Masterarbeit

Daniel Sabotta

Design eines CMOS-Testchips für die digitale und analoge Signalverarbeitung in ABS-Sensoren

Fachhochschule Westküste
Fachbereich Technik

Fachhochschule Westküste - University of Applied Sciences
Faculty of Engineering

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

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

Daniel Sabotta

Design eines CMOS-Testchips für die digitale und analoge Signalverarbeitung in ABS-Sensoren

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

Betreuender Prüfer : Prof. Dr.-Ing. Karl-Ragmar Riemschneider
Zweitgutachter : Prof. Dr.-Ing. Stephan Hußmann

Abgegeben am 18. Januar 2013

Daniel Sabotta

Thema der Masterarbeit

Design eines CMOS-Testchips für die digitale und analoge Signalverarbeitung in ABS-Sensoren

Stichworte

CMOS-Chip, ABS-Sensoren, Digitale Signalverarbeitung, Cadence

Kurzzusammenfassung

Die vorliegende Arbeit befasst sich mit der Implementierung von Teilsystemen für die Zustandserkennung in ABS-Sensoren auf einem CMOS-Chip. Zu diesen Teilsystemen gehört ein Protokollgenerator für die Kommunikation des ABS-Sensors mit dem ABS-Steuergerät über das AK-Protokoll, ein Algorithmus für die Zustandserkennung von ABS-Sensoren und Teile der analogen Signalvorverarbeitung.

Daniel Sabotta

Title of the paper

Design of a CMOS test chip for the digital and analog signal processing in ABS sensors

Keywords

CMOS chip, ABS sensors, Digital Signalprocessing, Cadence

Abstract

This master's thesis deals with the implementation of subsystems, used for the detection of the ABS sensor state, on a CMOS chip. These subsystems are a protocol generator for the communication between the ABS sensor and the ABS control unit via the AK protocol, an algorithm for the detection of the ABS sensor state and parts of the analog signal preprocessing.

Danksagung

An dieser Stelle möchte ich mich bei meinem betreuenden Prüfer Prof. Dr.-Ing. Karl-Ragmar Riemschneider, der es mir ermöglicht hat, diese Masterarbeit zu erstellen, bedanken. Weiterhin bedanke ich mich bei meinem Zweitgutachter Prof. Dr.-Ing. Stephan Hußmann.

Besonderer Dank gilt Herrn Dipl.-Ing. (FH) Martin Krey für seinen fachlichen Rat und seine tatkräftige Unterstützung. Außerdem bedanke ich mich bei dem gesamten Forschungsteam der ESZ-ABS und BATSEN Projekte für das angenehme und kollegiale Arbeitsklima.

Weiterer Dank geht an Gerhard Dörflein und Hermann Meier von NXP für ihre technische Unterstützung.

Vor allem aber möchte ich meiner Familie danken, ohne deren Unterstützung ich mein Studium niemals hätte absolvieren können.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 ESZ-ABS

Die meisten heutigen Pkws sind mit einem Antiblockiersystem (ABS) ausgestattet, was dem Blockieren der Räder bei starkem Bremsen entgegenwirkt. Für die Funktion des ABS muss die Raddrehzahl ermittelt werden. An der Radnarbe ist hierfür ein Encoderrad montiert. Dieses Encoderrad moduliert ein Magnetfeld, welches von einem Sensor aufgenommen wird [22].

Im Rahmen des Forschungsprojekts Experimentelle digitale Signalverarbeitung und Zustandserkennung für ABS-Sensoren (ESZ-ABS) soll die Zuverlässigkeit der ABS-Sensoren erhöht werden.

Moderne ABS-Sensoren basieren auf dem Hall-Effekt oder dem anisotropen magnetoresistiven Effekt (AMR) und liefern ein digitales Ausgangssignal, das von dem ABS-Steuergerät ausgewertet werden kann. Dies bedeutet, dass sie mikroelektronische Schaltungen beinhalten.

Der AMR-Effekt beschreibt die Änderung eines Widerstandes in Abhängigkeit eines äußeren Magnetfeldes. Ist das Magnetfeld in oder gegen die Stromrichtung ausgerichtet, ist der Widerstand am größten. Liegt das Magnetfeld hingegen senkrecht zur Stromrichtung an, ist der elektrische Widerstand am kleinsten. Siehe [21].

In ABS-Sensoren wird dieser AMR-Effekt ausgenutzt, indem vier Widerstände zu einer Wheatstonesche Messbrücke (siehe Abbildung 1.1) verschaltet werden. Dabei wird die Differenzspannung U_{diff} ausgewertet. Bewegen sich die magnetischen Pole des Encoderrades an der Messbrücke vorbei, so sollte im Idealfall ein sinusförmiges Signal an der Differenzspannung zu messen sein (siehe Abbildung 1.2).

In Abbildung 1.3 ist ein ABS-Sensor zu sehen. Er enthält die AMR-Messbrücke, eine Mixed-Signal ASIC[1] und einen Line-Driver- und Supply-ASIC. Der Line-Driver- und Supply-ASIC vrersorgt die AMR-Messbrücke und den Mixed-Signal-ASIC mit Spannung und treibt den Strom für das digitale Protokoll zum ABS-Steuergerät. Die AMR-Messbrücke liefert über zwei Anschlüsse dem Mixed-Signal-ASIC die Brückenspannung. Der Mixed-Signal-ASIC wertet die Brückenspannung aus und generiert das Protokoll.

[1] **ASIC** Anwendungsspezifische integrierte Schaltung (Application Specific Integrated Circuit)

Das Protokoll, mit dem Daten von dem ABS-Sensor an das ABS-Steuergerät gesendet werden, ist in [14] als „AK-Protocol“ definiert, dabei steht AK für Arbeitskreis. In Abbildung 1.4 ist ein Beispiel dieses AK-Protokolls zu sehen. Dabei wird bei jedem Nulldurchgang des Sensorsignals ein Speedpulse gesendet (28 mA). Zwischen den Speedpulsen werden Daten im Manchester-Code gesendet, die unter anderem die Drehrichtung des Rades angeben (14 mA).

Durch Veränderung der Position verzerrt sich das Ausgangssignal der Messbrücke. Die Auswertung der gesamten harmonischen Verzerrung (THD) hat gezeigt, dass mit ihr eine Aussage über die Qualität des Signals getroffen werden kann [24]. Das AK-Protokoll beinhaltet Bits, dessen Verwendung noch nicht definiert sind. Diese noch unbelegten Bits könnten dafür genutzt werden dem ABS-Steuergerät die Information über die Signalqualität mitzuteilen.

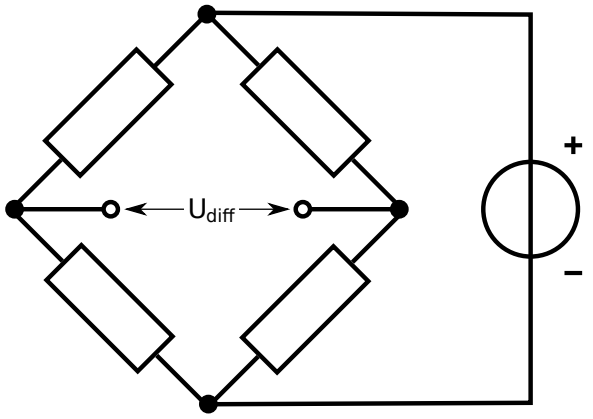


Abbildung 1.1: Wheatstonesche Messbrücke.

1.1.1 Stand des Projekts

Es stehen dem Forschungsprojekt ESZ-ABS eine Reihe von Messeinrichtungen zur Verfügung, mit denen sowohl das Verhalten der Sensoren in Magnetfeldern [23], als auch an realen Encoderrädern [23, 38] analysiert werden kann. Des Weiteren wurden, auf Basis von Mikrokontrollern, Testplatinen entwickelt, die die Grundfunktionen des Sensors umsetzen.

Um einen Schritt weiter in Richtung Mikroelektronik zu gehen, wurden die digitalen Grundfunktionen (darunter die Berechnung des THD und die Logik des Busprotokolls, mit dem der ABS-Sensor mit dem ABS-Steuergerät kommuniziert) in VHDL umgesetzt und auf einem Field Programmable Gate Array (FPGA) implementiert [17, 16]. Auch analoge Signal-

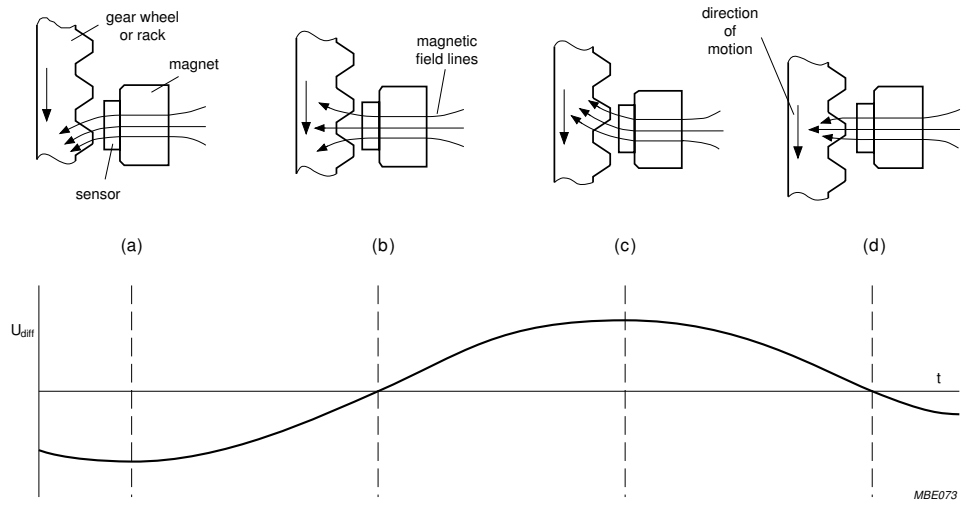


Abbildung 1.2: Differenzspannung der Messbrücke in Abhängigkeit der Encoderposition. – Im Idealfall ist das Ausgangssignal der AMR-Messbrücke eine sinusförmige Spannung. Aus [33].

verarbeitung ist für die Auswertung des Differenzsignals notwendig. Dafür wurde zuletzt im Rahmen einer Abschlussarbeit eine Verstärkerbank entwickelt [32].

Der grundsätzliche Signalfluss des ABS-Sensors ist in Abbildung 1.5 gezeigt: Das von der AMR-Messbrücke kommende, durch das Encoderrad angeregte, Sensorsignal wird zunächst analog verstärkt und der enthaltene Offset kompensiert. Nach der Quantisierung des Signals können die enthaltenen Harmonischen bestimmt werden. Aus den Harmonischen kann schließlich der THD des Sensorsignals berechnet werden. Aus den erzeugten Daten kann der Protokollgenerator das über den Bus gehende Signal erzeugen, was von dem Line-Driver umgesetzt wird.

Die analoge Verstärkung wurde mit der Offsetkompensation im Rahmen einer Abschlussarbeit auf einer Leiterplatte aufgebaut [32]. Die Bestimmung der Harmonischen und Berechnung des THD wurde in einer weiteren Abschlussarbeit auf einem FPGA implementiert [17]. Der THD berechnet sich nach der Formel aus Gleichung 1.1. Da für diese Berechnung unendlich viele Harmonische notwendig wären, wurde sie auf die Berechnung des THD mit den ersten fünf Harmonischen (HD5) beschränkt (siehe Gleichung 1.2).

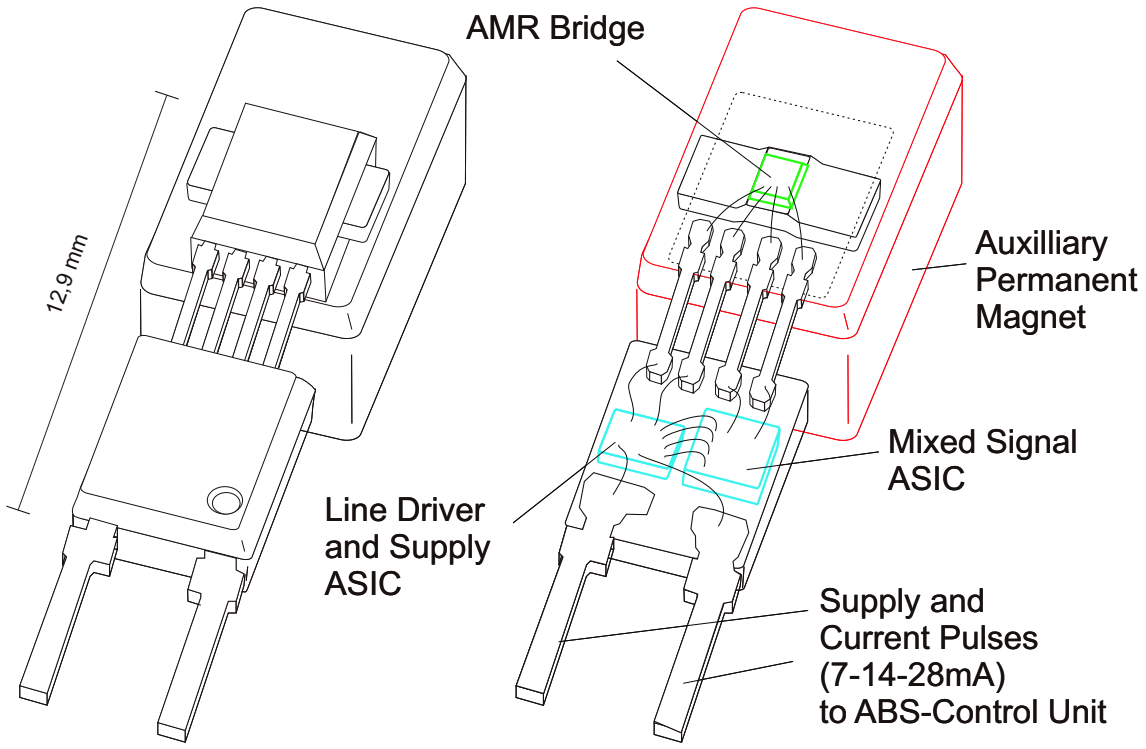


Abbildung 1.3: Aufbau eines ABS-Sensors mit AMR-Messbrücke. – Die AMR-Messbrüke liefert dem Mixed Signal ASIC die Brückenspannung. Dieser wertet die Brückenspannung aus und erzeugt das Ausgabeprotokoll. Das Ausgabeprotokoll wird von dem Line Driver and Supply ASIC über den Bus gesendet. Des Weiteren versorgt der Line Driver and Supply ASIC die Messbrücke und den Mixed Signal ASIC mit der Versorgungsspannung. Abbildung modifiziert aus [26].

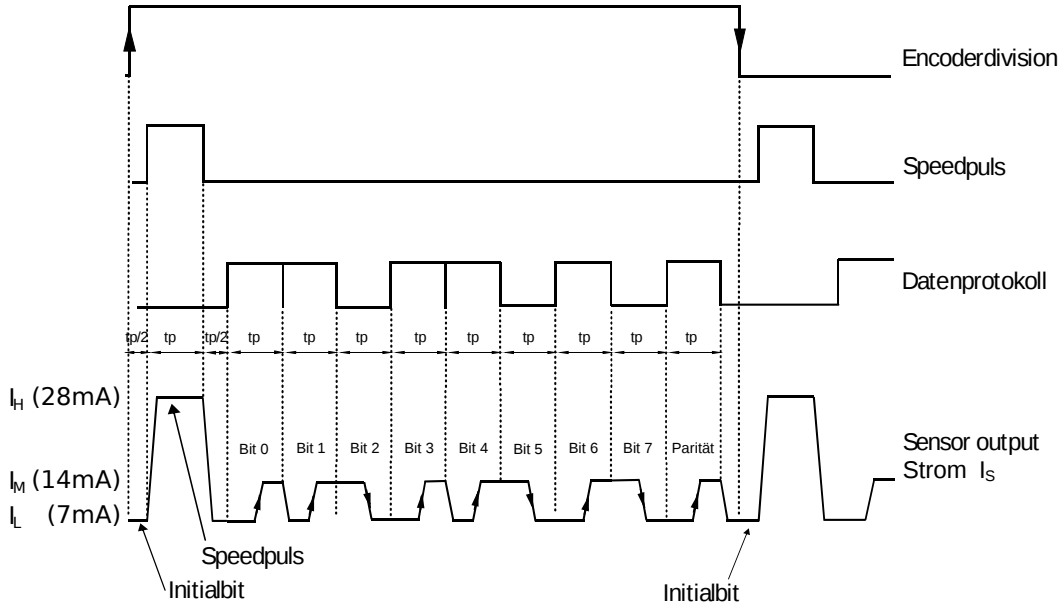


Abbildung 1.4: AK-Protokoll des ABS-Sensors. – Encoderdivision zeigt die Nulldurchgänge des Sensorsignals. Der Speedpulse wird bei diesen Nulldurchgängen ausgegeben. Dem Speedpulse folgen bis zu acht Daten- und ein Paritätsbit. Unten ist der Ausgangsstrom des Sensors I_S zu sehen. Dabei entspricht der Speedpulse einem Ausgangssignal von 28 mA und die Datenbits 14 mA. Aus [14].

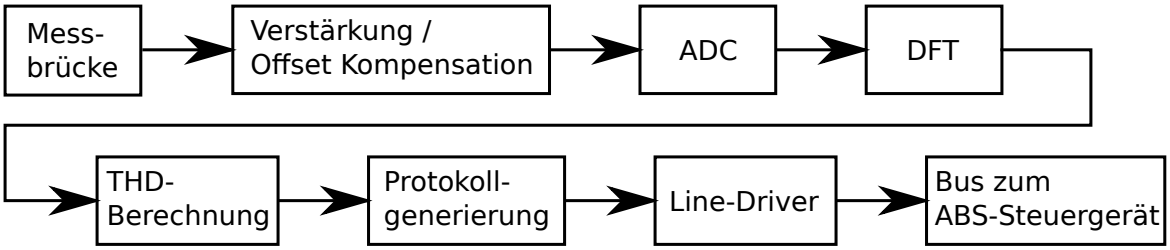


Abbildung 1.5: Signalflussdiagramm des ABS-Sensors. – Das von der Messbrücke aufgenommene Sensorsignal wird zunächst analog verstärkt und der in dem Signal enthaltene Offset wird kompensiert. Nach der Quantisierung des Sensor Signals, mittels eines Analog-Digital-Umsetzers, werden die Harmonischen des Sensorsignals durch eine DFT bestimmt. Aus den Harmonischen kann der THD des Sensorsignals berechnet werden. Aus den somit gesammelten Daten kann das Protokoll erzeugt und von dem Line-Driver ausgegeben werden.

$$\mathrm{THD} = \sqrt{\frac{\sum_{n=2}^{\infty} H_n^2}{\sum_{n=1}^{\infty} H_n^2}} \tag{1.1}$$

$$\mathrm{HD5} = \sqrt{\frac{H_2^2 + H_3^2 + H_4^2 + H_5^2}{H_1^2 + H_2^2 + H_3^2 + H_4^2 + H_5^2}} \tag{1.2}$$

1.1.2 Ziel dieser Arbeit

Um noch einen Schritt weiter in Richtung Mikroelektronik zu gehen wurde im Rahmen dieser Abschlussarbeit ein CMOS-Testchip[2] gefertigt, der Teile der digitalen und analogen Hardware enthält. Dabei soll die Berechnung des THD und in erster Linie die Logik des Ausgabeprotokolls umgesetzt werden. Des Weiteren sollen einige Komponenten der analogen Signalverarbeitung in dem CMOS-Testchip implementiert werden.

Ziel ist es dabei, eine erste Abschätzung der Realisierbarkeit der Signalverarbeitung auf dem CMOS-Chip zu geben. Dabei wird der Flächenbedarf und die Stromaufnahme berücksichtigt. Da der, im Rahmen dieser Arbeit, entworfene CMOS-Chip in einem nicht aktuellem Prozess gefertigt wurde, muss später eine Abschätzung auf modernere Prozesse erfolgen.

1.2 CMOS-Chipdesign

1.2.1 Grundlagen

In einem CMOS Halbleiterprozess werden p- und n-Kanal-MOSFETs[3] auf gemeinsamen Substrat verwendet. Dabei dienen die p-Kanal-MOSFETs als „Pullup“ und die n-Kanal-MOSFETs als „Pulldown“-Pfad. Ein Pfad ist, abhängig von den Eingängen, immer durchgeschaltet. In Abbildung 1.6 ist als Beispiel ein NOR-Gatter in der CMOS-Technologie zu sehen. Siehe [20, 10].

[2] **CMOS** Complementary Metal Oxide Semiconductor
[3] **MOSFET** Metall-Oxid-Halbleiter-Feldeffekttransistor

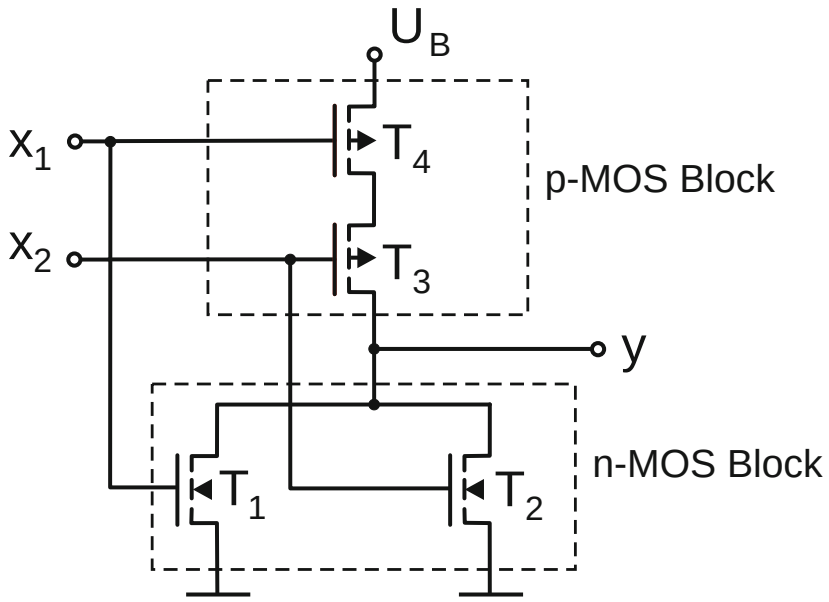


Abbildung 1.6: Grundsätzlicher Aufbau eines komplexen CMOS-Gatters, bestehend aus je einem Block mit n-Kanal MOSFET („Pulldown“) und einem Block mit p-Kanal MOSFET („Pullup“), am Beispiel des NOR-Gatters. Aus [20].

1.2.2 Standardzellen

Es ist möglich CMOS-Chips vollkommen spezifisch, durch das Entwerfen jeder geometrischen Struktur, zu entwickeln (Full-Custom-Design). Da digitale Schaltwerke auf die Beschreibung mittels logischer Verknüpfungen zurückzuführen sind, ist es sinnvoll, auf eine Bibliothek mit logischen Gattern, den Standardzellen, zurückzugreifen [39]. Hierbei handelt es sich um eine vom Prozess und Hersteller abhängige Datenbank, die Logikgatter und Speicherelemente (z. B. Flipflops) enthält. So ist es nicht notwendig jedes Element mit Transistoren aufzubauen, sondern es kann die Logik mit einer Hardwarebeschreibungssprache, wie z. B. VHDL[4] beschrieben und anschließend synthetisiert werden.

Die Standardzellen, der im Rahmen dieser Arbeit eingesetzten Bibliothek, besitzen alle eine identische Höhe. So können sie zeilenweise aneinander gelegt werden (siehe Abbildung 1.7). Wenn jede zweite Zeile um 180 ° gedreht wird, können sich immer zwei Zeilen eine Leitung der Versorgungsspannung teilen. Mit diesem Verfahren ist es möglich, die Logikgatter platzsparend aneinander zu legen.

Die einzelnen Logikgatter werden auf dem Chip über Leitungen auf den Metalllagen verbunden. Damit für die Leitungsführung genügend Platz vorhanden ist, können zwischen den

[4] **VHDL** Very High Speed Integrated Circuit Hardware Description Language

notwendigen Gattern leere Zellen eingefügt werden, über die Leitungen gelegt werden können. Ebenso ist es möglich, Reihen zwischen den Standardzellen für die Leitungsführung frei zu lassen.

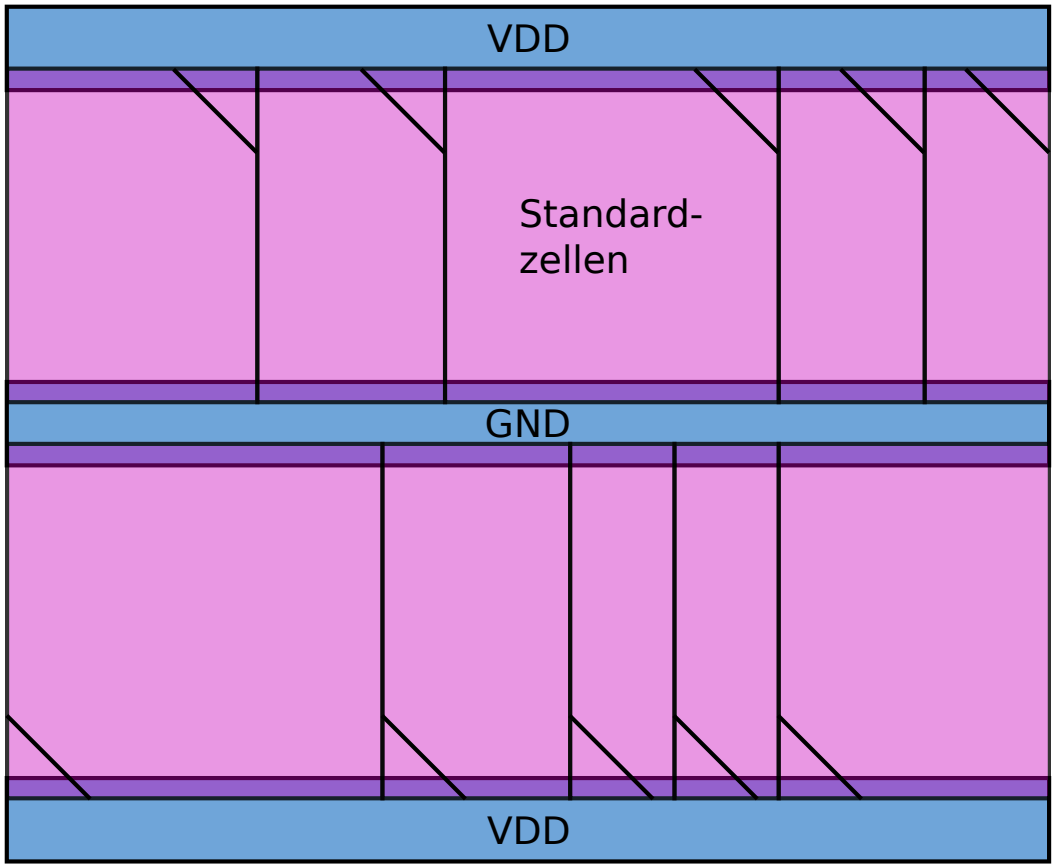


Abbildung 1.7: Standardzellen in Reihe. – Jede zweite Reihe von Standardzellen ist um 180 ° gedreht. So können sich immer zwei Reihen einen Anschluss der Versorgungsspannung teilen.

2 Analyse

2.1 Workflow

In diesem Abschnitt wird kurz der Workflow des Entwurfs eines Mixed-Signal-Chips erläutert, wie er im Laufe der vorliegenden Arbeit erarbeitet wurde. Es hat sich herausgestellt, dass es sinnvoll ist, die digitalen und analogen Module unabhängig voneinander zu entwerfen und anschließend zu einem Design zusammen zu führen.

2.1.1 Entwurf digitaler Logik

Der hier beschriebene Workflow beginnt in dem Register-Transfer Level (RTL) mit einer bereits entwickelten und auf einem FPGA implementierten Logik. Er endet nach dem Fertigstellen des Layouts und des Schaltplans. Der Workflow der digitalen Logik ist in Abbildung 2.1 dargestellt. Er orientiert sich stark an dem in [9], [11] und [39] beschriebenen Vorgehen.

RTL-Ebene

Ausgangspunkt dieses Workflows ist eine in einer Hardwarebeschreibungssprache (HDL), wie VHDL oder Verilog, beschriebene Logik im Register-Transfer Level. Diese Logik kann mit einer Testumgebung (Testbench), die ebenfalls mit einer HDL beschrieben ist, im Register-Transfer Level simuliert werden. Bei dieser Simulationgeht es um die Logik. Die spätere Platzierung oder die Eigenschaften des verwendeten Prozesses und die der Standardbibliotheken haben hier keinen Einfluss auf die Simulation. Die Simulation kann mit beliebigen HDL-Simulatoren (wie z. B. Modelsim oder GHDL) durchgeführt werden.

Gate Level-Ebene

Um in die Darstellungsebene des Gate Levels zu gelangen, muss der HDL-Code aus dem Register-Transfer Level mit einem geeignetem Synthesewerkzeug synthetisiert werden. Die Ausgabe des Synthesewerkzeuges ist eine Netzliste, in der die einzelnen Elemente der Standardbibliothek so mit einander verbunden sind, wie es das Synthesewerkzeug aus dem HDL-Code interpretiert. An dieser Stelle nehmen die Eigenschaften des Prozesses und der Standardbibliotheken in Form von Gatterlaufzeiten Einzug in das System. Die Netzliste kann

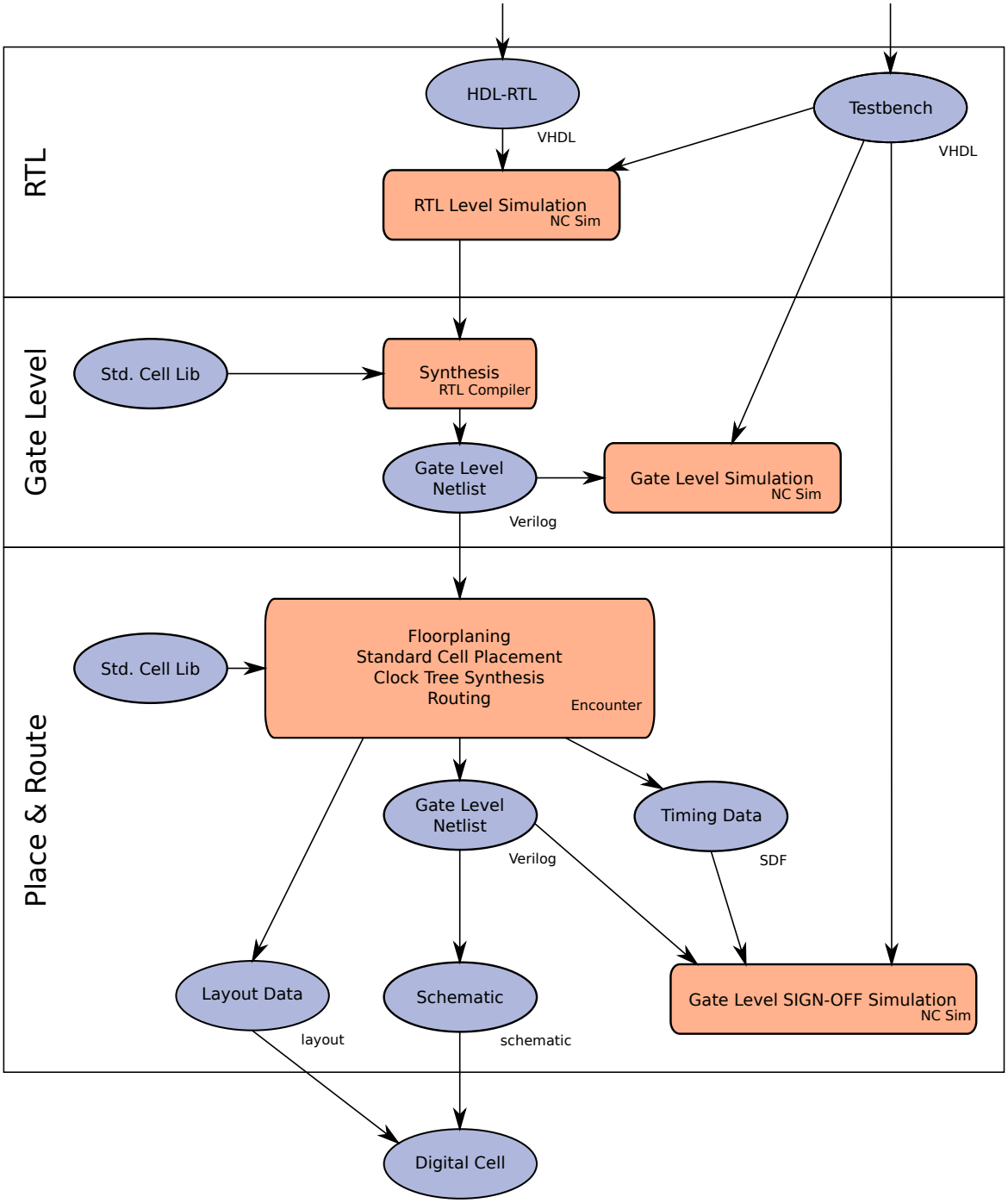


Abbildung 2.1: Workflow der Implementierung digitaler Schaltungen auf CMOS-Chips. – Die Veranschaulichung beginnt mit der Beschreibung der Hardware im Register-Transfer Level und endet bei der digitalen Zelle, bestehend aus dem Layout und der Schematic. Modifiziert aus [9].

wieder mit der Testbench simuliert werden. Dabei sind die Gatterlaufzeiten in SDF[1]-Dateien gespeichert, die in die Simulation eingebunden werden müssen.

Place & Route-Ebene

Mithilfe eines Place & Route-Tools kann aus der Netzliste ein Layout erstellt werden. Hierbei muss zunächst ein Floorplan erstellt werden. Der Floorplan legt grob die Platzierung des Cores, also die Digitalzellen aus der Standardbibliothek und dessen Ring für die Spannungsversorgung fest.

Die Standardzellen werden dann in Reihen platziert. Dabei ist darauf zu achten, dass genügend Platz zwischen den Zellen bzw. den Reihen vorhanden ist, um eine Verdrahtung zu ermöglichen.

In diesem Schritt muss auch ein Netzwerk zum Verteilen des Taktes generiert werden. Dies ist notwendig, da der Takt eine große Anzahl an Eingängen zu bedienen hat, was ein einziger Treiber meist nicht alleine bewerkstelligen kann. Dabei muss darauf geachtet werden, dass der Takt an allen Takteingängen möglichst zeitgleich ankommt. In Abbildung 2.2 ist ein einfaches Beispiel eines Taktbaumes zu sehen.

Des Weiteren werden in diesem Schritt die einzelnen digitalen Zellen miteinander verbunden. Ist der anfangs gewählte Floorplan zu klein, können nicht alle Verbindungen verlegt werden, ohne gegen die vom Prozess und Hersteller definierten Entwurfs-Regeln zu verstoßen. Wurde der Floorplan zu groß gewählt, bleibt Chip-Fläche ungenutzt.

Aus dem erstellten Layout muss nun eine neue Netzliste erzeugt werden, da unter anderem durch den Taktbaum, neue Zellen in die Logik eingefügt wurden. Außerdem muss jetzt eine neue SDF-Datei erzeugt werden, die zu den Gatterlaufzeiten auch die parasitären Effekte der Leitungsführung berücksichtigt. Mit diesen Dateien und der Testbench kann die Simulation auf der Place & Route Ebene durchgeführt werden.

2.1.2 Entwurf analoger Schaltungen

Der Einstieg in das Analogdesign beginnt mit dem Erstellen einer Schematic (siehe Abbildung 2.3). In der Schematic werden die analogen Komponenten aus der Standardbibliothek miteinander verbunden. Gleichzeitig kann eine Testschematic erstellt werden, in der die zu erstellende Schaltung zur Simulation eingebettet werden kann.

[1] **SDF** Standard Delay Format – IEEE-Standard für Timing-Daten. (`http://www.eda.org/sdf/`)

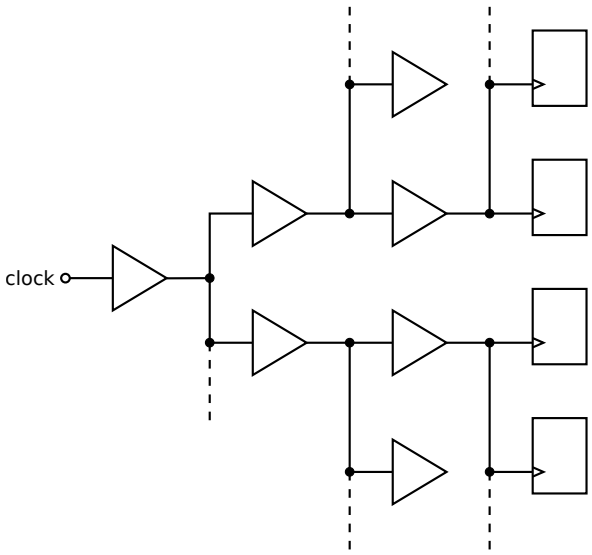


Abbildung 2.2: Skizze eines Taktbaumes. – Die einzelnen Treiber werden nur mit einer festgelegten Anzahl von Eingängen belastet. So verteilt sich der Takt nach jedem Treiber, wie die Äste eines Baumes.

Aus der Schematic der Schaltung kann das Layout erstellt werden. Mit dem Layout ziehen dann durch die Leitungsführung weitere parasitäre Effekte in die Schaltung ein. Nachdem die parasitären Effekte extrahiert wurden, kann eine weitere Simulation mit der Testschematic, die Postlayout-Simulation, durchgeführt werden.

Das Layout und die Schematic bilden zusammen die entwickelte analoge Zelle.

2.1.3 Zusammenführung der analogen und digitalen Module

Aus den erzeugten Digital- und Analog- Zellen kann die gesamte Schematic des Chips erstellt werden. An dieser Stelle werden auch Pads in die Schematic eingefügt, an denen später die Pins des Gehäuses über Drähte angeschlossen werden (Bonding). Aus dieser Schematic kann das Layout des Chips erstellt werden. Aus dem Layout können die parasitären Effekte des Chips extrahiert werden (Extracted View).

Mit der Extracted View kann das Verhalten des Chips, mit allen extrahierten parasitären Effekten, simuliert werden. Aus Zeitgründen wurde in dieser Arbeit auf diese Simulation verzichtet.

Die Layout-Daten müssen einen Design Rule Check (DRC) durchlaufen, um Fehler im Layout auszuschließen. Die Entwurfsregeln sind dabei vom Hersteller abhängig. Um sicherzugehen, dass das Layout auch mit dem Schaltplan übereinstimmt, muss ein Layout versus

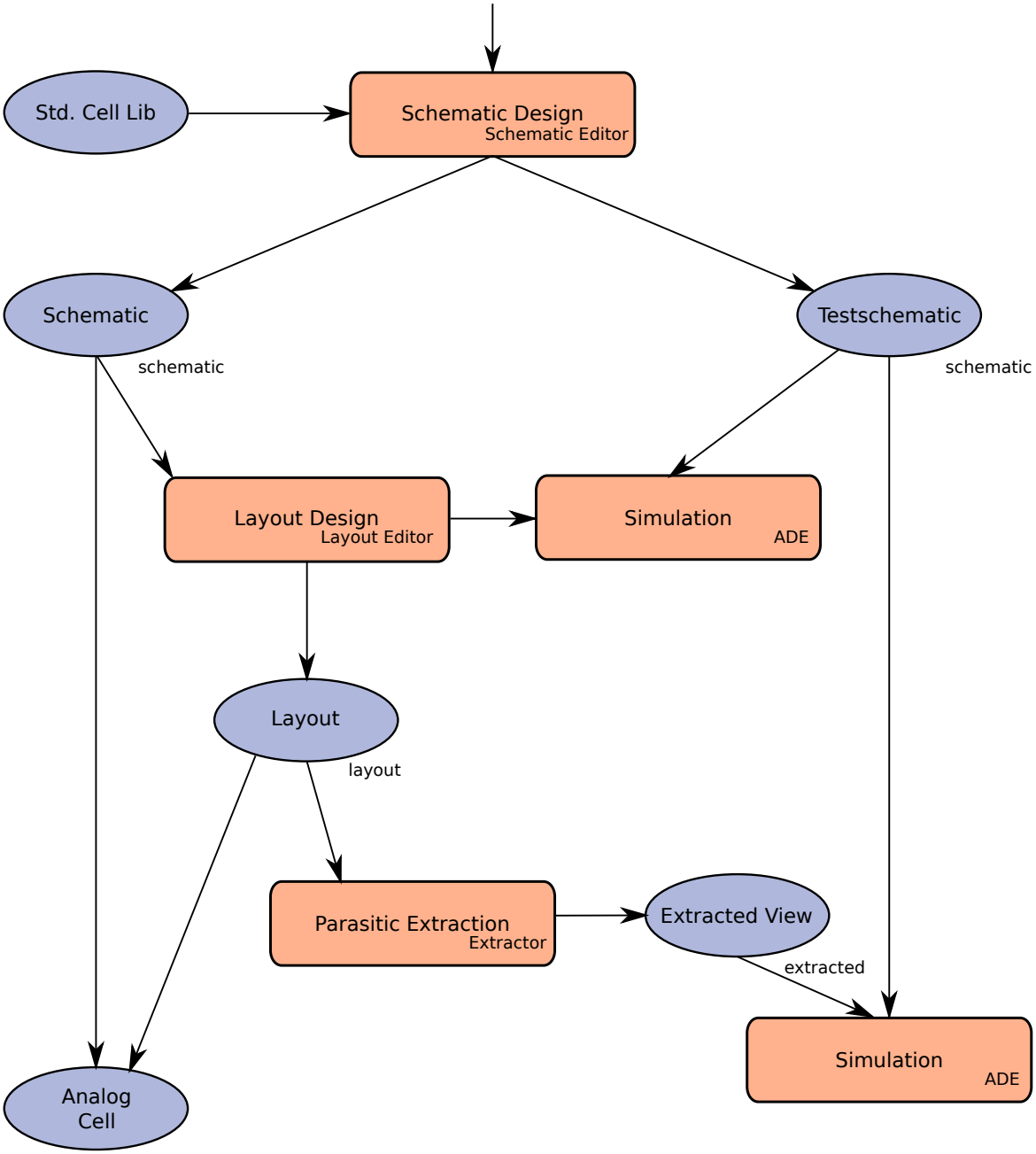


Abbildung 2.3: Workflow der Implementierung analoger Schaltungen auf CMOS-Chips. – Der Workflow der analogen Schaltungen beginnt mit dem Erstellen der Schematic und einer Testschematic, die den Funktionsnachweis ermöglicht. Er endet mit dem Erstellen der analogen Zelle, die die Schematic und das Layout enthält.

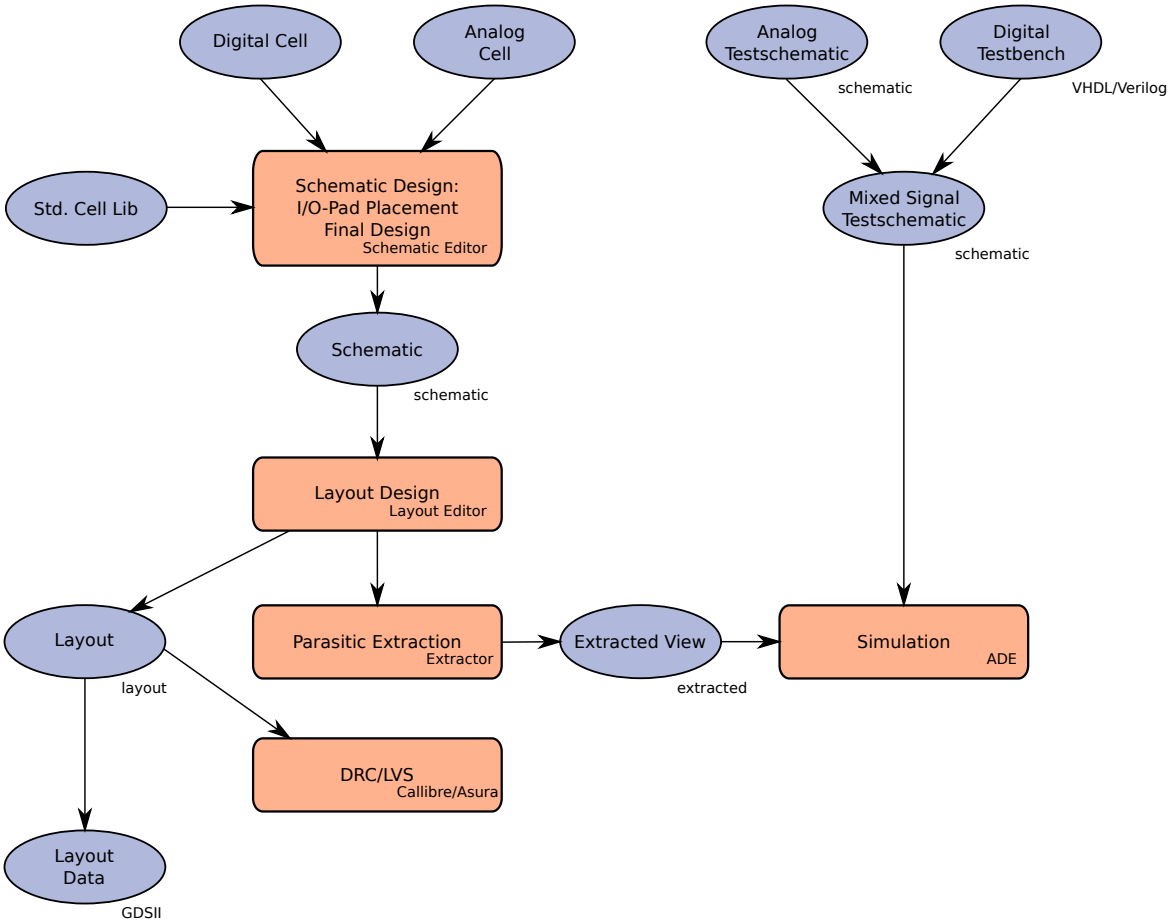


Abbildung 2.4: Workflow des Zusammenführens analoger und digitaler Schaltungen. – Der Workflow dieser Abbildung beginnt mit den digitalen und analogen Zellen (siehe Abbildungen 2.1 und 2.3) und endet mit dem Erstellen einer GDSII-Datei.

Schematic (LVS) Test durchgeführt werden. Hierbei wird die Leitungsführung im Layout, sowie das Vorhandensein der einzelnen Komponenten mit der Schematic verglichen. Ist das Chiplayout frei von DRC und LVS Fehlern kann eine GDSII Datei erzeugt werden, welche die notwendigen Layout-Daten für die Herstellung enthält.

2.2 Cadence Toolchain

Die Cadence Toolchain bietet eine Fülle von Programmen zum Erstellen und Simulieren von integrierten Schaltkreisen. Der CMOS-Testchip, der im Rahmen dieser Arbeit erstellt wurde, wurde mit Cadence IC6.1.4 und dem HIT-Kit v4.00 von Austriamicrosystems entworfen. Dabei wurde nur ein Bruchteil, der von Cadence zur Verfügung gestellten Programme, benötigt.

Die Toolchain von Cadence/Austriamicrosystems bietet meist mehrere Programme für eine Aufgabe, hier werden nur kurz die Aufgaben einer Auswahl von Programmen aufgeführt, die im Rahmen dieser Arbeit verwendet wurden. Die Toolchain richtet sich dabei stark nach dem Workflow aus Abschnitt 2.1.

Das Hauptprogramm der Cadence Toolchain nennt sich Virtuoso. Hieraus werden der Layout Editor, der Schematic Editor, der Library Manager und weitere Programme gestartet. Es existieren in dieser Toolchain aber auch Programme, die direkt (ohne Virtuoso) gestartet werden können.

Die Cadence Toolchain enthält einen HDL-Simulator, namens NC Sim. Mit diesem Simulator ist es möglich sowohl Verilog als auch VHDL zu simulieren. Das Programm greift beim Kompilieren und Elaborieren auf eine Vielzahl weiterer Programme zurück.

Um aus dem HDL-Code eine Netzliste mit Standardzellen zu generieren, wird der RTL Compiler genutzt. Aus der daraus gewonnenen Netzliste kann mit dem Programm Encounter RTL Compiler ein Layout gefertigt werden. Das Programm Encounter RTL Compiler beherrscht ebenso das Erstellen des Floorplans und Taktbaumes, wie auch das Routen und die Zellplatzierung. Beide Programme lassen sich aus einem Terminal starten und bieten die Möglichkeit, komplett per Tcl-Skript[2] bedient zu werden, was die Reproduzierbarkeit deutlich erhöht und es ermöglicht, kleine Änderungen am Design leicht vorzunehmen.

[2]Tcl ist eine Open Source-Skriptsprache.

2.3 Testmöglichkeiten

Bei dem Enwurf von Logikschaltungen mit programmierbaren Logikbauelementen, wie FPGAs oder CPLDs[3], kann die entwickelte Logik auf dem Chip getestet und, falls notwendig, modifiziert werden. So können Fehler in der Logik nachträglich behoben werden. Des Weiteren gibt es für die Entwicklung auf programmierbaren Logikbauelementen die Möglichkeit, interne Zustände über z. B. die JTAG-Schnittstelle auszulesen, was die Fehlersuche erleichtert. Digitale Logik auf CMOS-Chips haben diese Möglichkeiten nicht. Wurde der Chip gefertigt, ist es nicht möglich die Beschaltung nachträglich zu verändern. In diesem Abschnitt werden einige Möglichkeiten beschrieben, die Logik auf CMOS-Chips zu testen.

