kommt, ist ebenfalls im Header ausgewiesen. Auch die erzeugte „ENTITY“ hat den entsprechenden Namen. Die erzeugten Datentypen der „ENTITY“ entsprechen den Datentypen in Simulink, welche dahinter auskommentiert sichtbar sind. Anschliessend folgen alle Blöcke, welche in Simulink im NCO-Block sind, als „COMPONENT“. Nach der Definition der internen Signale folgt die Eigentliche Logik.

```
1  -- -------------------------------------------------------------
2  --
3  -- File Name: hdlsrc/DDC\nco.vhd
4  -- Created: 2011-06-29 09:54:20
5  --
6  -- Generated by MATLAB 7.11 and Simulink HDL Coder 2.0
7  --
8  -- -------------------------------------------------------------
9
10
11 -- -------------------------------------------------------------
12 --
13 -- Module: nco
14 -- Source Path: DDC/DigitalDownConverter/nco
15 -- Hierarchy Level: 1
16 --
17 -- -------------------------------------------------------------
18 LIBRARY IEEE;
19 USE IEEE.std_logic_1164.ALL;
20 USE IEEE.numeric_std.ALL;
21
22 ENTITY nco IS
23   PORT( clk : IN std_logic;
24         reset : IN std_logic;
25         enb_1_32_0 : IN std_logic;
26         enb_const_rate : IN std_logic;
27         freq : IN std_logic_vector(31 DOWNTO 0); -- sfix32_En31
28         phase_offset : IN std_logic_vector(31 DOWNTO 0); -- sfix32_En31
29         alphasin : OUT std_logic_vector(23 DOWNTO 0); -- sfix24_En21
30         cos_1 : OUT std_logic_vector(23 DOWNTO 0) -- sfix24_En22
31         );
32 END nco;
33
```

```
34
35 ARCHITECTURE rtl OF nco IS
36
37   -- Component Declarations
38   COMPONENT cos
39     PORT( clk : IN std_logic;
40           reset : IN std_logic;
41           enb_1_32_0 : IN std_logic;
42           angle : IN std_logic_vector(23 DOWNTO 0); -- sfix24_En21
43           sin : OUT std_logic_vector(23 DOWNTO 0); -- sfix24_En22
44           cos_1 : OUT std_logic_vector(23 DOWNTO 0) -- sfix24_En22
45           );
46   END COMPONENT;
47
48   -- Component Configuration Statements
49   FOR ALL : cos
50     USE ENTITY work.cos(rtl);
51
52   -- Signals
53   SIGNAL alpha0_out1 : signed(23 DOWNTO 0); -- sfix24_En22
54   SIGNAL freq_signed : signed(31 DOWNTO 0); -- sfix32_En31
55   SIGNAL Unit_Delay_out1 : signed(31 DOWNTO 0); -- sfix32_En31
56   SIGNAL Unit_Delay1_out1 : signed(31 DOWNTO 0); -- sfix32_En31
57   SIGNAL Sum_out1 : signed(31 DOWNTO 0); -- sfix32_En31
58   SIGNAL phase_offset_signed : signed(31 DOWNTO 0); -- sfix32_En31
59   SIGNAL Sum1_out1 : signed(31 DOWNTO 0); -- sfix32_En31
60   SIGNAL Sum1_out1_1 : signed(31 DOWNTO 0); -- sfix32_En31
61   SIGNAL pi_out1 : signed(23 DOWNTO 0); -- sfix24_En21
62   SIGNAL pi_out1_1 : signed(23 DOWNTO 0); -- sfix24_En21
63   SIGNAL Product_mul_temp : signed(55 DOWNTO 0); -- sfix56_En52
64   SIGNAL Product_out1 : signed(23 DOWNTO 0); -- sfix24_En21
65   SIGNAL Product_out1_1 : signed(23 DOWNTO 0); -- sfix24_En21
66   SIGNAL Product_out1_2 : signed(23 DOWNTO 0); -- sfix24_En21
67   SIGNAL sin_cos_out1 : std_logic_vector(23 DOWNTO 0); -- ufix24
68   SIGNAL sin_cos_out2 : std_logic_vector(23 DOWNTO 0); -- ufix24
69   SIGNAL sin_cos_out1_signed : signed(23 DOWNTO 0); -- sfix24_En22
70   SIGNAL sin_cos_out1_1 : signed(23 DOWNTO 0); -- sfix24_En22
71   SIGNAL Add_sub_cast : signed(24 DOWNTO 0); -- sfix25_En22
72   SIGNAL Add_sub_cast_1 : signed(24 DOWNTO 0); -- sfix25_En22
73   SIGNAL Add_sub_temp : signed(24 DOWNTO 0); -- sfix25_En22
74   SIGNAL Add_out1 : signed(23 DOWNTO 0); -- sfix24_En21
75   SIGNAL sin_cos_out2_signed : signed(23 DOWNTO 0); -- sfix24_En22
76   SIGNAL sin_cos_out2_1 : signed(23 DOWNTO 0); -- sfix24_En22
77
78 BEGIN
79   u_sin_cos : cos
80     PORT MAP( clk => clk,
81               reset => reset,
```

```
82              enb_1_32_0 => enb_1_32_0,
83              angle => std_logic_vector(Product_out1_2), -- sfix24_En21
84              sin => sin_cos_out1, -- sfix24_En22
85              cos_1 => sin_cos_out2 -- sfix24_En22
86              );
87
88  alpha0_out1 <= to_signed(0, 24);
89
90  freq_signed <= signed(freq);
91
92  Unit_Delay_process : PROCESS (clk, reset)
93  BEGIN
94    IF reset = '1' THEN
95      Unit_Delay_out1 <= to_signed(0, 32);
96    ELSIF clk'EVENT AND clk = '1' THEN
97      IF enb_1_32_0 = '1' THEN
98        Unit_Delay_out1 <= freq_signed;
99      END IF;
100   END IF;
101 END PROCESS Unit_Delay_process;
102
103
104 Sum_out1 <= Unit_Delay_out1 + Unit_Delay1_out1;
105
106 Unit_Delay1_process : PROCESS (clk, reset)
107 BEGIN
108   IF reset = '1' THEN
109     Unit_Delay1_out1 <= to_signed(0, 32);
110   ELSIF clk'EVENT AND clk = '1' THEN
111     IF enb_1_32_0 = '1' THEN
112       Unit_Delay1_out1 <= Sum_out1;
113     END IF;
114   END IF;
115 END PROCESS Unit_Delay1_process;
116
117
118 phase_offset_signed <= signed(phase_offset);
119
120 Sum1_out1 <= Unit_Delay1_out1 + phase_offset_signed;
121
122 Product_in_pipe0_process : PROCESS (clk, reset)
123 BEGIN
124   IF reset = '1' THEN
125     Sum1_out1_1 <= to_signed(0, 32);
126   ELSIF clk'EVENT AND clk = '1' THEN
127     IF enb_1_32_0 = '1' THEN
128       Sum1_out1_1 <= Sum1_out1;
129     END IF;
```

```
130   END IF;
131 END PROCESS Product_in_pipe0_process;
132
133
134 pi_out1 <= to_signed(6588397, 24);
135
136 Product_in_pipe1_process : PROCESS (clk, reset)
137 BEGIN
138   IF reset = '1' THEN
139     pi_out1_1 <= to_signed(0, 24);
140   ELSIF clk'EVENT AND clk = '1' THEN
141     IF enb_const_rate = '1' THEN
142       pi_out1_1 <= pi_out1;
143     END IF;
144   END IF;
145 END PROCESS Product_in_pipe1_process;
146
147