Mit einer feinen Messspitze wäre es möglich Signale direkt auf dem Chip zu messen. Dabei können die Nadeln nur auf der obersten Metalllage aufgesetzt werden, was beim Routen zu berücksichtigen ist. Da die Strukturen auf CMOS-Chips zu klein sind um die Messspitzen manuell zu platzieren, kann diese Variante in dem ESZ-ABS-Projekt nicht angewendet werden.

Eine Alternative zum direkten Messen auf dem Chip ist das Herausführen von Signalen über Pins (Bond Out). Dabei ist zu beachten, dass die Logik dafür verändert werden muss. Das kann dazu führen, dass sich das Zeitverhalten der Logik verändert. Ein Nachteil dieses Verfahrens ist, dass jedes zu messende Signal einen Pin am Gehäuse benötigt.

Signale, die für den Entwurf besonders entscheidend sind, können aufgetrennt und aus dem Chip geführt werden (siehe Abbildung 2.5). Das Signal kann so überprüft und falls es in dem Chip nicht richtig erzeugt wird, durch eine externe Quelle generiert werden. Allerdings sind für dieses Verfahren, für jedes Signal, zwei Pins notwendig und sollte deshalb nur für kritische Signale verwendet werden. Durch Separation von einzelnen Modulen durch diese Methode, ist es möglich die Module autark zu betreiben.

Kombinatorische Logik kann getestet werden, indem man einen Testvektor an die Eingänge legt und die Ausgänge beobachtet. Bei einer kombinatorischen Logik mit n Eingängen sind maximal 2^n Testvektoren notwendig. Da bei sequenzieller Logik die Ausgänge nicht nur von dem Eingangsvektor, sondern auch von den gespeicherten Werten abhängt, sind deutlich mehr Testvektoren notwendig. Des Weiteren müssen sich die Register in einem definierten Anfangszustand befinden, um eine Aussage über die Korrektheit der Ergebnisse treffen zu können. Um fertigungsbedingte Fehler in den Flipflops zu erkennen, kann auf einen Test mittels einer Scan-Chain zurückgegriffen werden. Hierbei werden die D-Flipflops mit zwei zusätzlichen Eingängen ausgestattet (siehe Abbildung 2.6). Alle zu testenden Flipflops werden über die zusätzlichen Eingänge (in Abbildung 2.6 rechts: Q_{i-1}) zu einem Schieberegister

[3] **CPLD** Complex Programmable Logic Device

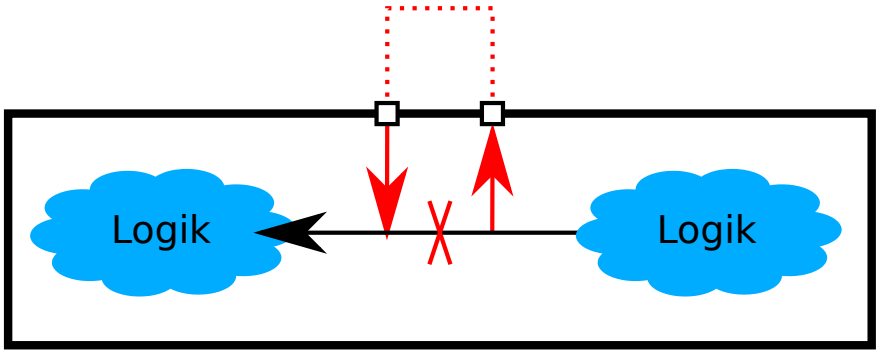


Abbildung 2.5: Funktionsweise der Testpins. – Die beiden Logikblöcke (blaue Wolken) kommunizieren in dieser Abbildung miteinander (schwarzer Pfeil). Durchtrennt man diese Leitung und führt sie aus dem Chip, können diese vor dem Wiedereinspeisen überprüft werden (rote Linien).

zusammengeschaltet. Wird an den Anfang des Schieberegisters eine definierte Bitreihenfolge gelegt, wird diese durch alle Flipflops hindurch getaktet und kommt mit einer Verzögerung am Ende des Schieberegisters wieder heraus. Mit dieser Methode kann man zumindest Flipflops kontrollieren. Die 0.35 µm CMOS Libraries (C35) von Austriamicrosystems bietet für die Scan-Chain Flipflops mit einem zweiten gemultiplexten Eingang an (z. B. DFS1 und DFS3). Mithilfe einer Scan-Chain können die Register auf definierte Werte gesetzt werden, um das Testen sequenzieller Logik zu vereinfachen. Bei diesem Verfahren ist zu beachten, dass durch das Hinzufügen der Multiplexer der kritische Pfad verlängert wird. Eine Scan-Chain wurde im Rahmen dieser Arbeit nicht implementiert.

2.4 Modularisierung

Die Funktionen des CMOS-Testchips wurden auf höchster Entwurfsebene in drei Module unterteilt. Die Unterteilung beruht auf dem Verwendungszweck und auf dem Grad der Entwicklung.

In Abbildung 2.7 ist die Aufteilung der Module grob dargestellt. Die analoge Signalverarbeitung ist auf digitale Steuersignale angewiesen und liefert auch digitale Ausgangswerte. Deshalb muss dieses Modul mit der digitalen Signalverarbeitung kommunizieren können. Das Modul für die Protokollausgabe ist in der Entwicklung am weitesten fortgeschritten und wurde so implementiert, dass es ohne die beiden Module für die Signalverarbeitung lauffähig ist. Die Schnittstelle zwischen der digitalen Signalverarbeitung und dem Protokollgenerator wurde in dieser Arbeit nicht entworfen.

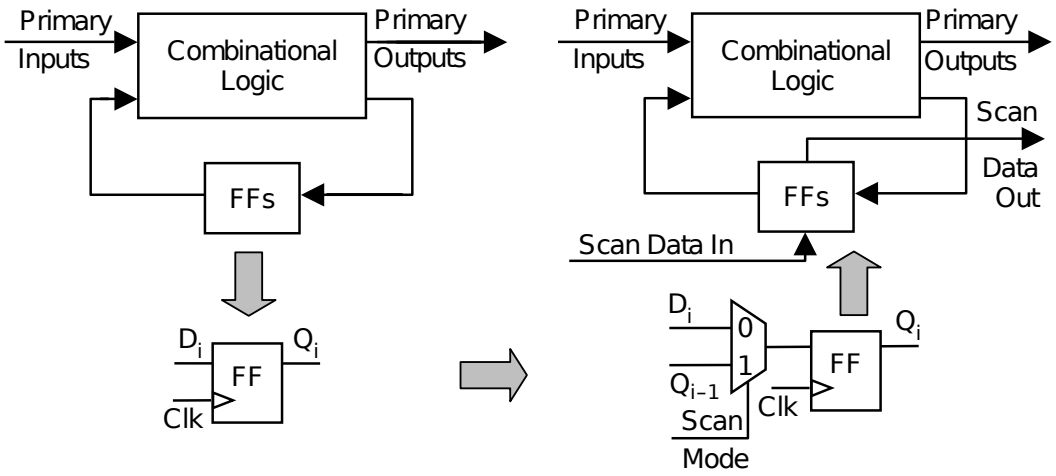


Abbildung 2.6: Scan-Chain-Flipflop. – Links die schematische Darstellung einer digitalen Logik ohne und rechts mit Scan-Chain-Flipflop. Modifiziert aus [45, Seite 30].

Durch die Modularisierung wird versucht, einzelne Module des CMOS-Testchips testen zu können, auch wenn andere nicht funktionsfähig sind.

2.4.1 Protokollgenerator

Der Protokollgenerator erzeugt das in [14] beschriebene Busprotokoll (siehe Abschnitt 1.1 und [16]). Dieses Modul wurde bereits in der FPGA-Implementierung ausgiebig getestet, weshalb es separat betrieben wird.

An den Protokollgenerator werden die Daten angelegt, die an den Linedriver gesendet werden sollen. Er generiert daraus den Manchestercode und setzt zwei Signale, die den Speedpulse (28 mA) und den Datapulse (14 mA) repräsentieren.

Erkennt das Modul an dem eingehenden Startsignal einen High-Pegel, dann wird mit der Ausgabe der anliegenden Datenbits im Manchestercode begonnen. Wird innerhalb von 150 ms kein neue Protokollausgabe gestartet, beginnt das Modul selbstständig ein sogenanntes Ruheprotokoll auszugeben [16]. Das Paritätsbit wird von dem Protokollgenerator selbst generiert. Abbbildung 2.8 zeigt das Blockschaltbild des Protokollgenerators.

Die Anzahl der übermittelten Bits ist von der Frequenz des Sensorsignals abhängig. Bei geringer Frequenz werden mehr Bits übertragen als bei hohen Frequenzen (siehe Tabelle 2.1).

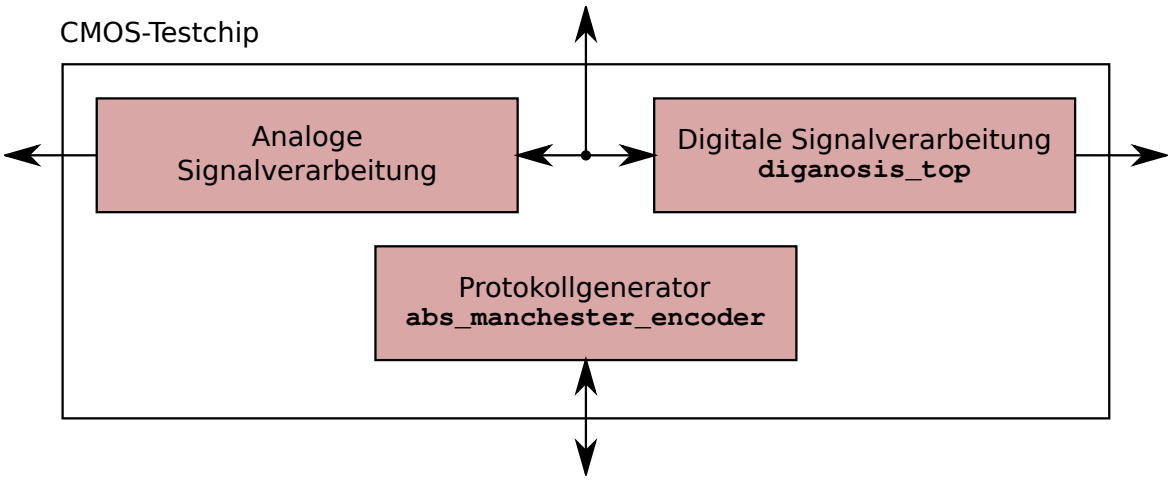


Abbildung 2.7: Modularisierung des CMOS-Testchips auf höchster Entwurfsebene. – Der CMOS-Testchip ist in drei Module unterteilt. Die beiden Signalverarbeitungsteile sind über Signale miteinander verbunden. Das Ausgabeprotokoll ist von den anderen Modulen unabhängig. Allerdings teilen sich alle Module die Spannugsversorgung.

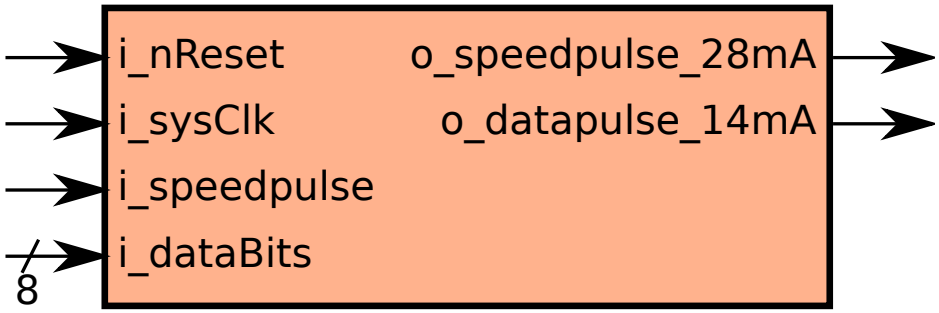


Abbildung 2.8: Protokollgenerator. – Von der Schnittstelle zur digitalen Signalverarbeitung kommen die Daten, die gesendet werden sollen. Mit dem eingehenden Speedpulse, bei jedem Nulldurchgang des Sensorsignals, wird signalisiert, dass die Daten gesendet werden sollen. Die beiden Ausgangssignale signalisieren dem Linedriver, welcher Srompegel auf den Bus gelegt werden soll.

Tabelle 2.1: Anzahl der Datenbits im AK-Protokoll in Abhängigkeit der Encoderfrequenz. – Die Geschwindigkeiten beziehen sich auf ein Encoderrad mit 96 Inkrementen (48 Zähnen). Quelle: [14].

Signalfrequenz	Ungefähre Geschwindigkeit	Zu sendende Datenbits
$< 1.800\,\text{Hz}$	$135\,\frac{\text{km}}{\text{h}}$	Bit 0 bis Bit 8
$< 2.000\,\text{Hz}$	$150\,\frac{\text{km}}{\text{h}}$	Bit 0 bis Bit 7
$< 2.200\,\text{Hz}$	$165\,\frac{\text{km}}{\text{h}}$	Bit 0 bis Bit 6
$< 2.400\,\text{Hz}$	$180\,\frac{\text{km}}{\text{h}}$	Bit 0 bis Bit 5
$< 2.800\,\text{Hz}$	$210\,\frac{\text{km}}{\text{h}}$	Bit 0 bis Bit 4
$< 3.200\,\text{Hz}$	$240\,\frac{\text{km}}{\text{h}}$	Bit 0 bis Bit 3
$< 4.000\,\text{Hz}$	$300\,\frac{\text{km}}{\text{h}}$	Bit 0 bis Bit 2
$< 5.000\,\text{Hz}$	$375\,\frac{\text{km}}{\text{h}}$	Bit 0 bis Bit 1

2.4.2 Digitale Signalverarbeitung

Die digitale Signalverarbeitung, wie sie schon in einem FPGA implementiert wurde [17], berechnet den THD des Sensorsignals U_{diff} mithilfe der ersten fünf Harmonischen (HD5) (sieheGleichung 1.2). Das Top-Level mit allen eingebundenen Modulen der FPGA-Implementation ist in Abbildung 2.9 zu sehen. Es wird in diesem Verfahren immer eine komplette Periode des Sensorsignals abgetastet. Dabei wird davon ausgegangen, dass sich die Periode nur langsam ändert. Durch Detektieren der Nulldurchgänge wird bestimmt, wie lang die Periode ist und daraufhin die Abtastrate eingestellt.

An den FPGA angeschlossene ADCs liefern die Differenzspannung (U_{diff}) und die Halbbrückenspannungen des Sensors. Die ADCs werden mithilfe der Komponente TL_ADC ausgelesen. Die Nulldurchgänge und damit die Periodenlänge des Sensorsignals wird mit dem Smart Comparator (SMART_COMPARATOR) [17] bestimmt. Die Drehrichtung des Rades wird mithilfe des Moduls DR_DETECTION erkannt.

Aus Gleichung 1.2 ist ersichtlich, dass die ersten fünf Harmonischen für die Berechnung des HD5 benötigt werden. Die Harmonischen werden mittels einer diskreten Fourier-Transformation (DFT) der Differenzspannung ermittelt. Diese DFT ist in dem Block TL_SAMPLE implementiert. Die für die DFT benötigten Sinus- und Kosinuswerte sind in einer Lookup Table (LUT) gespeichert. Diese LUT beinhaltet eine Logik, die die Anzahl der abgespeicherten Werte minimiert [17].

In der Komponente TL_CALC wird letztendlich der HD5 aus den Harmonischen berechnet. Um den HD5 zu bestimmen (Gleichung 1.2) wurden folgende arithmetische Operationen implementiert:

- Quadrierer,

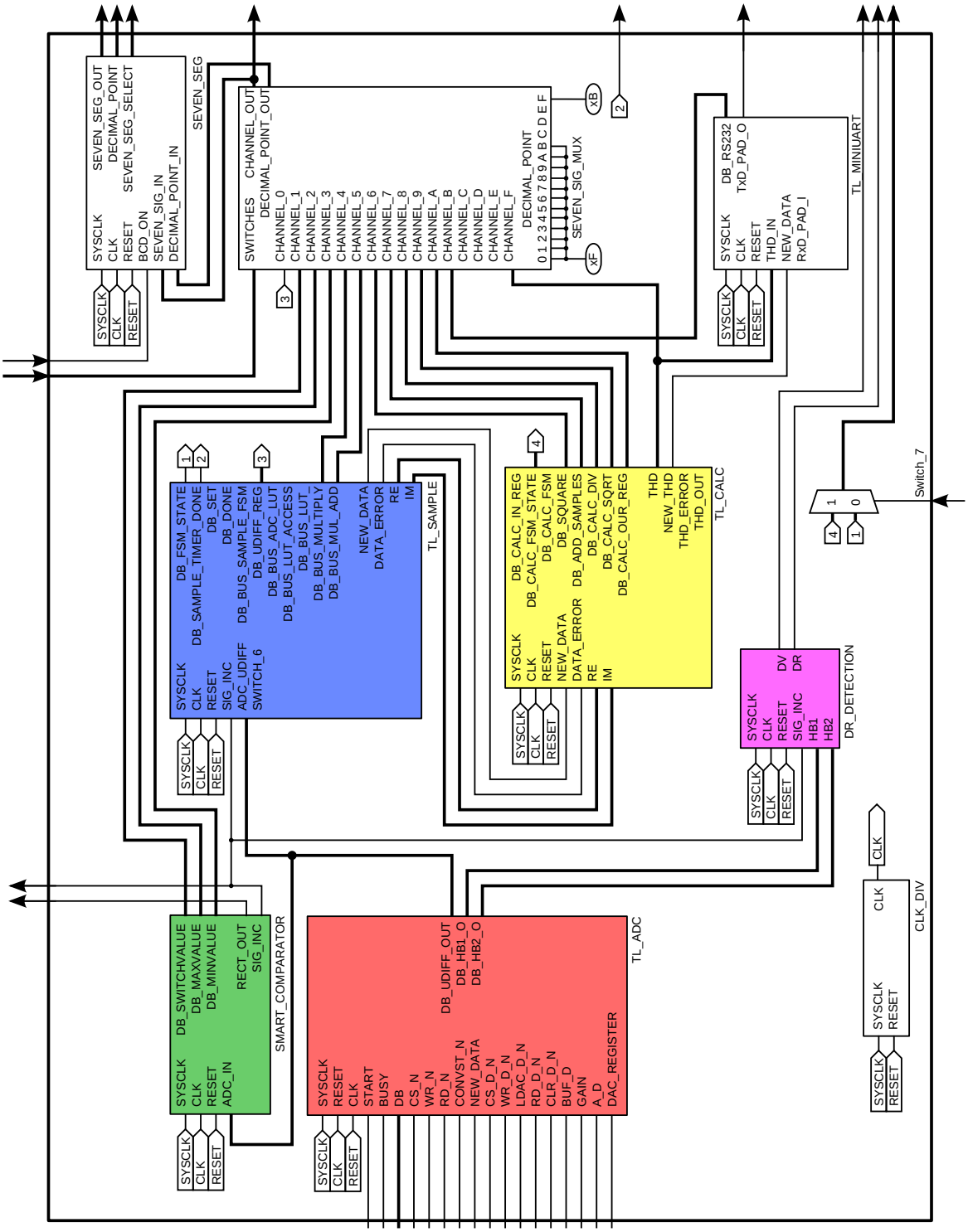


Abbildung 2.9: Top Level der FPGA-Implementation der digitalen Signalverarbeitung. – Die entscheidenden Komponenten wurden hier farblich markiert: TL_ADC dient zur Kommunikation mit den ADCs. SMART_COMPARATOR erkennt die Nulldurchgänge. TL_SAMPLE berechnet die Harmonischen des Sensorsignals. TL_CALC berechnet den HD5. DR_DETECTION bestimmt die Drehrichtung des Rades. Abbildung modifiziert aus [17].

- Addierer,
- Dividierer und
- Radizierer.

Durch die Verwendung von Multiplexern war es möglich, arithmetische Einheiten mehrfach zu benutzen.

Ferner sind noch Komponenten für die Ausgabe der Daten implementiert (SEVEN_SIG_MUSX, SEVEN_SEG und TL_MINIUART), die im Rahmen dieser Arbeit allerdings nicht verwendet wurden.

Für die Implementierung auf dem CMOS-Testchip wurde die digitale Signalverarbeitung, wie in Abschnitt 3 beschrieben wird, verändert.

2.4.3 Analoge Signalverarbeitung

In vorgehenden Arbeiten wurde das Sensorsignal mit geregelten Verstärkern analog vorverstärkt. Darüber hinaus wurde der in dem analogen Signal enthaltene Offset kompensiert. Siehe hierzu [24, 32, 40]. In der vorliegenden Arbeit wurden lediglich Teile der analogen Hardware implementiert.

2.5 Arbeitsplan

Als Ausgangspunkt dienen die in VHDL beschriebenen digitalen Module: Der Protokollgenerator und die digitale Signalverarbeitung mit der Berechnung der Harmonischen und des THD. Zunächst soll der Protokollgenerator, anschließend die digitale Signalverarbeitung und zum Schluss die analoge Signalverarbeitung erstellt werden. Danach sollen alle Module zu dem kompletten CMOS-Testchip vereint werden.

Dabei wurde ein strenges Zeitfenster gesetzt, da der CMOS-Testchip auf einem Multi-Project Wafer gefertigt werden sollte. Aus Zeitmangel wurden die Simulationen der digitalen Module auf funktionale Simulationen beschränkt. Für das analoge Modul muss aus dem gleichen Grund auf umfangreiche Simulationen verzichtet werden. Da die Simulation des kompletten Chips viel Zeit in Anspruch nehmen würde, ist diese komplett entfallen.

Während der Chip in der Produktion ist, wurde eine erste Teststrategie entworfen. Das primäre Ziel ist nicht einen, in der Serienproduktion notwendigen, vollständigen Test zu entwerfen, sondern lediglich festzustellen, ob grundsätzliche Fehler im Design vorhanden sind. Weiterführende Tests wurden auf einen späteren Zeitpunkt verschoben.

3 Entwurf

3.1 Protokollgenerator

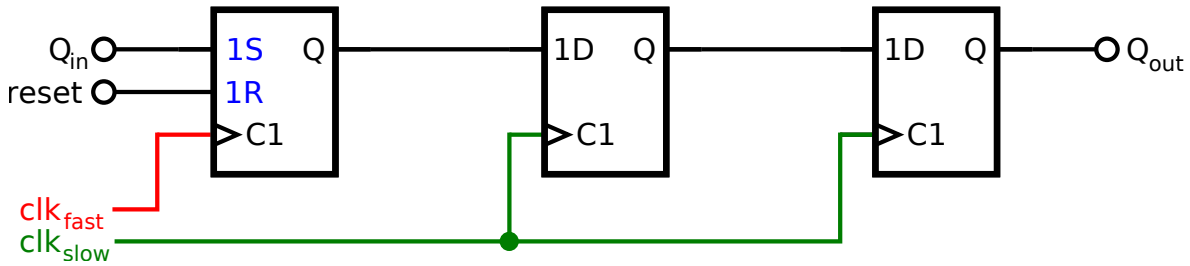
Der Protokollgenerator erzeugt, wie erwähnt, das AK-Protokoll, das von dem Line-Driver ASIC umgesetzt wird. Er besitzt dabei ein Eingangssignal, das die Protokollausgabe startet und acht Dateneingänge. Die Daten werden über zwei Ausgangssignale gesendet (siehe Abbildung 2.8). Das eine Ausgangssignal dient dazu, dem ABS-Steuergerät einen Nulldurchgang des Sensorsignals mitzuteilen (o_speedpulse_28mA). Mit dem zweiten Ausgangssignal werden die Daten zwischen den Speedpulsen übermittelt (o_datapulse_14mA). Eine Darstellung aller, in dem Protokollgenerator enthaltenen, Module und der Quelltext sind in Anhang B.1 zu finden.

3.1.1 Besonderheiten der Chip-Implementation

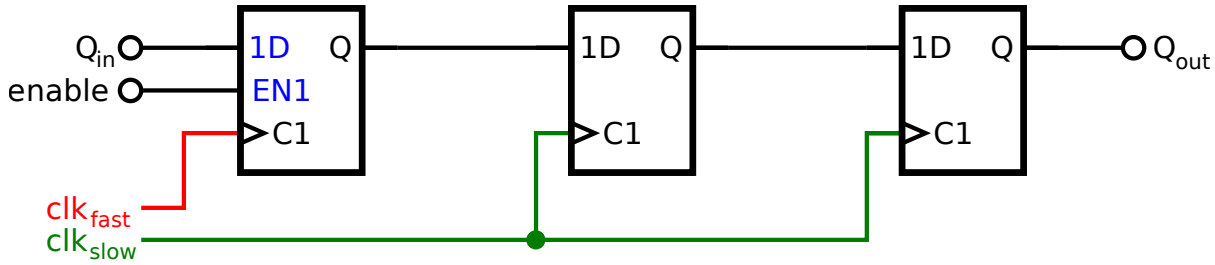
Die Taktfrequenz der Logik wurde auf 16 MHz festgelegt. Der Protokollgenerator benötigt einen Takt von 40 kHz, um die seriellen Daten korrekt ausgeben zu können [16]. Mithilfe eines Zählers wird der interne Takt (40 kHz) mit einem Tastgrad von ungefähr 50 % erzeugt.

Da der Protokollgenerator mit einem heruntergeteiltem Takt betrieben wird, müssen alle eingehenden Signale zu dem langsameren Takt synchronisiert werden, um metastabile Zustände zu vermeiden. In Abbildung 3.1a ist die Verschaltung der Eingangsflipflops zu sehen, wie sie für das eingehende Speedpulse-Signal verwendet werden. Dabei handelt es sich bei dem Flipflop, das auf dem schnelleren Takt arbeitet, um ein taktflankengesteuertes RS-Flipflop. Dies ist notwendig, da kein eingehendes Speedpulse-Signal übersehen werden darf. Das RS-Flipflop wird von dem Protokollgenerator zurückgesetzt, sobald die Ausgabe des Protokolls gestartet wurde. Die Dateneingänge werden ebenfalls mit drei Flipflops synchronisiert, dabei handelt es sich aber um drei D-Flipflops (siehe Abbildung 3.1b). Das Erste der drei D-Flipflops besitzt einen Enable-Eingang. Dieser ist mit dem eingehenden Speedpulse verbunden, sodass die Daten mit einem eingehenden Speedpulse gespeichert werden.

Die Synchronisation wäre nicht notwendig, wenn der interne Takt des Protokollgenerators mit der gleichen Taktfrequenz, wie der Systemtakt laufen würde. Dafür müssten allerdings weitere Zähler implementiert werden, damit das AK-Protokoll weiterhin eingehalten werden kann. Des Weiteren würde sich die Leistungsaufnahme aufgrund des höheren Taktes erhöhen.



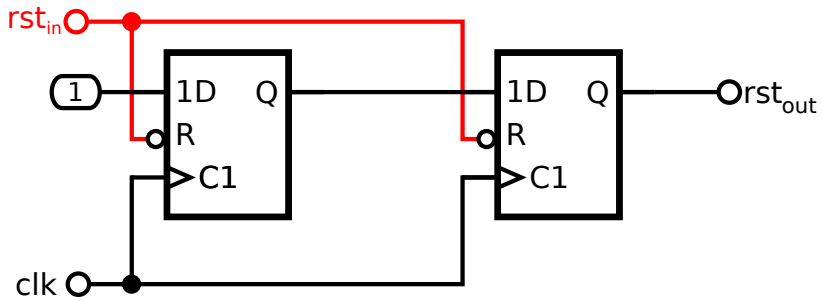
3.1a Flipflops zum Synchronisieren des eingehenden Speedpulses. – Das RS-Flipflop (links) läuft mit dem Systemtakt von 16 MHz. Die beiden D-Flipflops werden mit dem internen Takt von 40 kHz betrieben. Diese Schaltung ist notwendig, um metastabile Zustände der Eingangssignale zu vermeiden und auf jeden eingehenden Speedpulse zu reagieren.



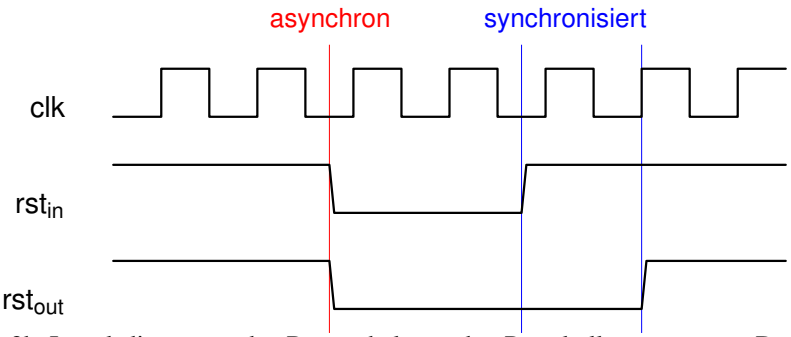
3.1b Flipflops zum Synchronisieren der eingehenden Datensignale. – Das erste Flipflop (links) läuft mit dem Systemtakt von 16 MHz. Um die anliegenden Daten zu übernehmen, muss an dem Enable-Eingang einen High-Pegel anliegen. Die beiden anderen Flipflops werden mit dem internen Takt von 40 kHz betrieben. Diese Schaltung ist notwendig, um metastabile Zustände der Eingangssignale zu vermeiden und die Daten mit dem eingehenden Speedpulse zu übernehmen.

Abbildung 3.1: Filpflops zum Synchronisieren der Eingangssignale des Protokollgenerators. – In 3.1a die Synchronisation des Speedpulse Signals und in 3.1b die Synchronisation der Datensignale.

In dem Protokollgenerator wurden nur Flipflop mit asynchronem Reset verwendet. Flipflops mit synchronem Reset sind in der Standardbibliothek nicht vorhanden und müssten aus einem Flipflop, einem Inverter und einem NAND-Gatter synthetisiert werden. Damit die Flipflops nicht in einen metastabilen Zustand geraten, wenn der Reset in der nähe der Taktflanke losgelassen wird, wurde die Resetlogik aus Abbildung 3.2a implementiert. Dadurch, dass der eingehenden Reset an die asynchronen Reseteingänge der Flipflops gelegt ist, wird ein eingehender Reset sofort (asynchron) an den Ausgang der Schaltung weitergereicht. Wenn der eingehende Reset losgelassen wird, übernimmt das erste Flipflop den anstehenden High-Pegel nach dem ersten Takt, das Zweite übernimmt diesen nach dem zweiten Takt. Auch in diesem Fall dienen die beiden in Reihe geschalteten Flipflops dazu, metastabile Zustände am Ausgang der Schaltung zu vermeiden. Das Loslassen des Resets erfolgt somit für alle Flipflops synchron zum Takt (siehe Abbildung 3.2b). Siehe [13].



3.2a Resetschaltung des Protokollgenerators. – Diese Schaltung ermöglicht ein asynchrones Setzen und ein synchrones Zurücksetzen des Resets



3.2b Impulsdiagramm der Resetschaltung des Protokollgenerators. – Das Setzen des Resets erfolgt asynchron und das Zurücksetzen des Resets synchron zum Takt.

Abbildung 3.2: Resetschaltung des Protokollgenerators.

Da der RTL Compiler von Cadence keine Typdeklarationen[1] unterstützt, mussten die Zustände als Konstanten deklariert werden. Dies bietet zudem den Vorteil, dass die Zustandscodierung bekannt ist, was das Debuggen des Chips später erleichtert.

[1] **type** STATES **is** (S0, S1, S2);

3.1.2 Testmöglichkeiten

Der Zustandsautomat in dem Protokollgenerator bietet eine einfache Methode, Fehler in der Logik zu erkennen, indem der aktuelle Zustand über drei Pins ausgegeben wird. So kann der Zustand mithilfe eines Logikanalysators ermittelt und möglicherweise auftretende Fehler in der Hardware schnell eingegrenzt werden. Verhält sich der Zustandsautomat nicht wie erwartet, so sind weitere Aus- und Eingänge vorgesehen, über die der Protokollgenerator zu Debuggen ist.

Für die Funktion sind vor allem die internen Reset-Signale, sowie der intern heruntergeteilte Takt wichtig, da ohne diese keine der Flipflops arbeiten würden. Um diese Signale zu kontrollieren wurden sie „aufgetrennt“ und aus dem Chip geführt (siehe Abbildung 2.5). Wurde die Funktionalität der Logik, die diese Signale treiben nachgewiesen, so können die Ausgänge direkt auf die Eingänge gelegt werden. Andernfalls kann ein extern erzeugtes Signal eingespeist werden.

Das Signal i_speedpulse startet die Ausgabe eines Protokolls. In späteren Implementationen soll es von der digitalen Signalverarbeitung gesendet werden, wenn diese einen Nulldurchgang des Sensorsignals registriert. Es ist ein entscheidendes Element des Protokollgenerators, denn ohne einen eingehendes Signal wird nur das Ruheprotokoll ausgegeben. Aus diesem Grund wird es auf die gleiche Weise wie die Takt- und Reset-Signale ausgegeben.

Ähnlich verhält es sich mit dem Shift-Register, das die zu sendenen Halbbits des Manchestercodes enthält. Damit es möglich ist, das Shift-Register durch ein eingespeistes Signal zu ersetzen, ist es notwendig zu erfahren, wann ein neues Bit angelegt werden soll. Dazu wurde das Steuersignal (shiftManchsterReg) ebenfalls an einen Ausgang gelegt.

Diese Testmöglichkeit zielt nicht darauf ab, die Funktionalität der einzelnen Logik-Gatter und Flipflops zu testen, sondern dient dazu, fehlerhafte Blöcke zu umgehen.

3.1.3 Funktionale Simulation (Register-Transfer Level)

Beschreibung der Testbench

Die Funktion des Protokollgenerators wurde mit einer funktionalen Simulation getestet. Dabei werden unterschiedliche Daten an die Eingänge gelegt und das Verhalten der Schaltung manuell analysiert. Zum funktionalen Test wurde eine Testbench entworfen, die die acht möglichen Ausgabebitlängen, den Ruhepuls und die Unterbrechung des Ruhepulses stimuliert. Die Bitlängen in Abhängigkeit der Encoderfrequenz sind in Tabelle 2.1 zu sehen.

In der Testbench werden zuerst die acht verschiedenen Bitlängen durchlaufen. Dabei sind die, in Abschnitt 3.2.2 beschriebenen, Ein- und Ausgangspaare direkt miteinander verbunden, können aber optional über die Testbench eingespeist werden. Anschließend wird der

Ruhepuls komplett stimuliert und dann zweimal der Ruhepuls ausgegeben, der dann während der Ausgabe an zwei verschiedenen Stellen im Ruhepuls, durch zu sendende Daten, unterbrochen wird.

Ergebnisse der Simulation

Die Simulation verhält sich wie erwartet. Abbildung 3.3 zeigt die Ausgabe des Protokollgenerators von acht Bit. Das Signal sensorOutput ist ein von der Testbench, aus den beiden Ausgangssignalen (o_speedpulse_28mA und o_datapulse_14mA) erzeugtes Signal, was den Strom auf der Busleitung darstellen soll. Die grün markierten Flanken sind die Ausgangsdaten im Manchester Code. Die Ausgangsdaten entsprechen dem Eingangswert ($11011010_{\text{bin}} = \text{DA}_{\text{hex}}$).

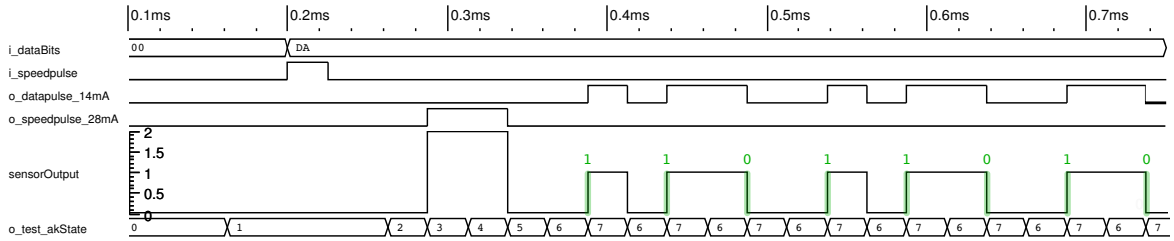


Abbildung 3.3: Simulation des Protokollgenerators. – Die Abbildung zeigt die Ausgabe von 8 Bit (alle Datenbits ohne Paritätsbit). Das Signal sensorOutput ist ein, aus den beiden Ausgangssignalen (o_speedpulse_28mA und o_datapulse_14mA) in der Testbench erzeugtes Signal. Die grün markierten Flanken sind die Ausgangsdaten im Manchester Code. Die Ausgangsdaten entsprechen dem Eingangswert ($11011010_{\text{bin}} = \text{DA}_{\text{hex}}$).

3.2 Digitale Signalverarbeitung

Zusätzlich zum HD5, dessen Berechnung auf den ersten fünf Harmonischen beruht, wurde noch die Berechnung des HDI implementiert. Der HDI berechnet sich aus der Gesamtleistung P_{ges} und der Leistung der Grundschwingung (erste Harmonische) P_1 nach Gleichung 3.1. Dazu wurde versucht, möglichst die Module der HD5-Berechnung mit zu benutzen. Es mussten allerdings, da in der HD5-Berechnung nicht notwendig, noch Subtrahierer eingesetzt werden. Der Signalfluss ist in Abbildung 3.4 dargestellt.

$$\mathrm{HDI} = \sqrt{\frac{P_{\mathrm{ob}}}{P_{\mathrm{ges}}}} = \sqrt{\frac{P_{\mathrm{ges}} - P_1}{P_{\mathrm{ges}}}} \tag{3.1}$$

$$P_{\mathrm{ges}} = \frac{1}{N}\left(\sum_{n=0}^{N-1} x_n^2 - \left(\sum_{n=0}^{N-1} x_n\right)^2\right) \tag{3.2}$$

$$P_1 = |X_1|^2 \tag{3.3}$$

3.2.1 Besonderheiten der Chip-Implementation

Die Zustände mussten auch, wie im Protokollgenerator in VHDL, mithilfe von Konstantendeklarationen codiert werden (siehe Abschnitt 3.1.1).

Da auf dem Chip nur eine begrenzte Anzahl von Ein- und Ausgabeports zur Verfügung stehen, werden die HD5- und HDI-Ergebnisse seriell ausgegeben. Die Ausgabe ist dabei stark an den SPI-Bus[2] angelehnt. Abbildung 3.5 zeigt die Funktionsweise der seriellen Ausgabe. Die Signalleitung hd_ser liefert einen High-Pegel, während die Daten seriell, beginnend mit dem HDI (dabei wird das MSB[3] zuerst gesendet), ausgegeben werden. Dabei werden die Daten synchron zum Systemtakt, bei steigender Taktflanke, ausgegeben. Beim Einlesen der Werte bietet es sich an, auf die negative Taktflanke zu reagieren, da das Bit dann stabil anliegen wird.

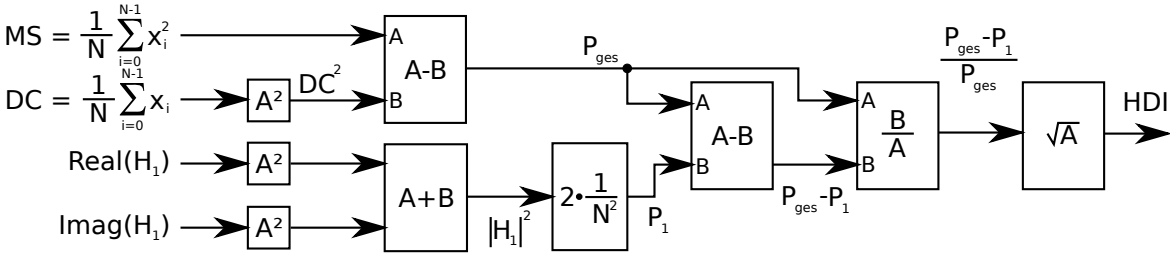


Abbildung 3.4: Signalfluss der HDI-Berechnung. – Als Eingangssignale dienen die erste Harmonische, das Quadrat des Quadratischen Mittels (MS)[4]und die Leistung des Gleichanteils (DC).

[2] **SPI** Serial Peripheral Interface

[3] **MSB** Most Significant Bit

[4] Dabei ist anzumerken, dass nicht der Effektivwert (RMS; Root Mean Square), sondern dessen Quadrat angewendet wird.

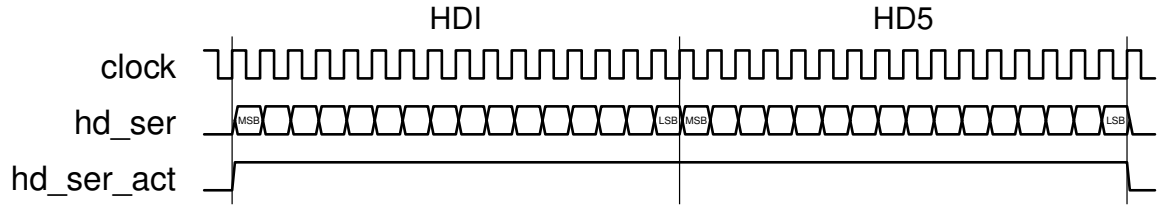


Abbildung 3.5: Serielle Ausgabe der HDI- und HD5-Werte. – Während der Datenübertragung (synchron zum Systemtakt) liefert das Signal hd_ser_act einen High-Pegel. Zuerst wird der HDI, dann der HD5 übertragen (beginnend mit dem MSB).

Steuerleitung (i_mux_mode)	Funktion
„01“	Die Signale der internen digitalen Signalverarbeitung werden über bidirektionale Pins aus dem Chip geführt.
„10“	Die Signale der internen analogen Signalverarbeitung werden über bidirektionale Pins aus dem Chip geführt.
„11“ „00“	Die internen Signalverarbeitungs Module arbeiten miteinander; alle Signale werden zusätzlich aus dem Chip geführt.

Tabelle 3.1: Betriebsmodi des Multiplexers (analog_mux)

3.2.2 Testmöglichkeiten im Zusammenspiel mit der analogen Hardware

Das analoge und das digitale Signalverarbeitungsmodul kommunizieren über Steuer- und Datenleitungen miteinander. Damit es möglich ist, sowohl die digitale als auch die analoge Hardware unabhängig voneinander zu testen, wurde ein Multiplexer für die digitalen Signale entwickelt, dessen Ausgänge auf bidirektionale Pads gelegt wurden. In Abbildung 3.6 ist die Funktionsweise des Multiplexers skizziert. Über Steuerleitungen lässt sich die Richtung des Signalflusses einstellen. Die drei Betriebsmodi des Multiplexers sind kurz in Tabelle 3.1 beschrieben. Im Normalfall (i_mux_mode = „11“ oder „00“) sind die beiden Module über den Multiplexer direkt miteinander verbunden. An den, aus dem Chip geführten Signalen, kann man die Kommunikation beobachten. In den anderen beiden Modi kann entweder nur mit dem analogen (i_mux_mode = „10“) oder dem digitalen Modul (i_mux_mode = „01“) über Pins kommuniziert werden. Alle Signale zu dem unbenutzten Modul werden auf Low-Pegel gesetzt. Über Steuerleitungen an den Pads kann eingestellt werden, ob es sich um einen Eingang oder Ausgang handelt.

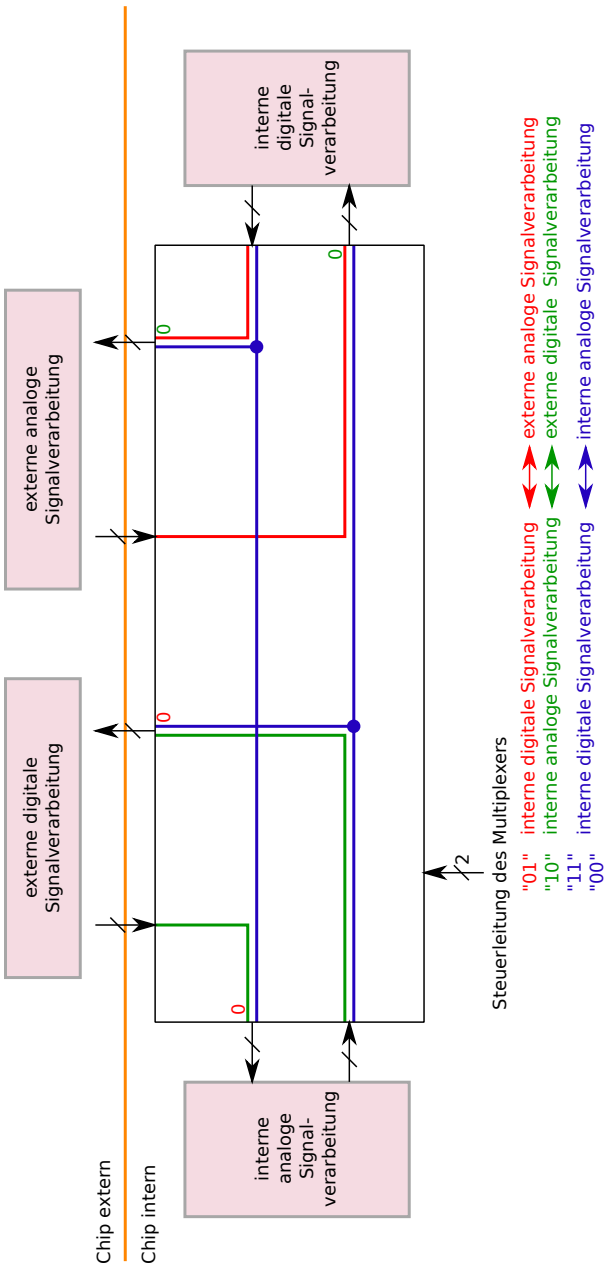


Abbildung 3.6: Darstellung der Funktionsweise des Multiplexers (analog_mux). – In dieser Abbildung sind auf der linken Seite die Signale des analogen Signalverarbeitungsteils, rechts die des digitalen Signalverarbeitungsteils und oben die extern eingespeisten Signale, angeordnet. Über die Steuerleitungen des Multiplexers kann die Richtung des Signalflusses eingestellt werden. Die farblichen Markierungen geben den Signalfluss bei den Kombinationen der Steuerleitungen an. Siehe Anhang B.2.2.

Um die DFT überprüfen zu können, werden die die Real- und Imaginärteile der Harmonischen seriell ausgegeben. Die jeweils 26 Bit breiten Real- und Imaginärteile werden wie der HD5 und HDI seriell ausgegeben (siehe Abschnitt 3.2.1).

Bei der Berechnung der THD-Werte sind zwei Zustandsautomaten beteiligt, deren Zustände über fünf Pins gemultiplext ausgegeben werden. Mit einem Eingangssignal kann ausgewählt werden, welche der beiden Zustände ausgegeben werden soll. Das Multiplexen war aufgrund der Beschränkung der Pinanzahl notwendig.

Ein wichtiger Teil der Signalverarbeitung ist das Erkennen der Nulldurchgänge des Sensorsignals. Zur Kontrolle wird ein Signal (o_zero_cross) ausgegeben, das einen Impuls liefert, wenn ein Nulldurchgang des Sensorsignals erkannt wurde.

Des Weiteren werden zwei Signale ausgegeben, die Fehler in der Berechnung anzeigen. Das Signal o_error_dev wird High, wenn die Periodenlänge des Sensorsignals zu sehr von der letzten Periodenlänge abweicht. Ist die Periode des Sensorsignals zu groß (größer oder gleich dem maximalen Zählerstand), oder kleiner als ein Schwellwert, dann wird das Signal o_error_ovf_unf gesetzt.

3.2.3 Funktionale Simulation (Register-Transfer Level)

Beschreibung der Testbench

Die im Rahmen dieser Arbeit neu entstandenen Komponenten wurden erst einzeln in dem Register-Transfer Level simuliert. Erst nachdem die Funktion der Komponenten gegeben war, wurden sie dem digitalen Signalverarbeitungsmodul hinzugefügt.

Für die funktionale Simulation wurde eine Testbench geschrieben, in der eine Komponente eingebunden wird, die die analoge Signalverarbeitung simuliert. So kann die digitale Signalverarbeitung mit realen, vorher aufgenommenen, Sensorsignalen simuliert werden. Dies wird dadurch ermöglicht, dass der Simulator CSV[5]-Dateien einlesen kann. Diese CSV-Dateien enthalten die Sensorsignale, die zuvor an einem Radmessplatz aufgenommen wurden [38].

Um die Ergebnisse der Berechnung anzuzeigen, wurde ein Prozess in die Testbench geschrieben, der die seriell ausgegebenen HDI-und HD5-Werte deserialisiert und in Prozent ausgibt. Daneben existiert eine weitere Testbench, mit der der eingebaute Multiplexer getestet werden kann.

[5] **CSV** Comma-separated values

Ergebnisse der Simulation

Abbildung 3.7 ist ein Ausschnitt eines Sensorsignals mit dem, von der Logik berechneten HDI- und HD5-Werten, zu sehen. Das Signal i_data_u_diff zeigt das quantisierte Sensorsignal (von dem ADC-Simulator erzeugt). Anhand des Signals o_zero_cross ist zu erkennen, dass die Nulldurchgänge des Sensorsignals erkannt wurden (auf dem Sensorsignal ist ein Offset addiert). Die seriell über die Signale o_hd_ser und o_hd_ser_act ausgegebenen Daten werden von der Testbench eingelesen und in Prozent dargestellt (discrete (hd5Result_real) und discrete (hdiResult_real)).