148 Product_mul_temp <= Sum1_out1_1 * pi_out1_1;
149 Product_out1 <= Product_mul_temp(54 DOWNTO 31) + ("0" & (Product_mul_temp(55)
      AND (Product_mul_temp(30) OR Product_mul_temp(29) OR Product_mul_temp(28)
      OR Product_mul_temp(27) OR Product_mul_temp(26) OR Product_mul_temp(25) OR
      Product_mul_temp(24) OR Product_mul_temp(23) OR Product_mul_temp(22) OR
      Product_mul_temp(21) OR Product_mul_temp(20) OR Product_mul_temp(19) OR
      Product_mul_temp(18) OR Product_mul_temp(17) OR Product_mul_temp(16) OR
      Product_mul_temp(15) OR Product_mul_temp(14) OR Product_mul_temp(13) OR
      Product_mul_temp(12) OR Product_mul_temp(11) OR Product_mul_temp(10) OR
      Product_mul_temp(9) OR Product_mul_temp(8) OR Product_mul_temp(7) OR
      Product_mul_temp(6) OR Product_mul_temp(5) OR Product_mul_temp(4) OR
      Product_mul_temp(3) OR Product_mul_temp(2) OR Product_mul_temp(1) OR
      Product_mul_temp(0))));
150
151 Product_out_pipe_process : PROCESS (clk, reset)
152 BEGIN
153   IF reset = '1' THEN
154     Product_out1_1 <= to_signed(0, 24);
155   ELSIF clk'EVENT AND clk = '1' THEN
156     IF enb_1_32_0 = '1' THEN
157       Product_out1_1 <= Product_out1;
158     END IF;
159   END IF;
160 END PROCESS Product_out_pipe_process;
161
162
163 sin_cos_in_pipe_process : PROCESS (clk, reset)
164 BEGIN
165   IF reset = '1' THEN
166     Product_out1_2 <= to_signed(0, 24);
```

```
167     ELSIF clk'EVENT AND clk = '1' THEN
168       IF enb_1_32_0 = '1' THEN
169         Product_out1_2 <= Product_out1_1;
170       END IF;
171     END IF;
172   END PROCESS sin_cos_in_pipe_process;
173
174
175   sin_cos_out1_signed <= signed(sin_cos_out1);
176
177   sin_cos_out_pipe0_process : PROCESS (clk, reset)
178   BEGIN
179     IF reset = '1' THEN
180       sin_cos_out1_1 <= to_signed(0, 24);
181     ELSIF clk'EVENT AND clk = '1' THEN
182       IF enb_1_32_0 = '1' THEN
183         sin_cos_out1_1 <= sin_cos_out1_signed;
184       END IF;
185     END IF;
186   END PROCESS sin_cos_out_pipe0_process;
187
188
189   Add_sub_cast <= resize(alpha0_out1, 25);
190   Add_sub_cast_1 <= resize(sin_cos_out1_1, 25);
191   Add_sub_temp <= Add_sub_cast - Add_sub_cast_1;
192   Add_out1 <= Add_sub_temp(24 DOWNTO 1);
193
194   alphasin <= std_logic_vector(Add_out1);
195
196   sin_cos_out2_signed <= signed(sin_cos_out2);
197
198   sin_cos_out_pipe1_process : PROCESS (clk, reset)
199   BEGIN
200     IF reset = '1' THEN
201       sin_cos_out2_1 <= to_signed(0, 24);
202     ELSIF clk'EVENT AND clk = '1' THEN
203       IF enb_1_32_0 = '1' THEN
204         sin_cos_out2_1 <= sin_cos_out2_signed;
205       END IF;
206     END IF;
207   END PROCESS sin_cos_out_pipe1_process;
208
209
210   cos_1 <= std_logic_vector(sin_cos_out2_1);
211
212 END rtl;
```

J E-Mail Strahlstrom-Rekord

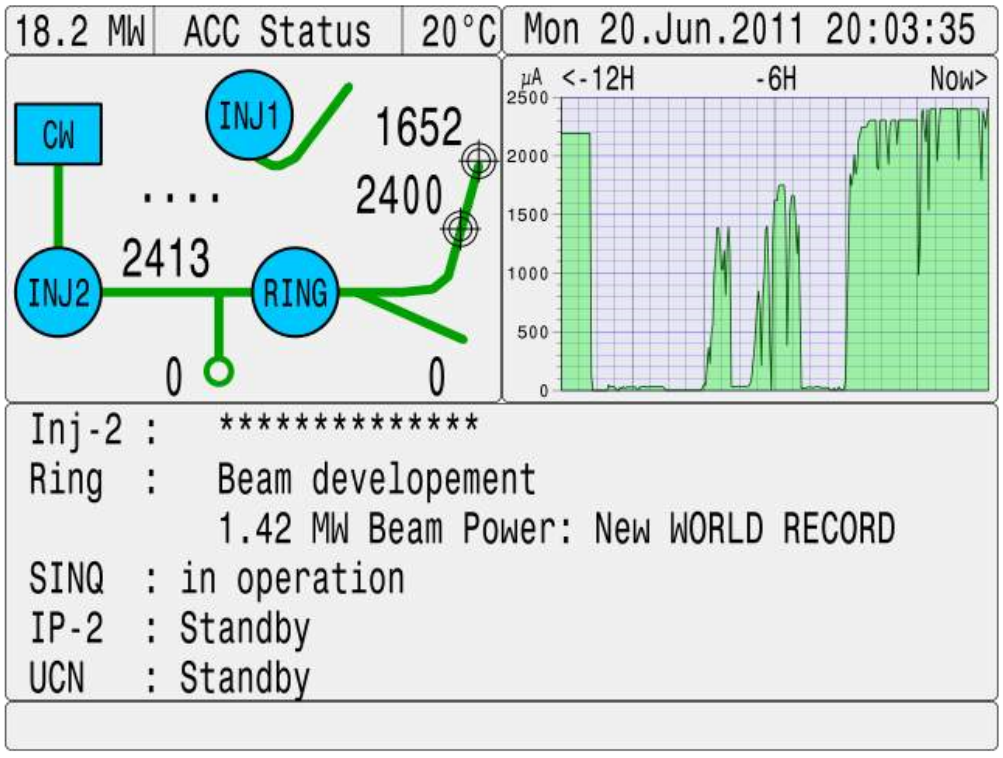
Von Seidel Mike <Mike.Seidel@psi.ch>
Betreff **HIPA Rekordstrom** 21.06.2011 08:39
An GFA-ABE <GFA-ABE@psi.ch>, GFA-ABK <GFA-ABK@psi.ch>, GFA-ATK <GFA-ATK@p: **34 weitere**
CC Rivkin Leonid <leonid.rivkin@psi.ch>, Clausen Kurt <kurt.clausen@psi.ch>, Allenspach Peter <peter

Liebe Kolleginnen und Kollegen

Seit zwei Jahren besitzen wir die BAG Bewilligung für Testbetrieb bei einem Maximalstrom von 2.4mA in der Protonenanlage. Gestern ist es in der Strahlentwicklung erstmalig gelungen, diesen neuen Rekord auch tatsächlich einzustellen.

Der hohe Strahlstrom ist möglich da die Verluste an der Extraktion des Ringzyklotrons auf Werte unter 1e-4 relativ zur nominalen Strahlleistung von 1.3MW abgesenkt wurden. Wir führen das auf verschiedene Massnahmen im letzten Shutdown zurück – die Beseitigung von vertikalen Aperturbeschränkungen, eine bessere Strahlqualität aus der neuen ECR Quelle sowie die Reduktion von 50Hz Restrippeln auf bestimmten empfindlichen Speisegeräten. Somit ist dieser schöne Erfolg das Ergebnis einer gemeinsamen Anstrengung von ganz verschiedenen Fachgruppen des PSI.

Vielen Dank an alle Beteiligten, Mike Seidel.



K Literaturverzeichnis

[Alf96] Alfke, Peter.
Xilinx Application Note XAPP 052: Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators.
http://www.xilinx.com/support/documentation/application_notes/xapp052.pdf.
Juli 1996

[And11] Andraka, Ray.
A survey of CORDIC algorithms for FPGA based computers.
http://www.andraka.com/files/crdcsrvy.pdf.
2011

[AVN11] AVNET.
AVNET PartBuilder (FPGA Lieferinformationen).
http://avnetexpress.avnet.com/store/em/EMController?action=compare&R=12845672+12251388+12847396&N=0&catalogId=500201&storeId=500201&langId=-1.
2011

[Don11] Donadio, Matthew P.
CIC Filter Introduction.
http://www.mikrocontroller.net/attachment/51932/cic2.pdf.
2011

[Eng11] Engineering, Hunt.
Digital DownConversion (DDC) using FPGA.
http://www.hunteng.co.uk/support/ddc.htm.
2011

[Int07] Intersil.
Data Sheet ISL5216 (FN6013.3).
http://www.intersil.com/data/fn/fn6013.pdf.
Juli 2007

[Joh10] Johansen, Ernst.
VME pDBPM Design Description.
Paul Scherrer Institut, 5232 Villigen PSI.
2010

[Kar10] Karrenberg, Ulrich:
Signale Prozesse Systeme.
5., neu bearb. u. erw. Aufl.
Berlin Heidelberg : Springer-Verlag, 2010. –
ISBN 978–3–642–01863–3

[Mat11a] MathWorks.
CIC Decimation.