Die Simulation zeigt, dass die digitale Signalverarbeitung Daten ausgibt, die annähernd mit den erwarteten Werten übereinstimmen. Abweichungen zu den theoretisch berechneten Werten, liegen unter anderem an der Quantisierung. Genauere Untersuchungen wurden im Rahmen dieser Arbeit nicht durchgeführt.

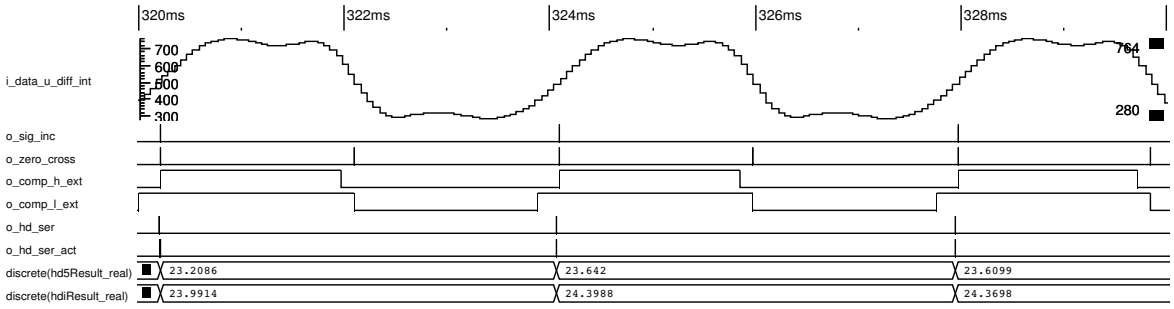


Abbildung 3.7: Simulation der digitalen Signalverarbeitung. – i_data_u_diff ist das quantisierte Sensorsignal vom ADC-Simulator. discrete (hd5Result_real) und discrete (hdiResult_real) sind die von der Testbench aufbereiteten Ergebnisse der digitalen Signalverarbeitung.

3.3 Analoge Hardware

Auf dem CMOS-Testchip wurde nicht die komplette analoge Signalverarbeitung, wie sie für die Aufarbeitung des Sensorsignals notwendig wäre, realisiert. Es wurden stattdessen einzelne Bauelemente implementiert. Der komplette Schaltplan der analogen Hardware in dem CMOS-Testchip ist in Anhang E.1 zu finden.

3.3.1 Komponenten

Operationsverstärker

Für die Aufbereitung des Sensorsignals wird eine Verstärkerschaltung mit drei Operationsverstärkern benötigt. Für die vorliegende Arbeit wurden diese Operationsverstärker aus der Standardbibliothek ohne Beschaltung in dem CMOS-Testchip implementiert. Das bedeutet, dass sie für erste Tests noch extern beschaltet werden müssen. Jeder Operationsverstärker benötigt eine Konstantstromquelle, die jeweils 11.5 µA liefert. Die analogen Ein- und Ausgänge wurden aus dem CMOS-Testchip geführt. Die digitalen Steuersignale sind in dem CMOS-Testchip auf konstante Pegel gelegt, sodass die Operationsverstärker immer aktiv sind.

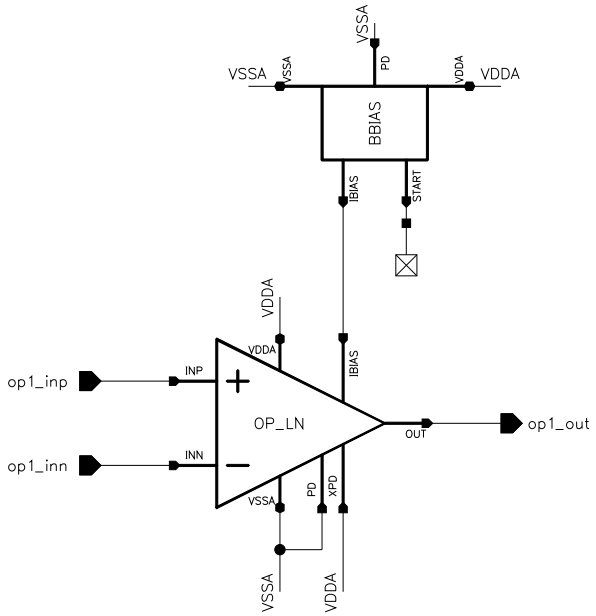


Abbildung 3.8: Beschaltung der Operationsverstärker im CMOS-Testchip. – Über dem Operationsverstärker ist die Konstantstromquelle zu sehen. Alle Steuereingänge des Operationsverstärkers und der der Konstantstromquelle wurden auf konstante Pegel gelegt, sodass der Operationsverstärker immer aktiv ist.

Komparatoren

Zur Bestimmung der Nulldurchgänge wurde in der FPGA-Implementierung ein digitaler Smart Comparator verwendet [17]. Für die vorliegende Arbeit wurde allerdings auf zwei Komparatoren mit einer nachgeschaltetn Logik zurückgegriffen. Die Schaltschwellen der

Komparatoren müssen extern über analoge Eingänge eingestellt werden. Da die Ausgänge der Komparatoren als digitale Signale zu interpretieren sind, werden sie auf den in Kapitel 3.2.2 beschriebenen Multiplexer gegeben, um das Debuggen zu ermöglichen. Die digitalen Steuersignale sind, wie bei den Operationsverstärkern, im CMOS-Testchip auf konstante Pegel gelegt (siehe Abbildung 3.9).

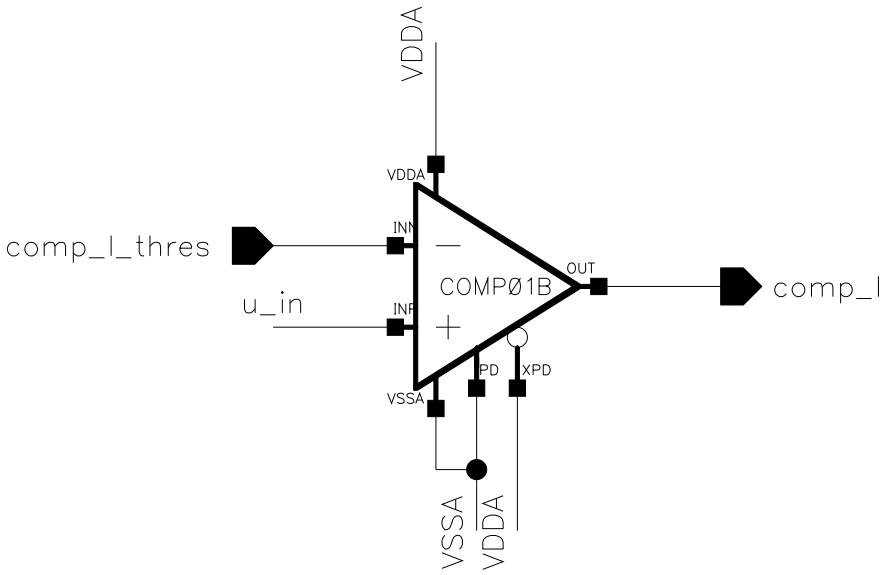


Abbildung 3.9: Beschaltung der Komparatoren im CMOS-Testchip. – Die Steuereingänge wurden auf konstante Pegel gelegt, sodass die Komparatoren immer aktiv sind.

Power-on Reset

Damit nach dem Einschalten der Betriebsspannung die digitalen Module in einen definierten Zustand gesetzt werden, benötigt man einen Reset, der ausgelöst wird, sobald die Betriebsspannung stabil ist. Dafür wurde ein Power-on Reset (POR) Modul implementiert. Der Ausgang dieses Moduls folgt dem Spannungsverlauf der Betriebsspannung, sobald diese einen gewissen Pegel überschritten hat (siehe Abbildung 3.14b). Der Ausgang des POR ist somit analog, kann aber als ein digitales Signal interpretiert werden. Um die Funktionsweise deutlich zu machen, wird der Ausgang über einen analogen Pin ausgegeben.

Analog-Digital-Umsetzer

Der ADC beinhaltet sowohl analoge wie auch digitale Hardware. In diesem Fall handelt es sich um einen 10 Bit ADC, der nach dem Prinzip der sukzessiven Approximation ar-

beitet. Die Steuerung des ADCs kann durch eine externe Beschaltung oder durch das digitale Signalverarbeitungs-Modul auf dem CMOS-Testchip erfolgen (siehe Multiplexer in Abschnitt 3.2.2). Für den Digitalteil des ADCs ist eine Verbindung mit der digitalen Versorgungsspannung notwendig (siehe Abbildung 3.10).

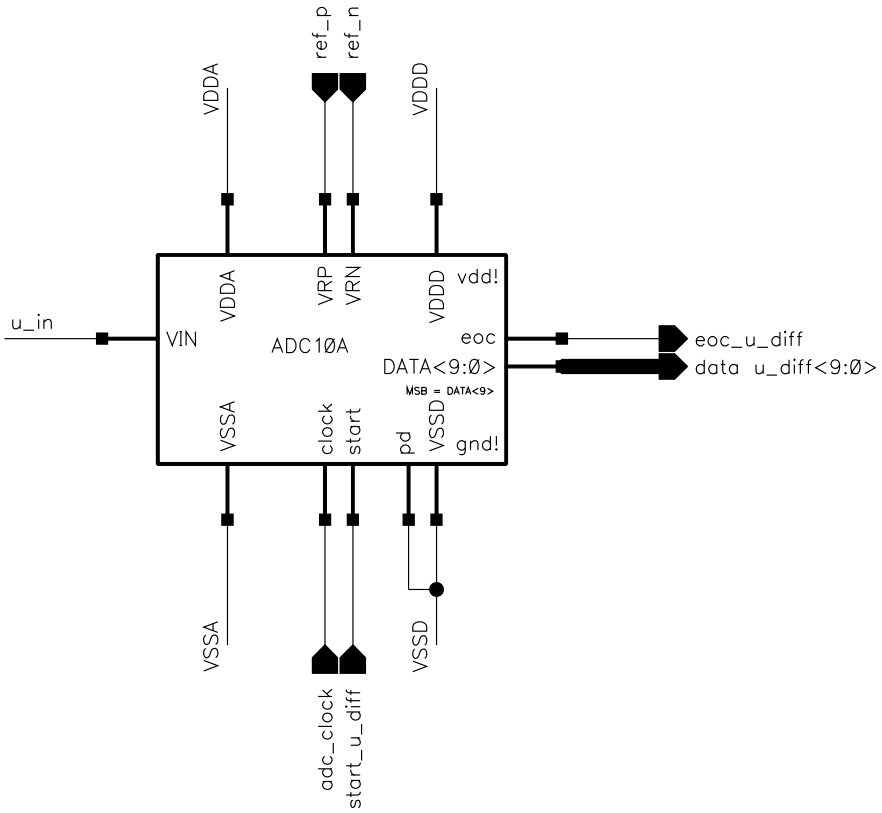


Abbildung 3.10: Beschaltung des Analog-Digital-Umsetzers im CMOS-Testchip. – Der Power-Down-Eingang wurde auf konstanten Low-Pegel gelegt. Der ADC ist somit betriebsbereit.

3.3.2 Simulation der Schaltung

Mithilfe der Simulation wird die korrekte Verschaltung der analogen Komponenten überprüft. Dazu wurde die Testschaltung aus Anhang E.2 entworfen.

Operationsverstärker

Die Funktion der Operationsverstärker wurde überprüft, indem nichtinvertierende Verstärker mit unterschiedlichen Verstärkungsfaktoren aufgebaut wurden. In Abbildung 3.11 ist

exemplarisch die Beschaltung des Operationsverstärkers OP1 gezeigt, in Abbildung 3.12 die Ein- und Ausgänge aller drei Operationsverstärker.

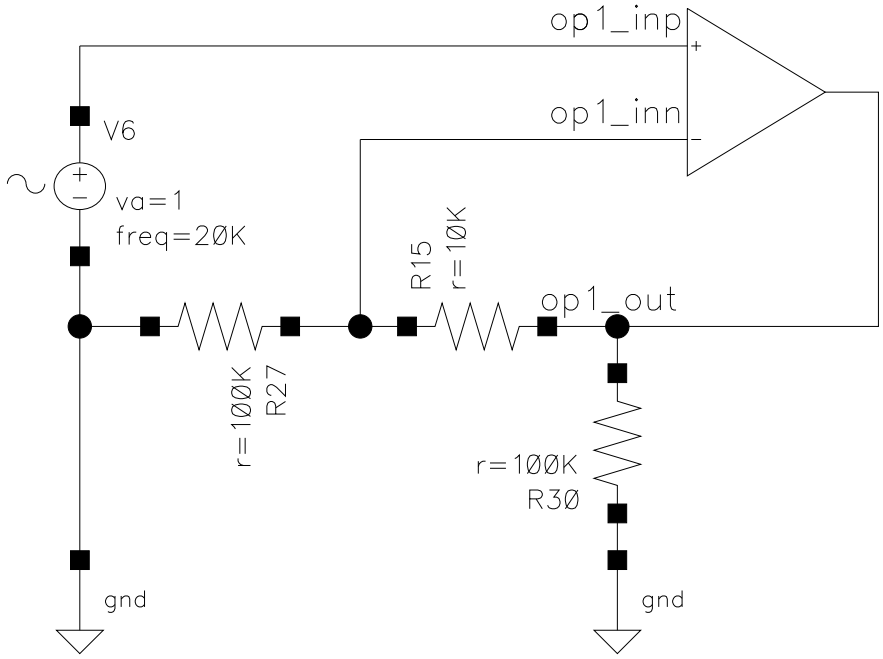


Abbildung 3.11: Simulationsbeschaltung der Operationsverstärker. – Die Operationsverstärker wurden als nichtinvertierende Verstärker aufgebaut. Exemplarisch ist in dieser Abbildung die Beschaltung des OP1 gezeigt. Verstärkungsfaktor $v_1 = 1 + \frac{R_{15}}{R_{27}} = 1.1$.

Komparatoren

An die Komparatoren wurde eine statische Schwellenspannung angelegt: $U_{\text{comp\_h\_thres}} = 2.15\,\text{V}$ und $U_{\text{comp\_l\_thres}} = 1.05\,\text{V}$. Als Eingangsspannung wurde eine Sinusspannung $u_{\text{in}} = 1.65\,\text{V} + 0.75\,\text{V}\sin\left(2\pi \cdot 100\,\text{kHz} \cdot t\right)$ angelegt und die Ausgänge der Komparatoren (comp_h und comp_l) überprüft. Abbildung 3.13 zeigt die Ein- und Ausgänge der Komparatoren.

Power-on Reset

Um den POR zu testen, wurde als Versorgungsspannung eine Spannungsquelle mit dreiecksförmiger Spannung angelegt. Wie in Abbildung 3.14a zu sehen, folgt der Ausgang der Versorgungsspannung, sobald diese eine Schwellwert überschreitet. Dieses Verhalten ist mit dem im Datenblatt beschriebenen identisch (siehe Abbildung 3.14b).

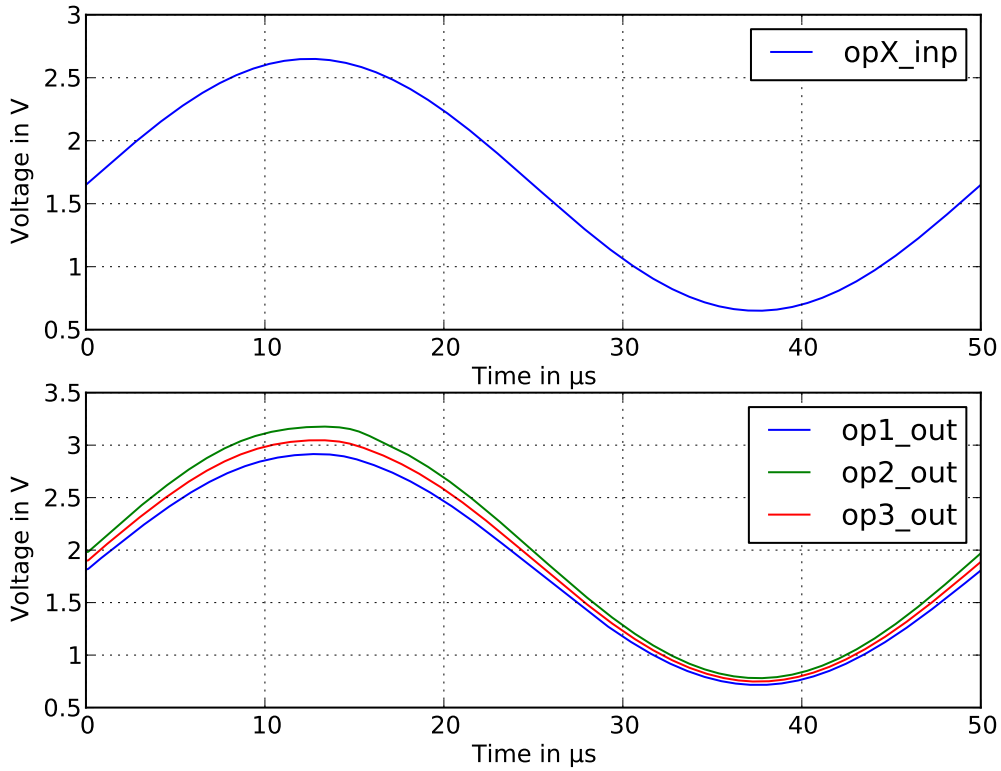


Abbildung 3.12: Simulationsergebnis der Operationsverstärker als nichtinvertierende Verstärker. – Die nichtinvertierenden Verstärker wurden mit den Verstärkungsfaktoren $v_1 = 1.10$ für OP1, $v_2 = 1.20$ für OP2 und $v_3 = 1.15$ für OP3 aufgebaut. Das obere Signal opX_inp zeigt das Eingangssignal, das an alle OPs angelegt wurde.

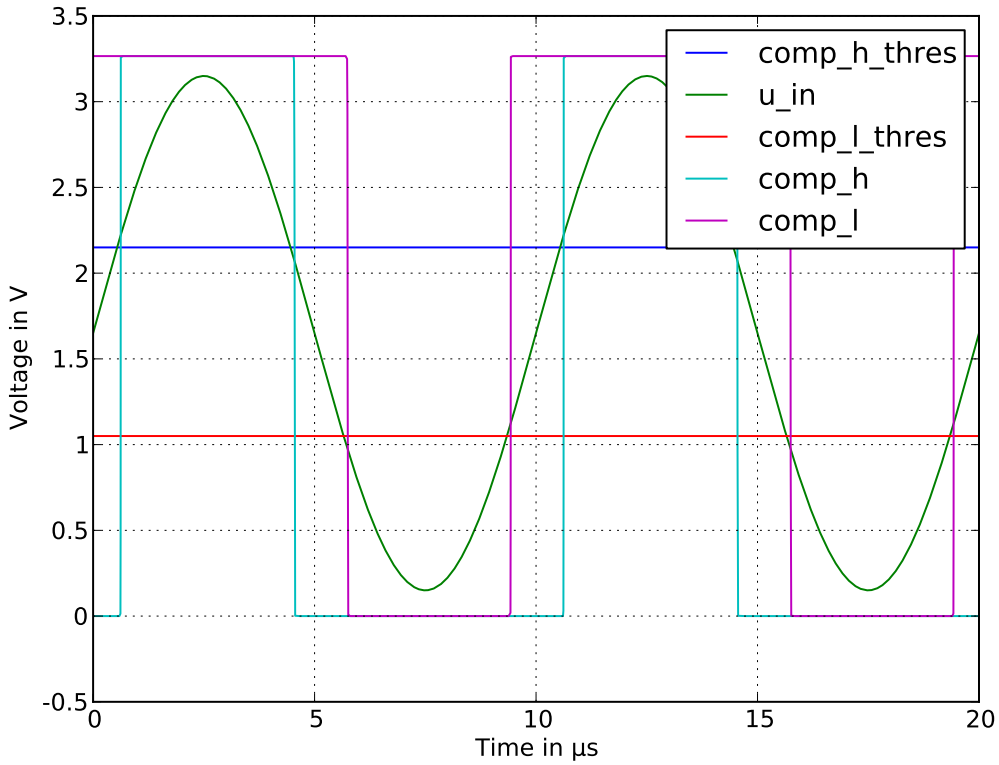
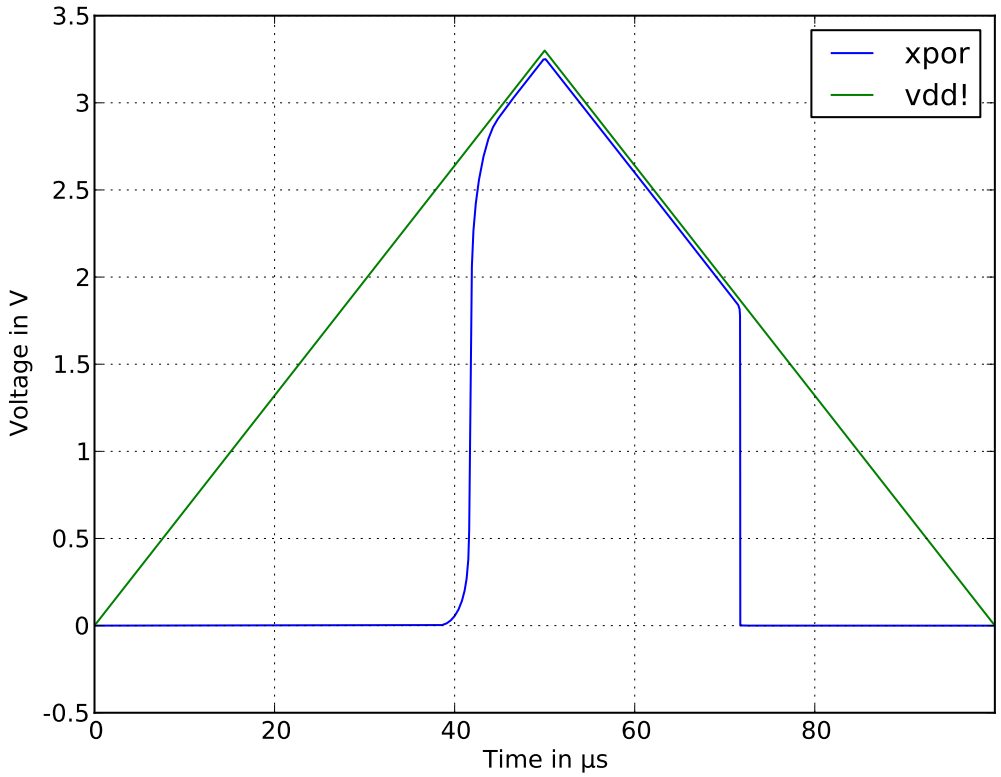
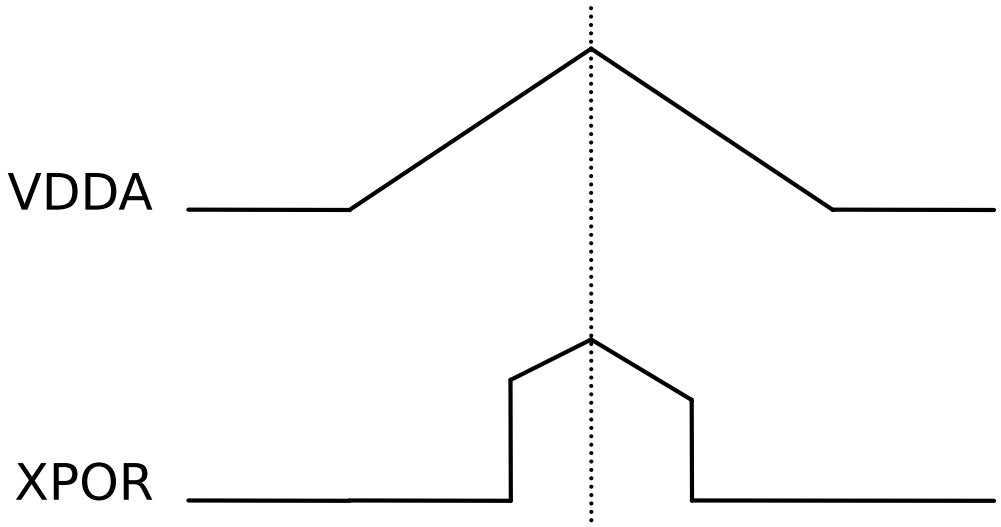


Abbildung 3.13: Simulationsergebnis der Komparatoren. – In grün ist die Eingangspannung der Komparatoren (u_in) abgebildet. Die Schwellenspannungen liegen bei 2.15 V (comp_h_thres in blau) und bei 1.05 V (comp_l_thres in rot). Die Ausgänge der Komparatoren (comp_h und comp_l) liefern einen High-Pegel, wenn die Eingangsspannung über dem jeweiligen Schwellenwert liegt. Die Abweichungen der Ausgangsspannugnen sind auf die unterschiedlichen Verstärkungsfaktoren zurückzuführen.



3.14a Simulation des Power-on Reset. – Verlauf der sich aufbauenden Versorgungsspannung (vdd!) und des Ausgangs des POR.



3.14b Verhalten des Power-on Resetslaut Datenblatt. – Oben ist der Verlauf der Versorgungsspannung und unten der Ausgang des POR zu sehen. Modifiziert aus [5].

Abbildung 3.14: Verhalten des Power-on Reset (simuliert 3.14a und nach Datenblatt 3.14b).

Analog-Digital-Umsetzer

Die Funktion des Analog-Digital-Umsetzer (ADC) wurde getestet, indem die Referenzspannungen (3.3 V) angeschlossen und ein Takt (1 MHz) angelegt wurde. Als Eingangsspannung dient beispielsweise eine Gleichspannung von 2.30 V. Nach elf Takten liefert der ADC 1011001001_{bin} bzw. $2C9_{\text{hex}}$ bzw. 713_{dec} als Ausgangswert, was nach Gleichung 3.4 einer Spannung von 2.30 V entspricht.

$$\frac{0\text{x}2\text{C}9}{0\text{x}3\text{FF}} \cdot V_{ref} = 0.969697 \cdot 3.3\,\text{V} = 2.3\,\text{V} \tag{3.4}$$

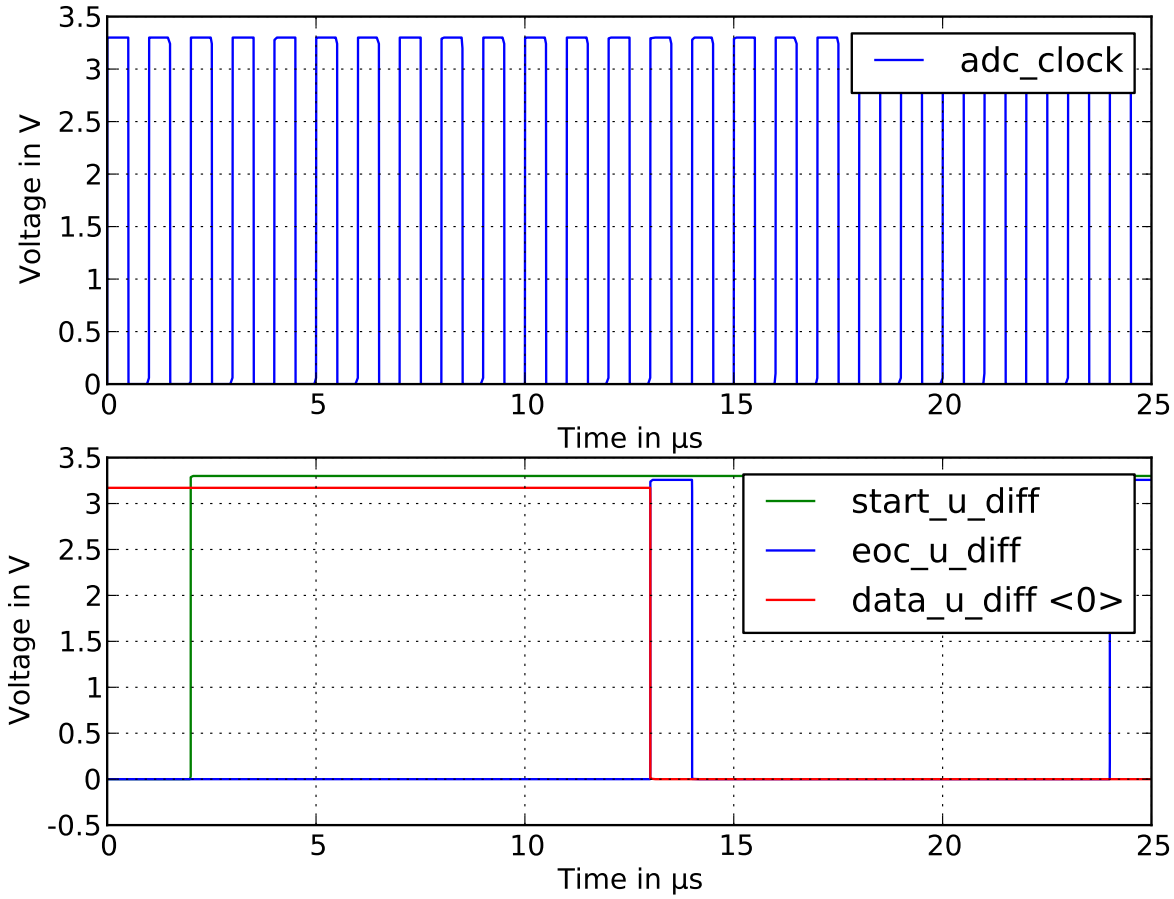


Abbildung 3.15: Simulation des Analog-Digital-Umsetzers. – Zu sehen sind die Steuersignale des ADC und dessen Takt. Das Signal start_u_diff startet die Umsetzung und das Signal eoc_u_diff signalisiert das Ende einer Umsetzung.

Tabelle 3.2: Ausgangssignale des ADC bei 2.3 V Eingangsspannung. – Diese Signal-Pegel liegen, nachdem das Signal eoc_u_diff von Low- auf High-Pegel gewechselt hat, an den Ausgängen des ADC an (siehe Abbildung 3.15). Dabei ist data_u_diff <0> das LSB.

Signalname	Pegel
data_u_diff <0>	High
data_u_diff <1>	Low
data_u_diff <2>	Low
data_u_diff <3>	High
data_u_diff <4>	Low
data_u_diff <5>	Low
data_u_diff <6>	High
data_u_diff <7>	High
data_u_diff <8>	Low
data_u_diff <9>	High

4 Realisierung: Synthese und Layout

4.1 Synthetisieren und Platzieren der digitalen Hardware

Wie im Anhang G exemplarisch beschrieben, wurde aus der in VHDL beschriebenen Logik eine Schaltung, bestehend aus Zellen der Standardbibliothek, synthetisiert.

Die Schaltung aus Zellen der Standardbibliothek wurden mit dem Programm Encounter RTL Compiler aus der Cadence-Toolchain in ein Layout überführt. Dabei platziert das Programm die Standardzellen in Reihen, wie in Abbildung 1.7 zu sehen. Hierbei kann Einfluss darauf genommen werden, wie viel Platz zwischen den Reihen und wie viel Freiraum innerhalb der Reihen vorhanden ist. Dieser Freiraum ist für die Leiterbahnführung zwischen den Ein- und Ausgängen der einzelnen Zellen notwendig. Die Flächenausnutzung der digitalen Module des CMOS-Testchips beträgt ungefähr 60 %. Auf Leerzeilen zwischen den Reihen, die ebenfalls für die Leiterbahnführung benutzt werden können, wurde verzichtet.

Versuche, die Schaltung des Signalverarbeitungsteils mit drei Metalllagen für die Leiterbahnführung zu erstellen, sind fehlgeschlagen. Der Grund dafür war, dass es der Autorouter-Algorithmus nicht geschafft hat, alle Signalleitungen zu verlegen, ohne gegen Design-Regeln zu verstoßen. Deshalb wurden vier Metalllagen für die Verdrahtung gewählt.

4.2 Gatelevel und Postlayout-Simulationen

4.2.1 Protokollgenerator

4.2.1.1 Gatelevel-Simulation

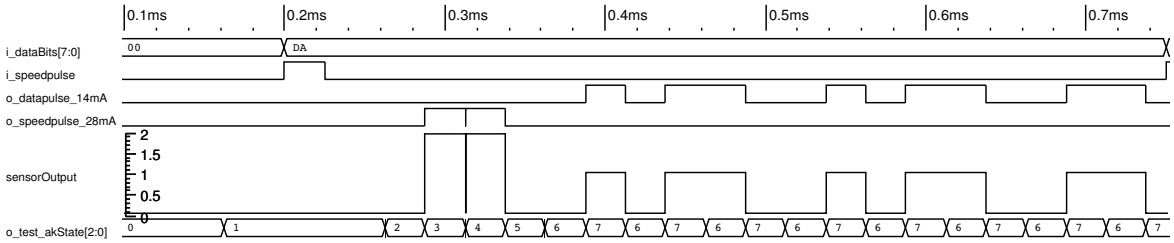
Bei dieser Simulation werden, wie oben erwähnt, die Gatterlaufzeiten der Zellen aus der Standardbibliothek berücksichtigt. Die Netzliste des Protokollgenerators liegt als Verilog-Datei vor. Als Testbench wird, die schon bei der RTL-Simulation verwendete, genutzt. Dies erlaubt einen direkten Vergleich der Ergebnisse.

Es fallen keine Unterschiede zu der RTL-Simulation auf, was unter anderem an der, im Vergleich zu den Gatterlaufzeiten, niedrigen Taktfrequenz der Logik liegt. Die Ergebnisse zeigen, dass die Logik korrekt synthetisiert wurde, da sich die Gatelevel-Simulation genauso wie die RTL-Simulation verhält.

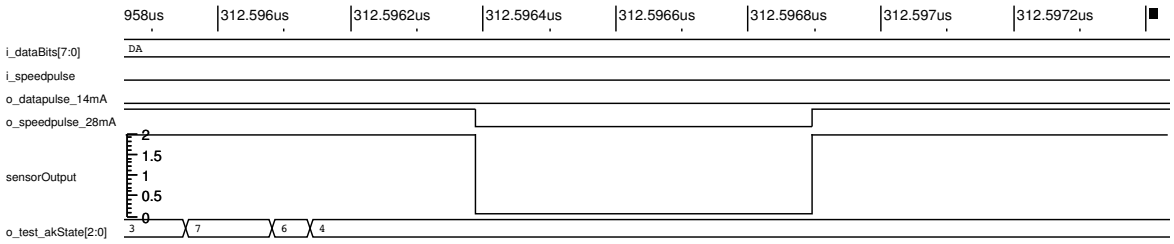
4.2.1.2 Postlayoutsimulation

Für diese Simulationen wurden die parasitären Effekte aus dem digitalem Layout extrahiert. Diese spiegeln sich bei dieser Simulation als Laufzeiten zwischen den einzelnen Gattern wieder. Diese Laufzeiten werden ebenfalls in einer SDF-Datei abgespeichert. Die Simulation wird auch wieder mit der gleichen Testbench, wie in den anderen beiden Simulationen durchgeführt.

Die Postlayoutsimulation verhält sich, bis auf kleine Details, wie die RTL Simulation. In der Abbildung 4.1b sieht man, dass für kurze Zeiten die Signale ihren Pegel wechseln. Vor allem in dem Ausgangssignal ist für 508 ps eine Pegeländerung zu sehen. Dieses Verhalten trat in der RTL Simulation nicht auf und ist mit Laufzeiten der kombinatorischen Ausgangslogik zu begründen.



4.1a Ausschnitt der Postlayoutsimulation des Protokollgenerators. In dem Signal o_speedpulse_28mA ist ein Einbruch des Pegels bei ungefähr 0.31 ms zu erkennen.



4.1b Ausschnitt der Postlayoutsimulation auf den Einbruch des Signals o_speedpulse_28mA vergrößert. Zu erkennen ist, dass der Einbruch 508 ps andauert.

Abbildung 4.1: Postlayoutsimulation des Protokollgenerators. – In 4.1a ist die Ausgabe von acht Bit im Manchestercode dargestellt. 4.1b zeigt den Einbruch des Ausgangssignals o_speedpulse_28mA.

4.2.2 Digitale Signalverarbeitung

Bei der Postlayoutsimulation des digitalen Signalverabreitungs-Moduls sind keine Unterschiede zwischen der Postlayout-, der Gatelevel- und der RTL-Simulation zu erkennen. Bei diesem Modul wurden allerdings die einzelnen Signale nicht so detailliert untersucht wie bei dem Protokollgenerator. Es wurden als Funktionsnachweis die HDI- und HD5-Werte betrachtet. In Abbildung 4.2 ist ein Ausschnitt der Postlayout-Simulation zu sehen.

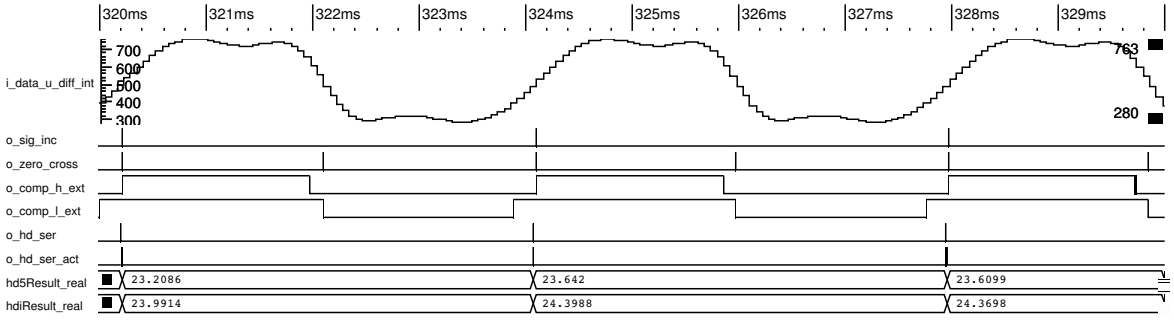


Abbildung 4.2: Postlayoutsimulation der digitalen Signalverarbeitung. – Das Signal i_data_u_diff_int zeigt das Brückensignal des Sensors. Die daraus berechneten HD5- und HDI-Werte werden über die Signale o_hd_ser und o_hd_ser_act ausgegeben und mit den Signalen hd5Result_real hdiResult_real visualisiert.

4.3 Layout der analogen Hardware

Bei dem Erstellen des Layouts wurde nicht auf einen Autorouter zurückgegriffen. Alle Komponenten wurden manuell platziert und geroutet. Die Komponenten wurden, wie in Abbildung 4.3 gezeigt, platziert. Dabei wurden die analogen Ein- und Ausgänge an der rechten und unteren Seite des Layouts platziert. So können die analogen Signale direkt aus dem CMOS-Testchip geführt werden. Die Breite der Leiterbahnen entsprechen in diesem Layout der Leiterbahndicke der einzelnen Komponenten. Jede Komponente wurde mit einem Ring der Versorgungsspannung umgeben, um diese möglichst gut anschließen zu können.

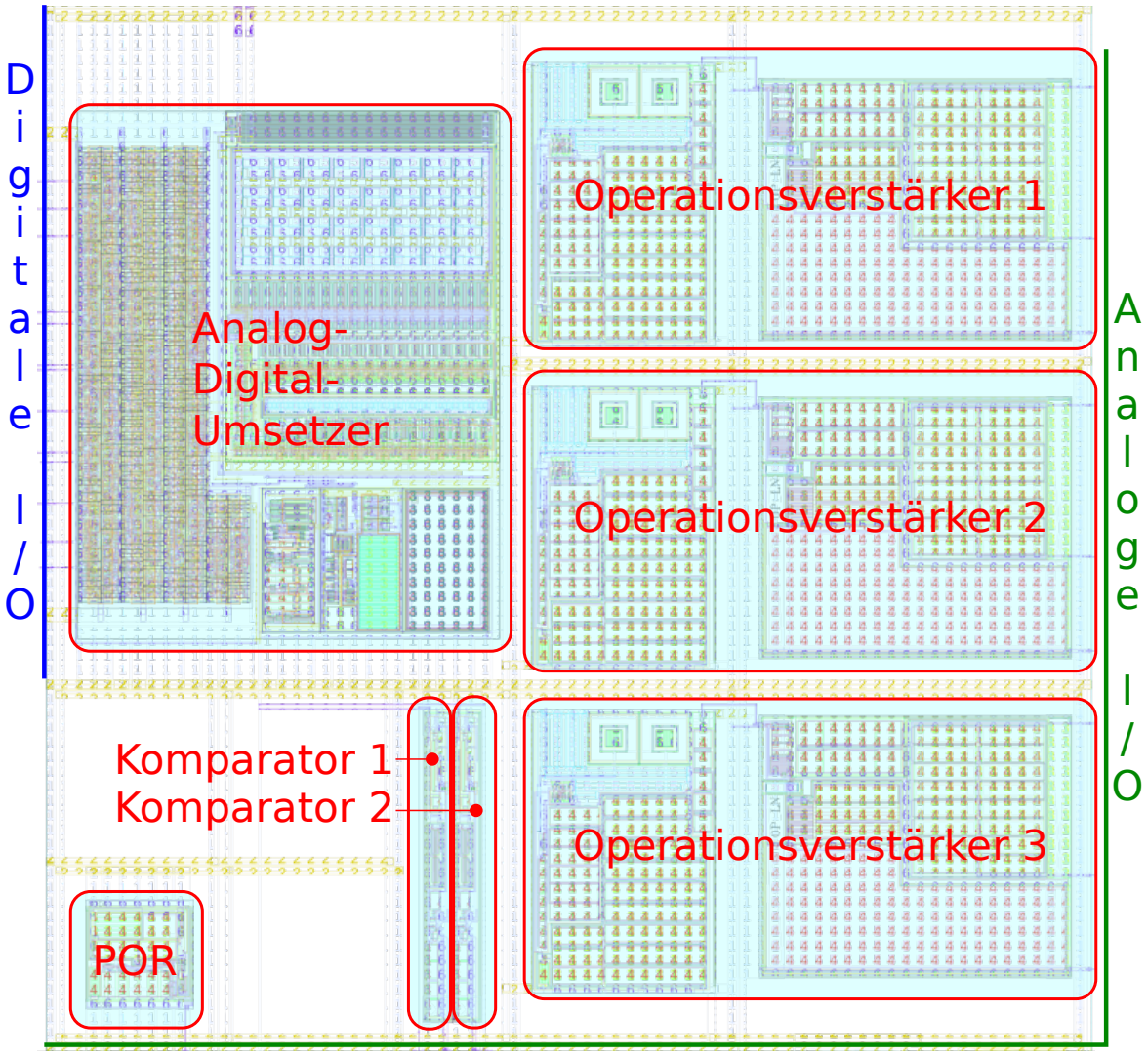


Abbildung 4.3: Positionierung der analogen Komponenten im Layout. – Die Zellen wurden so platziert, dass am rechten Rand die digitalen Aus- und Eingänge des ADC liegen. Die analogen Anschlüsse wurden an den linken und unteren Rand gelegt.

Simulation des Layouts

Das Layout wurde mithilfe der gleichen Testschaltung simuliert wie die Schematic. Im Verhalten sind keine Unterschiede zu erkennen. Wie bereits erwähnt, wurde die Funktionalität nur grob geprüft.

4.4 Erstellen des Pinouts

Die aneinandergereihten Padzellen bilden einen ESD-Schutz-Ring[1], durch den die Versorgungsspannung und Masse geschliffen wird. Um das analoge Modul mit einer anderen Versorgungsspannung als die digitalen Module zu betreiben, muss dieser Ring aufgebrochen werden. Zu diesem Zweck existieren in der Standardbibliothek Trennzellen, die den Ring auftrennen, sodass der ESD-Schutz möglichst erhalten bleibt.

Die, in dem CMOS-Testchip verbauten Pads sind mit einer kurzen Beschreibung in Tabelle 4.1 zu finden. Für die analogen Pins wurden Pads mit einem Eingangswiderstand von 200 Ω gewählt. Es sind die Pads mit dem niedrigsten Serienwiderstand, die einen ESD-Schutz bieten [7]. Für die meisten digitalen Eingänge wurden Standard-CMOS-Eingangspuffer gewählt. Optional zu setzende Eingänge, wie z. B. die Steuerleitungen des Multiplexers, wurden über Pads mit Pull-Up bzw. Pull-Down Widerständen aus dem Chip geführt. Damit ist der CMOS-Testchip auch ohne Beschaltung der optionalen Eingänge funktionsfähig. Die Eingänge der Takte sind mit speziellen Pads realisiert worden. Pads für die Versorgungsspannungen bedienen sowohl den Kern des Chips, als auch den ESD-Schutzring. Als Gehäuse wurde ein Ceramic Leaded Chip Carrier (CLCC) mit 84 Pins vorgesehen. Abbildung A.1 in Anhang A zeigt das komplette Pinout des CMOS-Testchips.

4.5 Erstellen des Gesamtlayouts

In Abbildung 4.4 ist das komplette Layout mit markierten Modulen des CMOS-Testchips zu sehen. Um die digitalen Ein- und Ausgänge möglichst einfach von den analogen Ein- und Ausgängen zu trennen, wurde die analoge Signalverarbeitung in eine Ecke gelegt (unten rechts). Da die analoge Signalverarbeitung über digitale Signale mit der digitalen Signalverarbeitung kommuniziert, wurde diese direkt darüber platziert (der Multiplexer ist Teil der digitalen Signalverarbeitung). Der Protokollgenerator kann autark betrieben werden und wurde deshalb zuletzt in die freie Ecke platziert.

[1] **ESD** Elektrostatische Entladung (electrostatic discharge)

Tabelle 4.1: Im CMOS-Testchip verwendete Pads.

Typ	Beschreibung
Digitale Eingänge:	
ICP	CMOS Eingangspuffer
ICDP	CMOS Eingangspuffer mit Pull-Down-Widerstand
ICUP	CMOS Eingangspuffer mit Pull-Up-Widerstand
ICCK4P	CMOS Takt Eingangspuffer (4 mA)
Digitale Ausgänge:	
BU2P	Ausgangspuffer (2 mA)
Digital Bidirektional:	
BBC4P	Bidirektionaler CMOS Puffer (4 mA)
Analog:	
APRIO200P	Analoger Ein- / Ausgang mit 200 Ω Serienwiderstand
Versorgung:	
VDD3ALLP	Versorgungsspannung Pad (Versorgt die Ausgangspuffer, den Kern und die Peripherie; 3.3 V)
GND3ALLP	Masse Pad (Versorgt die Ausgangspuffer, den Kern und die Peripherie)

Wie zu erkennen, ist die Fläche des CMOS-Testchips zu 72.83 % belegt. Das liegt daran, dass viele Pins zum Test und Debugging benötigt wurden. Da die Pads definierte Abmaße besitzen, leitet sich daraus die Chipfläche ab (padbestimmtes Design).

An den Ecken sind Zellen platziert, die den ESD-Ring verbinden. Da an zwei Seiten die Trennzellen eingefügt wurden, um die analogen von den digitalen Pads zu trennen, mussten auf den gegenüberliegenden Seiten Füllzellen eingefügt werden, um die Länge auszugleichen, damit der ESD-Ring geschlossen ist.

4.6 Design Rule Check

Nachdem das Layout fertiggestellt war, wurde der Design Rule Check (DRC) durchgeführt. Dabei traten diverse Fehler auf. Einige dieser Fehler konnten ignoriert, andere manuell behoben werden. Eine Auswahl der DRC-Fehler werden hier kurz erläutert.

Viele der DRC-Fehler traten in den Zellen der Standardbibliothek auf, was daran liegt, dass diese Zellen stark optimiert sind. Diese Fehler können ignoriert werden [46].

Nach dem Routen traten beim DRC insgesamt 47 Fehler auf, die dem Antenna Effect zuzuordnen sind. Wird ein Gate an eine große Fläche der unteren Metalllagen angebunden, kann es beim Ätzen dieser Metalllagen dazu kommen, dass das Metall wie eine „Antenne“ wirkt und Ionen aufnimmt, die das Potential erhöhen. Dabei ist es möglich, dass die Gatespannung so groß wird, dass das Gateoxid irreparabel beschädigt wird (Abbildung 4.5a). Um diesen Effekt zu verhindern, ist es notwendig kurz vor dem Gate auf die höchste Metalllage zu wechseln, um die an das Gate angebundene Fläche der unteren Metalllage zu verringern (siehe Abbildung 4.5b) [35, Seite 634].

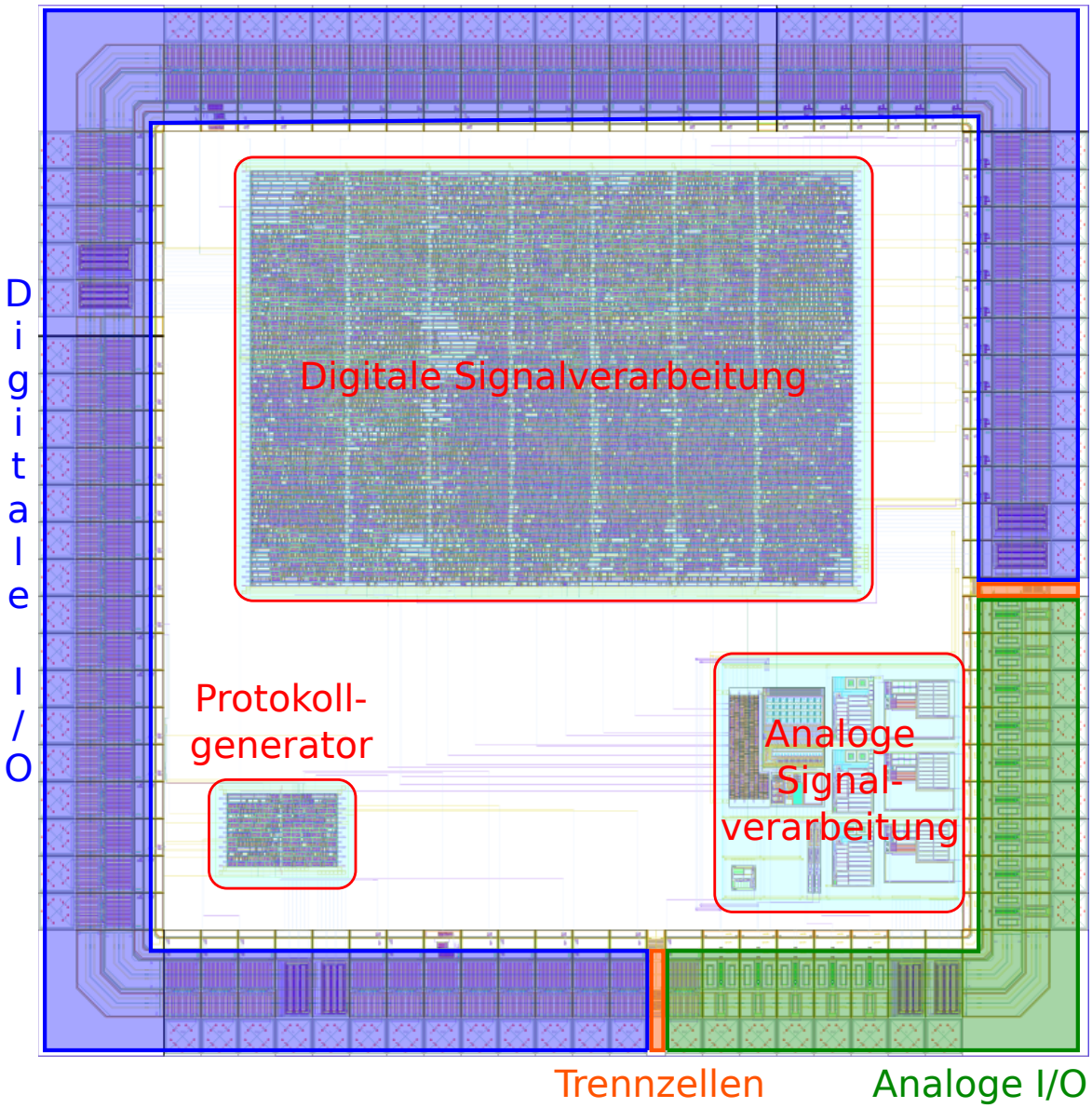
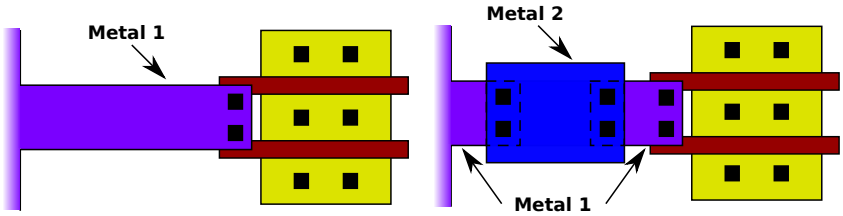


Abbildung 4.4: Layout des gesamten CMOS-Testchips. – Die Module der analogen Signalverarbeitung wurden mit allen analogen Anschlüssen in die untere linke Ecke platziert. Die restlichen Anschlüsse sind digitale Ein- und Ausgänge.



4.5a Das Gate kann bei dem Ätzvorgang irreparabel beschädigt werden.

4.5b Durch das Wechseln des Gateanschlusses auf eine höhere Metallage kann das Zerstören des Gates vermieden werden.

Abbildung 4.5: Antenna Effect. – Gates, an denen eine große Fläche der unteren Metalllagen angeschlossen ist, können während des Ätzens irreparabel beschädigt werden (4.5a). Ein Weg diesen Effekt zu verhindern ist, kurz vor dem Gate auf eine höher gelegene Metalllage zu wechseln, um die an das Gate angebundene Fläche der ersten Metalllage zu verringern (4.5b). Abbildung modifiziert aus [35, Seite 634].

5 Testen des produzierten Chips

5.1 Testplan

Im Rahmen dieser Arbeit wurden lediglich kurze Tests durchgeführt. Um den vollen Funktionsumfang zu testen, fehlte leider die hierfür notwendige Zeit. Im Projekt ESZ-ABS wird der Chip mittels einer Testplatine später genauer untersucht. In diesem Kapitel wird das Vorgehen beim Testen des CMOS-Testchips beschrieben. Dabei orientiert sich das Testen an den Simulationen.

Grob gliedert sich das Vorgehen dabei in zwei Schritte: Zunächst folgt ein Minimaltest, um zu überprüfen, ob der Chip überhaupt eine Funktion zeigt. Es könnte durch Design- oder Produktionsfehler vorkommen, dass ein Exemplar oder gar alle Chips nicht funktionsfähig sind. Anschließend können die einzelnen Module genauer untersucht werden.

Das Vorgehen des Minimaltests ist in Anhang F zu finden. Dabei werden zunächst die Komponenten des Protokollgenerators überprüft, anschließend die analogen Komponenten. Die Überprüfung der digitalen Signalverarbeitung setzt bei diesem Vorgehen eine funktionsfähige analoge Hardware voraus. Als Plattform für diesen Minimaltest dient eine Platine mit einem PLCC84-Sockel bei der alle Pins an 2 mm-Federstecker geführt sind. Hiermit kann eine Verdrahtung über Kabel erfolgen, was für erste Funktionsüberprüfungen ausreicht. Wurde die Funktion der analogen Komponenten nachgewiesen, können diese mit der digitalen Signalverarbeitung verbunden werden und mithilfe eines Funktionsgenerators eine Spannung eingespeist werden. Mit einem Oszilloskop (Logikanalyser) kann die Ausgabe (HDI und HD5) überprüft werden.

5.2 Erstinbetriebnahme

Für die erste Inbetriebnahme wurde der CMOS-Testchip in eine Platine mit einem PLCC84-Sockel verbaut. Diese Platine ist für jeden Pin des CMOS-Testchips mit Buchsen für 2 mm-Federstecker ausgestattet. Zuerst wurde lediglich die Spannungsversorgung der digitalen Logik angeschlossen. Dabei ging das Labornetzteil sofort in die, auf 1 mA eingestellte, Strombegrenzung. Nachdem die Strombegrenzung schrittweise bis auf 10.01 mA erhöht wurde, stieg die Spannung trotzdem nicht über 0.75 V. Dieses Fehlverhalten ähnelt sehr stark dem Verhalten einer Diode, was auf einen Kurzschluss hindeutet. Ein weiteres Exemplar des CMOS-Testchips zeigte das gleiche Verhalten. Die Stromaufnahme der analogen

Tabelle 5.1: Stromaufnahme des CMOS-Testchips bei erster Inbetriebnahme. – Die Werte zeigen das Verhalten einer Diode, was auf einen Kurzschluss der Versorgungsspannung zurückzuführen ist. Alle Werte wurden mit einem FLUKE 45 Dual Display Multimeter aufgenommen.

Strom	Spannung
1.03 mA	0.40 V
5.02 mA	0.70 V
6.04 mA	0.71 V
10.01 mA	0.75 V

Signalverarbeitung lag in dem erwarteten Bereich. Nach Analyse des Layouts wurde festgestellt, dass die Spannungsversorgung des Protokollgenerators verpolt angeschlossen wurde. Somit konnten anfangs nur die Operationsverstärker, die Komparatoren und der POR getestet werden, da sie die Versorgungsspannung der digitalen Logik nicht benötigen.

Da es nicht ohne weiteres möglich ist, Leitungen auf einem Chip zu verändern, war es nur Dank günstigen Umständen und gutem Kontakt des ESZ-ABS-Projekts zu der Halbleiterindustrie möglich, den CMOS-Testchip trotzdem komplett in Betrieb zu nehmen. Die dazu verwendeten Methoden sind keinesfalls das Standardvorgehen in der Chipherstellung.

Um die digitale Signalverarbeitung testen zu können, wurde die Spannungsversorgung des Protokollgenerators mithilfe eines Lasers durchtrennt. Dies war möglich, da die Spannungsversorgung über die oberste Metalllage angeschlossen ist und sich keine notwendigen Strukturen darunter befinden. In Abbildung 5.1 ist ein Ausschnitt des Layouts zu sehen, an dem die Versorgungsspannung durchtrennt wurde. Um den Protokollgenerator testen zu können, mussten die Anschlüsse der Versorgungsspannung vertauscht werden. Dazu wurde ein Focused Ion Beam (FIB) eingesetzt. Mit dem FIB ist es möglich, Oberflächenschichten abzutragen und sogar nachträglich neue Leiterbahnen auf dem Chip zu erzeugen. Das Ergebnis des FIB ist in Abbildung 5.2 zu sehen. Dabei sind die Leiterbahnen an den Stellen, an denen sie durchtrennt wurden, rot markiert. Die grün markierten Leiterbahnen wurden neu hinzugefügt.

5.3 Weiterführende Tests

Um den CMOS-Testchip ausgiebig zu testen, wird zur Zeit eine Testplatine entwickelt. Diese Platine soll an ein Nexys2-Board[1] (auf dem ein Spartan-3E FPGA von Xilinx verbaut ist) angeschlossen werden. Über das auf dem Nexys2-Board verbaute FPGA können die

[1] FPGA Development Kit von Digilent.

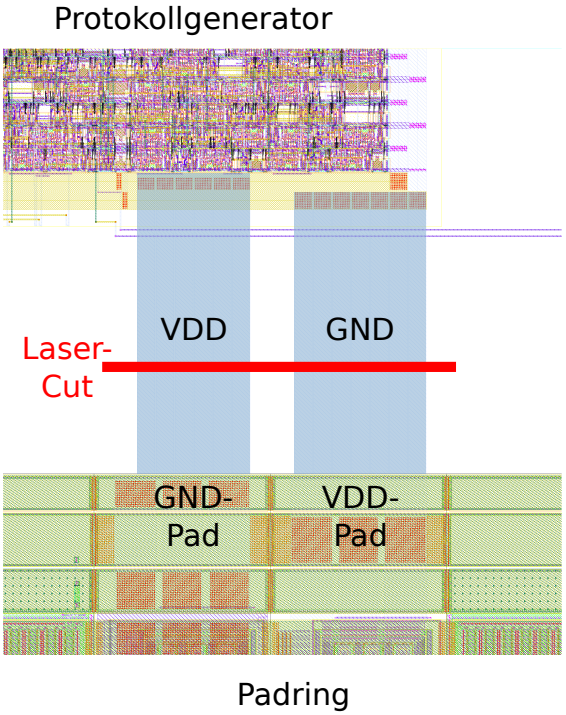


Abbildung 5.1: Platzierung des Laserschnittes auf dem CMOS-Testchip. – Besonders günstige Umstände machten es möglich die Spannungsversorgung des Protokollgenerators mittels eines Lasers durchtrennen zu lassen. So konnten die anderen Module des CMOS-Testchips getestet werden. Die rote Linie zeigt die Position, an der die Spannungsversorgung durchtrennt wurde.

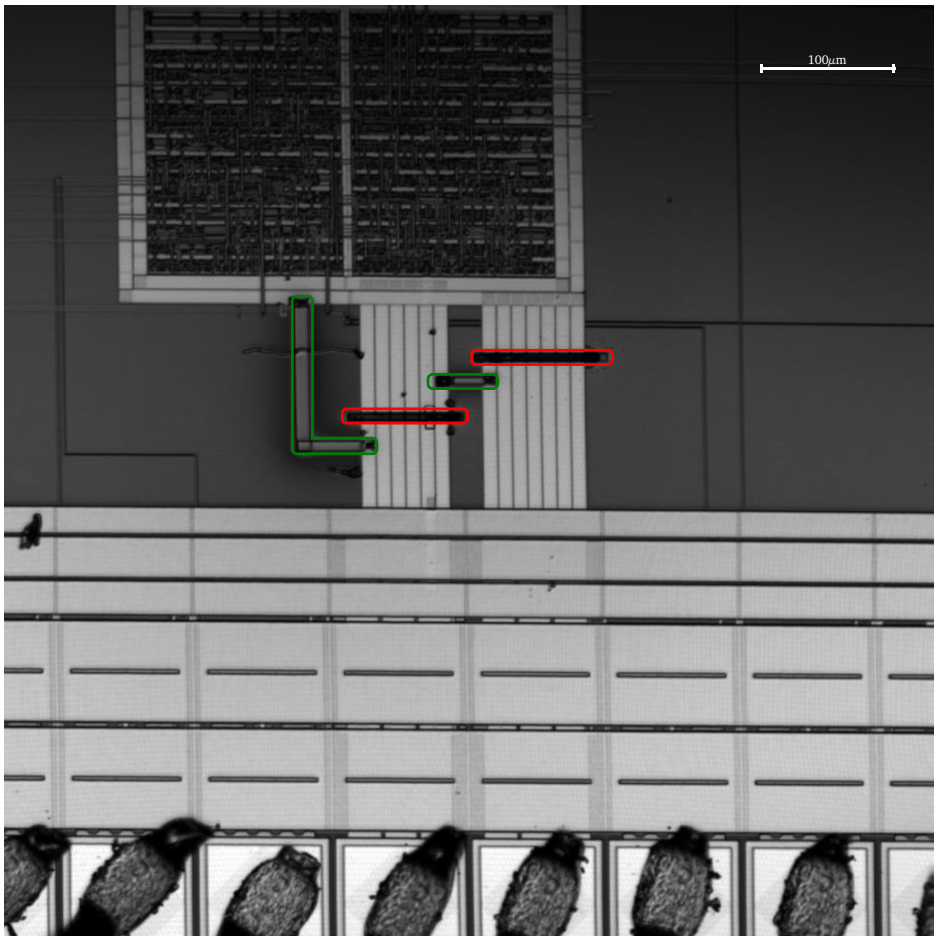


Abbildung 5.2: Ausschnitt des CMOS-Testchips nach der Bearbeitung mittels FIB. – Die Leiterbahnen wurden an den rot markierten Stellen durchtrennt. Die grün markierten Leiterbahnen wurden neu hinzugefügt. Diese Bearbeitung ermöglichte das Testen des Protokollgenerators.

Eingangssignale des CMOS-Testchips gesetzt und dessen Ausgabe analysiert bzw. an einen Computer gesendet und dort analysiert werden.

Der Protokollgenerator kann getestet werden, indem die Eingangssignale auf einen definierten Wert gelegt und die Ausgangssignale mit einem Oszilloskop oder im FPGA verglichen werden. Als Grundlage für diesen Test kann die Testbench des Protokollgenerators dienen. Steuert man mit den beiden Ausgängen des Protokollgenerators zwei Stromquellen an, so kann das eigentliche AK-Protokoll erzeugt und ausgewertet werden.

Über den Multiplexer können die Module der analogen und digitalen Signalverarbeitung unabhängig voneinander getestet werden. Allerdings ist diese Aussage nur bedingt korrekt, da der Multiplexer in dem Modul der digitalen Signalverarbeitung integriert ist. Zumindest können beim Erproben der digitalen Signalverarbeitung Einflüsse der analogen Signalverarbeitung wie z. B. Quantisierungsfehler und das Rauschen des Analog-Digital-Umsetzers ausgeschlossen werden. Hierzu können digitale Werte direkt von dem FPGA an den CMOS-Testchip gesendet werden. Dafür müssten der ADC und die Komparatoren simuliert werden. Die Simulatoren wurden bereits für die Testbench geschrieben und können als Grundlage verwendet werden. Die Ergebnisse (HDI und HD5) könnten seriell aus dem CMOS-Testchip ausgelesen werden und an einen Computer gesendet werden. Auf diesem können die Daten des CMOS-Testchips mit Werten verglichen werden, die auf dem Computer mit hoher Präzision berechnet wurden.

5.4 Ergebnisse

Bei den ersten Tests muss berücksichtigt werden, dass der Versuchsaufbau mit relativ langen Leitungen aufgebaut wurde. Des Weiteren teilen sich alle Module eine Spannungsversorgung, die auch gleichzeitig als Referenzspannung für den ADC dient. Alle Schwellenwerte wurden statisch mit veränderbaren Widerständen aufgebaut und manuell eingestellt. Für erste Versuche ist dieser Testaufbau ausreichend. Für spätere Tests wird allerdings ein Aufbau auf einer Platine empfohlen.

5.4.1 Protokollgenerator

Der Protokollgenerator verhält sich, nach der Korrektur mittels FIB, wie erwartet. Alle Komponenten wurden, wie in Anhang F beschrieben getestet. Allerdings stand zu dem Zeitpunkt des Tests kein Funktionsgenerator zur Verfügung, der ein 16 MHz Taktsignal generieren konnte, deshalb wurde ein Systemtakt von 12 MHz eingestellt.

Abbildung 5.3a zeigt die Ausgabe eines vollen Protokolls. In Grün ist der eingehende Speedpulse zu sehen. Danach startet die Ausgabe des Protokolls (rot und blau). Abbildung 5.3b

zeigt den Ruhepuls. Wie zu erkennen ist, liegen die Ruhepulse 200 ms statt den geforderten 150 ms auseinander. Dieses Verhalten ist durch die niedrigere Taktfrequenz zu erklären: $200\,\text{ms} \cdot \frac{12\,\text{MHz}}{16\,\text{MHz}} = 200\,\text{ms} \cdot 0.75 = 150\,\text{ms}$.

Das, in der Postlayout-Simulation aufgetretene Verhalten des Ausgangssignals o_speedpulse_28mA, ist mit einem Oszilloskop an der Hardware nicht mehr zu messen. Dies könnte an dem Tiefpassverhalten des Ausgangspads und der nachfolgenden Leitung liegen.

5.4.2 Analoge Signalverarbeitung

Operationsverstärker

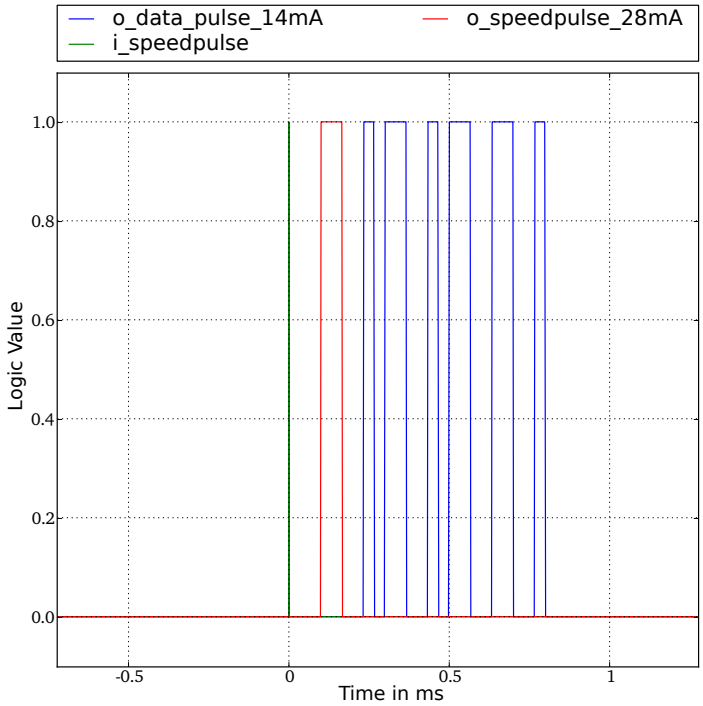
Die Operationsverstärker wurden, wie in der Simulation, als nichtinvertierender Verstärker aufgebaut (siehe Abbildung 5.4). Der Serienwiderstand im Pad des Ausgangspins ($R_{Pad} = 200\,\Omega$) hat einen geringen Einfluss auf den Verstärkungsfaktor. Damit ergibt sich eine theoretische Verstärkung mit einem Verstärkungsfaktor von $v = 1 + \frac{20\,\text{k}\Omega + 200\,\Omega}{100\,\text{k}\Omega} = 1.2002$. Die Widerstände der Eingangspins sind aufgrund des hohen Eingangswiderstandes der Operationsverstärker zu vernachlässigen. In Abbildung 5.5 ist exemplarisch die Ein- und Ausgangsspannung des OP2 zu sehen. Der gemessene Verstärkungsfaktor (über alle Messpunkte gemittelt) liegt bei: $v = 1.20068$.

Komparatoren

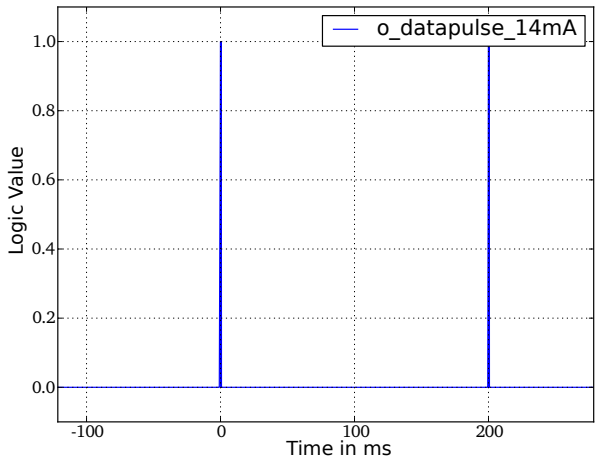
Die Schwellen der Komparatoren wurden auf comp_l_thres $= 1.45\,\text{V}$ und auf comp_h_thres $= 1.87\,\text{V}$ eingestellt. An den Eingang wurde eine Sinusspannung angelegt. Beim Überschreiten der Schwellspannungen wechseln die Ausgangspins der Komparatoren wie erwartet auf High-Pegel (siehe Abbildung 5.6).

Power-on Reset

Das Verhalten des POR (siehe Abbildung 5.7) deckt sich mit dem in dem Datenblatt (Abbildung 3.14b) beschriebenen.



5.3a Ausgabe des Protokollgenerators. Die Ausgabe des Protokolls wird hier mit einem eingehenden Speedpulse i_speedpulse gestartet. Nachdem der ausgehende Speedpulse gesendet wurde (o_speedpulse_28mA), folgen die acht Daten- und ein Paritätsbit (o_datapulse_14mA).



5.3b Wird 150 ms lang von dem Protokollgenerator kein eingehender Speedpulse registriert, so wird das Ruheprotokoll gesendet. In diesem Fall wird das Ruheprotokoll erst nach 200 ms gesendet, da der CMOS-Testchip mit 12 MHz anstatt den 16 MHz betrieben wurde, für die das System ausgelegt ist.

Abbildung 5.3: Test des Protokollgenerators. – 5.3a: Ausgabe des Speedpulses und acht Daten-, sowie Paritätsbit. 5.3b: Ausgabe des Ruheprotokolls.

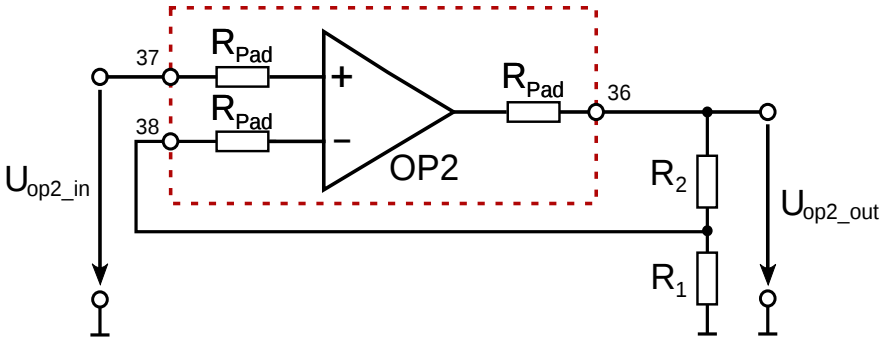


Abbildung 5.4: Testaufbau eines Operationsverstärkers des CMOS-Testchips. – Der Operationsverstärker (OP2) befindet sich ebenso wie die Serienwiderstände R_{Pad} auf dem CMOS-Testchip. Hier ist der Operationsverstärker als nichtinvertierender Verstärker aufgebaut. Der Verstärkungsfaktor beträgt theoretisch: $v = 1 + \frac{R_2 + R_{Pad}}{R_1} = 1.2002$ mit $R_{Pad} = 200\,\Omega$, $R_1 = 100\,k\Omega$ und $R_2 = 20\,k\Omega$.

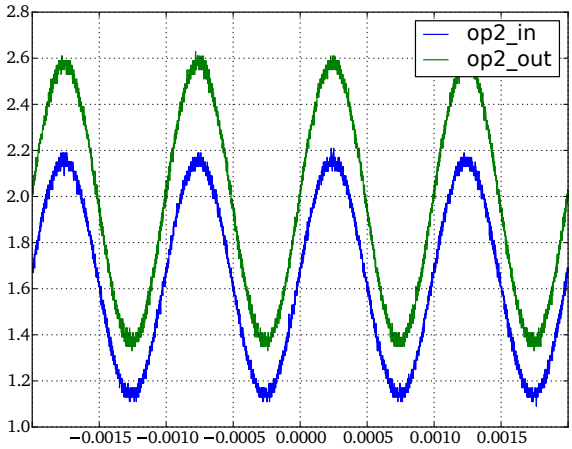


Abbildung 5.5: Interner Operationsverstärker als nichtinvertierender Verstärker. – Hier ist der Operationsverstärker 2 (OP2) als nichtinvertierender Verstärker aufgebaut (Verstärkungsfaktor $v = 1 + \frac{20\,k\Omega + 200\,\Omega}{100\,k\Omega} = 1.2002$). Der aus den gemessenen Spannungen ermittelte Verstärkungsfaktor beträgt, gemittelt über alle Messwerte, $v_{real} = \frac{U_{out}}{U_{in}} = 1.20068$.

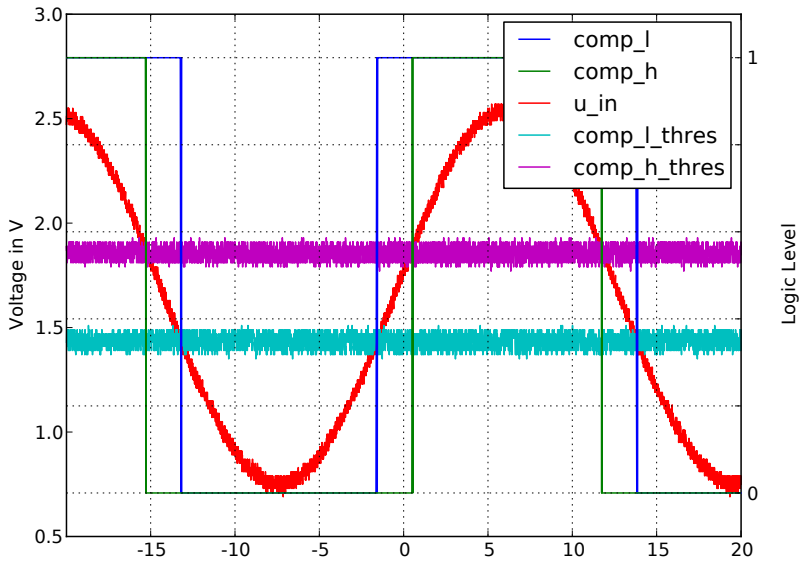


Abbildung 5.6: Test der Komparatoren. – Die Signale comp_l und comp_h sind digital. Ihre Pegel können an der rechten Skala abgelesen werden.

Analog-Digital-Umsetzer

Bei einer Eingangsspannung von 2.30 V lieferte der ADC einen digitalen Wert von $2BC_{16}$. Ein Vergleich mit den erwarteten, idealen Werten ist in Tabelle 5.2 zu sehen. Wie zu erkennen ist, weicht schon das vierte Bit (2^6) von dem erwarteten Ergebnis ab. Die hohe Abweichung von dem theoretisch erwarteten Wert liegt zum einen daran, dass die Referenzspannung an derselben Spannungsquelle wie die analoge und digitale Versorgungsspannung angeschlossen ist. Zum anderen diente als Versuchsaufbau nur ein Sockel mit 2 mm-Federstecker.

5.4.3 Digitale Signalverarbeitung

Die digitale Signalverarbeitung wurde in Kombination mit der analogen Signalverarbeitung getestet. Dazu wurden verschiedene Spannungen mit einem Signalgenerator erzeugt. Der serielle HDI und HD5 wurde mit einem Oszilloskop aufgenommen und auf einem Computer ausgewertet. Dazu wurde das Oszilloskop über die Ethernet-Schnittstelle ausgelesen und die seriellen Daten, mit dem Python-Script aus Anhang D, in Prozent umgerechnet und ausgegeben. In Tabelle 5.3 sind die aufgenommenen HDI- und HD5-Werte aufgelistet, mit denen der CMOS-Testchip getestet wurde. Als Vergleichswert ist ein theoretisch

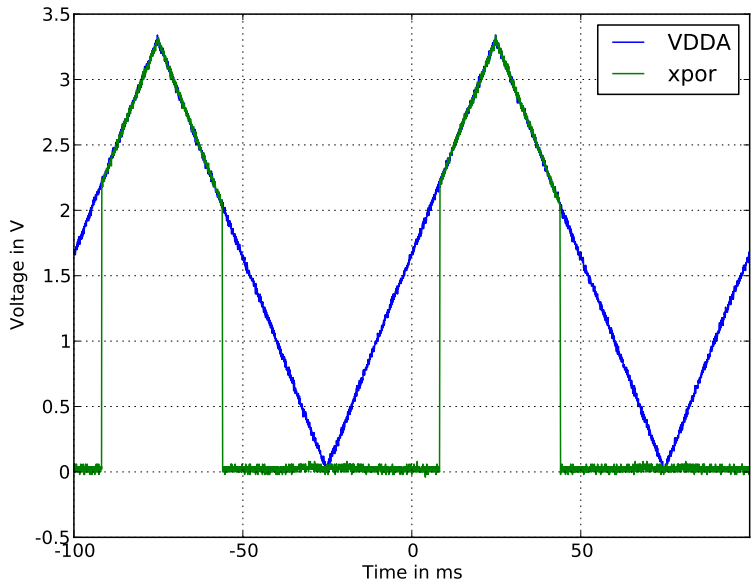


Abbildung 5.7: Test des Power-on Reset. An die Spannungsversorgung wurde eine Dreieckspannung angelegt (VDDA: blau). Der Ausgang des POR ist in grün dargestellt (xpor).

Tabelle 5.2: Test des ADC – Vergleich der idealen und gemessenen Ausgabebits. – Schon das vierte Bit (2^6) weicht von dem erwarteten Ergebnis ab. Die große Abweichung lässt sich unter anderem dadurch erklären, dass die Versorgungsspannungsquelle auch als Referenzspannung für den ADC genutzt wurde.

	Binärer Ausgangswert									
	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
ideal	1	0	1	1	0	0	1	0	0	1
gemessen	1	0	1	0	1	1	1	1	0	0

Tabelle 5.3: Vom CMOS-Testchip berechnete HDI und HD5 Werte.

Eingespeistes Signal			HD5	HDI	Theoretischer HD5
Signalform	Frequenz	Spitzen-spannung			
Sinus	37 Hz	0.90 V	0.404 %	4.697 %	0.000 %
	1037 Hz	0.50 V	5.940 %	8.408 %	0.000 %
	1037 Hz	0.25 V	2.014 %	11.612 %	0.000 %
Dreieck	37 Hz	0.90 V	11.864 %	13.268 %	11.728 %
	1037 Hz	0.50 V	12.404 %	14.755 %	11.728 %
	1037 Hz	0.25 V	17.136 %	19.894 %	11.728 %

berechneter HD5-Wert angegeben. Er berechnet sich aus Gleichung 1.2. Die Abweichungen der gemessenen Werte von den Vergleichswerten kommen unter anderem durch den Versuchsaufbau zustande und sollten später mithilfe der Testplatine überprüft werden. Die hohen Abweichungen bei einer Spitzenspannung von $\hat{u} = 0.25\,\mathrm{V}$ sind außerdem dadurch zu begründen, dass der ADC nicht vollständig ausgesteuert ist. Dies könnte mit einer analogen Regelverstärkung, wie sie in [24, 32, 40] bereits eingesetzt wurde, verbessert werden. Dennoch konnte gezeigt werden, dass die digitale Signalverarbeitung im Zusammenspiel mit der analogen Signalverarbeitung funktionsfähig ist.

6 Bewertung und Fazit

6.1 Chip-Flächenbedarf

Die Gesamtfläche des Chips beträgt $2831.4\,\mu m \times 2831.4\,\mu m = 8.016\,83\,mm^2$. Die Verteilung der Fläche auf die einzelnen Module ist in Abbildung 6.1 und Tabelle 6.1 aufgeschlüsselt. Die Leiterbahnen, die außerhalb der Module liegen, wurden für diese Betrachtung vernachlässigt. Wie deutlich zu erkennen ist, dominiert der Padring die Fläche mit 42.25 % und 27.17 % der Chipfläche sind ungenutzt. Das hängt beides damit zusammen, dass viele Pads für das Debugging benötigt wurden und somit die Anzahl der Pads die Chipfläche bestimmt.

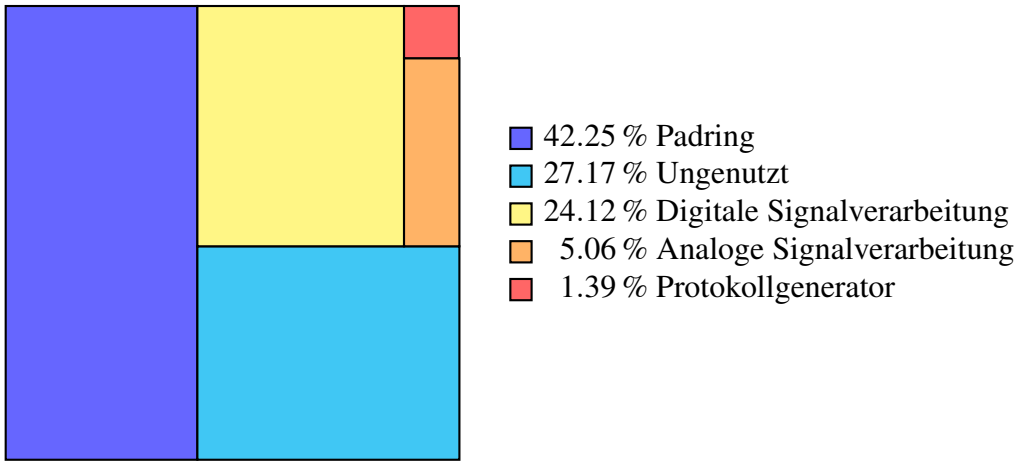


Abbildung 6.1: Prozentuale Verteilung der gesamten Chipfläche auf die einzelnen Module. – Der absolute Flächenbedarf ist in Tabelle 6.1 zu sehen. Die Leiterbahnen, die außerhalb der Module liegen wurden für diese Betrachtung vernachlässigt.

Eine Aussage über den Flächenbedarf der kompletten analogen Signalverarbeitung kann nicht getroffen werden, da nur einzelne Bauelemente implementiert wurden. Die folgenden Betrachtungen des Flächenbedarfs beschränken sich auf die von dem Synthesewerkzeug berechnete Fläche.

Der Protokollgenerator belegt lediglich 1.39 % der Chipfläche (siehe Abbildung 6.1). Damit bietet dieses Modul wenig Potential, um die gesamte Chipfläche durch Optimierung

Tabelle 6.1: Verteilung der gesamten Chipfläche auf die einzelnen Module. – Die Leiterbahnen, die außerhalb der Module liegen, wurden für diese Betrachtung vernachlässigt.

Modul	Fläche	Relative Fläche
Padring	3 387 443.30 µm^2	42.25 %
Ungenutzt	2 178 397.66 µm^2	27.17 %
Digitale Signalverarbeitung	1 933 761.60 µm^2	24.12 %
Analoge Signalverarbeitung	405 769.00 µm^2	5.06 %
Protokollgenerator	111 454.40 µm^2	1.39 %
Summe	8 016 825.96 µm^2	100.00 %

zu verkleinern. Insgesamt wurden 276 Zellen aus der Standardbibliothek für den Protokollgenerator implementiert. Davon sind 83 Zellen Flipflops, die 64.29 % der Fläche des Protokollgenerators ausmachen. Die restlichen 193 Zellen, kombinatorische Logik, belegen nur 35.71 % der Gesamtfläche des Protokollgenerators.

In Abbildung 6.2 (Tabelle 6.2) ist der Flächenbedarf der einzelnen Komponenten der digitalen Signalverarbeitung aufgeführt. Weit über die Hälfte der Fläche der digitalen Signalverarbeitung, wird für die Berechnung der HD5- und HDI-Werte benötigt. Die DFT-Implementation besteht aus mehreren Komponenten, die zusammen 23 % der Fläche der digitalen Signalverarbeitung belegen. Abbildung 6.3 (Tabelle 6.3) zeigt, dass der Großteil der Fläche der digitalen Signalverarbeitung für sequenzielle Logik, also für Flipflops benötigt wird. Die Anzahl der verwendeten Flipflops bietet also einen guten Ansatz, um die Chipfläche zu reduzieren. Des Weiteren bietet die Berechnung der HD5- und HDI- Werte einen Ansatzpunkt, um die Chipfläche zu reduzieren, da dieser Teil 68.99 % der Fläche der digitalen Signalverarbeitung einnimmt.

Der CMOS-Testchip wurde mit der 0.35 µm-Technologie gefertigt. Der Flächenbedarf der digitalen Module kann für eine erste Abschätzung auf modernere CMOS-Technologien skaliert werden[1]. Dabei wird davon ausgegangen, dass sich die Seitenlängen der Standardzellen linear mit der Prozessgröße ändern [10, 44]. Da die Fläche der Standardzelle das Quadrat zweier Längen ist, fließt dieser Faktor quadratisch ein. In Gleichung 6.1 wird die Fläche der digitalen Signalverarbeitung als Beispiel auf den 0.18 µm-Prozess skaliert. Tabelle 6.4 zeigt den geschätzten Flächenbedarf der digitalen Signalverarbeitung in einem 0.18 µm- und 0.13 µm-Prozess.

[1] Analoge CMOS-Schaltungen können nicht skaliert werden.

$$
\begin{aligned}
S &= \frac{\text{Technologie 1}}{\text{Technologie 2}} = \frac{0.35\,\mu\text{m}}{0.18\,\mu\text{m}} = 0.514 \\
A_{\text{Technologie 2}} &= A_{\text{Technologie 1}} \cdot S^2 \\
&= 1\,933\,761.60\,\mu\text{m}^2 \cdot 0.26448 = 511\,460.21\,\mu\text{m}^2
\end{aligned} \tag{6.1}
$$

Um die komplette Auswertung des Sensorsignals auf einem Chip zu realisieren, müssten noch weitere Komponenten hinzugefügt werden. Zum einen fehlt die Schnittstelle zwischen der HD-Berechnung und dem Ausgabeprotokoll, zum anderen ist nur ein kleiner Teil der analogen Signalverarbeitung auf dem CMOS-Testchip implementiert worden. Da allerdings noch 27 % von den 8 mm^2 der Chipfläche ungenutzt sind und für den ABS-Sensor deutlich weniger Pads benötigt werden, halte ich die Möglichkeit, die komplette Signal-Auswertung auf einem Chip unterzubringen, für realistisch. Für die Realisierbarkeit spricht auch, dass bei dem 0.13 µm-Prozess nur weniger als ein Siebtel der Fläche für die Digitale Logik benötigt wird. Da die analogen Komponenten nicht skaliert werden können und auch die analoge Signalverarbeitung nicht vollständig implementiert wurde, kann hierüber keine Aussage getroffen werden.

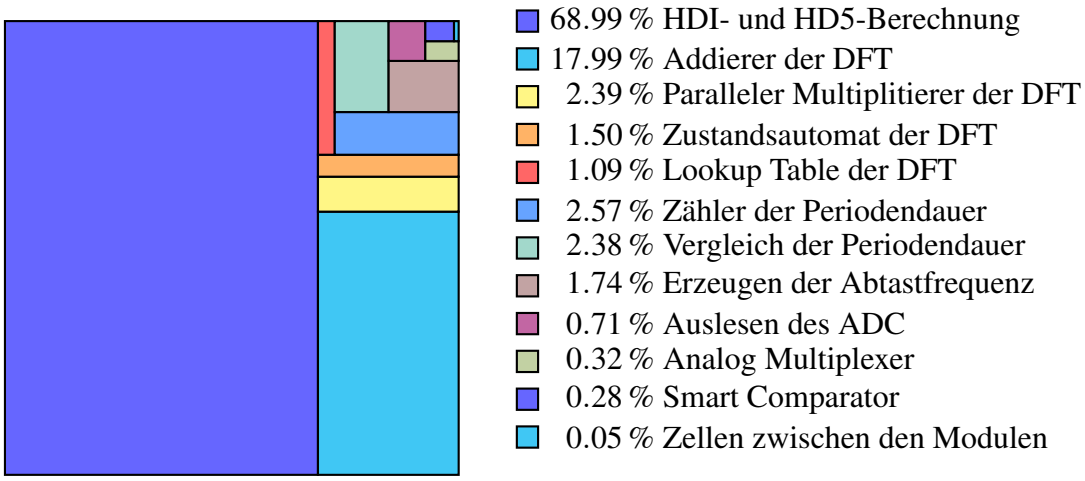


Abbildung 6.2: Prozentuale Verteilung der Fläche der digitalen Signalverarbeitung auf implementierte Komponenten. – In dieser Aufschlüsselung wird nur der Flächenbedarf der Standardzellen betrachtet. Der zustätzliche Platzbedarf für das Routen wird hier vernächlässigt.

Tabelle 6.2: Verteilung der Fläche der digitalen Signalverarbeitung auf implementierte Komponenten. – In dieser Aufschlüsselung werden nur die Standardzellen betrachtet. Der zusätzliche Platzbedarf für das Routen wird hier vernachlässigt.

Modul	Anzahl Standardzellen	Fläche in μm^2	(in %)
HDI- und HD5-Berechnung (calc_hd_i)	4541	751059.40	(68.99)
Addierer der DFT (add_i)	714	195904.80	(17.99)
Paralleler Multiplitierer der DFT (mul_para_i)	216	25971.40	(2.39)
Zustandsautomat der DFT (fsm_calc_harm_i)	212	16307.20	(1.50)
Lookup Table der DFT (lut_i)	165	11848.20	(1.09)
Zähler der Periodendauer (period_timer_i)	160	28009.80	(2.57)
Vergleich der Periodendauer (period_time_chk_i)	158	25916.80	(2.38)
Erzeugen der Abtastfrequenz (sample_timer_i)	132	18964.40	(1.74)
Auslesen des ADC (sample_i)	63	7680.40	(0.71)
Analog Multiplexer (analog_mux_i)	39	3530.80	(0.32)
Smart Comparator (smart_comp_i)	16	3003.00	(0.28)
Zellen zwischen den Modulen	6	527.80	(0.05)
Summe	6422	1088724.00	(100.00)

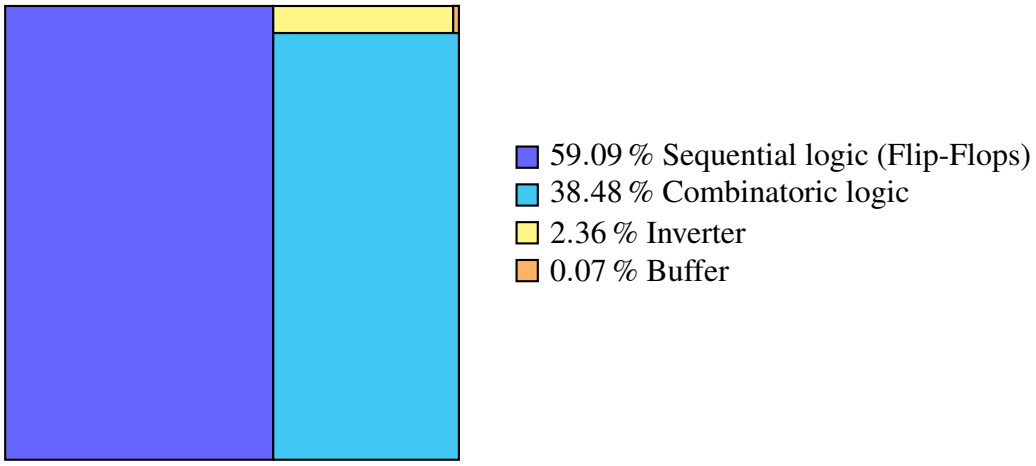


Abbildung 6.3: Prozentuale Verteilung der Fläche der digitalen Signalverarbeitung auf sequentielle und kombinatorische Logik. – Diese Abbildung zeigt, dass ein Großteil der Fläche der digitalen Signalverarbeitung von sequentieller Logik belegt wird. Vergleiche Tabelle 6.3

Tabelle 6.3: Verteilung der Fläche der digitalen Signalverarbeitung auf sequentielle und kombinatorische Logik. Vergleiche Abbildung 6.3

Typ	Anzahl Standardzellen	Fläche	Relative Fläche
Sequential logic (Flip-Flops)	1906	643 370.00 µm^2	59.09 %
Combinatoric logic	3803	418 927.60 µm^2	38.48 %
Inverter	700	25 716.60 µm^2	2.36 %
Buffer	13	709.80 µm^2	0.07 %
Summe	6422	1 088 724.00 µm^2	100.00 %

Tabelle 6.4: Skalierung des Flächenbedarfs der digitalen Signalverarbeitung des CMOS-Testchips auf andere CMOS-Technologien. – Dies ist eine Abschätzung des Flächenbedarfs bei der Verwendung anderer Technologien.

Technologie	Fläche
0.35 µm	1 933 761.60 µm^2
0.18 µm	511 460.21 µm^2
0.13 µm	266 780.17 µm^2

6.2 Leistungsaufnahme und Taktfrequenz

Die Verlustleistung von digitalen CMOS-Schaltungen hängt maßgeblich von der Taktfrequenz und der Versorgungsspannung ab. Gleichung 6.2 zeigt, wie sich die dynamische Verlustleistung P_{dyn} berechnet [20, Seite 336 ff.]. Dabei ist f die Taktfrequenz, U_B die Betriebsspannung und C_L eine kapazitive Last. Es ist in Bezug auf die Leistungsaufnahme, also eine möglichst niedrige Taktfrequenz wünschenswert. Die statischen Verluste sind in der Regel zu vernachlässigen [20]. Die Stromaufnahme der Logik der ABS-Sensoren darf laut [17, Seite 26-27] bei etwa 2 mA bis 3 mA liegen.

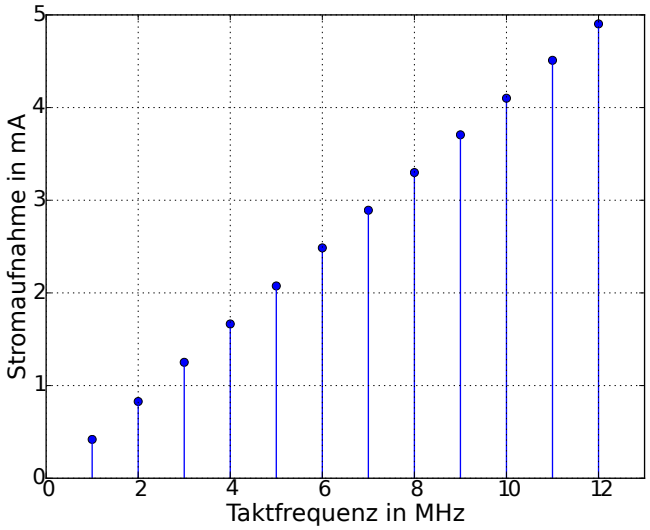
$$P_{dyn} = C_L f U_B^2 \tag{6.2}$$

Die Bits des Ausgabeprotokolls müssen mit einer Frequenz von 40 kHz geschaltet werden. Daraus resultiert, dass der Protokollgenerator mit keiner niedrigeren Taktfrequenz betrieben werden kann. Bedingt durch den Zustandsautomaten und der Flipflops für die Synchronisation beginnt die Ausgabe erst zwei Takte nach Eingang des Startsignals. Eine höhere Taktfrequenz würde lediglich die Reaktionszeit des Protokollgenerators verringern.

Die Berechnung des HDI und des HD5 benötigt, nachdem alle 64 benötigten Samples eingelesen wurden 465 Takten. Das entspricht bei einem Systemtakt von 16 MHz 29.0625 µs.

In Abbildung 6.4 ist die gemessene Stromaufnahme des CMOS-Testchips bei verschiedenen Frequenzen aufgelistet. Dabei wurde nur die Stromaufnahme der digitalen Signalverarbeitung gemessen. Wie erwartet steigt die Stromaufnahme linear mit der Frequenz (Gleichung 6.2).