Matlab-Hilfe: Signal Processing Blockset -> Blocks -> Filtering -> Multirate Filters -> CIC Decimation.
2011

[Mat11b] MathWorks.
DSP System Toolbox Demos.
http://www.mathworks.com/products/dsp-system/demos.html.
2011

[Mat11c] MathWorks.
DSP System Toolbox Filter Chain Demo.
http://www.mathworks.com/products/dsp-system/demos.html?file=/products/demos/shipping/dsp/ddcfilterchaindemo.html#1.
2011

[Mat11d] MathWorks.
DSP System Toolbox GSM Digital Down Converter.
http://www.mathworks.com/products/dsp-system/demos.html?file=/products/demos/shipping/dsp/dspDigitalDownConverter.html.
2011

[Mat11e] MathWorks.
FIR Decimation.
Matlab-Hilfe: Signal Processing Blockset -> Blocks -> Filtering -> Multirate Filters -> FIR Decimation.
2011

[Mat11f] MathWorks.
FIR Decimation.
Matlab-Hilfe: Simulink HDL Coder -> User's Guide -> Guide to Supported Blocks and Block Implementations -> Block Implementation Parameters.
2011

[Mat11g] MathWorks.
GSM Digital Down Converter.
Matlab-Hilfe: Signal Processing Blockset -> Demos -> Simulink Demos -> Communications -> GSM Digital Down Converter.
2011

[Mat11h] MathWorks.
Increasing NCO Spurious-Free Dynamic Range.
http://www.mathworks.co.uk/products/communications/demos.html?file=/products/demos/shipping/comm/commncopnseqdither.html.
2011

[Mat11i] MathWorks.
Integrating Xilinx System Generator and Simulink HDL Coder (Application Guideline).
http://www.mathworks.com/mason/tag/proxy.html?dataid=10718&fileid=59570.
2011

[Mat11j] MathWorks.
Random Number Block.
http://www.mathworks.com/help/toolbox/simulink/slref/randomnumber.html.
2011

[Mat11k] MathWorks.

Simulink: Digital Waveform Generation: Approximating a Sine Wave.
http://www.mathworks.ch/products/simulink/demos.html?file=/products/demos/shipping/simulink/sldemo_wavethd_script.html.
2011

[Mat11l] MathWorks.
Simulink HDL Coder 2 User's Guide.
http://www.mathworks.com/help/pdf_doc/slhdlcoder/slhdlcoder_ug.pdf.
2011

[Mat11m] MathWorks.
Simulink HDL Coder 2.1.
http://www.mathworks.com/products/datasheets/pdf/simulink-hdl-coder.pdf.
2011

[Mat11n] MathWorks.
Simulink HDL Coder: CORDIC Algorithm Using Simulink® Blocks.
http://www.mathworks.com/products/slhdlcoder/demos.html?file=/products/demos/shipping/hdlcoder/hdlcodercordic.html.
2011

[Mat11o] MathWorks.
Simulink HDL Coder: Generate HDL code from Simulink models and MATLAB code.
http://www.mathworks.com/products/slhdlcoder/.
2011

[Mat11p] MathWorks.
Simulink Help NCO.
http://www.mathworks.ch/help/toolbox/dsp/ref/nco.html.
2011

[Mat11q] MathWorks.
Why do I receive a memory allocation error when I try to run my Simulink model?
http://www.mathworks.com/support/solutions/en/data/1-19HOC/.
2011

[Mie11] Mietke, Detlef.
Amplitudenmodulation aus mathematischer Sicht.
http://www.elektroniktutor.de/signale/am.html.
2011

[Mül05] Müller, Urs.
ISL 5216: Einstellungen des DDC für den ersten Prototypen.
Paul Scherrer Institut, 5232 Villigen PSI.
2005

[New11] NewWaveInstruments.
Linear Feedback Shift Registers.
http://www.newwaveinstruments.com/resources/articles/m_sequence_linear_feedback_shift_register_lfsr.htm.
2011

[PSI11a] PSI.
Paul Scherrer Institut Homepage.
www.psi.ch.
2011

[PSI11b] PSI.