Neben der Taktfrequenz ist die Leistungsaufnahme digitaler CMOS-Schaltungen stark von der Betriebsspannung U_B abhängig. Nach Gleichung 6.2 fließt diese quadratisch in die dynamische Verlustleistung ein. Laut [10] kann die Betriebsspannung von CMOS-Schaltungen,



Taktfrequenz	Stromaufnahme
1 MHz	0.419 mA
2 MHz	0.828 mA
3 MHz	1.251 mA
4 MHz	1.665 mA
5 MHz	2.075 mA
6 MHz	2.486 mA
7 MHz	2.892 mA
8 MHz	3.299 mA
9 MHz	3.706 mA
10 MHz	4.101 mA
11 MHz	4.510 mA
12 MHz	4.903 mA

Abbildung 6.4: Stromaufnahme des CMOS-Testchips in Abhängigkeit der Taktfrequenz. – Die Stromaufnahme steigt linear zur Taktfrequenz.

genau wie die Länge der Standardzellen, linear zu der Prozessgröße skaliert werden. Damit könnte in einem 0.13 µm-Prozess eine Betriebsspannung von $U_B = 1.2\,\mathrm{V}$ eingesetzt werden. Die dynamische Verlustleistung ist folglich quadratisch und die Stromaufnahme linear von der Prozessskalierung abhängig. Tabelle 6.5 zeigt die geschätzte Betriebsspannung, die Stromaufnahme und die daraus berechnete Verlustleistung für modernere CMOS-Technologien.

Bei der Stromaufnahme und der Taktfrequenz sehe ich noch einige Probleme auf das Projekt zu kommen. Die Stromaufnahme des Chips liegt weit über den zulässigen Werten. Auch nach der Skalierung auf modernere CMOS-Technologien zeigt sich, dass die digitale Signalverarbeitung noch 2 mA Strom benötigt. Dieser kann aber nur durch eine niedrigere Taktfrequenz gesenkt werden.

6.3 Erweiterungsvorschläge

Um die Lücke zwischen der HDI- und HD5-Berechnung und dem Protokollgenerator zu schließen, wird ein Modul benötigt, das die Berechnungen auswertet und danach den drei Fehlerbits des Protokolls einen Wert zuordnet. Dies könnte im einfachsten Fall durch eine Lookup Table geschehen.

Diese Arbeit hat nur eine erste Abschätzung für die Realisierbarkeit der Umsetzung betrachtet. Um weitere Aussagen zu treffen, müsste der Algorithmus zur Annäherung an den THD weiter untersucht werden.

Tabelle 6.5: Skalierung der Leistungsaufnahme der digitalen Signalverarbeitung des CMOS-Testchips auf andere CMOS-Technologien. – Dies ist eine Abschätzung der Betriebsspannung und Stromaufnahme bei der Verwendung anderer Technologien bei einer Taktfrequenz von 12 MHz.

Technologie	Betriebsspannung	Stromaufnahme	Leistung
0.35 µm	3.3 V	4.903 mA	16.180 mW
0.18 µm	1.7 V	2.522 mA	4.287 mW
0.13 µm	1.2 V	1.821 mA	2.185 mW

Tabelle 6.6: Flächenbedarf von D-Flipflops. – Flipflops mit synchronem Reset sind in der Standardbibliothek nicht vorhanden und müssen aus einem Flipflop ohne Reseteingang, einem Inverter und einem NAND-Gatter synthetisiert werden. Dabei ist nicht die für die Verdrahtung benötigte Fläche berücksichtig.

Typ	Standardzelle	Fläche
D-Flipflop ohne Reset	DF1	232.960 μm^2
D-Flipflop mit asynchronem Reset	DFC1	320.320 μm^2
D-Flipflop mit synchronem Reset	DF1, INV3, NAN22	305.760 μm^2

Wie oben gezeigt, wird ein Großteil der Chipfläche der digitalen Module von Flipflops belegt. Bei den Flipflops sehe ich eine weitere Möglichkeit Ressourcen einzusparen. Wie in Tabelle 6.6 zu sehen ist, benötigen Flipflops ohne Reseteingang deutlich weniger Fläche als Flipflops mit Reseteingang. Es sollten Überlegungen angestellt werden, ob jedes der implementierten Flipflops einen Reseteingang benötigt. Da so eventuell Fläche eingespart werden könnte.

Es sollte angestrebt werden, die Stromaufnahme der Logik zu minimieren, was nur durch eine geringere Taktung möglich ist. Dafür könnte untersucht werden, wie oft und wie schnell es sinnvoll ist, den THD des Sensorsignals zu bestimmen.

Weitere Versuche sollten die analoge Signalaufbereitung komplett (inklusive Regelung) auf einem Chip realisieren. Vermutlich wird der Flächenbedarf aufgrund von benötigten Widerständen weiter ansteigen.

Abkürzungsverzeichnis

ABS	Antiblockiersystem
ADC	Analog-Digital-Umsetzer (Analog-to-Digital-Converter)
AMR	Anisotroper Magnetoresistiver Effekt
ASIC	Anwendungsspezifische integrierte Schaltung (Application Specific Integrated Circuit)
BIST	Build In Self Test
CLCC	Ceramic Leaded Chip Carrier
CMOS	Complementary Metal Oxide Semiconductor
CPLD	Complex Programmable Logic Device
CSV	Comma-separated values
DRC	Design Rule Check
DFT	Diskrete Fourier-Transformation
ESD	Elektrostatische Entladung (electrostatic discharge)
ESZ-ABS	Experimentelle digitale Signalverarbeitung und Zustandserkennung für ABS-Sensoren
FIB	Focused Ion Beam
FPGA	Field Programmable Gate Array
HD5	Berechnung der gesamten harmonischen Verzerrung mithilfe der ersten fünf Harmonischen
HDL	Hardwarebeschreibungssprache (Hardware Description Language)
LSB	Least Significant Bit
LUT	Lookup Table
LVS	Layout versus Schematic
MOSFET	Metall-Oxid-Halbleiter-Feldeffekttransistor
MSB	Most Significant Bit
POR	Power-on Reset

RTL Register-Transfer Level

SDF Standard Delay Format – IEEE-Standard für Timing-Daten.

SPI Serial Peripheral Interface

Tcl Open Source-Skriptsprache

THD Gesamte Harmonische Verzerrung (Total Harmonic Distortion)

VHDL Very High Speed Integrated Circuit Hardware Description Language

Literaturverzeichnis

[1] *Cadence SOC Encounter Tutorial*. – URL http://eeweb.poly.edu/labs/nanovlsi/tutorials/soctutorials/Tutorial_Encounter.html. – Zugriffsdatum: 03.05.2012

[2] *Converting your Existing Libraries from CDB to OA*. – URL www.cs.utah.edu/cadence/data/cdb2oa.pdf. – Zugriffsdatum: 03.05.2012

[3] ARVIND: *Automatic Placement and Routing using Cadence Encounter*. 2008. – URL http://csg.csail.mit.edu/6.375/6_375_2008_www/handouts/tutorials/tut5-enc.pdf

[4] AUSTRIAMICROSYSTEMS AG: *First Encounter - P&R Flow*. – URL http://asic.austriamicrosystems.com/hitkit/hk400/fe/index.html. – Zugriffsdatum: 04.05.2012

[5] AUSTRIAMICROSYSTEMS AG: *Analog Standard Cell – POR - CMOS Power On Reset*. Revision B, 11-Mar-08, März 2008

[6] AUSTRIAMICROSYSTEMS AG: *0.35 um CMOS C35 Design Rules*. Rev.: 9.0, 2011. – URL http://asic.ams.com/cgi-sbin/agreement?ENG-183_rev9.pdf. – Zugriffsdatum: 28.09.2012

[7] AUSTRIAMICROSYSTEMS AG: *0.35um Analog Standard Cells*. Rev.: 9.0, 2012. – URL http://asic.ams.com/databooks/c35_a/index.html. – Zugriffsdatum: 28.09.2012

[8] AUSTRIAMICROSYSTEMS AG: *The Bonding process*, 2012. – URL http://asic.ams.com/hitkit/tools/bond_editor/index.html. – Zugriffsdatum: 08.08.2012

[9] AUSTRIAMICROSYSTEMS AG: *Digital Design Flow*, 2012. – URL http://asic.ams.com/hitkit/general/DigitalDesignFlow2012.pdf. – Zugriffsdatum: 23.12.2012

[10] BAKER, R. J.: *CMOS Circuit Design, Layout, and Simulation*. 3rd ed. Wiley-IEEE Press, September 2010

[11] BRUNVAND, Erik: *Digital VLSI Chip Design with Cadence and Synopsys CAD Tools*. 1 edition. Addison Wesley, März 2009

[12] CADENCE: *Designing Power Mesh Structures Application Note*, Juli 2002. – URL http://support.cadence.com

[13] CUMMINGS, Clifford E. ; MILLS, Don ; GOLSON, Steve: *Asynchronous & Synchronous Reset – Design Techniques - Part Deux*. 2003

[14] DAIMLER AG: *Requirement Specifications for Standardized Interface for Wheel Speed Sensors with Additional Information „AK-Protokoll“*. 4.0, Februar 2008

[15] DASALUKUNTE, Deepak: *Digital IC-Project and Verification – Place and Route*, Lund University, . – URL `http://www.eit.lth.se/fileadmin/eit/courses/etin01/slides/Place_and_Route.pdf`. – Zugriffsdatum: 04.05.2012

[16] DRESCHHOFF, Jan-Heiner: *VHDL - Implementierung des Ausgabeprotokolls von ABS - Sensoren*, Fakultät Technik und Informatik der Hochschule für Angewandte Wissenschaften Hamburg, Studienarbeit, 2009

[17] DRESCHHOFF, Jan-Heiner: *FPGA-Prototyp der Signalverarbeitung für ABS-Sensoren mit Diagnosefunktion*, Fakultät Technik und Informatik der Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2010

[18] EUROPRACTICE: *AMS C35 cmos + opto technology overview (MPW)*. – URL `http://www.europractice-ic.com/technologies_AMS.php?tech_id=cmos`. – Zugriffsdatum: 28.09.2012

[19] GOSCH, Karl: *Application Note – Power Supply Concept*. Draft v1.0. austriamicrosystems AG (Veranst.), 2012. – URL `http://asic.ams.com/appnotes/digital/Power_Supply_Concept_Draft1.0.pdf`

[20] GÖBEL, Holger: *Einführung in die Halbleiter-Schaltungstechnik*. 4., bearbeitete und erweiterte Auflage. Springer, 2011

[21] GÖPEL, Wolfgang ; HESSE, Joachim ; ZEMEL, J. N. ; BOLL, Richard ; OVERSHOTT, Kenneth J.: *Magnetic Sensors. (Bd. 5): A Comprehensive Survey (Sensors a Comprehensive Survey)*. 1. Wiley-VCH, März 1990

[22] HEISSING, Bernhard (Hrsg.) ; ERSOY, Metin (Hrsg.) ; GIES, Stefan (Hrsg.): *Fahrwerkhandbuch: Grundlagen, Fahrdynamik, Komponenten, Systeme, Mechatronik, Perspektiven (ATZ/MTZ-Fachbuch)*. 3., überarb. u. erw. Aufl. 2011. Vieweg+Teubner Verlag, Juni 2011

[23] IVANOV, Kalin: *Fehlersichere Automatisierung eines Encoder-Messplatzes zur Untersuchung von ABS-Sensoren*, Fakultät Technik und Informatik der Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2011

[24] JEGENHORST, Niels: *Entwicklung eines Controllersystems zur Zustandserkennung von ABS-Sensoren*, Fakultät Technik und Informatik der Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2009

[25] KESEL, Frank ; BARTHOLOMÄ, Ruben: *Entwurf von digitalen Schaltungen und Systemen mit HDLs und FPGAs: Einführung mit VHDL und SystemC*. Oldenbourg Wissenschaftsverlag, Februar 2006

[26] KREY, Martin ; RIEMSCHNEIDER, Dr.-Ing. Karl-Ragmar: Signalverarbeitung zur Funktionsdiagnose bei magnetischen Sensoren. In: *EForum* Jahrgang 2011 (2011), S. 20 bis 25

[27] MÄDER, Andreas: *Layoutsynthese*. MIN-Fakultät der Universität Hamburg, 2006. – URL `http://tams-www.informatik.uni-hamburg.de/research/vlsi/edaTools/doc/06/layoutSyn.pdf`. – Zugriffsdatum: 03.05.2012

[28] MÄDER, Andreas: *VHDL- und mixed-mode Netzlistensimulation*. MIN-Fakultät der Universität Hamburg, 2006. – URL `http://tams-www.informatik.uni-hamburg.de/research/vlsi/edaTools/doc/06/netlistSim.pdf`. – Zugriffsdatum: 09.05.2012

[29] MÄDER, Andreas: *Cadence Grundlagen*. MIN-Fakultät der Universität Hamburg, 2009. – URL `http://tams-www.informatik.uni-hamburg.de/research/vlsi/edaTools/doc/09/cadence.pdf`. – Zugriffsdatum: 03.05.2012

[30] MÄDER, Andreas: *Full-Custom Design - Cadence IC v6.1.31 + AMS Hit-Kit v4.00*. MIN-Fakultät der Universität Hamburg, November 2009. – URL `http://tams-www.informatik.uni-hamburg.de/research/vlsi/edaTools/doc/09/fcDesign.pdf`. – Zugriffsdatum: 03.05.2012

[31] MICHAEL, Dr. M. ; NEOPHYTOU, Stelios: *Cadence RTL Compiler Ultra Tutorial*. ELECTRICAL AND COMPUTER ENGINEERING DEPARTMENT, UNIVERSITY OF CYPRUS, 2007. – URL `http://www.eng.ucy.ac.cy/mmichael/courses/ECE407/lecture_notes/RTL%20Logic%20Synthesis%20Tutorial.pdf`. – Zugriffsdatum: 03.05.2012

[32] OSTERMANN, Robert: *Experimentalaufbau einer kombinierten logarithmischen und linearen Verstärkerbank für ABS-Sensoren*, Fakultät Technik und Informatik der Hochschule für Angewandte Wissenschaften Hamburg, Bachelorthesis, 2012

[33] PHILIPS SEMICONDUCTORS: *General Rotational speed measurement*, 1998

[34] PIOREK, Markus: *Hard- und Software eines Messplatzes zur Harmonischenanalyse bei magnetischen Sensoren*, Fakultät Technik und Informatik der Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2011

[35] RAZAVI, Behzad: *Design of Analog CMOS Integrated*. First Edition. McGraw-Hill Companies, Oktober 2003

[36] RODRIGUES, Joachim: *Physical Placement with Cadence SoCEncounter 7.1*, November 2008. – URL http://www.eit.lth.se/fileadmin/eit/courses/etin01/manual_etc/SEtraining2.pdf. – Zugriffsdatum: 03.05.2012

[37] SANTOS, Marcelino ; GOMES, Sílvia: *Standard Cell Placement and Routing Tutorial*, 2007. – URL https://dspace.ist.utl.pt/bitstream/2295/290575/1/tutorial_encounter_versao2.pdf

[38] SCHOERMER, Christian: *Konstruktion und Automatisierung eines Radmessplatzes für ABS mit Encodern verschiedener Automobil-Hersteller*, Fakultät Technik und Informatik der Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2011

[39] SIEMERS, Christian (Hrsg.) ; SIKORA, Axel (Hrsg.): *Taschenbuch Digitaltechnik*. 2., vollständig überarbeitete Auflage. Carl Hanser Verlag GmbH & CO. KG, September 2007

[40] STAHL, Martin: *Controllersystem zur Verstärkungsregelung und Offsetkompensation für ABS-Sensoren mit Diagnosefunk*, Fakultät Technik und Informatik der Hochschule für Angewandte Wissenschaften Hamburg, Bachelorthesis, 2010

[41] STAN, Mircea R.: *RTL Logic Synthesis Tutorial*. ECE Department – University of Virginia (Veranst.), September 2006. – URL http://www.ee.virginia.edu/~mrs8n/soc/rc_tutorial.html. – Zugriffsdatum: 03.05.2012

[42] STAN, Mircea R.: *VHDL/Verilog Simulation Tutorial*. ECE Department, University of Virginia (Veranst.), September 2006. – URL http://www.ee.virginia.edu/~mrs8n/soc/sim_tutorial.html. – Zugriffsdatum: 09.05.2012

[43] STAN, Prof. M.: *Backend Design Tutorial*, September 2006. – URL http://www.ee.virginia.edu/~mrs8n/soc/enc_tutorial.html. – Zugriffsdatum: 03.05.2012

[44] THUSELT, Frank: *Physik der Halbleiterbauelemente: Einführendes Lehrbuch für Ingenieure und Physiker (German Edition)*. 2. Auflage. Springer, Juni 2011

[45] WANG, Laung-Terng ; WU, Cheng-Wen ; WEN, Xiaoqing: *VLSI Test Principles and Architectures: Design for Testability (The Morgan Kaufmann Series in Systems on Silicon)*. Morgan Kaufmann, Juli 2006

[46] ZANE, Prof. Regan A.: *Cadence Design Tools – Adding I/O Pads to the Design*. 2010. – URL http://ecee.colorado.edu/~ecen5837/cadence/Adding_IO.html. – Zugriffsdatum: 08.11.2012

A Pinout des CMOS-Testchip

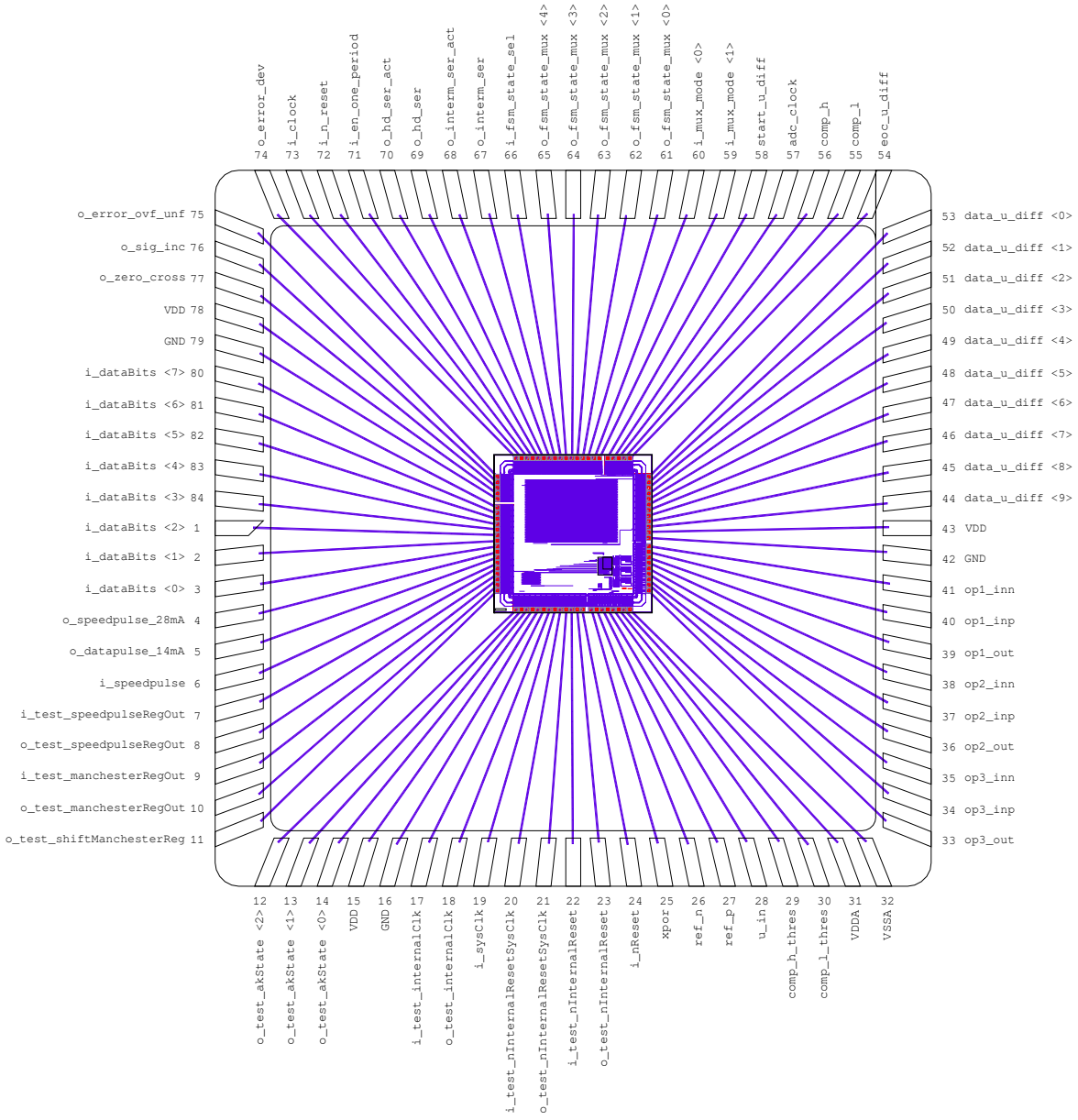


Abbildung A.1: Pinout des CMOS-Testchip.

B Quelltext und Blockschaltbilder

Die hier abgedruckten Quelltexte sind nicht diejenigen, aus denen der Chip synthetisiert wurde. Aus Gründen der besseren Lesbarkeit wurde der Quelltext noch mit Kommentaren versehen. Der Quelltext aus denen der Chip synthetisiert wurde, ist im SVN-Repository[1] unter der Revisionsnummer r1507 zu finden.

B.1 Protokollgenerator

[1] `svn+ssh://ssh.informatik.haw-hamburg.de/srv/svn/rmp`

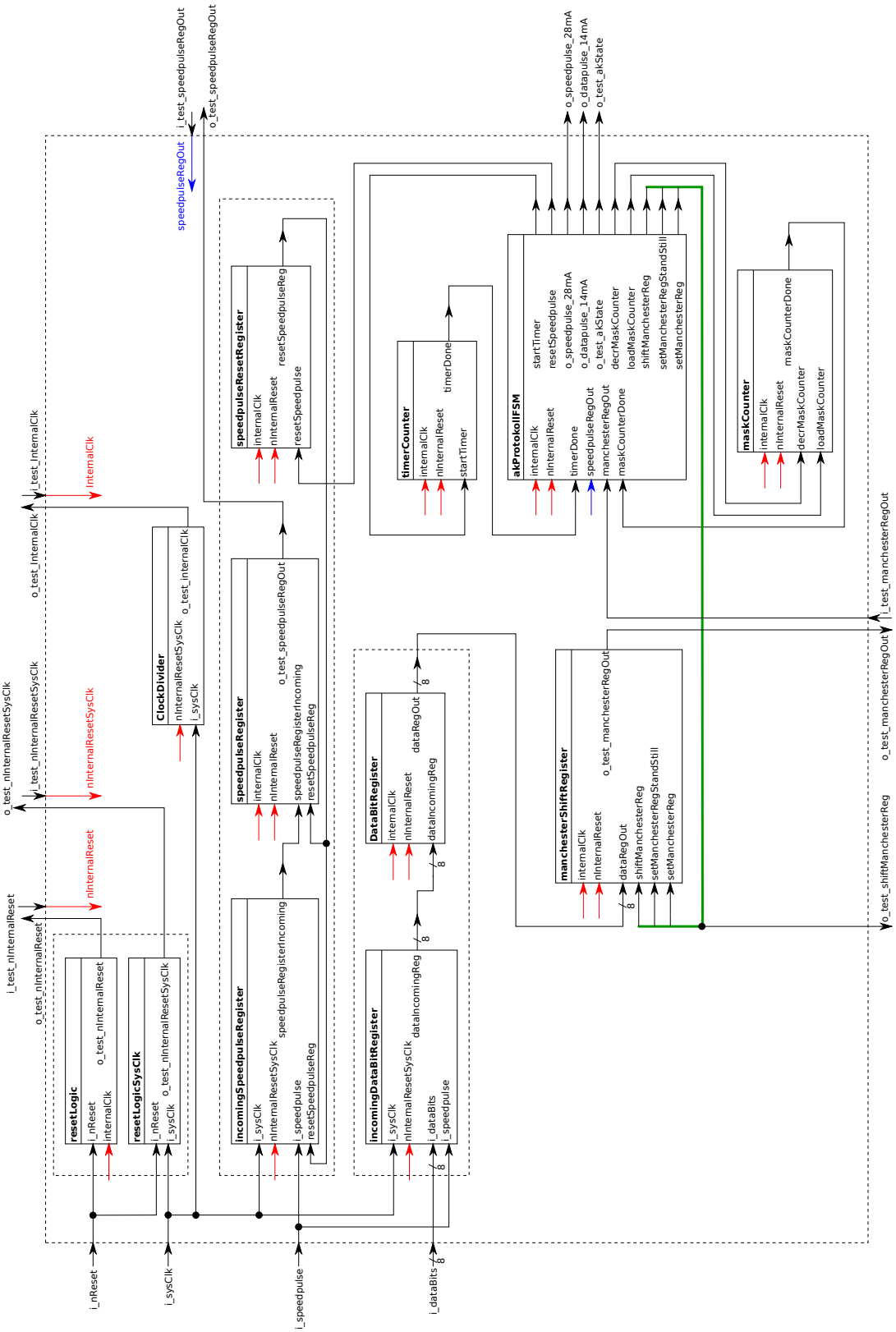


Abbildung B.1: Blockschaltbild des Protokollgenerators. – Siehe Quelltext B.1

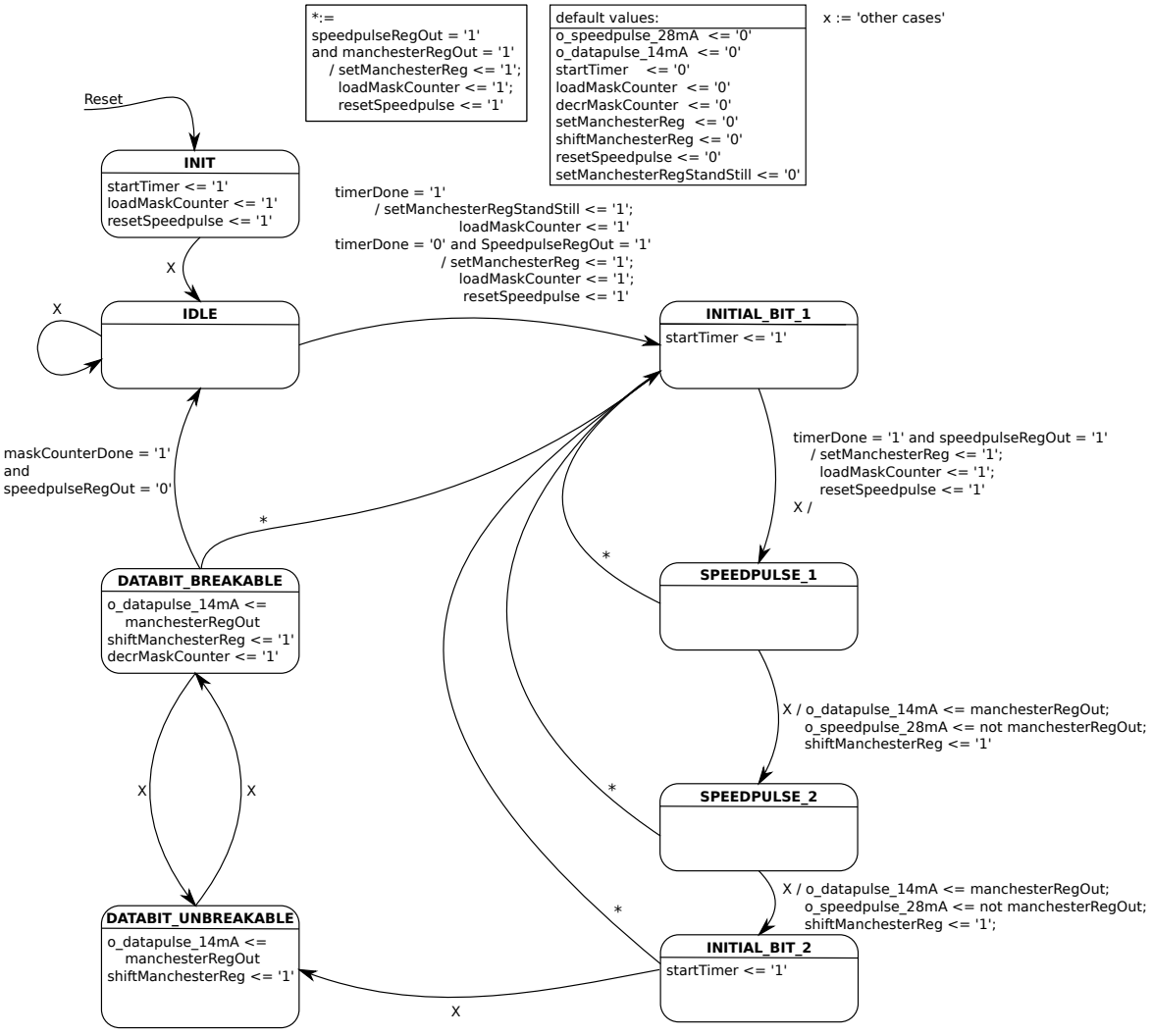


Abbildung B.2: Zustandsdiagramm des Protokollgenerators. – Siehe Quelltext B.1

```
-------------------------------------------------------------------------------
-- Title      : abs_menchster_encoder
-- Project    : ESZ-ABS
-------------------------------------------------------------------------------
-- File       : abs_manchester_encoder.vhdl
-- Author     : Jan-Heiner Dreschhoff (hdreschhoff {at} gmail.com)
--              Daniel Sabotta
-- Company    : HAW Hamburg
-- Created    : 2012-05-25
-- Last update: 2012-08-21
-------------------------------------------------------------------------------
-- Description: abs_menchester_encoder
-------------------------------------------------------------------------------
-- Copyright (c) 2012
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author         Description
-- 2009-09-15  0        Dreschhoff     start of coding
-- 2009-09-17  0        Dreschhoff     Wrote Timer and Clockdivider
--                                     Started FSM Design
-- 2009-09-18  0        Dreschhoff     changed internal Clockspeed to 80kHz
--                                     wrote first Version of FSM
--                                     added "after 9ns" for testing
--                                     and debugging
-- 2009-09-21  0        Dreschhoff     rewrote FSM, added Registers
-- 2009-09-22  0        Dreschhoff     wrote Test_abs_manchester.vhd
--                                     Issues while passing Data from i_dataBits to REG_IN
-- 2009-09-23  0        Dreschhoff     fixed FSM and man_reg Issues.
--                                     Timer now starts on INITIAL_BIT_1.
-- 2009-09-24  0        Dreschhoff     fixed timing Issues,
--                                     renamed Signals,
--                                     fixed CLK why is maskCounterReg=="XXXX" on Start?
--                                     asynchronous i_nReset?
-- 2009-09-25  0        Dreschhoff     fixed maskCounterReg issue by rooting i_sysClk on CLK while i_nReset
-- 2009-09-26  1        Dreschhoff     cleaned up:
--                                       > Sensitivitylist of FSM
--                                       > States INITIAL_BIT_1, INITIAL_BIT_2 and DATABIT_UNBREAKABLE
-- 2009-10-01  1        Dreschhoff     Seperated Data- and Controlpath,
--                                     extended FSM so STANDSTILL_TIMEOUT Manchestercode can be Interrupted by actual
    Information.
-- 2009-10-07  1        Dreschhoff     detected CLK issues in ISE (longest path 23ns) changed clk divider accordingly
--                                     i_nReset now asynchronous
-- 2009-10-15  1        Dreschhoff     Removed SIG_LVL from design via comments
--                                     Removed STANDSTILL_TIMEOUT_OUT and START_BLOCKING altogether
--                                     Removed all Latches
-- 2009-10-20  1        Dreschhoff     Changed Code and Timing to suit an external 10 MHz Clock
-- 2009-11-06  1        Dreschhoff     Changed akProtokollState INITIAL_BIT_1 to better suit wrapper
-- 2012-05-16  2        Sabotta        Formatting
-- 2012-05-23  2        Sabotta        timer replaced
--                                     removed a undefined state in clk_gen
--                                     changed signal/process names
--                                     changed clock to 16MHz
--                                     mask counter counts only 9 Bit (8 downto 0)
-- 2012-05-29  2        Sabotta        inserting a buffer for incoming speedpulse
--                                     remove incoming data mask (how many bits to send)
-- 2012-05-30  2        Sabotta        remove clockenable / inserting second clockdomain
-- 2012-06-01  2        Sabotta        inserting asynchron reset logic (set reset asynchron and release reset synchron)
-- 2012-06-04  2        Sabotta        inserting manual statecode
--                                     enabling to set the state externally
-- 2012-06-06  2        Sabotta        diable reset on faling edge
--                                     remove the possibility to set the state externally
-- 2012-06-07  2        Sabotta        inserting testsignals
-- 2012-06-11  2        Sabotta        changing dutycycle of internal clk to 50%
-- 2012-08-21  2        Sabotta        formatted and commented
-------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;



entity abs_manchester_encoder is

  generic(
    -- Most significant bit of the clock divider
    CLK_DIVIDER_MSB    : integer := 7;
    -- Clock divider (half period) value to reach a 40kHz clock (1 / 25us): 16MHz/40kHz = 0x190
    clk_divider      : std_logic_vector(7 downto 0) := x"C8"; --x"190";

    -- Most significant bit of Timer
    NUM_TIMER_MSB    : integer := 12;
    -- the standstill-protocoll will be send after STANDSTILL_TIMEOUT times 25us
```

```
    -- this should be 150ms: 150ms/25us = 0x1770 (0x1770 - 3 clock cycles [FSM and counter delay] = 0x176D)
    STANDSTILL_TIMEOUT : std_logic_vector(12 downto 0) := "1" & x"76D";

    -- MSB of Manchester Register
    MAN_REG_MSB        : integer := 19
  );



  port(
    i_sysClk   : in std_logic; -- System clock (16MHz)
    i_nReset   : in std_logic; -- Active low reset

    -- INPUTS --
    i_speedpulse   : in std_logic;                    -- Incoming speed pulse
    i_dataBits     : in std_logic_vector(7 downto 0); -- Incoming data

    -- OUTPUTS --
    o_speedpulse_28mA : out std_logic; -- Controlsignal for the 28mA driver
    o_datapulse_14mA  : out std_logic; -- Controlsignal for the 14mA driver

    -- TEST INPUTS --
    -- feed in an external signal if internal generation doesn't work
    -- (it exist a o_test_-output-pin for each i_test_-pin ro bridge them externly if the module works)
    i_test_nInternalReset        : in std_logic; -- internal Reset Signal
    i_test_nInternalResetSysClk : in std_logic; -- internal Reset Signal on system Clock
    i_test_internalClk           : in std_logic; -- internal Clock (should be 40kHz)
    i_test_speedpulseRegOut      : in std_logic; -- output of speedpulse register
    i_test_manchesterRegOut      : in std_logic; -- output of manchester register

    -- TEST OUTPUTS --
    o_test_nInternalReset        : out std_logic; -- internal Reset Signal
    o_test_nInternalResetSysClk : out std_logic; -- internal Reset Signal on system Clock
    o_test_internalClk           : out std_logic; -- internal Clock (should be 40kHz)
    o_test_speedpulseRegOut      : out std_logic; -- output of speedpulse register
    o_test_manchesterRegOut      : out std_logic; -- output of manchester register

    o_test_shiftManchesterReg    : out std_logic; -- shift left (get next bit to send)
    o_test_akState               : out std_logic_vector(2 downto 0)  -- control the state of akProtokollFSM
  );
end entity abs_manchester_encoder;


architecture behavioral of abs_manchester_encoder is

  ------------------
  --    signals   --
  ------------------

  -- reset logic
  signal nInternalReset        : std_logic;  -- reset for all processen on internalClk
  signal nResetOut             : std_logic;  -- reset for all processen on internalClk (process output)
  signal nResetRegister        : std_logic;  -- register to avoid metastable states
  signal nInternalResetSysClk : std_logic;  -- reset for all processen on i_sysClk
  signal nResetOutSysClk       : std_logic;  -- reset for all processen on i_sysClk  (process output)
  signal nResetRegisterSys     : std_logic;  -- register to avoid metastable states

  -- ClockDivider
  signal clkCounterReg   : std_logic_vector (CLK_DIVIDER_MSB downto 0);  -- Internal counting signal from 0 to CLK_DIVIDER
  signal clkDivOut       : std_logic;  -- 40kHz clock
  signal internalClk     : std_logic;  -- 40kHz clock

  -- speedpulse register
  signal speedpulseRegOut           : std_logic; -- internal incoming speedpulse
  signal resetSpeedpulse            : std_logic; -- reset saved speedpulse
  signal resetSpeedpulseReg         : std_logic; -- reset saved speedpulse (i_sysClk domain)
  signal speedpulseRegisterIncoming : std_logic; -- save incoming speedpulse (monoflop)
  signal speedpulseReg              : std_logic_vector (1 downto 0);-- register to avoid metastable states

  -- databit register
  signal dataIncomingReg  : std_logic_vector (7 downto 0);  -- saves incoming dataBits (i_sysClk domain)
  signal dataReg1         : std_logic_vector (7 downto 0);  -- register to avoid metastable states
  signal dataReg2         : std_logic_vector (7 downto 0);  -- register to avoid metastable states
  signal dataRegOut       : std_logic_vector (7 downto 0);  -- data to work with

  -- timer
  signal timerCounterReg  : std_logic_vector(NUM_TIMER_MSB downto 0); -- Counter signal register from STANDSTILL_TIMEOUT
      downto 0
  signal startTimer       : std_logic; -- Starts the timer
  signal timerDone        : std_logic; -- Time's up

  -- ManchesterShiftRegister
  signal shiftManchesterReg         : std_logic; -- Shift left
  signal setManchesterReg           : std_logic; -- Write incoming data into the ManchesterShiftRegister
```

```
  signal setManchesterRegStandStill : std_logic; -- Write incoming data into the ManchesterShiftRegister for stand still protocoll
  signal manchesterRegOut           : std_logic; -- Shiftregister output (next bit to send)
  signal manchesterRegister         : std_logic_vector(MAN_REG_MSB downto 0); -- Stores the values

  -- maskCounter
  signal loadMaskCounter  : std_logic; -- Load iDatamask into maskCounter
  signal decrMaskCounter  : std_logic; -- Decrement maskCounter
  signal maskCounterDone  : std_logic; -- maskCounter is 0
  signal maskCounterReg   : std_logic_vector(3 downto 0); -- Counter signal register



  -- akProtokollFSM
  subtype typeAKProtokollStates is std_logic_vector(2 downto 0); -- akProtokollFSM states
  constant INIT                 : typeAKProtokollStates := "000"; -- Initial state
  constant IDLE                 : typeAKProtokollStates := "001"; -- Idle state
  constant INITIAL_BIT_1        : typeAKProtokollStates := "010"; -- Send first initial bit (befor speedpulse)
  constant SPEEDPULSE_1         : typeAKProtokollStates := "011"; -- Send first half of speedpulse
  constant SPEEDPULSE_2         : typeAKProtokollStates := "100"; -- Send second half of speedpulse
  constant INITIAL_BIT_2        : typeAKProtokollStates := "101"; -- Send second initial bit (after speedpulse)
  constant DATABIT_UNBREAKABLE  : typeAKProtokollStates := "110"; -- Send first half of data bit (next state MUST be DATABIT_BREAKABLE)
  constant DATABIT_BREAKABLE    : typeAKProtokollStates := "111"; -- Send first half of data bit

  signal akProtokollState, nextAkProtokollState : typeAKProtokollStates; -- akProtokollFSM state registers


begin

  ----------------
  -- Reset logic --
  ----------------

  -- purpose: create a asynchornous reset at incoming i_nReset signal
  --          and release reset synchronously (2 cycle delay)
  --          for internalClk
  --          (see "Asynchronous & Synchronous Reset Design Techniques - Part Deux")
  -- type   : sequential
  -- inputs : internalClk, i_nReset
  -- outputs: nInternalReset
  resetlogic: process (internalClk, i_nReset)
  begin
    if i_nReset = '0' then
      nResetRegister <= '0';
      nResetOut <= '0';
    elsif internalClk = '1' and internalClk'event then
      nResetRegister <= '1';
      nResetOut <= nResetRegister;
    end if;
  end process;

  ----- testpins -----
  o_test_nInternalReset <= nResetOut;
  nInternalReset <= i_test_nInternalReset;
  --nInternalReset <= nResetOut; -- without testpins

  -- purpose: create a asynchornous reset at incoming i_nReset signal
  --          and release reset synchronously (2 cycle delay)
  --          for i_sysClk
  -- type   : sequential
  -- inputs : i_sysClk, i_nReset
  -- outputs: nInternalResetSysClk
  resetLogicSysClk: process (i_sysClk, i_nReset)
  begin
    if i_nReset = '0' then
      nResetRegisterSys <= '0';
      nResetOutSysClk <= '0';
    elsif i_sysClk = '1' and i_sysClk'event then
      nResetRegisterSys <= '1';
      nResetOutSysClk <= nResetRegisterSys;
    end if;
  end process;

  ----- testpins -----
  o_test_nInternalResetSysClk <= nResetOutSysClk;
  nInternalResetSysClk <= i_test_nInternalResetSysClk;
  --nInternalResetSysClk <= ; -- without testpins

  ----------------
  --  Clockdivider --
  ----------------

  -- purpose: generates an 40kHz enable signal
  -- type   : sequential
```

```
  -- inputs : i_sysClk, i_nReset
  -- outputs: internalClk
  ClockDivider : process(i_sysClk, nInternalResetSysClk)
  begin
    if nInternalResetSysClk = '0' then
      clkCounterReg <= (others => '0');
      clkDivOut     <= '0';
    elsif i_sysClk = '1' and i_sysClk'event then
      clkCounterReg <= clkCounterReg + 1;
      --clkDivOut <= '0';
      if clkCounterReg >= clk_divider-1 then
        clkDivOut <= not clkDivOut;
        clkCounterReg <= (others => '0');
      end if;-- clkCounterReg
    end if;-- i_sysClk
  end process ClockDivider;

  ----- testpins -----
  o_test_internalClk <= clkDivOut;
  internalClk <= i_test_internalClk;
  --internalClk <= clkDivOut; -- without testpins


  ---------------------------------
  -- Synchronize incoming Speedpulse --
  ---------------------------------

  -- purpose: saves incoming speedpule in i_sysClk-Domain (monoflop)
  -- type   : sequential
  -- inputs : i_sysClk, nInternalReset, i_speedpulse, resetSpeedpulseReg
  -- outputs: speedpulseRegisterIncoming
  incomingSpeedpulseRegister: process (i_sysClk, nInternalResetSysClk)
  begin
    if nInternalResetSysClk = '0' then
      speedpulseRegisterIncoming <= '0';
    elsif i_sysClk = '1' and i_sysClk'event then
      if i_speedpulse = '1' then
        speedpulseRegisterIncoming <= '1';
      elsif resetSpeedpulseReg = '1' then
        speedpulseRegisterIncoming <= '0';
      end if;
    end if;
  end process;

  -- purpose: saves incoming speedpule in internalClk-Domain
  --          to avoid metastable states
  -- type   : sequential
  -- inputs : i_sysClk, nInternalReset, i_speedpulse, resetSpeedpulseReg
  -- outputs: speedpulseReg (= speedpulseRegOut)
  speedpulseRegister: process (internalClk, nInternalReset)
  begin
    if nInternalReset = '0' then
      speedpulseReg <= (others => '0');
    elsif internalClk = '1' and internalClk'event then
      if resetSpeedpulseReg = '1' then
        speedpulseReg <= (others => '0');
      else
        speedpulseReg(1) <= speedpulseReg(0);
        speedpulseReg(0) <= speedpulseRegisterIncoming;
      end if;
    end if;
  end process;

  ----- testpins -----
  o_test_speedpulseRegOut <= speedpulseReg(1);
  speedpulseRegOut <= i_test_speedpulseRegOut;
  -- speedpulseRegOut <= speedpulseReg(1); -- without testpins


  -- purpose: save resetsignal for speedpulse register one cycle
  -- type   : sequential
  -- inputs : internalClk, nInternalReset, resetSpeedpulse
  -- outputs: resetSpeedpulseReg
  speedpulseResetRegister: process (internalClk, nInternalReset)
  begin
    if nInternalReset = '0' then
      resetSpeedpulseReg <= '0';
    elsif internalClk = '1' and internalClk'event then
      resetSpeedpulseReg <= resetSpeedpulse;
    end if;
  end process;


  ---------------------------------
  -- Synchronize Incoming Data --
```

```
  ------------------------------

  -- purpose: saves incoming databits in i_sysClk-Domain when i_speedpulse is 1
  -- type   : sequential
  -- inputs : i_sysClk, nInternalReset, i_speedpulse, i_dataBits
  -- outputs: dataIncomingReg
  incomingDataBitRegister: process (i_sysClk, nInternalResetSysClk)
  begin
    if nInternalResetSysClk = '0' then
      dataIncomingReg <= (others => '0');
    elsif i_sysClk = '1' and i_sysClk'event then
      if i_speedpulse = '1' then
        dataIncomingReg <= i_dataBits;
      end if;
    end if;
  end process;

  -- purpose: saves incoming data in internalClk-Domain
  --          to avoid metastable states
  -- type   : sequential
  -- inputs : internalClk, nInternalReset, dataIncomingReg
  -- outputs: dataReg2 (= dataRegOut)
  DataBitRegister: process (internalClk, nInternalReset)
  begin
    if nInternalReset = '0' then
      dataReg1 <= (others => '0');
      dataReg2 <= (others => '0');
    elsif internalClk = '1' and internalClk'event then
      dataReg1 <= dataIncomingReg;
      dataReg2 <= dataReg1;
    end if;
  end process;

  dataRegOut <= dataReg2; -- output of databit register


  -------------
  --  Timer  --
  -------------

  -- purpose: counts from STANDSTILL_TIMEOUT downto 0
  --          generate timerDone signal, when time is up
  -- type   : sequential
  -- inputs : i_sysClk, nInternalReset, signals
  -- outputs: timerDone
  timerCounter : process(internalClk, nInternalReset)
  begin
    if nInternalReset = '0' then      -- synchronous i_nReset sets signals to default
      timerCounterReg <= STANDSTILL_TIMEOUT;
      timerDone <= '0';

    elsif internalClk = '1' and internalClk'event then
      timerDone <= '0';
      if startTimer = '1' then            -- get new time from FSM
        timerCounterReg <= STANDSTILL_TIMEOUT; -- pass time to timer
      else
        if timerCounterReg = 0 then
          timerDone <= '1';
        else
          timerCounterReg <= timerCounterReg - 1;
        end if;-- timerCounterReg = 0
      end if;
    end if; -- internalClk
  end process timerCounter;


  ---------------------------
  -- manchesterShiftRegister --
  ---------------------------

  -- purpose: a shift Register that is set with setManchesterReg == 1
  --          and shifted with SHIFT_MAN_REG == 1
  --          sends o_datapulse_14mA(MSB) to FSM for Output
  -- type   : sequential
  -- inputs : i_sysClk, nInternalReset, internalClk, dataRegOut, shiftManchesterReg, setManchesterReg,
      setManchesterRegStandStill
  -- outputs: manchesterRegister
  manchesterShiftRegister : process(internalClk, nInternalReset)
    variable parity : std_logic;
  begin
    if nInternalReset = '0' then
      manchesterRegister <= (others => '0');

    elsif internalClk = '1' and internalClk'event then
      -- Calculate parity
```

```
      parity := dataRegOut(7) xor dataRegOut(6) xor dataRegOut(5) xor dataRegOut(4) xor dataRegOut(3) xor dataRegOut(2)
            xor dataRegOut(1) xor dataRegOut(0);

      -- Fill register with data bits
      if setManchesterReg = '1' then
        manchesterRegister <=      "00"     -- for Speed_Pulse
              & not dataRegOut(7) & dataRegOut(7)
              & not dataRegOut(6) & dataRegOut(6)
              & not dataRegOut(5) & dataRegOut(5)
              & not dataRegOut(4) & dataRegOut(4)
              & not dataRegOut(3) & dataRegOut(3)
              & not dataRegOut(2) & dataRegOut(2)
              & not dataRegOut(1) & dataRegOut(1)
              & not dataRegOut(0) & dataRegOut(0)
              & not parity        & parity     ;
      elsif setManchesterRegStandStill = '1' then
        manchesterRegister <= "11"          -- Standstill (send last databits again)
              & not dataRegOut(7) & dataRegOut(7)
              & not dataRegOut(6) & dataRegOut(6)
              & not dataRegOut(5) & dataRegOut(5)
              & not dataRegOut(4) & dataRegOut(4)
              & not dataRegOut(3) & dataRegOut(3)
              & not dataRegOut(2) & dataRegOut(2)
              & not dataRegOut(1) & dataRegOut(1)
              & not dataRegOut(0) & dataRegOut(0)
              & not parity        & parity     ;

      elsif shiftManchesterReg = '1' then       -- SHIFT Register
        manchesterRegister <= manchesterRegister(MAN_REG_MSB-1 downto 0) & '0';

      end if;-- setManchesterReg
    end if;-- i_nReset
  end process manchesterShiftRegister;

  -- testpins
  o_test_manchesterRegOut <= manchesterRegister(MAN_REG_MSB);
  manchesterRegOut <= i_test_manchesterRegOut;
  o_test_shiftManchesterReg <= shiftManchesterReg;

  -- manchesterRegOut <= manchesterRegister(MAN_REG_MSB); -- without testpins


  ------------------
  --  maskCounter  --
  ------------------

  -- purpose: A counter, that counts from iDataMask downto 0
  --          loadMaskCounter = 1 loads the counter
  --          decrMaskCounter = 1 decrements the counter
  --          maskCounterReg = 0 signals that the counter is 0
  -- type   : sequential
  -- inputs : i_sysClk, nInternalReset, loadMaskCounter, decrMaskCounter
  -- outputs: frameDone
  maskCounter : process(internalClk, nInternalReset)
  begin
    if nInternalReset = '0' then -- all outputs = 0 on synchronous i_nReset
      maskCounterReg <= x"0";
      maskCounterDone <= '0';

    elsif internalClk = '1' and internalClk'event then
      if loadMaskCounter = '1' then      -- SET FRAME
        maskCounterReg <= x"8";
        maskCounterDone <= '0';
      elsif maskCounterReg = 0 then        -- FRAME DONE
        maskCounterDone <= '1';
      elsif decrMaskCounter = '1' then    -- SUBTRACTION
        maskCounterDone <= '0';
        maskCounterReg <= maskCounterReg - 1;
      end if;-- loadMaskCounter
    end if;-- i_nReset
  end process maskCounter;


  ------------------
  --  akProtokollFSM --
  ------------------

  ---- purpose: Stateregister
  ---- type   : sequential
  ---- inputs : i_sysClk, internalClk, nInternalReset, nextAkProtokollState
  ---- outputs: akProtokollState
  akProtokollStateregister : process(internalClk, nInternalReset)
  begin
    if nInternalReset = '0' then -- Default akProtokollState is INIT, so Timer starts after PowerOn
      akProtokollState <= INIT;
```

```
505     elsif internalClk = '1' and internalClk'event then
          akProtokollState <= nextAkProtokollState;
        end if;-- i_nReset
      end process akProtokollStateregister;

510   -- TESTOUTPUT
      o_test_akState <= akProtokollState;

      -- purpose: Controles the output signals (see statediagram)
      -- type   : combinational
515   -- inputs : akProtokollState, timerDone, speedpulseRegOut, manchesterRegOut, maskCounterDone
      -- outputs: o_speedpulse_28mA, o_datapulse_14mA, startTimer, loadMaskCounter, decrMaskCounter, setManchesterReg,
          shiftManchesterReg, resetSpeedpulse, setManchesterRegStandStill
      -- states : INIT                : Initial state
      --          IDLE                : Idle state (Mealy)
      --          INITIAL_BIT_1       : Send first initial bit (befor speedpulse) (Mealy)
520   --          SPEEDPULSE_1        : Send first half of speedpulse (Mealy)
      --          SPEEDPULSE_2        : Send second half of speedpulse (Mealy)
      --          INITIAL_BIT_2       : Send second initial bit (after speedpulse) (Mealy)
      --          DATABIT_UNBREAKABLE : Send first half of data bit (next state MUST be DATABIT_BREAKABLE)
      --          DATABIT_BREAKABLE   : Send first half of data bit (Mealy)
525   akProtokollLogic : process(akProtokollState, timerDone, speedpulseRegOut, manchesterRegOut, maskCounterDone)
      begin
        -- set default values
        o_speedpulse_28mA    <= '0';
        o_datapulse_14mA     <= '0';
530     startTimer       <= '0';
        loadMaskCounter    <= '0';
        decrMaskCounter    <= '0';
        setManchesterReg   <= '0';
        shiftManchesterReg  <= '0';
535     resetSpeedpulse <= '0';
        setManchesterRegStandStill <= '0';

        case akProtokollState is

540       -- Initial state
          when INIT =>
            startTimer <= '1';
            loadMaskCounter <= '1';
            resetSpeedpulse <= '1';
545         nextAkProtokollState <= IDLE;

          -- Idle state (Mealy)
          when IDLE =>
            -- send data (if timerDone = 1 standstill)
550         if timerDone = '1'then
              setManchesterRegStandStill <= '1';  -- load manchesterregister (lower speedpulse, when speedpulseRegOut = 0)
              loadMaskCounter <= '1';    -- loadMask (send all 9 bits, when speedpulseRegOut = 0)
              nextAkProtokollState <= INITIAL_BIT_1;
            elsif speedpulseRegOut = '1' then
555           setManchesterReg <= '1';  -- load manchesterregister (lower speedpulse, when speedpulseRegOut = 0)
              loadMaskCounter <= '1';    -- loadMask (send all 9 bits, when speedpulseRegOut = 0)
              resetSpeedpulse <= '1'; -- Reset speedpulse register
              nextAkProtokollState <= INITIAL_BIT_1;
            else
560           nextAkProtokollState <= IDLE;
            end if;-- timerDone = '1' or speedpulseRegOut = '1'

          -- Send first initial bit (befor speedpulse) (Mealy)
          when INITIAL_BIT_1 =>
565         startTimer <= '1';  -- start timer, so after 150ms a new standstill will be sent
            nextAkProtokollState <= SPEEDPULSE_1;    --
            if timerDone <= '1' and speedpulseRegOut = '1' then --only occurs on STANDSTILL_TIMEOUT signal (start "normal"
                transmission)
              setManchesterReg <= '1';  -- pass Data to Shiftregister
              loadMaskCounter <= '1';    -- load Mask
570           resetSpeedpulse <= '1'; -- Reset speedpulse register
            end if;-- timerDone <= '1' and speedpulseRegOut = '1'

          -- Send first half of speedpulse (Mealy)
          when  SPEEDPULSE_1 =>
575         if speedpulseRegOut = '1' and manchesterRegOut = '1' then --only occurs on STANDSTILL signal (start "normal"
                transmission)
              setManchesterReg <= '1';  -- load Data into Shiftregister
              loadMaskCounter <= '1';    -- load mask
              resetSpeedpulse <= '1'; -- Reset speedpulse register
              nextAkProtokollState <= INITIAL_BIT_1;   -- Send first initial bit
580         else
              o_datapulse_14mA <= manchesterRegOut;   -- send pseudo speedpulse on standstill
              o_speedpulse_28mA <= not manchesterRegOut;    -- send Speedpule when not sandstill
              shiftManchesterReg <= '1';          -- next databit
              nextAkProtokollState <= SPEEDPULSE_2;
585         end if;
```

```
      -- Send second half of speedpulse (Mealy)
      when SPEEDPULSE_2 =>
        if speedpulseRegOut = '1' then -- only occurs on STANDSTILL signal (start "normal" transmission)
             ???????????????????? Hier noch "manchesterRegOut = '1' then" wie bei SPEEDPULSE_1 ?????????
          setManchesterReg <= '1';  -- load Data into Shiftregister
          loadMaskCounter <= '1';   -- load mask
          resetSpeedpulse <= '1'; -- Reset speedpulse register
          nextAkProtokollState <= INITIAL_BIT_1;   -- Send first initial bit
        else
          o_datapulse_14mA <= manchesterRegOut;    -- send pseudo speedpulse on standstill
          o_speedpulse_28mA <= not manchesterRegOut;  -- send Speedpule when not sandstill
          shiftManchesterReg <= '1';          -- next databit
          nextAkProtokollState <= INITIAL_BIT_2;
        end if;

      -- Send second initial bit (after speedpulse) (Mealy)
      when INITIAL_BIT_2 =>
        if speedpulseRegOut = '1' then -- only occurs on STANDSTILL signal (start "normal" transmission)
             ???????????????????? Hier noch "manchesterRegOut = '1' then" wie bei SPEEDPULSE_1 ?????????
          setManchesterReg <= '1';  -- load Data into Shiftregister
          loadMaskCounter <= '1';   -- load mask
          resetSpeedpulse <= '1'; -- Reset speedpulse register
          nextAkProtokollState <= INITIAL_BIT_1;   -- Send first initial bit
        else
          o_datapulse_14mA <= '0';
          nextAkProtokollState <= DATABIT_UNBREAKABLE;
        end if;

      -- Send first half of data bit (next state MUST be DATABIT_BREAKABLE)
      when DATABIT_UNBREAKABLE =>
        o_datapulse_14mA <= manchesterRegOut;    -- send first half of databit
        shiftManchesterReg <= '1';               -- next databit
        nextAkProtokollState <= DATABIT_BREAKABLE;

      -- Send first half of data bit (Mealy)
      when DATABIT_BREAKABLE =>
        o_datapulse_14mA <= manchesterRegOut;    -- send second half of databit
        shiftManchesterReg <= '1';          -- next databit
        decrMaskCounter <= '1';           -- decrement mask counter
        if speedpulseRegOut = '1' then          -- interrupt transmission at new incoming speedpulse
          setManchesterReg <= '1';        -- pass Data to Shiftregister
          loadMaskCounter <= '1';         -- load mask
          resetSpeedpulse <= '1'; -- Reset speedpulse register
          nextAkProtokollState <= INITIAL_BIT_1;
        elsif maskCounterDone = '1' then        -- no more bits to send
          nextAkProtokollState <= IDLE;         -- !!!TIMER is still counting down the 150ms!!!
        else
          nextAkProtokollState <= DATABIT_UNBREAKABLE;        -- there is still Data to be send? SEND IT FOR GLORY!!!
        end if;

      -- needed since state is "handcoded"
      when others =>
        nextAkProtokollState <= INIT;

    end case;
  end process akProtokollLogic;


end behavioral;
```

Quelltext B.1: abs_manchester_encoder.vhd
– Protokollgenerator. Blockschaltbild: Abbildung B.1; Zustandsdiagramm: Abbildung B.2

B.2 Digitale Signalverarbeitung (diagnosis_top.vhd)

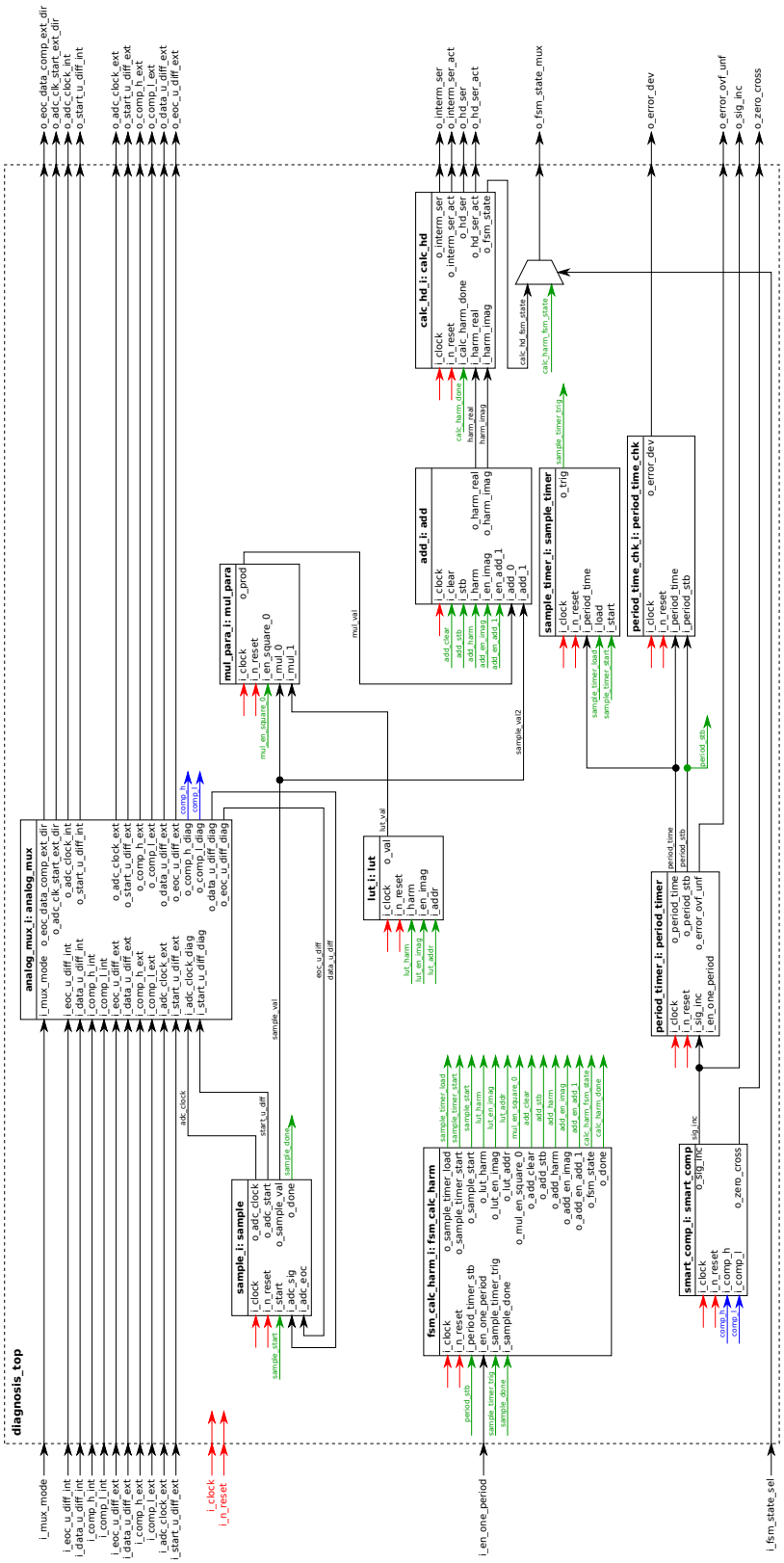


Abbildung B.3: Blockschaltbild der Digitalen Signalverarbeitung. – Siehe Quelltext B.2.

```
-------------------------------------------------------------------------------
-- Entity: diagnosis_top
-------------------------------------------------------------------------------
-- Copyright 2012
-- Filename           : diagnosis_top.vhd
-- Creation date      : 2012-06-25
-- Author(s)          : Martin Krey
-- Version            : 1.00
-- Description        : Top Level of Diagnosis Logic Block
--
-------------------------------------------------------------------------------
-- File History:
-- Date          Version  Author   Comment
-------------------------------------------------------------------------------
library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

package diagnosis_top_pkg is

  component diagnosis_top
    port(
            i_clock    : in std_logic;
            i_n_reset  : in std_logic;

            ---------------------------------------
            -- connection to internal analog module
            ---------------------------------------
            o_adc_clock_int    : out std_logic;
            o_start_u_diff_int : out std_logic;
            i_eoc_u_diff_int   : in std_logic;
            i_data_u_diff_int  : in unsigned(9 downto 0);

            i_comp_h_int : in std_logic;
            i_comp_l_int : in std_logic;

            ---------------------------------------
            -- connection to external analog module
            -- using bidirectional pins
            ---------------------------------------
            -- changes mux mode
            i_mux_mode : in unsigned(1 downto 0);

            -- direction bits (0 = output)
            o_eoc_data_comp_ext_dir : out std_logic;
            o_adc_clk_start_ext_dir : out std_logic;

            -- configuration for external analog module
            o_adc_clock_ext    : out std_logic;
            o_start_u_diff_ext : out std_logic;
            i_eoc_u_diff_ext   : in std_logic;
            i_data_u_diff_ext  : in unsigned(9 downto 0);

            i_comp_h_ext : in std_logic;
            i_comp_l_ext : in std_logic;

            -- configuration for testing internal analog
            i_adc_clock_ext    : in std_logic;
            i_start_u_diff_ext : in std_logic;
            o_eoc_u_diff_ext   : out std_logic;
            o_data_u_diff_ext  : out unsigned(9 downto 0);

            o_comp_h_ext : out std_logic;
            o_comp_l_ext : out std_logic;

            ---------------------------------------

            i_en_one_period : in std_logic;

            o_sig_inc    : out std_logic;
            o_zero_cross : out std_logic;

            -- fsm state
            i_fsm_state_sel : in std_logic;
            o_fsm_state_mux : out unsigned(4 downto 0);

            -- serial intermediate results output
            o_interm_ser     : out std_logic;
            o_interm_ser_act : out std_logic;

            -- serial hd output
            o_hd_ser     : out std_logic;
            o_hd_ser_act : out std_logic;

            -- error output
```

```
            o_error_dev     : out std_logic;
            o_error_ovf_unf : out std_logic
    );
  end component;

end package;


-------------------------------------------------------------------------------

library ieee;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;
library work;
    use work.global_pkg.all;
    use work.analog_mux_pkg.all;
    use work.smart_comp_pkg.all;
    use work.period_timer_pkg.all;
    use work.sample_timer_pkg.all;
    use work.sample_pkg.all;
    use work.fsm_calc_harm_pkg.all;
    use work.lut_pkg.all;
    use work.mul_para_pkg.all;
    use work.add_pkg.all;
    use work.period_time_chk_pkg.all;
    use work.calc_hd_pkg.all;

entity diagnosis_top is
    port(
            i_clock    : in std_logic;
            i_n_reset  : in std_logic;

            ---------------------------------------
            -- connection to internal analog module
            ---------------------------------------
            o_adc_clock_int     : out std_logic;
            o_start_u_diff_int  : out std_logic;
            i_eoc_u_diff_int    : in std_logic;
            i_data_u_diff_int   : in unsigned(9 downto 0);

            i_comp_h_int : in std_logic;
            i_comp_l_int : in std_logic;

            ---------------------------------------
            -- connection to external analog module
            -- using bidirectional pins
            ---------------------------------------
            -- changes mux mode
            i_mux_mode : in unsigned(1 downto 0);

            -- direction bits (0 = output)
            o_eoc_data_comp_ext_dir : out std_logic;
            o_adc_clk_start_ext_dir : out std_logic;

            -- configuration for external analog module
            o_adc_clock_ext     : out std_logic;
            o_start_u_diff_ext  : out std_logic;
            i_eoc_u_diff_ext    : in std_logic;
            i_data_u_diff_ext   : in unsigned(9 downto 0);

            i_comp_h_ext : in std_logic;
            i_comp_l_ext : in std_logic;

            -- configuration for testing internal analog
            i_adc_clock_ext     : in std_logic;
            i_start_u_diff_ext  : in std_logic;
            o_eoc_u_diff_ext    : out std_logic;
            o_data_u_diff_ext   : out unsigned(9 downto 0);

            o_comp_h_ext : out std_logic;
            o_comp_l_ext : out std_logic;

            ---------------------------------------

            i_en_one_period : in std_logic;

            o_sig_inc     : out std_logic;
            o_zero_cross  : out std_logic;

            -- fsm state
            i_fsm_state_sel : in std_logic;
            o_fsm_state_mux : out unsigned(4 downto 0);

            -- serial intermediate results output
```

```
            o_interm_ser     : out std_logic;
            o_interm_ser_act : out std_logic;

            -- serial hd output
            o_hd_ser     : out std_logic;
            o_hd_ser_act : out std_logic;

            -- error output
            o_error_dev     : out std_logic;
            o_error_ovf_unf : out std_logic
    );
end diagnosis_top;

architecture arch of diagnosis_top is
--    signal n_reset : std_logic;
    signal clock    : std_logic;
    signal adc_clock : std_logic;

    signal comp_h : std_logic;
    signal comp_l : std_logic;

    signal start_u_diff : std_logic;
    signal eoc_u_diff   : std_logic;
    signal data_u_diff  : unsigned(9 downto 0);

    signal sig_inc : std_logic;

    signal period_time : unsigned(24 downto 0);
    signal period_stb : std_logic;

    signal sample_timer_trig : std_logic;
    signal sample_timer_load : std_logic;
    signal sample_timer_start : std_logic;

    signal sample_start : std_logic;
    signal sample_done : std_logic;
    signal sample_val : signed(9 downto 0);
    signal sample_val2 : signed(19 downto 0);

    signal lut_harm : unsigned(3 downto 0);
    signal lut_addr : unsigned(5 downto 0);
    signal lut_en_imag : std_logic;
    signal lut_val : signed(9 downto 0);

    signal mul_en_square_0 : std_logic;
    signal mul_val : signed(19 downto 0);

    signal add_clear : std_logic;
    signal add_stb : std_logic;
    signal add_harm : unsigned(2 downto 0);
    signal add_en_imag : std_logic;
    signal add_en_add_1 : std_logic;

    signal calc_harm_done : std_logic;

    signal harm_real : harm_array_t;
    signal harm_imag : harm_array_t;

    signal calc_harm_fsm_state : unsigned(4 downto 0);
    signal calc_hd_fsm_state   : unsigned(3 downto 0);

begin

    analog_mux_i : analog_mux
        port map(
            -- changes mux mode
            i_mux_mode => i_mux_mode,

            -- direction bits (0 = output)
            o_eoc_data_comp_ext_dir => o_eoc_data_comp_ext_dir,
            o_adc_clk_start_ext_dir => o_adc_clk_start_ext_dir,

            -- connection to diagnosis module
            i_adc_clock_diag    => adc_clock,
            i_start_u_diff_diag => start_u_diff,
            o_eoc_u_diff_diag   => eoc_u_diff,
            o_data_u_diff_diag  => data_u_diff,

            o_comp_h_diag => comp_h,
            o_comp_l_diag => comp_l,

            -- connection to internal analog module
            o_adc_clock_int    => o_adc_clock_int,
            o_start_u_diff_int => o_start_u_diff_int,
            i_eoc_u_diff_int   => i_eoc_u_diff_int,
```

```
            i_data_u_diff_int  => i_data_u_diff_int,

            i_comp_h_int => i_comp_h_int,
            i_comp_l_int => i_comp_l_int,

            -- connection to external analog module
            o_adc_clock_ext    => o_adc_clock_ext,
            o_start_u_diff_ext => o_start_u_diff_ext,
            i_eoc_u_diff_ext   => i_eoc_u_diff_ext,
            i_data_u_diff_ext  => i_data_u_diff_ext,

            i_comp_h_ext => i_comp_h_ext,
            i_comp_l_ext => i_comp_l_ext,

            -- connection for testing internal analog
            i_adc_clock_ext    => i_adc_clock_ext,
            i_start_u_diff_ext => i_start_u_diff_ext,
            o_eoc_u_diff_ext   => o_eoc_u_diff_ext,
            o_data_u_diff_ext  => o_data_u_diff_ext,

            o_comp_h_ext => o_comp_h_ext,
            o_comp_l_ext => o_comp_l_ext
        );


    smart_comp_i : smart_comp
        port map(
            i_clock => i_clock,
            i_n_reset => i_n_reset,

            i_comp_h => comp_h,
            i_comp_l => comp_l,

            o_sig_inc => sig_inc,
            o_zero_cross => o_zero_cross
        );

    period_timer_i : period_timer
        port map(
            i_clock => i_clock,
            i_n_reset => i_n_reset,

            i_sig_inc => sig_inc,
            i_en_one_period => i_en_one_period,

            o_period_time => period_time,
            o_period_stb => period_stb,

            o_error_ovf_unf => o_error_ovf_unf
        );

    sample_timer_i : sample_timer
        generic map(
            sample_interval_correction => 4
            )
        port map(
            i_clock => i_clock,
            i_n_reset => i_n_reset,

            i_period_time => period_time,
            i_load => sample_timer_load,
            i_start => sample_timer_start,

            o_trig => sample_timer_trig
        );

    sample_i : sample
        generic map(
            adc_bias_val => 511
            )
        port map(
            i_clock => i_clock,
            i_n_reset => i_n_reset,

            i_start => sample_start,

            i_adc_sig => data_u_diff,
            i_adc_eoc => eoc_u_diff,
            o_adc_clock => adc_clock,
            o_adc_start => start_u_diff,

            o_sample_val => sample_val,
            o_done => sample_done
        );
```

```
    lut_i : lut
        port map(
            i_clock => i_clock,
            i_n_reset => i_n_reset,
            i_harm => lut_harm,
            i_en_imag => lut_en_imag,
            i_addr => lut_addr,

            o_val => lut_val
        );

    mul_para_i : mul_para
        generic map(
            mul_width => 10
        )
        port map(
            i_clock => i_clock,
            i_n_reset => i_n_reset,
            i_en_square_0 => mul_en_square_0,
            i_mul_0 => sample_val,
            i_mul_1 => lut_val,

            o_prod => mul_val
        );

    add_i : add
        port map(
            i_clock => i_clock,

            i_clear => add_clear,
            i_stb    => add_stb,
            i_harm   => add_harm,
            i_en_imag  => add_en_imag,
            i_en_add_1 => add_en_add_1,

            i_add_0 => mul_val,
            i_add_1 => sample_val2,

            o_harm_real => harm_real,
            o_harm_imag => harm_imag
        );

    fsm_calc_harm_i : fsm_calc_harm
        port map(
            i_clock => i_clock,
            i_n_reset => i_n_reset,

            i_period_timer_stb => period_stb,
            i_en_one_period => i_en_one_period,

            i_sample_timer_trig  => sample_timer_trig,
            o_sample_timer_load  => sample_timer_load,
            o_sample_timer_start => sample_timer_start,

            i_sample_done  => sample_done,
            o_sample_start => sample_start,

            o_lut_harm     => lut_harm,
            o_lut_en_imag => lut_en_imag,
            o_lut_addr     => lut_addr,

            o_mul_en_square_0 => mul_en_square_0,

            o_add_clear     => add_clear,
            o_add_stb       => add_stb,
            o_add_harm      => add_harm,
            o_add_en_imag  => add_en_imag,
            o_add_en_add_1 => add_en_add_1,

            o_fsm_state => calc_harm_fsm_state,

            o_done => calc_harm_done
        );

    period_time_chk_i : period_time_chk
        port map(
            i_clock => i_clock,
            i_n_reset => i_n_reset,

            i_period_time => period_time,
            i_period_stb => period_stb,

            o_error_dev => o_error_dev
        );
```

```
    calc_hd_i : calc_hd
        port map(
            i_clock => i_clock,
            i_n_reset => i_n_reset,

            i_calc_harm_done => calc_harm_done,

            i_harm_real => harm_real,
            i_harm_imag => harm_imag,

            o_interm_ser     => o_interm_ser,
            o_interm_ser_act => o_interm_ser_act,

            o_fsm_state      => calc_hd_fsm_state,

            o_hd_ser         => o_hd_ser,
            o_hd_ser_act     => o_hd_ser_act
        );

    sample_val2 <= to_signed(to_integer(sample_val),20);

    o_fsm_state_mux <= calc_harm_fsm_state when i_fsm_state_sel = '0' else
                       ('0' & calc_hd_fsm_state);

    o_sig_inc <= sig_inc;

end arch;
```

Quelltext B.2: diagnosis_top.vhd – Top-Level Entity der digitalen Signalverarbeitung. Siehe Abbildung B.3.

B.2.1 add.vhd

```
--------------------------------------------------------------------------------
-- Entity: add
--------------------------------------------------------------------------------
-- Copyright 2012
-- Filename           : add.vhd
-- Creation date      : 2012-06-07
-- Author(s)          : Martin Krey
-- Version            : 1.00
-- Description        :
--                      harmonic calculation
--------------------------------------------------------------------------------
-- File History:
-- Date          Version  Author   Comment
--------------------------------------------------------------------------------
library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;
    use work.global_pkg.all;

package add_pkg is

--    type harm_array_t is array(0 to 5) of signed(25 downto 0);

  component add
    port(
        i_clock   : in std_logic;

        i_clear    : in std_logic;
        i_stb      : in std_logic;
        i_harm     : in unsigned(2 downto 0);
        i_en_imag  : in std_logic;
        i_en_add_1 : in std_logic;

        i_add_0 : in signed(19 downto 0);
        i_add_1 : in signed(19 downto 0);

        o_harm_real : out harm_array_t;
        o_harm_imag : out harm_array_t
    );
  end component;

end package;
```

```
47  library ieee;
        use ieee.std_logic_1164.all;
        use ieee.numeric_std.all;
        use work.global_pkg.all;

52  entity add is
        port(
            i_clock    : in std_logic;

            i_clear    : in std_logic;
57          i_stb      : in std_logic;
            i_harm     : in unsigned(2 downto 0);
            i_en_imag  : in std_logic;
            i_en_add_1 : in std_logic;

62          i_add_0 : in signed(19 downto 0);
            i_add_1 : in signed(19 downto 0);

            o_harm_real : out harm_array_t;
            o_harm_imag : out harm_array_t
67      );
    end add;

    architecture arch of add is

72      signal harm_real : harm_array_t;
        signal harm_imag : harm_array_t;

        signal summand : signed(19 downto 0);

77      signal harm_idx : integer range 0 to 5;

    begin

        process(i_clock, i_clear)
82      begin
            if i_clear = '1' then
                -- clear all values
                for i in o_harm_real'range loop
                    harm_real(i) <= (others => '0');
87                  harm_imag(i) <= (others => '0');
                end loop;
            elsif i_clock = '1' and i_clock'event then
                if i_stb = '1' then
                    if i_en_imag = '0' then
92                      for i in o_harm_real'range loop
                            if harm_idx = i then
                                harm_real(i) <= harm_real(i) + summand;
                            end if;
                        end loop;
97                  elsif i_en_imag = '1' then
                        for i in o_harm_imag'range loop
                            if harm_idx = i then
                                harm_imag(i) <= harm_imag(i) + summand;
                            end if;
102                     end loop;
                    end if;
                end if;
            end if;
        end process;
107
        harm_idx <= to_integer(i_harm);

        summand <= i_add_1 when i_en_add_1 = '1' else
                   i_add_0;
112
        o_harm_real <= harm_real;
        o_harm_imag <= harm_imag;
    end arch;
```

Quelltext B.3: add.vhd – Aufsummieren der mit dem Sinus/Kosinus gewichteten Samples (DFT).

B.2.2 analog_mux.vhd

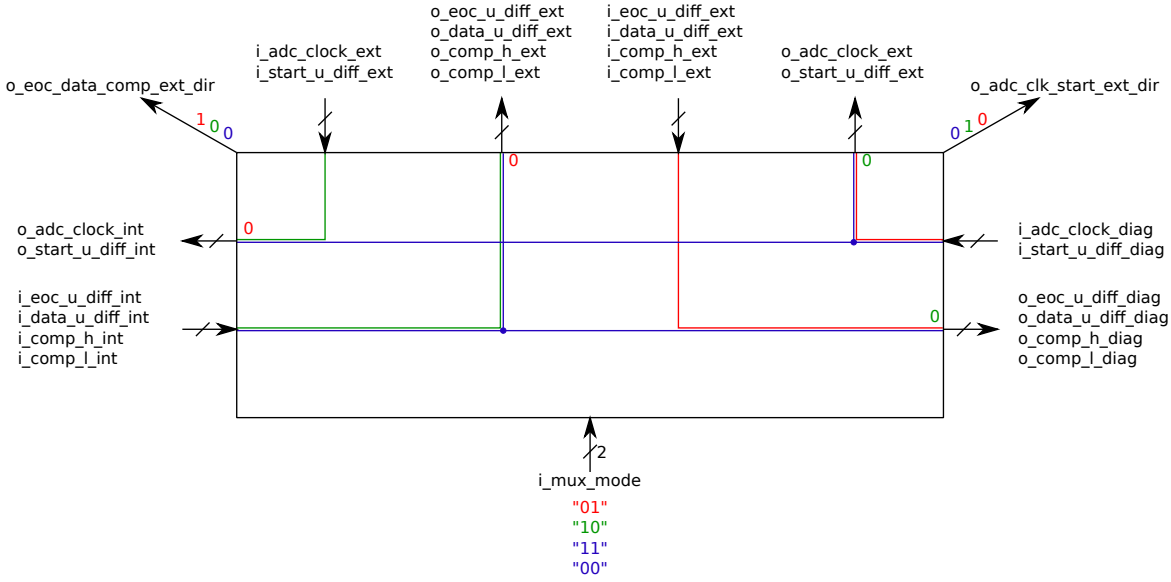


Abbildung B.4: Blockschaltbild des Multiplexers. – Siehe Abschnitt 3.2.2 und Quelltext B.4.

```
-------------------------------------------------------------------------------
-- Title      : Bidirectional Mux
-- Project    : ESZ-ABS
-------------------------------------------------------------------------------
-- File       : analog_mux.vhd
-- Author     : Daniel Sabotta, Martin Krey
-- Company    : HAW Hamburg
-- Created    : 2012-06-26
-- Last update: 2012-06-26
-------------------------------------------------------------------------------
-- Description: ...
-------------------------------------------------------------------------------
-- Copyright (c) 2012
-------------------------------------------------------------------------------

library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

package analog_mux_pkg is

  component analog_mux
    port(
            -- changes mux mode
            i_mux_mode : in unsigned(1 downto 0);

            -- direction bits (0 = output)
            o_eoc_data_comp_ext_dir : out std_logic;
            o_adc_clk_start_ext_dir : out std_logic;

            -- connection to diagnosis module
            i_adc_clock_diag     : in std_logic;
            i_start_u_diff_diag  : in std_logic;
            o_eoc_u_diff_diag    : out std_logic;
            o_data_u_diff_diag   : out unsigned(9 downto 0);

            o_comp_h_diag : out std_logic;
            o_comp_l_diag : out std_logic;

            -- connection to internal analog module
            o_adc_clock_int     : out std_logic;
            o_start_u_diff_int : out std_logic;
```

```
            i_eoc_u_diff_int    : in std_logic;
            i_data_u_diff_int   : in unsigned(9 downto 0);

            i_comp_h_int : in std_logic;
            i_comp_l_int : in std_logic;

            -- connection to external analog module
            o_adc_clock_ext     : out std_logic;
            o_start_u_diff_ext : out std_logic;
            i_eoc_u_diff_ext    : in std_logic;
            i_data_u_diff_ext   : in unsigned(9 downto 0);

            i_comp_h_ext : in std_logic;
            i_comp_l_ext : in std_logic;

            -- connection for testing internal analog
            i_adc_clock_ext     : in std_logic;
            i_start_u_diff_ext : in std_logic;
            o_eoc_u_diff_ext    : out std_logic;
            o_data_u_diff_ext   : out unsigned(9 downto 0);

            o_comp_h_ext : out std_logic;
            o_comp_l_ext : out std_logic
    );
  end component;

end package;

-------------------------------------------------------------------------------

library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

entity analog_mux is

    port(
            -- changes mux mode
            i_mux_mode : in unsigned(1 downto 0);

            -- direction bits (0 = output)
            o_eoc_data_comp_ext_dir : out std_logic;
            o_adc_clk_start_ext_dir : out std_logic;

            -- connection to diagnosis module
            i_adc_clock_diag    : in std_logic;
            i_start_u_diff_diag : in std_logic;
            o_eoc_u_diff_diag   : out std_logic;
            o_data_u_diff_diag  : out unsigned(9 downto 0);

            o_comp_h_diag : out std_logic;
            o_comp_l_diag : out std_logic;

            -- connection to internal analog module
            o_adc_clock_int     : out std_logic;
            o_start_u_diff_int : out std_logic;
            i_eoc_u_diff_int    : in std_logic;
            i_data_u_diff_int   : in unsigned(9 downto 0);

            i_comp_h_int : in std_logic;
            i_comp_l_int : in std_logic;

            -- connection to external analog module
            o_adc_clock_ext     : out std_logic;
            o_start_u_diff_ext : out std_logic;
            i_eoc_u_diff_ext    : in std_logic;
            i_data_u_diff_ext   : in unsigned(9 downto 0);

            i_comp_h_ext : in std_logic;
            i_comp_l_ext : in std_logic;

            -- connection for testing internal analog
            i_adc_clock_ext     : in std_logic;
            i_start_u_diff_ext : in std_logic;
            o_eoc_u_diff_ext    : out std_logic;
            o_data_u_diff_ext   : out unsigned(9 downto 0);

            o_comp_h_ext : out std_logic;
            o_comp_l_ext : out std_logic
    );
end analog_mux;

architecture arch of analog_mux is
begin
```

```
129     process(i_mux_mode,
                i_adc_clock_diag, i_start_u_diff_diag,
                i_eoc_u_diff_int, i_data_u_diff_int, i_comp_h_int, i_comp_l_int,
                i_eoc_u_diff_ext, i_data_u_diff_ext, i_comp_h_ext, i_comp_l_ext, i_adc_clock_ext, i_start_u_diff_ext
                )
134     begin

            o_eoc_u_diff_diag <= '0';
            o_data_u_diff_diag <= (others => '0');
            o_comp_h_diag <= '0';
139         o_comp_l_diag <= '0';

            o_adc_clock_int <= '0';
            o_start_u_diff_int <= '0';

144         o_adc_clk_start_ext_dir <= '1';     -- default is input
            o_adc_clock_ext <= '0';
            o_start_u_diff_ext <= '0';

            o_eoc_data_comp_ext_dir <= '1';     -- default is input
149         o_eoc_u_diff_ext <= '0';
            o_data_u_diff_ext <= (others => '0');
            o_comp_h_ext <= '0';
            o_comp_l_ext <= '0';

154         if i_mux_mode = "01" then
                ------------------------------
                -- external analog mode
                -- diag <-> ext
                ------------------------------
159

                o_eoc_u_diff_diag <= i_eoc_u_diff_ext;
                o_data_u_diff_diag <= i_data_u_diff_ext;
                o_comp_h_diag <= i_comp_h_ext;
                o_comp_l_diag <= i_comp_l_ext;
164

                o_adc_clock_int <= '0';
                o_start_u_diff_int <= '0';

                o_adc_clk_start_ext_dir <= '0';     -- enable outputs
169             o_adc_clock_ext <= i_adc_clock_diag;
                o_start_u_diff_ext <= i_start_u_diff_diag;

                o_eoc_data_comp_ext_dir <= '1';
                o_eoc_u_diff_ext <= '0';
174             o_data_u_diff_ext <= (others => '0');
                o_comp_h_ext <= '0';
                o_comp_l_ext <= '0';

            elsif i_mux_mode = "10" then
179             ------------------------------
                -- internal analog testing mode
                -- ext <-> int
                ------------------------------

184             o_eoc_u_diff_diag <= '0';
                o_data_u_diff_diag <= (others => '0');
                o_comp_h_diag <= '0';
                o_comp_l_diag <= '0';

189             o_adc_clock_int <= i_adc_clock_ext;
                o_start_u_diff_int <= i_start_u_diff_ext;

                o_adc_clk_start_ext_dir <= '1';
                o_adc_clock_ext <= '0';
194             o_start_u_diff_ext <= '0';

                o_eoc_data_comp_ext_dir <= '0';     -- enable outputs
                o_eoc_u_diff_ext <= i_eoc_u_diff_int;
                o_data_u_diff_ext <= i_data_u_diff_int;
199             o_comp_h_ext <= i_comp_h_int;
                o_comp_l_ext <= i_comp_l_int;

            else  --  i_mux_mode "00" or "11"
                ------------------------------
204             -- usual/sniffing mode
                -- diag <-> int
                -- diag/int -> ext (only outputs)
                ------------------------------

209             o_eoc_u_diff_diag <= i_eoc_u_diff_int;
                o_data_u_diff_diag <= i_data_u_diff_int;
                o_comp_h_diag <= i_comp_h_int;
                o_comp_l_diag <= i_comp_l_int;
```

```
214             o_adc_clock_int <= i_adc_clock_diag;
                o_start_u_diff_int <= i_start_u_diff_diag;

                -- sniffing outputs to external
                o_adc_clk_start_ext_dir <= '0';       -- enable outputs
219             o_adc_clock_ext <= i_adc_clock_diag;
                o_start_u_diff_ext <= i_start_u_diff_diag;

                o_eoc_data_comp_ext_dir <= '0';       -- enable outputs
                o_eoc_u_diff_ext <= i_eoc_u_diff_int;
224             o_data_u_diff_ext <= i_data_u_diff_int;
                o_comp_h_ext <= i_comp_h_int;
                o_comp_l_ext <= i_comp_l_int;
            end if;

229     end process;

end arch;
```

Quelltext B.4: analog_mux.vhd – Multiplexer für die Signale zwischen der digitalen und der analogen Signalverarbeitung für Testzwecke. Siehe Abbildung 3.6 und B.4.

B.2.3 calc_hd_add_samples.vhd

```
-------------------------------------------------------------------------------
-- Title      : calc_hd_add_samples
-- Project    : ESZ-ABS
-------------------------------------------------------------------------------
-- File       : calc_hd_add_samples.vhdl
-- Author     : Daniel Sabotta
--              based on: Jan-Heiner Dreschhoff
-- Company    : HAW Hamburg
-- Created    : 2012-06-28
-- Last update: 2012-06-29
-------------------------------------------------------------------------------
-- Description: Calculates the sum of the magnitude of the incoming complex
--              signal for hd5. The first complex signal for hdi!!
-------------------------------------------------------------------------------
-- Copyright (c) 2012
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version   Author      Description
--             0         Dreschhoff  init
-- 2012-06-28  1         Sabotta     remove Debugsignals and enable(CLK) and
--                                   XML-Comments
--                                   inserting the package declaration
-- 2012-06-29  1         Sabotta     inserting generic parameters
--                                   prepared for hdi: inserting extra outputs
--                                                     after first square
-- 2012-07-05  1         Sabotta     index problem 01F9D
--
-------------------------------------------------------------------------------

-- ################################## Package ###################################

library ieee;
   use ieee.std_logic_1164.all;
   use ieee.numeric_std.all;

library work;
  use work.global_pkg.all;

package calc_hd_add_samples_pkg is
  component calc_hd_add_samples is
    generic(
      out_width : integer := 44;
      square_val_in_width : integer := 20;    -- signed
      square_result_width : integer := 40     -- unsigned
    );
    port(
      i_sysclk       : in std_logic;
      i_n_reset      : in std_logic;

      i_start        : in std_logic;

      o_done         : out std_logic;

      i_harm_re      : in harm_array_t;
      i_harm_im      : in harm_array_t;

      NUMERATOR      : out unsigned(out_width-1 downto 0);
      DENOMINATOR    : out unsigned(out_width-1 downto 0);

      o_hdi_p1       : out unsigned(out_width-1 downto 0);
      o_hdi_p1_done  : out std_logic;

      o_hdi_gl       : out unsigned(out_width-1 downto 0);
      o_hdi_gl_done  : out std_logic
    );
  end component;
end calc_hd_add_samples_pkg;

-- ################################ end Package #################################

library ieee;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;

library work;
  use work.global_pkg.all;
  --~ use work.functions_pkg.all;
  --~ use work.operators_pkg.all;
  use work.calc_hd_square_pkg.all;

entity calc_hd_add_samples is
    generic(
```

```
83          out_width : integer := 44;
            square_val_in_width : integer := 20;   -- signed
            square_result_width : integer := 40   -- unsigned
        );
        port(
88          i_sysclk       : in std_logic;
            i_n_reset      : in std_logic;

            -- reset all internal signals and start calculation
            -- (to start a new calculation i_start must be 1 for one cycle)
93          i_start        : in std_logic;

            -- o_done is 1 if the whole calcualtion is completed (until i_start is 1)
            o_done         : out std_logic;

98          -- harmonic input
            i_harm_re      : in harm_array_t;
            i_harm_im      : in harm_array_t;

            -- result for hd5 calculation
103         -- NUMERATOR: sum of square of the absolute value of the harmonics 2 to 5
            -- DENOMINATOR: sum of square of the absolute value of  all harmonics
            NUMERATOR      : out unsigned(out_width-1 downto 0);
            DENOMINATOR    : out unsigned(out_width-1 downto 0);

108         -- square of the absolute value of the 0th harmonic (for hdi)
            o_hdi_p1       : out unsigned(out_width-1 downto 0);
            -- o_hdi_p1_done is 1 if o_hdi_p1_done is calculated
            o_hdi_p1_done : out std_logic;

113         -- suqare of i_harm_im[0]
            o_hdi_gl       : out unsigned(out_width-1 downto 0);
            -- o_hdi_gl_done is 1 if o_hdi_gl is calculated
            o_hdi_gl_done : out std_logic
        );
118 end calc_hd_add_samples;

    architecture behavior of calc_hd_add_samples is

      -- index of harmonics
123   signal index      : integer range 0 to 5;

      -- change between real and imaginary part of each harmonic
      signal real_not_im    : std_logic;

128   -- result of square
      signal square_result        : unsigned(square_result_width-1 downto 0);

      -- internal sum of the absolute value of the harmonics 2 to 5
      signal u25_sig       : unsigned(out_width-1 downto 0);
133   -- internal sum of the absolute valur of all harmonics
      signal uall_sig      : unsigned(out_width-1 downto 0);

      -- square_done is 1 if the square is calculated
      signal square_done  : std_logic;
138   --signal set_square   : std_logic;
      -- done_sig is 1 if the whole calculation is finished
      signal done_sig      : std_logic;
      --signal all_done_sig : std_logic;
      --  use each square only one time
143   signal once          : std_logic;

      -- square algorithm input
      signal square_in        : signed(square_val_in_width-1 downto 0);

148   -- some zeros to add up the squares
      signal zeros_for_adding : unsigned(out_width - square_result_width -1 downto 0);


      -- square of the absolute value of the 0th harmonic (for hdi)
153   signal hdi_p1         : unsigned(out_width-1 downto 0);
      -- o_hdi_p1_done is 1 if o_hdi_p1_done is calculated
      signal hdi_p1_done    : std_logic;
      -- suqare of i_harm_im[0]
      signal hdi_gl         : unsigned(out_width-1 downto 0);
158   -- o_hdi_gl_done is 1 if o_hdi_gl is calculated
      signal hdi_gl_done    : std_logic;

      -- start the square algorithm
      signal square_start             : std_logic;
163   -- initialize the square algorithm
      signal sqare_init : std_logic;

    begin
```

```
168   -- assign intenal signals to outputs
      NUMERATOR <= u25_sig;
      DENOMINATOR <= uall_sig;
      o_hdi_p1 <= hdi_p1;
      o_hdi_p1_done <= hdi_p1_done;
173   o_hdi_gl_done <= hdi_gl_done;
      o_done <= done_sig;

      -- fill eith zeros
      zeros_for_adding <= (others => '0');
178

      ------------------
      -- Control Logic --
      ------------------
183

      -- purpose: add up the squares and controls the square algorithm
      -- type   : sequential
      -- inputs : i_sysClk, nInternalReset, i_start, zeros_for_adding, square_result ...
      -- outputs: index, uall_sig, u25_sig, hdi_p1, hdi_p1_done, hdi_gl, hdi_gl_done, done_sig, square_start,  ...
188   ADD_DUMMY : process(i_sysclk, i_n_reset)
        variable temp_add   : unsigned(out_width-1 downto 0);
      begin
        if i_n_reset = '0' then
          index     <= 0 ;
193       real_not_im  <= '0';
          once      <= '0';
          uall_sig <= (others => '0');
          u25_sig  <= (others => '0');
          hdi_p1   <= (others => '0');
198       hdi_p1_done <= '0';
          done_sig <= '0';
          hdi_gl_done <= '0';
          o_hdi_gl      <= (others => '0');
          square_start <= '0';
203
        elsif i_sysclk = '1' and i_sysclk'event then
          square_start <= '0';

          -- reset internal signals and start the square algorithm
208       if i_start = '1' then
            once      <= '0';
            index   <= 0;
            real_not_im <= '0';
            uall_sig <= (others => '0');
213         u25_sig  <= (others => '0');
            hdi_p1 <= (others => '0');
            hdi_p1_done <= '0';
            o_hdi_gl      <= (others => '0');
            hdi_gl_done <= '0';
218         done_sig <= '0';
            square_start <= '1';

          -- suare is done and not all squares are added
          elsif square_done = '1' and done_sig = '0' then
223         -- start next suare
            square_start <= '1';

            -- add each square only one time
            if once = '0' then
228           -- change between imaginary and real part of the harmonic
              real_not_im <= not real_not_im;

              ---------------------
              -- hdi calculation --
233           ---------------------
              -- at first calculate the hdi_gl
              if hdi_gl_done = '0' then
                -- o_hdi_gl is the square of i_harm_imag[0]
                if real_not_im = '0' then
238               o_hdi_gl <= (zeros_for_adding & square_result);
                else
                  index <= 1;
                  hdi_gl_done <= '1';
                end if;
243
              -- then calculate hdi_p1
              elsif hdi_p1_done = '0' then
                -- hdi_p1 is the sum of squares of i_harm_re[1] and i_harm_imag[1]
                hdi_p1 <= hdi_p1 + (zeros_for_adding & square_result);
248             -- after adding the square of the real part start the hd5 calculation
                if real_not_im = '1' then
                  -- begin with the 5th harmonic
                  index <= 5;
                  hdi_p1_done <= '1';
```

```
253               end if;

                  ------------------
                  -- hd5 calculation --
                  ------------------
258               -- last (index 1) harmonic is only needed for uall_sig
                  elsif index = 1 then
                    uall_sig <= uall_sig + (zeros_for_adding & square_result);
                    if real_not_im = '1' then
                      -- whole calculatoin is done
263                   done_sig <= '1';
                    end if;

                  else -- index (all other harmonics)
                    -- add up the suares
268                 u25_sig <= u25_sig + (zeros_for_adding & square_result);
                    uall_sig <= uall_sig + (zeros_for_adding & square_result);

                    -- after adding up the real part select the next harmonic
                    if real_not_im = '1' then
273                   index <= index - 1;
                    end if;

                  end if; -- index
                  once <= '1';
278             end if; --once
              else -- square_done = '0' -- no new squares
                once <= '0';
              end if; -- square_done
            end if; -- i_n_reset
283       end process ADD_DUMMY;


          ------------------------
          -- mux the square inputs --
288       ------------------------

          -- purpose: multiplex the square input
          -- type   : combinational
          -- inputs : i_harm_re, i_harm_im, index, real_not_im
293       -- outputs: square_in
          process(i_harm_re, i_harm_im, index, real_not_im)
          begin
            -- use imaginary or real
            if real_not_im = '1' then
298           square_in <= i_harm_re(index)(25 downto 6);
            else
              if index = 0 then
                -- i_harm_im(0) is only 20 bit width and unsigned
                square_in <= to_signed(to_integer(i_harm_im(0)),20);
303           else
                -- for the rest uses the upper 20 bit
                square_in <= i_harm_im(index)(25 downto 6);
              end if;
            end if;
308       end process;



          ------------------
313       -- square algorithm --
          ------------------
          SQR : calc_hd_square
          generic map(
            in_width  => square_val_in_width,
318         out_width => square_result_width
          )
          port map(
            i_sysclk    => i_sysclk,
            i_n_reset   => i_n_reset,
323
            i_start     => square_start,

            o_done      => square_done,

328         i_sq_val    => square_in,
            o_sq_result => square_result
          );

        end behavior;
```

Quelltext B.5: calc_hd_add_samples.vhd – Berechnung der Summe der Beträge der komplexen Harmonischen.