Protonenbeschleuniger Benutzerlabor.
http://weblb1.psi.ch/forschung/benutzerlabor_protonen.shtml.
2011

[PSI11c] PSI.
Die Protonenbeschleunigeranlage des PSI.
http://www.psi.ch/media/protonenbeschleunigeranlagen.
2011

[TLL03] Tsoi, K.H. ; Leung, K.H. ; Leong, P.H.W.
Compact FPGA-based True and Pseudo Random Number Generators.
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.118.4430&rep=rep1&type=pdf.
April 2003

[Tru08] Truöl, Peter.
nat_ges_zh08.pdf.
http://www.physik.uzh.ch/~truoel/personal/nat_ges_zh08.pdf.
2008

[Wik11a] Wikidot.com.
Look-Up Tables.
http://tibasicdev.wikidot.com/lookuptables.
2011

[Wik11b] Wikipedia.
Cascaded-Integrator-Comb-Filter.
http://de.wikipedia.org/wiki/Cascaded-Integrator-Comb-Filter.
2011

[Wik11c] Wikipedia.
Formelsammlung Trigonometrie: Produkte der Winkelfunktionen.
http://de.wikipedia.org/wiki/Formelsammlung_Trigonometrie#Additionstheoreme.
2011

[Wik11d] Wikipedia.
I&Q-Verfahren.
http://de.wikipedia.org/wiki/I%26Q-Verfahren.
2011

[Wik11e] Wikipedia.
Linear Feedback Shift Registers.
http://en.wikipedia.org/wiki/Linear_feedback_shift_register.
2011

[Wik11f] Wikipedia.
Linear rückgekoppeltes Schieberegister.
http://de.wikipedia.org/wiki/Linear_rückgekoppeltes_Schieberegister.
2011

[Wik11g] Wikipedia.
Numerically-controlled oscillator: Mitigation techniques.
http://en.wikipedia.org/wiki/Numerically-controlled_oscillator.
2011

[Wik11h] Wikipedia.
Nyquist-Shannon-Abtasttheorem.
http://de.wikipedia.org/wiki/Nyquist-Shannon-Abtasttheorem.

2011

[Wik11i] Wikipedia.
Process gain.
http://en.wikipedia.org/wiki/Process_gain.
2011

[Wik11j] Wikipedia.
Spurious Free Dynamic Range.
http://de.wikipedia.org/wiki/Spurious_Free_Dynamic_Range.
2011

[Wik11k] Wikipedia.
Undersampling.
http://en.wikipedia.org/wiki/Undersampling.
2011

[Wik11l] Wiktionary.
Kavität.
http://de.wiktionary.org/wiki/Kavit%C3%A4t.
2011

[Xil11a] Xilinx.
Forumeintrag: DDC Core in the new version of ISE.
http://forums.xilinx.com/t5/Digital-Signal-Processing-IP-and/DDC-Core-in-the-new-version-of-ISE/td-p/8060.
2011

[Xil11b] Xilinx.
LogiCORE IP DUC/DDC Compiler v1.1.
http://www.xilinx.com/support/documentation/ip_documentation/duc_ddc_ds766.pdf.
2011

[Xil11c] Xilinx.
System Generator for DSP.
http://www.xilinx.com/tools/sysgen.htm.
2011

[Xil11d] Xilinx.
System Generator for DSP Getting Started Guide.
http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/sysgen_gs.pdf.
2011

[Xil11e] Xilinx.
Virtex-6 Family Overview.
http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf.
24 März 2011

Abbildungsverzeichnis

Tabellenverzeichnis

L MATLAB-Version und Toolboxen

MATLAB Version 7.11.0.584 (R2010b)
MATLAB License Number: 101072
Operating System: Microsoft Windows XP Version 5.1 (Build 2600: Service Pack 3)
Java VM Version: Java 1.6.0_17-b04 with Sun Microsystems Inc. Java HotSpot(TM) Client VM mixed mode

---

MATLAB	Version 7.11	(R2010b)
Simulink	Version 7.6	(R2010b)
Bioinformatics Toolbox	Version 3.6	(R2010b)
Control System Toolbox	Version 9.0	(R2010b)
Curve Fitting Toolbox	Version 3.0	(R2010b)
Data Acquisition Toolbox	Version 2.17	(R2010b)
Database Toolbox	Version 3.8	(R2010b)
EDA Simulator Link	Version 3.2	(R2010b)
Filter Design HDL Coder	Version 2.7	(R2010b)
Filter Design Toolbox	Version 4.7.1	(R2010b)
Fixed-Point Toolbox	Version 3.2	(R2010b)
Global Optimization Toolbox	Version 3.1	(R2010b)
Image Acquisition Toolbox	Version 4.0	(R2010b)
Image Processing Toolbox	Version 7.1	(R2010b)
Instrument Control Toolbox	Version 2.11	(R2010b)
MATLAB Builder JA	Version 2.2	(R2010b)
MATLAB Compiler	Version 4.14	(R2010b)
MATLAB Report Generator	Version 3.9	(R2010b)
Mapping Toolbox	Version 3.2	(R2010b)
Neural Network Toolbox	Version 7.0	(R2010b)
Optimization Toolbox	Version 5.1	(R2010b)
Parallel Computing Toolbox	Version 5.0	(R2010b)
Parallel Computing Toolbox	Version 5.0	(R2010b)
Partial Differential Equation Toolbox	Version 1.0.17	(R2010b)
RF Toolbox	Version 2.8	(R2010b)
Real-Time Workshop	Version 7.6	(R2010b)
Robust Control Toolbox	Version 3.5	(R2010b)

Signal Processing Blockset	Version 7.1	(R2010b)
Signal Processing Toolbox	Version 6.14	(R2010b)
SimHydraulics	Version 1.8	(R2010b)
SimPowerSystems	Version 5.3	(R2010b)
Simscape	Version 3.4	(R2010b)
Simulink 3D Animation	Version 5.2	(R2010b)
Simulink Control Design	Version 3.2	(R2010b)
Simulink Design Optimization	Version 1.2	(R2010b)
Simulink Fixed Point	Version 6.4	(R2010b)
Simulink HDL Coder	Version 2.0	(R2010b)
Simulink Verification and Validation	Version 3.0	(R2010b)
Spreadsheet Link EX	Version 3.1.2	(R2010b)
Stateflow	Version 7.6	(R2010b)
Stateflow Coder	Version 7.6	(R2010b)
Statistics Toolbox	Version 7.4	(R2010b)
Symbolic Math Toolbox	Version 5.5	(R2010b)
System Identification Toolbox	Version 7.4.1	(R2010b)
Wavelet Toolbox	Version 4.6	(R2010b)
Xilinx System Generator	Version 12.1	production build
xPC Target	Version 4.4	(R2010b)