B.2.4 calc_hd_div.vhd

```
-------------------------------------------------------------------------------
-- Title      : calc_hd_div
-- Project    : ESZ-ABS
-------------------------------------------------------------------------------
-- File       : calc_hd_div.vhdl
-- Author     : Daniel Sabotta
--              based on: Jan-Heiner Dreschhoff
-- Company    : HAW Hamburg
-- Created    : 2012-06-28
-- Last update: 2012-06-29
-------------------------------------------------------------------------------
-- Description: Divider
-------------------------------------------------------------------------------
-- Copyright (c) 2012
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author      Description
--             0        Dreschhoff  init
-- 2012-06-28  1        Sabotta     remove Debugsignals and enable(CLK) and
--                                  XML-Comments
--                                  inserting the package declaration
-- 2012-06-29  1        Sabotta     inserting generic parameters
--
-------------------------------------------------------------------------------

-- ################################## Package #################################


library ieee;
   use ieee.std_logic_1164.all;
   use ieee.numeric_std.all;

library work;
   use work.global_pkg.all;

package calc_hd_div_pkg is
  component calc_hd_div is
    generic(
      -- in- and output widths
      num_width      : integer := 44;
      den_width      : integer := 44;
      result_width   : integer := 44
    );

    port(
      SYSCLK     : in std_logic;
      -- low active reset
      reset      : in std_logic;

      -- save NUMERATOR and DENOMINATOR and start divison if SET = 1
      SET        : in std_logic;
      -- synchronous reset
      Sreset     : in std_logic;
      -- DONE = 1 if division is ready
      DONE       : out std_logic;

      -- input values
      NUMERATOR   : in unsigned(num_width-1 downto 0);
      DENOMINATOR : in unsigned(den_width-1 downto 0);

      -- output value
      RESULT      : out unsigned(result_width-1 downto 0)
      );
  end component;
end calc_hd_div_pkg;

-- ################################ end Package ###############################

library ieee;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;
library work;
  use work.global_pkg.all;



entity calc_hd_div is
  generic(
      -- in- and output widths
      num_width : integer := 44;
      den_width : integer := 44;
      result_width : integer := 44
```

```
83      );

  port(
      SYSCLK     : in std_logic;
      -- low active reset
88      reset      : in std_logic;

      -- save NUMERATOR and DENOMINATOR and start divison if SET = 1
      SET        : in std_logic;
      -- synchronous reset
93      Sreset     : in std_logic;
      -- DONE = 1 if division is ready
      DONE       : out std_logic;

      -- input values
98      NUMERATOR   : in unsigned(num_width-1 downto 0);
      DENOMINATOR : in unsigned(den_width-1 downto 0);

      -- output value
      RESULT      : out unsigned(result_width-1 downto 0)
103   );
end calc_hd_div;

architecture behavior of calc_hd_div is

108   -- set NUMERATOR and DENOMINATOR only at the first cycle while SET is 1
  signal once : std_logic;
  -- internal numerator
  signal num  : unsigned(num_width downto 0);
  -- internal denominator
113   signal den  : unsigned(den_width-1 downto 0);
  -- internal result
  signal res  : unsigned(result_width-1 downto 0);
  -- counting signal
  signal cnt  : integer range 0 to result_width;
118
begin



123

  -- purpose: divider algorithm
  -- type   : sequential
  -- inputs : SYSCLK, reset, NUMERATOR, DENOMINATOR, Sreset, SET
  -- outputs: RESULT, DONE
128   divider_process: process(SYSCLK, reset)
    variable temp_num : unsigned(num_width downto 0);
  begin
    if reset = '0' then
      cnt <= result_width;
133     DONE <= '0';
      once <= '0';
      res <= (others => '0');
      num <= (others => '0');
      den <= (others => '0');
138     RESULT <= (others => '0');


    elsif SYSCLK = '1' and SYSCLK'event then
      -- synchronous reset
143     if Sreset = '1' then
        once <= '0';
        res <= (others => '0');
        num <= (others => '0');
        den <= (others => '0');
148     DONE <= '0';
        RESULT <= (others => '0');

      elsif once = '0' then
        -- set internal signals
153     if SET = '1' then
          num <= '0' & NUMERATOR;
          den <= DENOMINATOR;
          cnt <= result_width;
          -- but only in the first cycle
158       once <= '1';
        end if;     --SET

      else -- once
        -- all bits passed through (set outputs)
163     if cnt = 0 then
          DONE <= '1';
          RESULT <= res;
        else -- cnt
          -- clear output
```

```
168             RESULT <= (others => '0');

                -- divider algorithm
                if num > ('0' & den) then
                  temp_num := (num - ('0' & den));
173               num <= temp_num(num_width-1 downto 0) & '0';
                  res(cnt-1) <= '1';

                else
                  num <= num (num_width-1 downto 0) & '0';
178               res(cnt-1) <= '0';
                end if;
                -- next bit
                cnt <= cnt - 1;
              end if; --counter
183         end if;   --Sreset
          end if;     --reset
        end process;

end behavior;
```

Quelltext B.6: calc_hd_div.vhd – Dividierer.

B.2.5 calc_hd_fsm.vhd

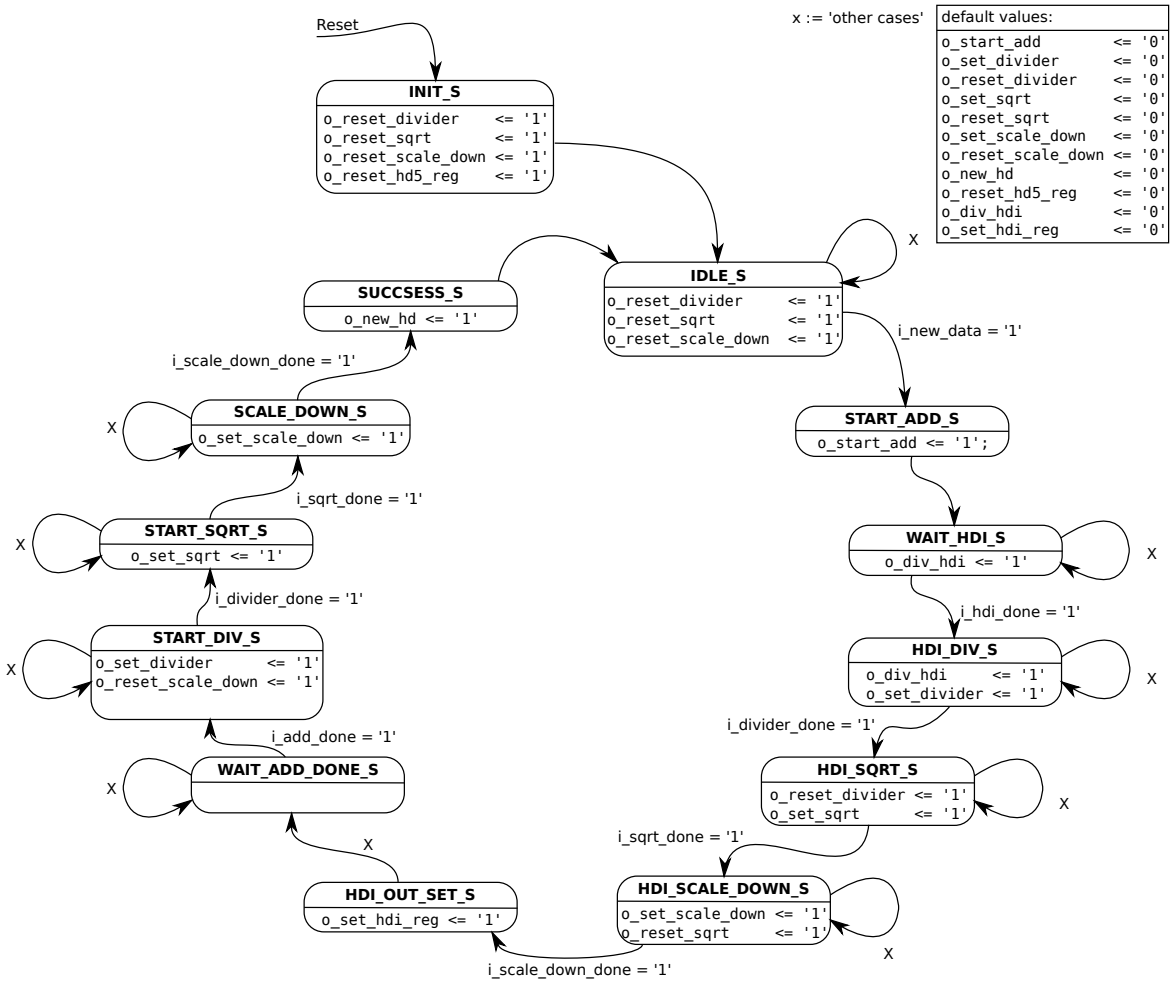


Abbildung B.5: Zustandsdiagramm des calc_hd-Moduls (Teil der Digitalen Signalverarbeitung). Siehe Quelltext B.7

```
-------------------------------------------------------------------------------
-- Title      : calc_hd_fsm
-- Project    : ESZ-ABS
-------------------------------------------------------------------------------
-- File       : calc_hd_fsm.vhdl
-- Author     : Daniel Sabotta
--              based on: Jan-Heiner Dreschhoff
-- Company    : HAW Hamburg
-- Created    : 2012-06-28
-- Last update: 2012-06-29
-------------------------------------------------------------------------------
-- Description: Finite State Machine to control the hdx calculation
-------------------------------------------------------------------------------
-- Copyright (c) 2012
-------------------------------------------------------------------------------
-- Revisions  :
-- Date          Version   Author       Description
--               0         Dreschhoff   init
-- 2012-06-28    1         Sabotta      remove Debugsignals and enable(CLK) and
--                                      XML-Comments
--                                      inserting the package declaration
-- 2012-06-29    1         Sabotta      inserting state declaration in this file
-- 2012-07-05    1         Sabotta      completely renewed (only one fsm)
--
-------------------------------------------------------------------------------

-- ################################## Package ################################

library ieee;
   use ieee.std_logic_1164.all;
   use ieee.numeric_std.all;

library work;
   use work.global_pkg.all;

package calc_hd_fsm_pkg is
  component calc_hd_fsm is
    port(
      -- allgemein
      i_sysclk     : in std_logic;
      i_n_reset    : in std_logic;

      i_new_data   : in std_logic;


      -- adder
      o_start_add   : out std_logic;
      i_add_done    : in std_logic;
      i_hdi_done          : in std_logic;

      -- divider
      o_set_divider   : out std_logic;
      o_reset_divider : out std_logic;
      i_divider_done  : in std_logic;

      -- squareroot
      o_set_sqrt    : out std_logic;
      o_reset_sqrt  : out std_logic;
      i_sqrt_done   : in std_logic;

      -- scaledown divider
      o_set_scale_down    : out std_logic;
      o_reset_scale_down  : out std_logic;
      i_scale_down_done   : in std_logic;

      -- Output signals
      o_new_hd    : out std_logic;
      o_reset_hd5_reg : out std_logic;

      -- hdi calculation
      o_div_hdi             : out std_logic;
      o_set_hdi_reg         : out std_logic;

      -- debug: state output
      o_state : out unsigned(3 downto 0)

    );
  end component;
end calc_hd_fsm_pkg;


-- ################################ end Package ##############################
```

```
library ieee;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;

library work;
  use work.global_pkg.all;


entity calc_hd_fsm is
  port(
    -- allgemein
    i_sysclk     : in std_logic;
    i_n_reset    : in std_logic;

    i_new_data   : in std_logic;

    -- adder
    o_start_add    : out std_logic;
    i_add_done     : in std_logic;
    i_hdi_done         : in std_logic;

    -- divider
    o_set_divider   : out std_logic;
    o_reset_divider : out std_logic;
    i_divider_done  : in std_logic;

    -- squareroot
    o_set_sqrt   : out std_logic;
    o_reset_sqrt : out std_logic;
    i_sqrt_done  : in std_logic;

    -- scaledown divider
    o_set_scale_down   : out std_logic;
    o_reset_scale_down : out std_logic;
    i_scale_down_done  : in std_logic;

    -- Output signals
    o_new_hd   : out std_logic;
    o_reset_hd5_reg : out std_logic;

    -- hdi calculation
    o_div_hdi            : out std_logic;
    o_set_hdi_reg        : out std_logic;

    -- debug: state output
    o_state : out unsigned(3 downto 0)

  );
end calc_hd_fsm;

architecture behavior of calc_hd_fsm is

  -- states
  subtype state_type is unsigned(3 downto 0);

  constant INIT_S            : state_type := "0000"; -- init
  constant IDLE_S            : state_type := "0001"; -- wait state
  constant START_ADD_S       : state_type := "0010"; -- start adding squares and |X|^2 for hdi
  constant WAIT_HDI_S        : state_type := "0011"; -- wait for hdis |X|^2
  constant HDI_DIV_S         : state_type := "0100"; -- divide for hdi calculation
  constant HDI_SQRT_S        : state_type := "0101"; -- sqrt for hdi
  constant HDI_SCALE_DOWN_S  : state_type := "0110"; -- scale down for hdi
  constant HDI_OUT_SET_S     : state_type := "0111"; -- save hdi result
  constant WAIT_ADD_DONE_S   : state_type := "1000"; -- wait for sums of squares
  constant START_DIV_S       : state_type := "1001"; -- dividing for hd5
  constant START_SQRT_S      : state_type := "1010"; -- suareroot for hd5
  constant SCALE_DOWN_S      : state_type := "1011"; -- scale down division for hd5
  constant SUCCSESS_S        : state_type := "1100"; -- all calculations done

  signal state, next_state  : state_type;

begin


  -- debug output
  o_state <= state;


  -- purpose: Stateregister
  -- type   : sequential
  -- inputs : i_sysclk, i_n_reset, state
  -- outputs: state
  state_register: process (i_sysclk, i_n_reset)
  begin
    if i_n_reset = '0' then
```

```
      state <= IDLE_S;  -- should be INIT_S ;-)

    elsif i_sysclk = '1' and i_sysclk'event then
      state <= next_state;
    end if;
  end process;



  -- purpose: Controles the output signals and state transitions
  --          (see statediagram)
  -- type   : combinational
  -- inputs : i_scale_down_done, i_sqrt_done, i_divider_done, i_add_donem,
  --          i_hdi_done, i_new_data, i_error
  -- outputs: o_reset_in_reg, o_start_add, o_set_divider, o_reset_divider,
  --          o_set_sqrt, o_reset_sqrt, o_set_scale_down, o_reset_scale_down,
  --          o_new_hd, o_hd_error, o_reset_hd5_reg, o_div_hdi, o_set_hdi_reg
  -- states : INIT_S            : init
  --          IDLE_S            : wait state
  --          START_ADD_S       : start adding squares and |X|^2 for hdi
  --          WAIT_HDI_S        : wait for hdis |X|^2
  --          HDI_DIV_S         : divide for hdi calculation
  --          HDI_SQRT_S        : sqrt for hdi
  --          HDI_SCALE_DOWN_S  : scale down for hdi
  --          HDI_OUT_SET_S     : save hdi result
  --          WAIT_ADD_DONE_S   : wait for sums of squares
  --          START_DIV_S       : dividing for hd5
  --          START_SQRT_S      : suareroot for hd5
  --          SCALE_DOWN_S      : scale down division for hd5
  --          SUCCSESS_S        : all calculations done

  state_logic: process (state, i_scale_down_done, i_sqrt_done, i_divider_done, i_add_done, i_hdi_done, i_new_data)
  begin
    -- default values
    o_start_add          <= '0';
    o_set_divider        <= '0';
    o_reset_divider      <= '0';
    o_set_sqrt           <= '0';
    o_reset_sqrt         <= '0';
    o_set_scale_down     <= '0';
    o_reset_scale_down   <= '0';
    o_new_hd             <= '0';
    o_reset_hd5_reg      <= '0';
    o_div_hdi            <= '0';
    o_set_hdi_reg        <= '0';

    next_state <= INIT_S;

    case state is

    when INIT_S            => -- init / wait state
      o_reset_divider      <= '1';
      o_reset_sqrt         <= '1';
      o_reset_scale_down   <= '1';
      o_reset_hd5_reg      <= '1';
      next_state <= IDLE_S;

    when IDLE_S            => -- wait state
      -- reset divider and sqrt
      o_reset_divider      <= '1';
      o_reset_sqrt         <= '1';
      o_reset_scale_down   <= '1';
      if i_new_data = '1' then
        next_state <= START_ADD_S;
      else
        next_state <= IDLE_S;
      end if;

    when START_ADD_S       => -- start adding squares and |X|^2 for hdi
      -- start pulse (only one cycle)
      o_start_add          <= '1';
      next_state <= WAIT_HDI_S;

    when WAIT_HDI_S        => -- wait for hdis |X|^2
      o_div_hdi <= '1';
      if i_hdi_done = '1' then
        next_state <= HDI_DIV_S;
      else
        next_state <= WAIT_HDI_S;
      end if;

    when HDI_DIV_S         => -- divide for hdi calculation
      o_div_hdi <= '1';
      o_set_divider <= '1';
```

```
      if i_divider_done = '1' then
        next_state <= HDI_SQRT_S;
      else
        next_state <= HDI_DIV_S;
      end if;

    when HDI_SQRT_S        => -- sqrt for hdi
      o_reset_divider      <= '1';
      o_set_sqrt <= '1';
      if i_sqrt_done = '1' then
        next_state <= HDI_SCALE_DOWN_S;
      else
        next_state <= HDI_SQRT_S;
      end if;

    when HDI_SCALE_DOWN_S => -- scale down for hdi
      o_set_scale_down <= '1';
      o_reset_sqrt <= '1';
      if i_scale_down_done = '1' then
        next_state <= HDI_OUT_SET_S;
      else
        next_state <= HDI_SCALE_DOWN_S;
      end if;

    when HDI_OUT_SET_S     => -- save hdi result
      o_set_hdi_reg <= '1'; -- set only one cycle
      next_state <= WAIT_ADD_DONE_S;

    when WAIT_ADD_DONE_S   => -- wait for sums of squares
      if i_add_done = '1' then
        next_state <= START_DIV_S;
      else
        next_state <= WAIT_ADD_DONE_S;
      end if;

    when START_DIV_S       => -- dividing for hd5
      o_set_divider <= '1';
      o_reset_scale_down <= '1';
      if i_divider_done = '1' then
        next_state <= START_SQRT_S;
      else
        next_state <= START_DIV_S;
      end if;

    when START_SQRT_S      => -- suareroot for hd5
      o_set_sqrt <= '1';
      if i_sqrt_done = '1' then
        next_state <= SCALE_DOWN_S;
      else
        next_state <= START_SQRT_S;
      end if;

    when SCALE_DOWN_S      => -- scale down division for hd5
      o_set_scale_down <= '1';
      if i_scale_down_done = '1' then
        next_state <= SUCCSESS_S;
      else
        next_state <= SCALE_DOWN_S;
      end if;

    when SUCCSESS_S        => -- all calculations done
      o_new_hd <= '1';
      next_state <= IDLE_S;

    when others            =>
      next_state <= INIT_S;

    end case;


  end process;

end behavior;
```

Quelltext B.7: calc_hd_fsm.vhd – Zustandsautomat zur Steuerung der HDI- und HD5-Berechnung.

B.2.6 calc_hd_harm_serilizer.vhd

```
--------------------------------------------------------------------------------
-- Title      : calc_hd_serializer
-- Project    : ESZ-ABS
--------------------------------------------------------------------------------
-- File       : calc_hd_serializer.vhd
-- Author     : Daniel Sabotta
-- Company    : HAW Hamburg
-- Created    : 2012-06-26
-- Last update: 2012-09-06
--------------------------------------------------------------------------------
-- Description: sends two harm_array_t serially (like SPI)
--              sending i_harm_real(0) at first then i_harm_imag(0) then
--              i_harm_real(1) and so on MSB first
--------------------------------------------------------------------------------
-- Copyright (c) 2012
--------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2012-06-26  0        Sabotta         created
-- 2012-09-06  0        Sabotta         commented
--
--------------------------------------------------------------------------------

-- ################################## Package ###################################

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

use work.global_pkg.all;

package calc_hd_harm_serializer_pkg is
  component calc_hd_harm_serializer is
    generic(
      -- number of bits in one harmonic
      one_harm_bits      : integer := 26;
      -- number of whole bits in array
      harm_array_t_bits : integer := 156
      -- (type harm_array_t is array(0 to 5) of signed(25 downto 0))
      );
    port(
      i_n_reset : in std_logic;
      i_clk     : in std_logic;

      -- parallel incoming data
      i_harm_real  : in harm_array_t;
      i_harm_imag  : in harm_array_t;
      -- incoming ready
      i_harm_ready : in std_logic;

      -- serial output (like SPI)
      -- data bit
      o_ser_out     : out std_logic;
      -- sending data
      o_ser_sending : out std_logic
      );
  end component;
end calc_hd_harm_serializer_pkg;

-- ################################ end Package #################################


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;

use work.global_pkg.all;

entity calc_hd_harm_serializer is
  generic(
      -- number of bits in one harmonic
      one_harm_bits      : integer := 26;
      -- number of whole bits in one array
      harm_array_t_bits : integer := 156
      -- (type harm_array_t is array(0 to 5) of signed(25 downto 0))
      );
    port(
      i_n_reset : in std_logic;
      i_clk     : in std_logic;

```

```
      -- parallel incoming data
      i_harm_real  : in harm_array_t;
      i_harm_imag  : in harm_array_t;
      -- incoming ready
      i_harm_ready : in std_logic;

      -- serial output (like SPI)
      -- data bit
      o_ser_out     : out std_logic;
      -- sending data
      o_ser_sending : out std_logic
      );
end entity calc_hd_harm_serializer;


architecture behavioral of calc_hd_harm_serializer is
  -- save and shift incoming data
  signal shift_register : std_logic_vector((2*harm_array_t_bits-1) downto 0);  -- 5 x 20 bits in einem harm_array_t
  signal load_shift_register : std_logic;

  -- the timer counts how many bits are sent
  -- start timer
  signal start_timer_register  : std_logic;
  -- enable timer
  signal enable_timer_register : std_logic;
  -- timer ready
  signal timer_register_rdy    : std_logic;
  -- counting register
  signal timer_register        : integer range (2*harm_array_t_bits) downto 0;

  -- state of the control FSM
  signal shift_control_state : std_logic_vector(1 downto 0);

begin

  ------------------
  -- Shift Register --
  ------------------

  -- purpose: shift register
  -- type   : sequential
  -- inputs : i_clk, i_n_reset, i_harm_real, i_harm_imag
  -- outputs: shift_register
  shift_registerP : process (i_clk, i_n_reset)
  begin
    if i_n_reset = '0' then
      -- reset shiftregister
      shift_register <= (others => '0');

    elsif i_clk = '1' and i_clk'event then
      -- load shift register if new data avalible and last data are sent
      if load_shift_register = '1' then
        for i in i_harm_imag'range loop
          -- load one harmoniac after the other into the shift register
          -- shift_register <= i_harm_real(0) & i_harm_imag_(0) & i_harm_real(1) & i_harm_imag_(1) & ...
          shift_register((2*one_harm_bits*(5-i+1)-1) downto (2*one_harm_bits*(5-i))) <= std_logic_vector(i_harm_real(i))
               & std_logic_vector(i_harm_imag(i));
        end loop;

      else
        -- shift the shiftregister
        shift_register <= shift_register((2*harm_array_t_bits - 2) downto 0) & '0';
      end if;
    end if;
  end process;

  -- set the data output (last bit in array)
  o_ser_out <= shift_register(2*harm_array_t_bits - 1);


  ----------------
  -- Control FSM --
  ----------------

  -- purpose: set/reset the timer, shift register and outputs
  -- type   : sequential
  -- inputs : i_clk, i_n_reset, i_harm_ready, timer_register_ready
  -- outputs: start_timer_register, enable_timer_register, load_shift_register
  shiftControlFSM : process (i_clk, i_n_reset)
  begin
    if i_n_reset = '0' then
      start_timer_register  <= '0';
      enable_timer_register <= '0';
      load_shift_register   <= '0';
      shift_control_state   <= "00";
```

```
167
    elsif i_clk = '1' and i_clk'event then
      -- default values
      start_timer_register  <= '0';
      enable_timer_register <= '0';
172   load_shift_register   <= '0';
      shift_control_state   <= "00";


      case shift_control_state is
177
        -- idle state (waiting for incoming data)
        when "00" =>
          shift_control_state <= "00";
          -- new data avalible
182       if i_harm_ready = '1' then
            -- go into read state, load shift register and start timer
            shift_control_state   <= "01";
            start_timer_register  <= '1';
            load_shift_register   <= '1';
187         enable_timer_register <= '1';
          end if;

        -- read state
        when "01" =>
192       -- go into send state and enable timer
          enable_timer_register <= '1';
          shift_control_state   <= "10";

        -- send state
197     when "10" =>
          -- if all bits are sent go into idle state
          if timer_register_rdy = '1' then
            shift_control_state <= "00";
          -- else send next bit
202       else
            shift_control_state   <= "10";
            enable_timer_register <= '1';
          end if;

207     when others =>
          shift_control_state <= "00";  -- read

      end case;
    end if;
212 end process;

  -- during sending o_ser_sending will be 1
  o_ser_sending <= '1' when shift_control_state = "10" else '0';

217 ------------
  -- Timer --
  ------------

  -- purpose: counts how many bits are send
222 -- type   : sequential
  -- inputs : i_clk, i_n_reset, start_timer_register, enable_timer_register
  -- outputs: timer_register
  timer_registerP : process (i_clk, i_n_reset)
  begin
227 if i_n_reset = '0' then
      timer_register <= 0;
    elsif i_clk = '1' and i_clk'event then
      -- start timer (synchronous reset)
      if start_timer_register = '1' then
232     timer_register <= 0;
      -- count, if enabled
      elsif enable_timer_register = '1' then
        timer_register <= timer_register + 1;
      end if;
237 end if;
  end process;

  -- signaling, that all bits are sent
  timer_register_rdy <= '1' when timer_register = (2*harm_array_t_bits - 1) else '0';
242
end behavioral;
```

Quelltext B.8: calc_hd_harm_serilizer.vhd – Serielle Ausgabe der Harmonischen für Testzwecke.

B.2.7 calc_hd_in_reg.vhd

```
1  ----------------------------------------------------------------------------
   -- Title      : calc_hd_in_reg
   -- Project    : ESZ-ABS
   ----------------------------------------------------------------------------
   -- File       : calc_hd_in_reg.vhdl
6  -- Author     : Daniel Sabotta
   --              based on: Jan-Heiner Dreschhoff
   -- Company    : HAW Hamburg
   -- Created    : 2012-06-28
   -- Last update: 2012-09-06
11 ----------------------------------------------------------------------------
   -- Description: Input Register
   ----------------------------------------------------------------------------
   -- Copyright (c) 2012
   ----------------------------------------------------------------------------
16 -- Revisions  :
   -- Date        Version   Author      Description
   --             0         Dreschhoff  init
   -- 2012-06-28  1         Sabotta     remove Debugsignals and enable(CLK) and
   --                                   XML-Comments
21 --                                   inserting the package declaration
   -- 2012-06-29  1         Sabotta     updating to six incoming signals
   -- 2012-09-06  1         Sabotta     commented
   --
   ----------------------------------------------------------------------------
26
   -- ################################## Package ###################################

   library ieee;
      use ieee.std_logic_1164.all;
31    use ieee.numeric_std.all;

   library work;
      use work.global_pkg.all;

36 package calc_hd_in_reg_pkg is
     component calc_hd_in_reg is
       port(
         SYSCLK     : in std_logic;
         reset      : in std_logic;
41
         -- load harmonics
         SET        : in std_logic;

         -- incoming harmonics
46       RE_IN_REG_i : in harm_array_t;
         IM_IN_REG_i : in harm_array_t;

         -- register output
         RE_IN_REG_o : out harm_array_t;
51       IM_IN_REG_o : out harm_array_t
             );
     end component;
   end calc_hd_in_reg_pkg;

56 -- ################################ end Package #################################


   library ieee;
     use ieee.std_logic_1164.all;
61   use ieee.numeric_std.all;
   library work;
     use work.global_pkg.all;

   entity calc_hd_in_reg is
66   port(
       SYSCLK     : in std_logic;
       reset      : in std_logic;

       -- load harmonics
71     SET        : in std_logic;

       -- incoming harmonics
       RE_IN_REG_i : in harm_array_t;
       IM_IN_REG_i : in harm_array_t;
76
       -- register output
       RE_IN_REG_o : out harm_array_t;
       IM_IN_REG_o : out harm_array_t
           );
81 end calc_hd_in_reg;
```

```
architecture behavior of calc_hd_in_reg is

  -- internal register
  signal re_sig : harm_array_t;
  signal im_sig : harm_array_t;

begin
  -- writer output
  RE_IN_REG_o <= re_sig;
  IM_IN_REG_o <= im_sig;


  -- purpose: register to save incoming harmonics
  -- type   : sequential
  -- inputs : SYSCLK, reset, RE_IN_REG_i, IM_IN_REG_i
  -- outputs: re_sig, im_sig
  in_regiser_p: process(SYSCLK, reset)
  begin
    if reset = '0' then
      for i in 0 to 5 loop
          re_sig(i) <= (others => '0');
          im_sig(i) <= (others => '0');
      end loop;
    elsif SYSCLK = '1' and SYSCLK'event then
      -- set everey harmonic
      if SET = '1' then
       for i in 0 to 5 loop
          re_sig(i) <= RE_IN_REG_i(i);
          im_sig(i) <= IM_IN_REG_i(i);
        end loop;
      end if; --set
    end if;     --reset
  end process;

end behavior;
```

Quelltext B.9: calc_hd_in_reg.vhd – Eingangsregister der calc_hd-Blocks.

B.2.8 calc_hd_sqrt.vhd

```
-------------------------------------------------------------------------------
-- Title      : calc_hd_sqrt
-- Project    : ESZ-ABS
-------------------------------------------------------------------------------
-- File       : calc_hd_sqrt.vhdl
-- Author     : Daniel Sabotta
--              based on: Jan-Heiner Dreschhoff
-- Company    : HAW Hamburg
-- Created    : 2012-06-28
-- Last update: 2012-09-06
-------------------------------------------------------------------------------
-- Description: calculates the squareroot
-------------------------------------------------------------------------------
-- Copyright (c) 2012
-------------------------------------------------------------------------------
-- Revisions  :
-- Date          Version   Author      Description
--               0         Dreschhoff  init
-- 2012-06-28    1         Sabotta     remove Debugsignals and enable(CLK) and
--                                     XML-Comments
--                                     inserting the package declaration
-- 2012-06-29    1         Sabotta     inserting generic parameters
-- 2012-09-06    1         Sabotta     commented
--
-------------------------------------------------------------------------------

-- ################################## Package ###################################

library ieee;
   use ieee.std_logic_1164.all;
   use ieee.numeric_std.all;

library work;
   use work.global_pkg.all;

package calc_hd_sqrt_pkg is
  component calc_hd_sqrt is
    generic(
```

```
      -- in- and output bitwidth
      val_in_width  : integer := 44;
      val_out_width  : integer := 22
    );

    port(
      SYSCLK : in std_logic;
      reset  : in std_logic;

      -- enables the squareroot algorithm
      SET    : in std_logic;

      -- synchronous reset
      Sreset  : in std_logic;

      -- calculation finished
      DONE   : out std_logic;

      -- in- and output values
      VAL_IN : in unsigned(val_in_width-1 downto 0);
      VAL_OUT: out unsigned(val_out_width-1 downto 0)
    );
  end component;
end calc_hd_sqrt_pkg;


-- ################################ end Package ##############################

library IEEE;
  use IEEE.std_logic_1164.all;
  use ieee.numeric_std.all;    -- for UNSIGNED

library work;
  use work.global_pkg.all;

entity calc_hd_sqrt is
  generic(
      -- in- and output bitwidth
      val_in_width  : integer := 44;
      val_out_width  : integer := 22
    );

    port(
      SYSCLK : in std_logic;
      reset  : in std_logic;

      -- enables the squareroot algorithm
      SET    : in std_logic;

      -- synchronous reset
      Sreset  : in std_logic;

      -- calculation finished
      DONE   : out std_logic;

      -- in- and output values
      VAL_IN : in unsigned(val_in_width-1 downto 0);
      VAL_OUT: out unsigned(val_out_width-1 downto 0)
    );
end calc_hd_sqrt;

architecture behaviour of calc_hd_sqrt is

  -- result
  signal q : unsigned(val_out_width-1 downto 0);
  -- original input
  signal a : unsigned(val_in_width-1 downto 0);
  -- indexing signal
  signal i : integer range 0 to val_out_width;


begin


  -- purpose: squareroot
  -- type   : sequential
  -- inputs : SYSCLK, reset, Sreset, SET, VAL_IN
  -- outputs: VAL_OUT
  process(SYSCLK, reset)
    variable left,right, r : unsigned(val_out_width+1 downto 0);  --input to adder/sub.r-remainder. really one bit more
          than val_out
    variable once : std_logic;
  begin
    if reset = '0' then
      q     <= (others => '0');
```

```
123       a      <= (others => '0');
          i      <= 0;
          r      := (others => '0');
          right  := (others => '0');
          left   := (others => '0');
128       once   := '0';
          VAL_OUT <= (others => '0');
          DONE    <= '0';

        elsif SYSCLK = '1' and SYSCLK'event then
133
          -- synchronous reset
          if Sreset = '1' then
            q      <= (others => '0');
            a      <= (others => '0');
138         i      <= 0;
            r      := (others => '0');
            right  := (others => '0');
            left   := (others => '0');
            VAL_OUT <= (others => '0');
143         DONE    <= '0';
            once   := '0';

          -- squareroot
          elsif SET = '1' then
148         -- load input valu only once
            if once = '0' then
              a <= VAL_IN;
              once := '1';

153         -- calculation done
            elsif i = val_out_width then
              DONE <= '1';
              VAL_OUT <= q;

158         -- calculate squareroot (see Dreschoffs Diplomarbeit)
            else
              right(0):='1';
              right(1):=r(val_out_width+1);
              right(val_out_width+1 downto 2):=q;
163
              left(1 downto 0):=a(val_in_width-1 downto val_in_width-2);
              left(val_out_width+1 downto 2):=r(val_out_width-1 downto 0);

              a(val_in_width-1 downto 2)<=a(val_in_width-3 downto 0);  --shifting by 2 bit.
168
              if ( r(val_out_width+1) = '1') then
                r := left + right;
              else
                r := left - right;
173           end if;

              q(val_out_width-1 downto 1) <= q(val_out_width-2 downto 0);
              q(0) <= not r(val_out_width+1);

178           i <= i + 1;
            end if;   -- once

          else
            DONE   <= '0';
183       end if;   --Sreset

        end if;    --reset
      end process;

188 end behaviour;
```

Quelltext B.10: calc_hd_sqrt.vhd – Quadratwurzel.

B.2.9 calc_hd_sub.vhd

```
-------------------------------------------------------------------------------
-- Title      : calc_hd_sub
-- Project    : ESZ-ABS
-------------------------------------------------------------------------------
-- File       : calc_hd_sub.vhdl
-- Author     : Daniel Sabotta
-- Company    : HAW Hamburg
-- Created    : 2012-06-29
-- Last update: 2012-06-29
-------------------------------------------------------------------------------
-- Description: Subtract with outputregister
--              difference = minuend - subtrahend ;-)
-------------------------------------------------------------------------------
-- Copyright (c) 2012
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author      Description
-- 2012-06-29  0        Sabotta     init
--
-------------------------------------------------------------------------------

-- ################################## Package ###################################


library ieee;
   use ieee.std_logic_1164.all;
   use ieee.numeric_std.all;

library work;
   use work.global_pkg.all;

package calc_hd_sub_pkg is
  component calc_hd_sub is
    generic(
      difference_width : integer := 20;
      subtrahend_width : integer := 20;
      minuend_width : integer := 20
    );

    port(
      i_sysclk      : in std_logic;
      i_n_reset     : in std_logic;

      i_minuend     : in unsigned (minuend_width-1 downto 0);
      i_subtrahend  : in unsigned (subtrahend_width-1 downto 0);
      o_difference  : out unsigned (difference_width-1 downto 0)
    );
  end component;
end calc_hd_sub_pkg;

-- ################################ end Package #################################

library ieee;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;
library work;
  use work.global_pkg.all;



entity calc_hd_sub is
  generic(
      difference_width : integer := 20;
      subtrahend_width : integer := 20;
      minuend_width : integer := 20
    );

    port(
      i_sysclk      : in std_logic;
      i_n_reset     : in std_logic;

      i_minuend     : in unsigned (minuend_width-1 downto 0);
      i_subtrahend  : in unsigned (subtrahend_width-1 downto 0);
      o_difference  : out unsigned (difference_width-1 downto 0)
    );
end calc_hd_sub;

architecture behavior of calc_hd_sub is

begin

    process(i_sysclk, i_n_reset)
```

```
    begin
      if i_n_reset = '0' then
        o_difference <= (others => '0');
      elsif i_sysclk = '1' and i_sysclk'event then
87        o_difference <= i_minuend - i_subtrahend;
      end if;     --reset
    end process;

end behavior;
```

Quelltext B.11: calc_hd_sub.vhd – Subtraktion.

B.2.10 calc_hd_TL.vhd

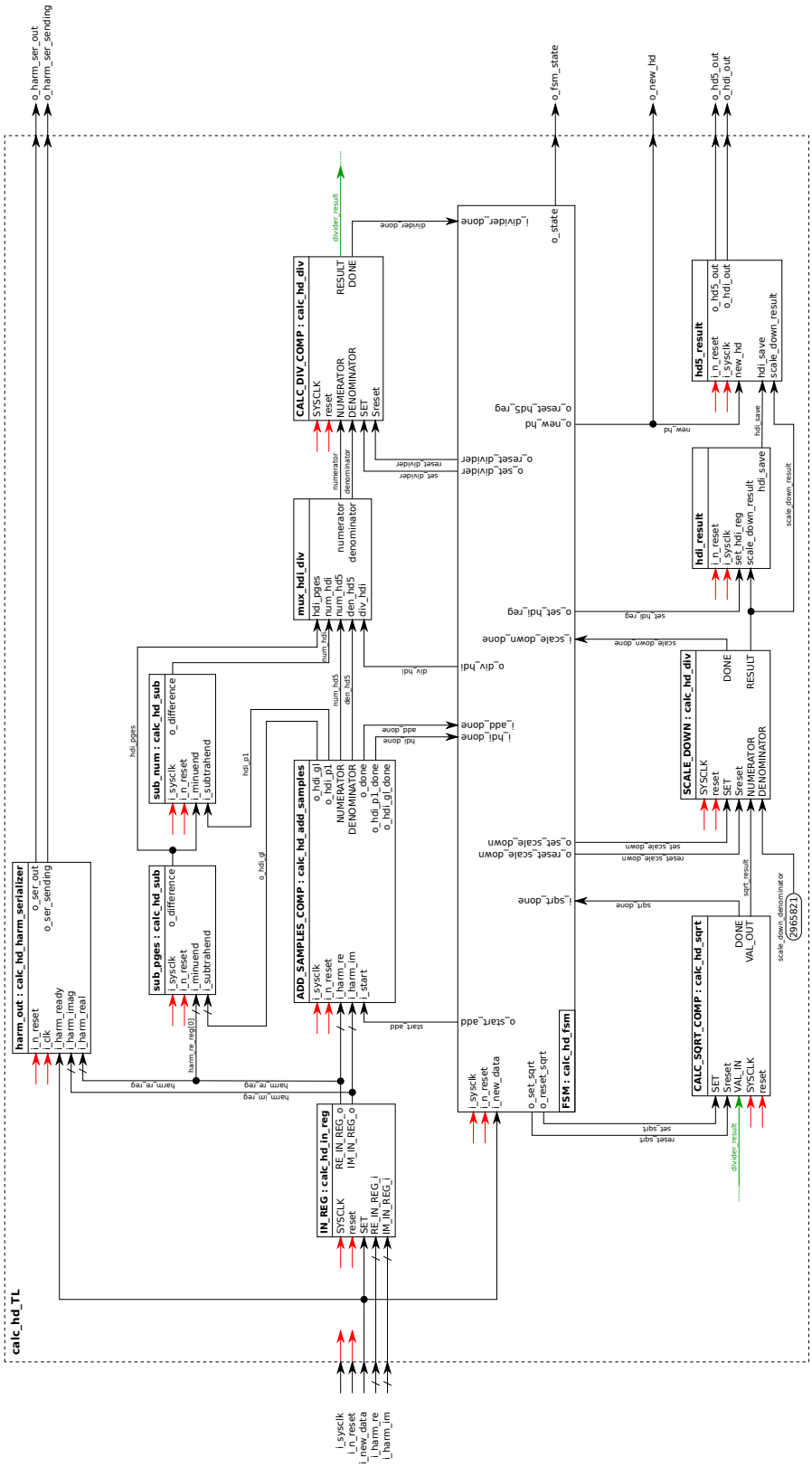


Abbildung B.6: Blockschaltbild des calc_hd_TL-Moduls (Teil der Digitalen Signalverarbeitung). Siehe Quelltext B.12

```
-------------------------------------------------------------------------------
-- Title      : calc_hd_TL
-- Project    : ESZ-ABS
-------------------------------------------------------------------------------
-- File       : calc_hd_TL.vhdl
-- Author     : Daniel Sabotta
--              based on: Jan-Heiner Dreschhoff
-- Company    : HAW Hamburg
-- Created    : 2012-06-28
-- Last update: 2012-09-06
-------------------------------------------------------------------------------
-- Description: Top level for hdi and hd5 calulation
-------------------------------------------------------------------------------
-- Copyright (c) 2012
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version   Author      Description
--             0         Dreschhoff  init
-- 2012-06-28  1         Sabotta     remove Debugsignals and enable(CLK) and
--                                   XML-Comments
--                                   inserting the package declaration
-- 2012-06-29  1         Sabotta     inserting generic parameters
-- 2012-07-05  1         Sabotta     new fsm inserted
--                                   output register renewed
-- 2012-09-06  1         Sabotta     commented
--
-------------------------------------------------------------------------------

-- ################################## Package ###################################
library ieee;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;

library work;
  use work.global_pkg.all;

package calc_hd_TL_pkg is
  component calc_hd_TL is
    generic(
    square_val_in_width           : integer := 20;
    square_val_out_width          : integer := 40;
    add_out_width                 : integer := 44;
    calc_div_comp_out_width       : integer := 44;
    sqrt_out_width                : integer := 22;
    scale_down_out_width          : integer := 22;
    scale_down_denominator_value  : integer := 2965821  -- 2_965_820.80 length of sqrt_out_width
                  -- um die hochskalierung bei der ersten division herauszurechnen also sqrt(2^[calc_div_comp_out_width
                       -1])
  );

  port(
    i_sysclk      : in std_logic;
    i_n_reset     : in std_logic;

    i_new_data    : in std_logic;

    i_harm_re     : in harm_array_t;
    i_harm_im     : in harm_array_t;

    o_new_hd      : out std_logic;
    o_hd5_out     : out unsigned(scale_down_out_width-1 downto 0);

    o_hdi_out     : out unsigned(scale_down_out_width-1 downto 0);

    o_harm_ser_out      : out std_logic;
    o_harm_ser_sending  : out std_logic;

    -- debug: state output
    o_fsm_state : out unsigned(3 downto 0)
  );
  end component;
end calc_hd_TL_pkg;

-- ################################ end Package #################################

library ieee;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;

library work;
  use work.global_pkg.all;
  use work.calc_hd_add_samples_pkg.all;
  use work.calc_hd_in_reg_pkg.all;
  use work.calc_hd_fsm_pkg.all;
  use work.calc_hd_div_pkg.all;
```

```
  use work.calc_hd_sqrt_pkg.all;
  use work.calc_hd_sub_pkg.all;
  use work.calc_hd_harm_serializer_pkg.all;

entity calc_hd_TL is
  generic(
    square_val_in_width           : integer := 20;
    square_val_out_width          : integer := 40;
    add_out_width                 : integer := 44;
    calc_div_comp_out_width       : integer := 44;
    sqrt_out_width                : integer := 22;
    scale_down_out_width          : integer := 22;
    scale_down_denominator_value  : integer := 2965821  -- 2_965_820.80 length of sqrt_out_width
                  -- um die hochskalierung bei der ersten dividion herauszu rechnen also sqrt(2^[
                      calc_div_comp_out_width-1])
  );

  port(
    i_sysclk      : in std_logic;
    i_n_reset     : in std_logic;

    i_new_data    : in std_logic;

    i_harm_re     : in harm_array_t;
    i_harm_im     : in harm_array_t;

    o_new_hd      : out std_logic;
    o_hd5_out     : out unsigned(scale_down_out_width-1 downto 0);

    o_hdi_out     : out unsigned(scale_down_out_width-1 downto 0);

    o_harm_ser_out      : out std_logic;
    o_harm_ser_sending  : out std_logic;

    -- debug: state output
    o_fsm_state : out unsigned(3 downto 0)
  );
end calc_hd_TL;

architecture structure of calc_hd_TL is

  signal new_hd :  std_logic;

  signal harm_re_reg     : harm_array_t;
  signal harm_im_reg     : harm_array_t;

  signal start_add       : std_logic;
  signal add_done      : std_logic;

  signal numerator         : unsigned(add_out_width-1 downto 0);
  signal denominator       : unsigned(add_out_width-1 downto 0);

  signal num_hd5            : unsigned(add_out_width-1 downto 0);
  signal den_hd5            : unsigned(add_out_width-1 downto 0);

  signal reset_divider  : std_logic;
  signal set_divider    : std_logic;
  signal divider_done   : std_logic;
  signal divider_result     : unsigned(calc_div_comp_out_width-1 downto 0);

  signal reset_sqrt : std_logic;
  signal set_sqrt   : std_logic;
  signal sqrt_done   : std_logic;
  signal sqrt_result     : unsigned(sqrt_out_width-1 downto 0);

  signal set_scale_down     : std_logic;
  signal reset_scale_down   : std_logic;
  signal scale_down_done    : std_logic;
  signal scale_down_result      : unsigned(scale_down_out_width-1 downto 0);

  signal reset_hd5_reg : std_logic;



  ------------------------------

  signal hdi_done            : std_logic;
  signal div_hdi             : std_logic;

  signal set_hdi_reg         : std_logic;

  signal hdi_pges : unsigned(27 downto 0);
  signal hdi_p1 : unsigned(43 downto 0);

  signal o_hdi_gl      : unsigned(43 downto 0);
```

```
    signal o_hdi_gl_done : std_logic;

    signal num_hdi : unsigned(41 downto 0);

    signal hdi_minuend_temp : unsigned(41 downto 0);
    signal hdi_subtrahend_temp : unsigned(41 downto 0);

    signal hdi_minuend_pges_temp : unsigned(27 downto 0);

    signal hdi_save      : unsigned(scale_down_out_width-1 downto 0);

    constant scale_down_denominator : unsigned (sqrt_out_width-1 downto 0) := to_unsigned(scale_down_denominator_value ,
        sqrt_out_width);

begin


 o_new_hd            <= new_hd;

-------
-- FSM --
-------

FSM : calc_hd_fsm
  port map(
      i_sysclk            => i_sysclk,
      i_n_reset            => i_n_reset,

      i_new_data          => i_new_data,

      o_start_add         => start_add,
      i_add_done          => add_done,

      o_set_divider       => set_divider,
      o_reset_divider     => reset_divider,
      i_divider_done      => divider_done,

      o_set_sqrt          => set_sqrt,
      o_reset_sqrt        => reset_sqrt,
      i_sqrt_done         => sqrt_done,

      o_set_scale_down    => set_scale_down,
      o_reset_scale_down  => reset_scale_down,
      i_scale_down_done   => scale_down_done,

      o_new_hd            => new_hd,
      o_reset_hd5_reg     => reset_hd5_reg,

      ------------------------

      i_hdi_done          => hdi_done,
      o_div_hdi           => div_hdi,
      o_set_hdi_reg       => set_hdi_reg,

      -- debug: state output
      o_state             => o_fsm_state

  );


  ----------
  -- IN_REG --
  ----------
  IN_REG : calc_hd_in_reg
  port map(
      SYSCLK    => i_sysclk,
      reset     => i_n_reset,

      SET       => i_new_data,

      RE_IN_REG_i => i_harm_re ,
      IM_IN_REG_i => i_harm_im,

      RE_IN_REG_o => harm_re_reg,
      IM_IN_REG_o => harm_im_reg
      );

  ---------------
  -- ADD_SAMPLES --
  ---------------

  ADD_SAMPLES_COMP : calc_hd_add_samples
    generic map(
      out_width => add_out_width,
```

```
      square_val_in_width => square_val_in_width,   -- signed
      square_result_width => square_val_out_width -- unsigned
    )
    port map(
      i_sysclk    => i_sysclk,
      i_n_reset    => i_n_reset,

      i_start  => start_add,
      o_done     => add_done,

      i_harm_re        => harm_re_reg,
      i_harm_im        => harm_im_reg,

      NUMERATOR       => num_hd5,
      DENOMINATOR     => den_hd5,

      o_hdi_p1         => hdi_p1,
      o_hdi_p1_done    => hdi_done,

      o_hdi_gl       => o_hdi_gl,
      o_hdi_gl_done => o_hdi_gl_done

      );

  ------------------
  -- CALC_DIV_COMP --
  ------------------
  CALC_DIV_COMP : calc_hd_div
    generic map(
      num_width => add_out_width,
      den_width => add_out_width,
      result_width => calc_div_comp_out_width
    )
    port map(
      SYSCLK    => i_sysclk,
      reset     => i_n_reset,

      SET       => set_divider,
      Sreset    => reset_divider,
      DONE      => divider_done,

      NUMERATOR   => numerator,
      DENOMINATOR => denominator,

      RESULT => divider_result
    );

  ---------------
  -- CALC_SQRT --
  ---------------
  CALC_SQRT_COMP : calc_hd_sqrt
  generic map(
    val_in_width  => calc_div_comp_out_width,
    val_out_width => sqrt_out_width
  )
  port map(
    SYSCLK    => i_sysclk,
    reset     => i_n_reset,

    SET       => set_sqrt,
    Sreset    => reset_sqrt,
    DONE      => sqrt_done,

    VAL_IN  => divider_result,
    VAL_OUT => sqrt_result

  );


  ---------------
  -- SCALE_DOWN --
  ---------------
  SCALE_DOWN : calc_hd_div
    generic map(
      num_width => 22,
      den_width => 22,
      result_width => 22
    )
    port map(
      SYSCLK    => i_sysclk,
      reset     => i_n_reset,

      SET       => set_scale_down,
      Sreset    => reset_scale_down,
      DONE      => scale_down_done,
```

```
338
      -- Test:
      --NUMERATOR   => "00" & x"02000",
      --DENOMINATOR => "00" & x"10000",
      NUMERATOR   => sqrt_result,
343   DENOMINATOR => scale_down_denominator,

      RESULT => scale_down_result
    );

348

  ------------
  -- SUB_PGES --
  ------------
  sub_pges: calc_hd_sub
353   generic map (
      difference_width => 28,
      subtrahend_width => 28,
      minuend_width => 28
    )
358
    port map(
      i_sysclk      => i_sysclk,
      i_n_reset     => i_n_reset,

363   i_minuend     => hdi_minuend_pges_temp,
      i_subtrahend  => o_hdi_gl(39 downto 12),
      o_difference  => hdi_pges
    );

368 -- adapting bit length
  hdi_minuend_pges_temp <= x"00" & unsigned(std_logic_vector(harm_re_reg(0)(25 downto 6)));

  ------------
  -- SUB_NUM --
373 ------------
  sub_num: calc_hd_sub
    generic map (
      difference_width => 42,
      subtrahend_width => 42,
378   minuend_width => 42
    )

    port map(
      i_sysclk      => i_sysclk,
383   i_n_reset     => i_n_reset,

      i_minuend     => hdi_minuend_temp,
      i_subtrahend  => hdi_subtrahend_temp,
      o_difference  => num_hdi
388 );

  -- adapting bit length
  hdi_minuend_temp <= ("00" & x"000" & hdi_pges);
  hdi_subtrahend_temp <=  "000" & x"000" & hdi_p1(43 downto 17); -- shift 17 bits  (>>17 <=> *2/N^2 with N=512) -- *2
       fuer um leistung aus spektrum zu berechnen! -- /N^2 fuer skalierung
393
  ------------------------------
  -- multiplexer for multiplication --
  ------------------------------
  -- purpose: multiplex the divider input
398 -- type   : combinational
  -- inputs : div_hdi, num_hdi, hdi_pges, num_hd5, den_hd5
  -- outputs: numerator, denominator
  mux_hdi_div: process(div_hdi, num_hd5, den_hd5, hdi_pges, num_hdi)
  begin
403   if div_hdi = '1' then
      numerator   <= "00" & num_hdi;      -- TODO: make numbers of zeros depending on generic
      denominator <= x"0000" & hdi_pges;
    else
      numerator   <= num_hd5;
408   denominator <= den_hd5;
    end if;
  end process;

  ------------------------
413 -- hdi save register --
  ------------------------
  -- purpose: save hdi
  -- type   : sequential
  -- inputs : i_sysclk, i_n_reset, scale_down_result
418 -- outputs: hdi_save
  hdi_result: process(i_n_reset, i_sysclk)
  begin
    if i_n_reset = '0' then
```

```
      hdi_save <= (others => '0');
    elsif i_sysclk = '1' and i_sysclk'event then
      if set_hdi_reg = '1' then
        hdi_save <= scale_down_result;
      end if;
    end if;
  end process;

  ------------------------
  -- hd result register --
  ------------------------
  -- purpose: save hdi and hd5
  -- type   : sequential
  -- inputs : i_sysClk, nInternalReset, scale_down_result, hdi_save
  -- outputs: o_hd5_out, o_hdi_out
  hd5_result: process(i_n_reset, i_sysclk)
  begin
    if i_n_reset = '0' then
      o_hd5_out <= (others => '0');
      o_hdi_out <= (others => '0');
    elsif i_sysclk = '1' and i_sysclk'event then
      if new_hd = '1' then
        o_hd5_out <= scale_down_result;
        o_hdi_out <= hdi_save;
      end if;
    end if;
  end process;


  ------------------------------
  -- intermediate serial out --
  ------------------------------
  harm_out: calc_hd_harm_serializer
    generic map(
      one_harm_bits      => 26,
      harm_array_t_bits => 156
    )
    port map(

      i_n_reset      => i_n_reset,
      i_clk          => i_sysclk,

      i_harm_real    => harm_re_reg,
      i_harm_imag    => harm_im_reg,
      i_harm_ready   => i_new_data,

      o_ser_out      => o_harm_ser_out,
      o_ser_sending => o_harm_ser_sending
    );

end structure;
```