M Fehlermeldungen in Matlab bei ausführen des DDC-Models

```
The model has 7 discrete rates, which is greater than the 6 discrete sample time colors
available. The 6th and slower rates have been marked using the orange sample time color.
Warning: Parameter overflow occurred for 'Threshold'. The parameter's value is outside the range that
the run-time data type can represent. The specified value was saturated to the closest representable
value. You can control this diagnostic on the Diagnostics pane of the Configuration Parameters dialog.
This originated from 'DDC_final/DigitalDownConverter/Cartesian2Polar/Controller/enabled␣counter/Switch'.
Warning: Parameter precision loss occurred for 'Table'. The parameter's value cannot be represented
exactly using the run-time data type. A small quantization error has occurred. You can control this
diagnostic on the Diagnostics pane of the Configuration Parameters dialog. This originated from
'DDC_final/DigitalDownConverter/Cartesian2Polar/CordicAlgorithm/ZComponent/Lookup␣Table'.
Warning: Parameter precision loss occurred for 'Gain'. The parameter's value cannot be represented
exactly using the run-time data type. A small quantization error has occurred. You can control this
diagnostic on the Diagnostics pane of the Configuration Parameters dialog. This originated from
'DDC_final/DigitalDownConverter/Cartesian2Polar/GainCorrection/Gain'.
Warning: Parameter precision loss occurred for 'a.h'. The parameter's value cannot be represented
exactly using the run-time data type. A small quantization error has occurred. You can control this
diagnostic on the Diagnostics pane of the Configuration Parameters dialog. This originated from
'DDC_final/DigitalDownConverter/FIR␣Decimation␣I1'.
Warning: Parameter precision loss occurred for 'a.h'. The parameter's value cannot be represented
exactly using the run-time data type. A small quantization error has occurred. You can control this
```

```
21 diagnostic on the Diagnostics pane of the Configuration Parameters dialog. This
       originated from
22 'DDC_final/DigitalDownConverter/FIR␣Decimation␣I2'.
23 Warning: Parameter precision loss occurred for 'a.h'. The parameter's value
       cannot be represented
24 exactly using the run-time data type. A small quantization error has occurred.
       You can control this
25 diagnostic on the Diagnostics pane of the Configuration Parameters dialog. This
       originated from
26 'DDC_final/DigitalDownConverter/FIR␣Decimation␣Q1'.
27 Warning: Parameter precision loss occurred for 'a.h'. The parameter's value
       cannot be represented
28 exactly using the run-time data type. A small quantization error has occurred.
       You can control this
29 diagnostic on the Diagnostics pane of the Configuration Parameters dialog. This
       originated from
30 'DDC_final/DigitalDownConverter/FIR␣Decimation␣Q2'.
31 Warning: Parameter precision loss occurred for 'Value'. The parameter's value
       cannot be represented
32 exactly using the run-time data type. A small quantization error has occurred.
       You can control this
33 diagnostic on the Diagnostics pane of the Configuration Parameters dialog. This
       originated from
34 'DDC_final/DigitalDownConverter/NCO/pi'.
35 Warning: Parameter precision loss occurred for 'Value'. The parameter's value
       cannot be represented
36 exactly using the run-time data type. A small quantization error has occurred.
       You can control this
37 diagnostic on the Diagnostics pane of the Configuration Parameters dialog. This
       originated from
38 'DDC_final/DigitalDownConverter/freq'.
39 Warning: Overflow occurred. This originated from 'DDC_final/DigitalDownConverter
       /NCO/Sum'.
40 Warning: Saturation occurred. This originated from
41 'DDC_final/DigitalDownConverter/Cartesian2Polar/CordicAlgorithm/ZComponent/
       Lookup␣Table'.
42 Warning: Saturation occurred. This originated from
43 'DDC_final/DigitalDownConverter/Cartesian2Polar/Controller/enabled␣counter/Sum'.
44 Warning: Overflow occurred. This originated from 'DDC_final/DigitalDownConverter
       /CIC
45 Decimation␣I'.
46 Warning: Overflow occurred. This originated from 'DDC_final/DigitalDownConverter
       /CIC
47 Decimation␣I'.
48 Warning: Overflow occurred. This originated from 'DDC_final/DigitalDownConverter
       /CIC
49 Decimation␣Q'.
```

```
50 Warning: Overflow occurred. This originated from 'DDC_final/DigitalDownConverter
       /CIC
51 Decimation␣I'.
52 Warning: Overflow occurred. This originated from 'DDC_final/DigitalDownConverter
       /CIC
53 Decimation␣Q'.
54 Warning: Overflow occurred. This originated from 'DDC_final/DigitalDownConverter
       /CIC
55 Decimation␣Q'.
56 Warning: Overflow occurred. This originated from 'DDC_final/DigitalDownConverter
       /CIC
57 Decimation␣I'.
58 Warning: Overflow occurred. This originated from 'DDC_final/DigitalDownConverter
       /CIC
59 Decimation␣Q'.
60 Warning: Overflow occurred. This originated from 'DDC_final/DigitalDownConverter
       /CIC
61 Decimation␣I'.
62 Warning: Overflow occurred. This originated from 'DDC_final/DigitalDownConverter
       /CIC
63 Decimation␣I'.
64 Warning: Overflow occurred. This originated from 'DDC_final/DigitalDownConverter
       /CIC
65 Decimation␣Q'.
66 Warning: Overflow occurred. This originated from 'DDC_final/DigitalDownConverter
       /CIC
67 Decimation␣Q'.
```

Erklärung der selbständigen Anfertigung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe.

Ort, Datum, Unterschrift