Quelltext B.12: calc_hd_TL.vhd – Toplevel Entity der HDI- und HD5-Berechnung. Siehe Abbildung B.6

B.2.11 calc_hd.vhd

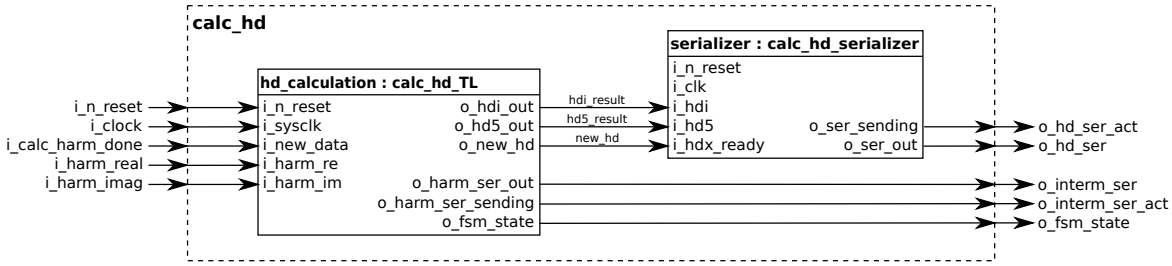


Abbildung B.7: Blockschaltbild des calc_hd-Moduls (Teil der Digitalen Signalverarbeitung). Siehe Quelltext B.13

```
--------------------------------------------------------------------------------
-- Entity: calc_hd
--------------------------------------------------------------------------------
-- Copyright 2012
-- Filename            : calc_hd.vhd
-- Creation date       : 2012-06-29
-- Author(s)           : Daniel Sabotta
-- Version             : 1.00
-- Description         : calculation of HD5 and HDI
--------------------------------------------------------------------------------
-- File History:
-- Date          Version  Author   Comment
--------------------------------------------------------------------------------
library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;
    use work.global_pkg.all;


package calc_hd_pkg is

--    type harm_array_t is array(0 to 5) of signed(25 downto 0);

  component calc_hd
    port(
        i_clock    : in std_logic;
        i_n_reset  : in std_logic;

        i_calc_harm_done : in std_logic;

        i_harm_real : in harm_array_t;
        i_harm_imag : in harm_array_t;

        -- serial intermediate results output
        o_interm_ser     : out std_logic;
        o_interm_ser_act : out std_logic;

        -- serial hd output
        o_hd_ser     : out std_logic;
        o_hd_ser_act : out std_logic;

        -- fsm state for debugging
        o_fsm_state    : out unsigned(3 downto 0)
    );
  end component;



end package;


--------------------------------------------------------------------------------

library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;
    use work.global_pkg.all;
    use work.calc_hd_TL_pkg.all;
```

```
    use work.calc_hd_harm_serializer_pkg.all;
    use work.calc_hd_serializer_pkg.all;

entity calc_hd is
    port(
        i_clock    : in std_logic;
        i_n_reset : in std_logic;

        i_calc_harm_done : in std_logic;

        i_harm_real : in harm_array_t;
        i_harm_imag : in harm_array_t;

        -- serial intermediate results output
        o_interm_ser      : out std_logic;
        o_interm_ser_act : out std_logic;

        -- serial hd output
        o_hd_ser      : out std_logic;
        o_hd_ser_act : out std_logic;

        -- fsm state for debugging
        o_fsm_state    : out unsigned(3 downto 0)
    );
end calc_hd;

architecture arch of calc_hd is
  signal new_hd : std_logic;
  signal hd5_result : unsigned(21 downto 0);
  signal hdi_result : unsigned(21 downto 0);

begin

-------------------------------------------------
---------------- Hier geprogged :-) -------------
-------------------------------------------------

hd_calculation: calc_hd_TL
  generic map(
    square_val_in_width           => 20,
    square_val_out_width          => 40,
    add_out_width                 => 44,
    calc_div_comp_out_width       => 44,
    sqrt_out_width                => 22,
    scale_down_out_width          => 22,
    scale_down_denominator_value  => 2965821  -- 2_965_820.80 mit length of sqrt_out_width
                  -- um die hochskalierung bei der ersten dividion herauszu rechnen also sqrt(2^[
                       calc_div_comp_out_width-1])
  )
  port map(
    i_sysclk      => i_clock,
    i_n_reset     => i_n_reset,

    i_new_data    => i_calc_harm_done,

    i_harm_re     => i_harm_real,
    i_harm_im     => i_harm_imag,

    o_new_hd      => new_hd,

    o_hd5_out     => hd5_result,
    o_hdi_out     => hdi_result,

    o_harm_ser_out     => o_interm_ser,
    o_harm_ser_sending => o_interm_ser_act,

    o_fsm_state => o_fsm_state

  );


serializer: calc_hd_serializer
    generic map(
      hdi_width => 16,
      hd5_width => 16
    )
    port map(

      i_n_reset   => i_n_reset,
      i_clk       => i_clock,

      i_hdi       => std_logic_vector(hdi_result(20 downto 5)),
      i_hd5       => std_logic_vector(hd5_result(20 downto 5)),
      i_hdx_ready => new_hd,
```

```
      o_ser_out    => o_hd_ser,
      o_ser_sending => o_hd_ser_act
    );

end arch;
```

Quelltext B.13: calc_hd – Toplevel Entity der HDI- und HD5-Berechnung inklusiver der seriellen Ausgabe. Siehe Abbildung B.7

B.2.12 fsm_calc_harm.vhd

```
------------------------------------------------------------------------
-- Entity: fsm_calc_harm
------------------------------------------------------------------------
-- Copyright 2012
-- Filename           : fsm_calc_harm.vhd
-- Creation date      : 2012-06-01
-- Author(s)          : Martin Krey
-- Version            : 1.00
-- Description        : Statemachine for harmonic sampling and
--                      harmonic calculation
------------------------------------------------------------------------
-- File History:
-- Date          Version  Author   Comment
------------------------------------------------------------------------
library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

package fsm_calc_harm_pkg is

  component fsm_calc_harm
    port(
            i_clock    : in std_logic;
            i_n_reset  : in std_logic;

            i_period_timer_stb : in std_logic;
            i_en_one_period    : in std_logic;

            i_sample_timer_trig  : in std_logic;
            o_sample_timer_load  : out std_logic;
            o_sample_timer_start : out std_logic;

            i_sample_done  : in std_logic;
            o_sample_start : out std_logic;

            o_lut_harm     : out unsigned(3 downto 0);
            o_lut_en_imag  : out std_logic;
            o_lut_addr     : out unsigned(5 downto 0);

            o_mul_en_square_0 : out std_logic;

            o_add_clear    : out std_logic;
            o_add_stb      : out std_logic;
            o_add_harm     : out unsigned(2 downto 0);
            o_add_en_imag  : out std_logic;
            o_add_en_add_1 : out std_logic;

            o_fsm_state : out unsigned(4 downto 0);

            o_done : out std_logic;

            o_fsm_next_state : out unsigned(4 downto 0)
    );
  end component;

end package;



------------------------------------------------------------------------

library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

entity fsm_calc_harm is
    port(
```

```
            i_clock    : in std_logic;
            i_n_reset  : in std_logic;

            i_period_timer_stb : in std_logic;
            i_en_one_period    : in std_logic;

            i_sample_timer_trig  : in std_logic;
            o_sample_timer_load  : out std_logic;
            o_sample_timer_start : out std_logic;

            i_sample_done  : in std_logic;
            o_sample_start : out std_logic;

            o_lut_harm     : out unsigned(3 downto 0);
            o_lut_en_imag  : out std_logic;
            o_lut_addr     : out unsigned(5 downto 0);

            o_mul_en_square_0 : out std_logic;

            o_add_clear    : out std_logic;
            o_add_stb      : out std_logic;
            o_add_harm     : out unsigned(2 downto 0);
            o_add_en_imag  : out std_logic;
            o_add_en_add_1 : out std_logic;

            o_fsm_state : out unsigned(4 downto 0);

            o_done : out std_logic;
            ------------------------------------
            o_fsm_next_state : out unsigned(4 downto 0)

    );
end fsm_calc_harm;

architecture arch of fsm_calc_harm is

--type state_fsm_t is (   s_wait_period_start,
--                       s_load_sample_timer,
--                       s_load_sample_timer_wait,
--                       s_start_sample,
--                       s_reset_first_sample,
--                       s_reset_harm_cnt,
--                       s_calc_sample_sum_square_0,
--                       s_calc_sample_sum_square_1,
--                       s_calc_sample_sum_square_2,
--                       s_calc_sample_sum_0,
--                       s_calc_sample_sum_1,
--                       s_calc_sample_sum_2,
--                       s_calc_sample_h_real_0,
--                       s_calc_sample_h_real_1,
--                       s_calc_sample_h_real_2,
--                       s_calc_sample_h_imag_0,
--                       s_calc_sample_h_imag_1,
--                       s_calc_sample_h_imag_2,
--                       s_inc_harm_cnt,
--                       s_calc_sample_end,
--                       s_inc_sample_counter,
--                       s_wait_sample_done,
--                       s_sample_done,
--                       s_set_last_sample,
--                       s_wait_sample_timer_trig,
--                       s_reset_sample_timer,
--                       s_done
--                   );

subtype state_fsm_t is unsigned(4 downto 0);

constant s_wait_period_start        : state_fsm_t := "00000";  -- 0
constant s_load_sample_timer        : state_fsm_t := "00001";  -- 1
constant s_load_sample_timer_wait   : state_fsm_t := "00010";  -- 2
constant s_start_sample             : state_fsm_t := "00011";  -- 3
constant s_reset_first_sample       : state_fsm_t := "00100";  -- 4
constant s_reset_harm_cnt           : state_fsm_t := "00101";  -- 5
constant s_calc_sample_sum_square_0 : state_fsm_t := "00110";  -- 6
constant s_calc_sample_sum_square_1 : state_fsm_t := "00111";  -- 7
constant s_calc_sample_sum_square_2 : state_fsm_t := "01000";  -- 8
constant s_calc_sample_sum_0        : state_fsm_t := "01001";  -- 9
constant s_calc_sample_sum_1        : state_fsm_t := "01010";  -- 10
constant s_calc_sample_sum_2        : state_fsm_t := "01011";  -- 11
constant s_calc_sample_h_real_0     : state_fsm_t := "01100";  -- 12
constant s_calc_sample_h_real_1     : state_fsm_t := "01101";  -- 13
constant s_calc_sample_h_real_2     : state_fsm_t := "01110";  -- 14
constant s_calc_sample_h_imag_0     : state_fsm_t := "01111";  -- 15
constant s_calc_sample_h_imag_1     : state_fsm_t := "10000";  -- 16
constant s_calc_sample_h_imag_2     : state_fsm_t := "10001";  -- 17
```

```
    constant s_inc_harm_cnt             : state_fsm_t := "10010";  -- 18
    constant s_calc_sample_end          : state_fsm_t := "10011";  -- 19
    constant s_inc_sample_counter       : state_fsm_t := "10100";  -- 20
    constant s_wait_sample_done         : state_fsm_t := "10101";  -- 21
156 constant s_sample_done              : state_fsm_t := "10110";  -- 22
    constant s_set_last_sample          : state_fsm_t := "10111";  -- 23
    constant s_wait_sample_timer_trig   : state_fsm_t := "11000";  -- 24
    constant s_reset_sample_timer       : state_fsm_t := "11001";  -- 25
    constant s_done                     : state_fsm_t := "11010";  -- 26
161
    signal state_fsm, next_state_fsm : state_fsm_t;

    signal sample_start_reset : std_logic;
    signal sample_start_set : std_logic;
166 signal sample_start : std_logic;

    signal sample_timer_start_reset : std_logic;
    signal sample_timer_start_set : std_logic;
    signal sample_timer_start : std_logic;
171
    -- signal sample_cnt : integer range 0 to 63;
    signal sample_cnt : unsigned(5 downto 0);
    signal sample_cnt_reset : std_logic;
    signal sample_cnt_inc : std_logic;
176 signal sample_cnt_end : std_logic;

    signal first_sample : std_logic;
    signal first_sample_set : std_logic;
    signal first_sample_reset : std_logic;
181
    signal last_sample : std_logic;
    signal last_sample_set : std_logic;
    signal last_sample_reset : std_logic;

186 signal harm_cnt : unsigned(3 downto 0);
    signal harm_cnt_reset : std_logic;
    signal harm_cnt_inc : std_logic;
    signal harm_cnt_end : std_logic;

191 signal one_period : std_logic;

    begin
        -- state changer
        process(i_n_reset, i_clock)
196     begin
            if i_n_reset = '0' then
                state_fsm <= s_wait_period_start;
            elsif i_clock = '1' and i_clock'event then
                state_fsm <= next_state_fsm;
201         end if;
        end process;

        -- statemachine
        process(state_fsm, i_period_timer_stb, i_sample_timer_trig, i_sample_done, first_sample, last_sample, sample_cnt_end,
              harm_cnt_end)
206     begin
            -- default values

            ---------------------------------
            -- registers
211         ---------------------------------
            first_sample_set <= '0';
            first_sample_reset <= '0';

            last_sample_set <= '0';
216         last_sample_reset <= '0';

            sample_start_set <= '0';
            sample_start_reset <= '0';

221         sample_timer_start_set <= '0';
            sample_timer_start_reset <= '0';

            sample_cnt_reset <= '0';
            sample_cnt_inc <= '0';
226
            harm_cnt_reset <= '0';
            harm_cnt_inc <= '0';

            ---------------------------------
231         -- others
            ---------------------------------
            o_sample_timer_load <= '0';
            o_done <= '0';
```

```
236         o_lut_en_imag <= '0';

            o_mul_en_square_0 <= '0';

            o_add_clear <= '0';
241         o_add_stb <= '0';
            o_add_en_imag <= '0';
            o_add_en_add_1 <= '0';

            next_state_fsm <= s_wait_period_start;
246
            ------------------------------
            -- fsm starts here
            ------------------------------
            case state_fsm is
251             when s_wait_period_start =>
                    -- wait for period start
                    if i_period_timer_stb = '1' then
                        next_state_fsm <= s_load_sample_timer;
                    else
256                     next_state_fsm <= s_wait_period_start;
                    end if;
                when s_load_sample_timer =>
                    -- calculate and save sample interval
                    o_sample_timer_load <= '1';
261                 o_add_clear <= '1';
                    -- reset all registers to start values
                    sample_cnt_reset <= '1';
                    first_sample_set <= '1';
                    last_sample_reset <= '1';
266                 sample_start_reset <= '1';
                    sample_timer_start_reset <= '1';
                    next_state_fsm <= s_load_sample_timer_wait;
                when s_load_sample_timer_wait =>
                    -- wait for takeover of sample interval in sample_timer
271                 next_state_fsm <= s_start_sample;
                when s_start_sample =>
                    -- start adc conversion and sample timer
                    sample_start_set <= '1';
                    sample_timer_start_set <= '1';
276                 -- first sample?
                    if first_sample = '1' then
                        -- no calculation needed
                        next_state_fsm <= s_reset_first_sample;
                    else
281                     -- do calculation for previous sample
                        next_state_fsm <= s_reset_harm_cnt;
                    end if;
                when s_reset_first_sample =>
                    -- clear first sample flag
286                 first_sample_reset <= '1';
                    next_state_fsm <= s_wait_sample_done;
                ------------------------------------------------------------
                ------------------------------
                -- calculations here
291             ------------------------------
                when s_reset_harm_cnt =>
                    harm_cnt_reset <= '1';
                    next_state_fsm <= s_calc_sample_sum_square_0;
                when s_calc_sample_sum_square_0 =>
296                 o_add_en_imag  <= '0';
                    o_add_en_add_1 <= '0';
                    o_mul_en_square_0 <= '1';
                    next_state_fsm <= s_calc_sample_sum_square_1;
                when s_calc_sample_sum_square_1 =>
301                 o_add_en_imag  <= '0';
                    o_add_en_add_1 <= '0';
                    o_mul_en_square_0 <= '1';
                    o_add_stb <= '1';
                    next_state_fsm <= s_calc_sample_sum_square_2;
306             when s_calc_sample_sum_square_2 =>
                    o_add_en_imag  <= '0';
                    o_add_en_add_1 <= '0';
                    o_mul_en_square_0 <= '1';
                    next_state_fsm <= s_calc_sample_sum_0;
311
                when s_calc_sample_sum_0 =>
                    o_add_en_imag  <= '1';
                    o_add_en_add_1 <= '1';
                    o_mul_en_square_0 <= '0';
316                 next_state_fsm <= s_calc_sample_sum_1;
                when s_calc_sample_sum_1 =>
                    o_add_en_imag  <= '1';
                    o_add_en_add_1 <= '1';
                    o_mul_en_square_0 <= '0';
```

```
321                 o_add_stb <= '1';
                    next_state_fsm <= s_calc_sample_sum_2;
                when s_calc_sample_sum_2 =>
                    o_add_en_imag  <= '1';
                    o_add_en_add_1 <= '1';
326                 o_mul_en_square_0 <= '0';
                    harm_cnt_inc <= '1';
                    next_state_fsm <= s_calc_sample_h_real_0;

                when s_calc_sample_h_real_0 =>
331                 o_lut_en_imag <= '0';
                    o_add_en_imag <= '0';
                    next_state_fsm <= s_calc_sample_h_real_1;
                when s_calc_sample_h_real_1 =>
                    o_lut_en_imag <= '0';
336                 o_add_en_imag <= '0';
                    o_add_stb <= '1';
                    next_state_fsm <= s_calc_sample_h_real_2;
                when s_calc_sample_h_real_2 =>
                    o_lut_en_imag <= '0';
341                 o_add_en_imag <= '0';
                    next_state_fsm <= s_calc_sample_h_imag_0;

                when s_calc_sample_h_imag_0 =>
                    o_lut_en_imag <= '1';
346                 o_add_en_imag <= '1';
                    next_state_fsm <= s_calc_sample_h_imag_1;
                when s_calc_sample_h_imag_1 =>
                    o_lut_en_imag <= '1';
                    o_add_en_imag <= '1';
351                 o_add_stb <= '1';
                    next_state_fsm <= s_calc_sample_h_imag_2;
                when s_calc_sample_h_imag_2 =>
                    o_lut_en_imag <= '1';
                    o_add_en_imag <= '1';
356                 if harm_cnt_end = '1' then
                        next_state_fsm <= s_calc_sample_end;
                    else
                        next_state_fsm <= s_inc_harm_cnt;
                    end if;
361
                when s_inc_harm_cnt =>
                    harm_cnt_inc <= '1';
                    next_state_fsm <= s_calc_sample_h_real_0;

366             when s_calc_sample_end =>
                    if last_sample = '1' then
                        -- all samples processed -> done
                        next_state_fsm <= s_done;
                    else
371                     next_state_fsm <= s_inc_sample_counter;
                    end if;
                ------------------------------------------------------------
                when s_inc_sample_counter =>
                    -- increment sample counter
376                 sample_cnt_inc <= '1';
                    next_state_fsm <= s_wait_sample_done;
                when s_wait_sample_done =>
                    if i_sample_done = '1' then
                        next_state_fsm <= s_sample_done;
381                 else
                        next_state_fsm <= s_wait_sample_done;
                    end if;
                when s_sample_done =>
                    -- stop adc
386                 sample_start_reset <= '1';
                    -- is period complete?
                    if sample_cnt_end = '1' then
                        -- calculation for last sample
                        next_state_fsm <= s_set_last_sample;
391                 else
                        -- next for sample timer
                        next_state_fsm <= s_wait_sample_timer_trig;
                    end if;
                when s_set_last_sample =>
396                 last_sample_set <= '1';
                    -- jump directly to calculation
                    next_state_fsm <= s_reset_harm_cnt;
                ------------------------------------------------------------
                when s_wait_sample_timer_trig =>
401                 -- wait for sample timer trigger
                    if i_sample_timer_trig = '1' then
                        next_state_fsm <= s_reset_sample_timer;
                    else
                        next_state_fsm <= s_wait_sample_timer_trig;
```

```
406                 end if;
            when s_reset_sample_timer =>
                -- reset sample timer
                sample_timer_start_reset <= '1';
                next_state_fsm <= s_start_sample;
411         ------------------------------------------------------------
            when s_done =>
                sample_timer_start_reset <= '1';
                o_done <= '1';
                next_state_fsm <= s_wait_period_start;
416         ------------------------------------------------------------
            when others =>
                next_state_fsm <= s_wait_period_start;
        end case;
    end process;
421


    -- sample_start signal reg
    process(i_n_reset, i_clock)
426 begin
        if i_n_reset = '0' then
            sample_start <= '0';
        elsif i_clock = '1' and i_clock'event then
            if sample_start_reset = '1' then
431             sample_start <= '0';
            elsif sample_start_set = '1' then
                sample_start <= '1';
            end if;
        end if;
436 end process;

    -- sample_timer_start signal reg
    process(i_n_reset, i_clock)
    begin
441     if i_n_reset = '0' then
            sample_timer_start <= '0';
        elsif i_clock = '1' and i_clock'event then
            if sample_timer_start_reset = '1' then
                sample_timer_start <= '0';
446         elsif sample_timer_start_set = '1' then
                sample_timer_start <= '1';
            end if;
        end if;
    end process;
451

    -- first_sample signal reg
    process(i_n_reset, i_clock)
    begin
        if i_n_reset = '0' then
456         first_sample <= '0';
        elsif i_clock = '1' and i_clock'event then
            if first_sample_reset = '1' then
                first_sample <= '0';
            elsif first_sample_set = '1' then
461             first_sample <= '1';
            end if;
        end if;
    end process;

466 -- last_sample signal reg
    process(i_n_reset, i_clock)
    begin
        if i_n_reset = '0' then
            last_sample <= '0';
471     elsif i_clock = '1' and i_clock'event then
            if last_sample_reset = '1' then
                last_sample <= '0';
            elsif last_sample_set = '1' then
                last_sample <= '1';
476         end if;
        end if;
    end process;

    -- sample counter
481 process(i_n_reset, i_clock)
    begin
        if i_n_reset = '0' then
            sample_cnt <= (others => '0');
            sample_cnt_end <= '0';
486     elsif i_clock = '1' and i_clock'event then
            if sample_cnt_reset = '1' then
                sample_cnt <= (others => '0');
                sample_cnt_end <= '0';
            elsif sample_cnt_inc = '1' and sample_cnt_end = '0' then
```

```
491                 sample_cnt <= sample_cnt + 1;
                end if;

                if sample_cnt = 63 then
                    sample_cnt_end <= '1';
496             else
                    sample_cnt_end <= '0';
                end if;

            end if;
501     end process;

        -- harm_cnt counter
        process(i_n_reset, i_clock)
        begin
506         if i_n_reset = '0' then
                harm_cnt <= (others => '0');
                harm_cnt_end <= '0';
            elsif i_clock = '1' and i_clock'event then
                if harm_cnt_reset = '1' then
511                 harm_cnt <= (others => '0');
                    harm_cnt_end <= '0';
                elsif harm_cnt_inc = '1' and harm_cnt_end = '0' then
                    harm_cnt <= harm_cnt + 1;
                end if;
516
                if harm_cnt = 5 then
                    harm_cnt_end <= '1';
                else
                    harm_cnt_end <= '0';
521             end if;

            end if;
        end process;

526     process(i_n_reset, i_clock)
        begin
            if i_n_reset = '0' then
                one_period <= '0';
            elsif i_clock = '1' and i_clock'event then
531             if i_period_timer_stb = '1' then
                    if i_en_one_period = '1' then
                        one_period <= '1';
                    else
                        one_period <= '0';
536                 end if;
                end if;
            end if;
        end process;

541

        o_sample_start <= sample_start;
        o_sample_timer_start <= sample_timer_start;

        o_lut_harm <= harm_cnt when one_period = '1' else
546                   harm_cnt(2 downto 0) & '0';

        o_lut_addr <= sample_cnt;

        o_add_harm <= harm_cnt(2 downto 0);
551
        o_fsm_state <= state_fsm;

        o_fsm_next_state <= next_state_fsm;

556 end arch;
```

Quelltext B.14: fsm_calc_harm.vhd – Zustandsautomat der DFT.

B.2.13 lut.vhd

```
--------------------------------------------------------------------------------
-- Entity: lut
--------------------------------------------------------------------------------
-- Copyright 2012
-- Filename           : lut.vhd
-- Creation date      : 2012-06-05
-- Author(s)          : Martin Krey
-- Version            : 1.00
-- Description        : cos/sin real/imag look-up table with 64 entries
--------------------------------------------------------------------------------
-- File History:
-- Date          Version  Author   Comment
--------------------------------------------------------------------------------
library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

package lut_pkg is

  component lut
    port(
            i_clock    : in std_logic;
            i_n_reset  : in std_logic;

            i_harm     : in unsigned(3 downto 0);
            i_en_imag  : in std_logic;
            i_addr     : in unsigned(5 downto 0);

            o_val : out signed(9 downto 0)
    );
  end component;

end package;


--------------------------------------------------------------------------------

library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

entity lut is
    port(
            i_clock    : in std_logic;
            i_n_reset  : in std_logic;

            i_harm     : in unsigned(3 downto 0);
            i_en_imag  : in std_logic;
            i_addr     : in unsigned(5 downto 0);

            o_val : out signed(9 downto 0)
    );
end lut;

architecture arch of lut is

    type array_t is array(0 to 16) of signed(9 downto 0);

    constant OMEGA : real := ieee.math_real.math_2_pi/real(64);

    --create LUT. calling this function creates an array of 17
    --            sin Values from >0degree to <90degree.
    function sin_lut_func return array_t is
        variable luts : array_t;
    begin
        for i in 0 to 16 loop
            luts(i) := to_signed(integer( ieee.math_real.sin(real(i)*omega)*
                                    real(2**(10-1)-1) ),
                                    luts(1)'length);
        end loop;
        return luts;
    end sin_lut_func;

    constant SIN_TAB : array_t := sin_lut_func;

    signal idx_0 : integer range 0 to 2**4;
    signal idx_1 : integer range 0 to 2**4;

    signal addr : unsigned(9 downto 0);
    signal quadrant : unsigned(1 downto 0);

begin
```

```
83
    addr <= i_addr * i_harm;

    idx_0 <= to_integer( addr(3 downto 0) );
    idx_1 <= 16 - to_integer( addr(3 downto 0) );
88
    quadrant <= addr(5 downto 4) + 1 when i_en_imag = '0' else     -- cos output
                addr(5 downto 4) + 0 when i_en_imag = '1';         -- sin output

    o_val <=  SIN_TAB(idx_0) when quadrant = "00" else
93            SIN_TAB(idx_1) when quadrant = "01" else
             -SIN_TAB(idx_0) when quadrant = "10" else
             -SIN_TAB(idx_1) when quadrant = "11";

    end arch;
```

Quelltext B.15: lut.vhd – LUT der DFT.

B.2.14 mul_para.vhd

```
    library ieee;
2     use ieee.std_logic_1164.all;
        use ieee.numeric_std.all;

    package mul_para_pkg is

7       component mul_para is
        generic(
            mul_width : natural
        );
        port(
12          i_clock    : in std_logic;
            i_n_reset : in std_logic;

            i_en_square_0 : in std_logic;
            i_mul_0 : in signed(mul_width - 1 downto 0);
17          i_mul_1 : in signed(mul_width - 1 downto 0);

            o_prod : out signed(2*mul_width - 1 downto 0)
        );
        end component;
22
    end mul_para_pkg;
    ------------------------------------------------------------------------
    library ieee;
        use ieee.std_logic_1164.all;
27      use ieee.numeric_std.all;

    entity mul_para is
        generic(
            mul_width : natural := 10
32      );
        port(
            i_clock    : in std_logic;
            i_n_reset : in std_logic;

37          i_en_square_0 : in std_logic;
            i_mul_0 : in signed(mul_width - 1 downto 0);
            i_mul_1 : in signed(mul_width - 1 downto 0);

            o_prod : out signed(2*mul_width - 1 downto 0)
42      );
    end mul_para;

    architecture behaviour of mul_para is

47      signal mul_1  : signed(mul_width - 1 downto 0);

    begin

        mul_1 <= i_mul_0 when i_en_square_0 = '1' else
52               i_mul_1;

        o_prod <= i_mul_0 * mul_1;

    end behaviour;
```

Quelltext B.16: mul_para.vhd – Multiplizierer der DFT.

B.2.15 period_time_chk.vhd

```
--------------------------------------------------------------------------------
-- Entity:
--------------------------------------------------------------------------------
-- Copyright 2012
-- Filename          : .vhd
-- Creation date     : 05.07.2012
-- Author(s)         : Martin Krey
-- Description       :
--
--------------------------------------------------------------------------------

library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

package period_time_chk_pkg is

    component period_time_chk
    generic(
        period_time_width : positive := 25
    );
    port(
        i_clock    : in std_logic;
        i_n_reset : in std_logic;

        i_period_time : in unsigned(period_time_width-1 downto 0);
        i_period_stb  : in std_logic;

        o_error_dev : out std_logic
    );
    end component;

end package;

--------------------------------------------------------------------------------

library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;
--    use ieee.std_logic_arith.all;

entity period_time_chk is
    generic(
        period_time_width : positive := 25
    );
    port(
        i_clock    : in std_logic;
        i_n_reset : in std_logic;

        i_period_time : in unsigned(period_time_width-1 downto 0);
        i_period_stb  : in std_logic;

        o_error_dev : out std_logic
    );
end period_time_chk;

architecture arch of period_time_chk is

    signal period_time_pre : unsigned(period_time_width-1 downto 0);
    signal period_time_diff : signed(period_time_width-1 downto 0);
    signal one_time : std_logic;
    signal error_dev : std_logic;

begin

    -- register for sample interval
    process(i_n_reset, i_clock)
    begin
        if i_n_reset = '0' then
            period_time_pre <= (others => '0');
            one_time <= '1';
            error_dev <= '0';
        elsif i_clock = '1' and i_clock'event then
            if i_period_stb = '1' and one_time = '0' then
                if (abs signed('0' & period_time_pre)-signed('0' & i_period_time)) > signed(period_time_pre srl 5) then
                    error_dev <= '1';
                else
                    error_dev <= '0';
                end if;
                period_time_pre <= i_period_time;
            elsif i_period_stb = '1' and one_time = '1' then
                one_time <= '0';
```

```
            end if;
        end if;
    end process;

    o_error_dev <= error_dev;

end arch;
```

Quelltext B.17: period_time_chk.vhd – Kontrolle der Periodenlänge.

B.2.16 period_timer.vhd

```
------------------------------------------------------------------------
-- Entity: period_timer
------------------------------------------------------------------------
-- Copyright ... 2010
-- Filename          : period_timer.vhd
-- Creation date     : 2010-10-08
-- Author(s)         : martin
-- Version           : 1.00
-- Description       : <short description>
------------------------------------------------------------------------
-- File History:
-- Date         Version  Author   Comment
-- 2010-10-08   1.00     martin     Creation of File
------------------------------------------------------------------------
library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

package period_timer_pkg is

  component period_timer
    generic(
        underflow_value   : positive := 12*16*64;
        period_time_width : positive := 25
    );
    port(
            i_clock : in std_logic;
            i_n_reset : in std_logic;

            i_sig_inc       : in std_logic;
            i_en_one_period : in std_logic;

            o_period_time : out unsigned(period_time_width-1 downto 0);
            o_period_stb  : out std_logic;

            o_error_ovf_unf : out std_logic
    );
  end component;

end package;


------------------------------------------------------------------------


library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

entity period_timer is
    generic(
        underflow_value   : positive := 12*16*64;
        period_time_width : positive := 25
    );
    port(
            i_clock : in std_logic;
            i_n_reset : in std_logic;

            i_sig_inc       : in std_logic;
            i_en_one_period : in std_logic;

            o_period_time : out unsigned(period_time_width-1 downto 0);
            o_period_stb  : out std_logic;

            o_error_ovf_unf : out std_logic
    );
```

```
end period_timer;

architecture arch of period_timer is

type state_fsm_t is (   s_init,
                        s_0,
                        s_1
                    );
signal state_fsm : state_fsm_t;

signal timer_val : unsigned(24 downto 0);
signal timer_ovf : std_logic;
signal timer_unf : std_logic;

signal one_period : std_logic;

signal period_time : unsigned(24 downto 0);
signal period_stb : std_logic;

signal period_error : std_logic;

begin

    process(i_n_reset, i_clock)
    begin
        if i_n_reset = '0' then
            state_fsm <= s_init;
      timer_val <= (others => '0');
      timer_ovf <= '0';
      timer_unf <= '0';
            period_time <= (others => '0');
            period_error <= '0';
            period_stb <= '0';
            one_period <= '0';
      elsif i_clock = '1' and i_clock'event then
            case state_fsm is
                when s_init =>
                    if i_sig_inc = '1' then
            one_period <= i_en_one_period;
                        state_fsm <= s_0;
                    end if;
                when s_0 =>
                    if i_sig_inc = '1' and one_period = '1' then
                        period_time <= timer_val;
                        if timer_unf = '0' and timer_ovf = '0' then
                            period_stb <= '1';
                            period_error <= '0';
                        else
                            period_stb <= '0';
                            period_error <= '1';
                        end if;
                        state_fsm <= s_1;
                    elsif i_sig_inc = '1' and one_period = '0' then
                        one_period <= '1';
                    end if;

                    if timer_val < underflow_value then
                        timer_unf <= '1';
                    else
                        timer_unf <= '0';
                    end if;

                    if timer_val = (2**period_time_width)-1 then
                        timer_ovf <= '1';
                    else
                        timer_val <= timer_val + 1;
                    end if;

                when s_1 =>
                    one_period <= i_en_one_period;
                timer_val <= (others => '0');
                timer_ovf <= '0';
                    period_stb <= '0';
                    state_fsm <= s_0;
            end case;
        end if;
    end process;

    o_period_time <= period_time;
    o_period_stb <= period_stb;

    o_error_ovf_unf <= period_error;

end arch;
```

Quelltext B.18: period_timer.vhd – Messen der Periodenlänge.

B.2.17 sample_timer.vhd

```
--------------------------------------------------------------------------------
-- Entity: sample_timer
--------------------------------------------------------------------------------
-- Copyright 2012
-- Filename            : sample_timer.vhd
-- Creation date       : 31.05.2012
-- Author(s)           : Martin Krey
-- Description         : This module controls ADC sampling depending on
--                       period_timer value, always 64 samples
--------------------------------------------------------------------------------

library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

package sample_timer_pkg is

    component sample_timer
    generic(
        sample_interval_correction : natural := 4;
        period_time_width : positive := 25;
        samples_per_period : positive := 64 -- ld(64) inactive
    );
    port(
        i_clock    : in std_logic;
        i_n_reset : in std_logic;

        i_period_time : in unsigned(period_time_width-1 downto 0);
        i_load : in std_logic;
        i_start : in std_logic;

        o_trig : out std_logic
    );
    end component;

end package;

--------------------------------------------------------------------------------

library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;
--    use ieee.std_logic_arith.all;

entity sample_timer is
    generic(
        sample_interval_correction : natural := 4;
        period_time_width : positive := 25;
        samples_per_period : positive := 64 -- ld(64) inactive
    );
    port(
        i_clock    : in std_logic;
        i_n_reset : in std_logic;

        i_period_time : in unsigned(period_time_width-1 downto 0);
        i_load : in std_logic;
        i_start : in std_logic;

        o_trig : out std_logic
    );
end sample_timer;

architecture arch of sample_timer is

-- http://www.velocityreviews.com/forums/t22799-log2-n.html
-- find minimum number of bits required to
-- represent N as an unsigned binary number
function log2_ceil(N: natural) return positive is
begin
    if N < 2 then
        return 1;
    else
```

```
        return 1 + log2_ceil(N/2);
    end if;
end;

constant ld_samples_per_period : positive := log2_ceil(samples_per_period);

-- take samples_per_period samples per signal period
-- fit the length of the compare value
signal sample_interval : unsigned(i_period_time'length-ld_samples_per_period downto 0);
signal sample_interval_cnt : unsigned(i_period_time'length-ld_samples_per_period downto 0);
signal trig : std_logic;

begin

    -- register for sample interval
    process(i_n_reset, i_clock)
    begin
        if i_n_reset = '0' then
            sample_interval <= (others => '0');
        elsif i_clock = '1' and i_clock'event then
            if i_load = '1' then
                sample_interval <= i_period_time(i_period_time'left downto ld_samples_per_period-1);
            end if;
        end if;
    end process;

    process(i_n_reset, i_clock)
    begin
        if i_n_reset = '0' then
            trig <= '0';
            sample_interval_cnt <= (others => '0');
        elsif i_clock = '1' and i_clock'event then
            if i_start = '0' then
                trig <= '0';
                sample_interval_cnt <= sample_interval;
            elsif i_start = '1' and trig = '0' then
                if sample_interval_cnt = sample_interval_correction then
                    trig <= '1';
                else
                    sample_interval_cnt <= sample_interval_cnt - 1;
                end if;
            elsif i_start = '1' and trig = '1' then
                trig <= '1';
            end if;
        end if;
    end process;

    o_trig <= trig;

end arch;
```

Quelltext B.19: sample_timer.vhd – Erzeugen der Abtastfrequenz.

B.2.18 sample.vhd

```
------------------------------------------------------------------------
-- Entity: sample
------------------------------------------------------------------------
-- Copyright 2012
-- Filename          : sample.vhd
-- Creation date     : 01.06.2012
-- Author(s)         : Martin Krey
-- Description       : Interface to ADC
--
------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

package sample_pkg is

    component sample
    generic(
        adc_bias_val : natural
    );
    port(
        i_clock    : in std_logic;
```

```
        i_n_reset : in std_logic;

        i_adc_sig : in unsigned(9 downto 0);
        i_adc_eoc : in std_logic;
        i_start   : in std_logic;

        o_adc_clock  : out std_logic;
        o_adc_start  : out std_logic;
        o_sample_val : out signed(9 downto 0);
        o_done       : out std_logic
    );
    end component;

end package;

-------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity sample is
    generic(
        adc_bias_val : natural
    );
    port(
        i_clock   : in std_logic;
        i_n_reset : in std_logic;

        i_adc_sig : in unsigned(9 downto 0);
        i_adc_eoc : in std_logic;
        i_start   : in std_logic;

        o_adc_clock  : out std_logic;
        o_adc_start  : out std_logic;
        o_sample_val : out signed(9 downto 0);
        o_done       : out std_logic
    );
end sample;

architecture arch of sample is

signal sample_data : std_logic;

signal adc_clk_reg : unsigned(3 downto 0) := (others => '0');
signal adc_start : std_logic;
signal sample_val : signed(9 downto 0) := (others => '0');
signal done : std_logic;

begin

    -- generate adc clock
    gen_adc_clock : process(i_n_reset, i_clock)
    begin
        if i_n_reset = '0' then
            adc_clk_reg <= (others => '0');
        elsif i_clock = '1' and i_clock'event then
            adc_clk_reg <= adc_clk_reg + 1;
        end if;
    end process;

    process(i_n_reset, i_clock)
    begin
        if i_n_reset = '0' then
            sample_data <= '0';
            sample_val <= (others => '0');
            adc_start <= '0';
            done <= '0';
        elsif i_clock = '1' and i_clock'event then
            if i_start = '0' then
                sample_data <= '0';
                adc_start <= '0';
                done <= '0';
            elsif i_start = '1' and sample_data = '0' then
                if i_adc_eoc = '1' then
                    sample_data <= '1';
                    adc_start <= '0';
                else
                    adc_start <= '1';
                end if;
            elsif i_start = '1' and sample_data = '1' then
                if i_adc_eoc = '0' then
                    sample_val <= signed(i_adc_sig) - to_signed(adc_bias_val, 10);
                    done <= '1';
                end if;
```

```
            end if;
        end if;
    end process;

    o_adc_clock <= adc_clk_reg(3);
    o_adc_start <= adc_start;
    o_sample_val <= sample_val;
    o_done <= done;

end arch;
```

Quelltext B.20: sample.vhd – Ansteuerung des ADC.

B.2.19 smart_comp.vhd

```
------------------------------------------------------------------------
-- Entity: smart_comp
------------------------------------------------------------------------
-- Copyright 2012
-- Filename           : smart_comp.vhd
-- Creation date      : 31.05.2010
-- Author(s)          : Martin Krey
-- Description        : implements smart comparator function,
--                      outputs sig_inc on rising edge of signal period
------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

package smart_comp_pkg is
  component smart_comp
  port(
        i_clock    : in std_logic;
        i_n_reset  : in std_logic;

        i_comp_h : in std_logic;
        i_comp_l : in std_logic;

        o_sig_inc     : out std_logic;
        o_zero_cross : out std_logic
  );
  end component;
end package;

------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity smart_comp is
  port(
        i_clock    : in std_logic;
        i_n_reset  : in std_logic;

        i_comp_h : in std_logic;
        i_comp_l : in std_logic;

        o_sig_inc     : out std_logic;
        o_zero_cross : out std_logic
  );
end smart_comp;

architecture arch of smart_comp is
    constant CDC_STAGES : natural := 2;

    type state_fsm_t is (   s_0,
                            s_1,
                            s_2,
                            s_3
                        );
    signal state_fsm : state_fsm_t;

    type cdc_reg_t is array(0 to CDC_STAGES-1) of std_logic;

    signal comp_h : cdc_reg_t;
    signal comp_l : cdc_reg_t;
```

```
    signal sig_inc : std_logic;
    signal zero_cross : std_logic;
begin

  -- synchronize input signals
  process(i_n_reset, i_clock)
  begin
        if i_n_reset = '0' then
            for i in 0 to CDC_STAGES-1 loop
                comp_h(i) <= '0';
                comp_l(i) <= '0';
            end loop;
        elsif i_clock = '1' and i_clock'event then
            comp_h(0) <= i_comp_h;
            comp_h(1) <= comp_h(0);

            comp_l(0) <= i_comp_l;
            comp_l(1) <= comp_l(0);
        end if;
  end process;

    -- control fsm to run through all comparator levels
    process(i_n_reset, i_clock)
    begin
        if i_n_reset = '0' then
            state_fsm <= s_0;
            sig_inc <= '0';
            zero_cross <= '0';
        elsif i_clock = '1' and i_clock'event then
            -- default values
            sig_inc <= '0';
            zero_cross <= '0';

            case state_fsm is
              when s_0 =>
                  if comp_l(CDC_STAGES-1) = '0' and comp_h(CDC_STAGES-1) = '0' then
                      state_fsm <= s_1;
                        zero_cross <= '1';
                  end if;
              when s_1 =>
                  if comp_l(CDC_STAGES-1) = '1' and comp_h(CDC_STAGES-1) = '0' then
                      state_fsm <= s_2;
                  end if;
              when s_2 =>
                  if comp_l(CDC_STAGES-1) = '1' and comp_h(CDC_STAGES-1) = '1' then
                      state_fsm <= s_3;
                      sig_inc <= '1';
                        zero_cross <= '1';
                  end if;
              when s_3 =>
                  if comp_l(CDC_STAGES-1) = '1' and comp_h(CDC_STAGES-1) = '0' then
                      state_fsm <= s_0;
                  end if;
            end case;
        end if;
    end process;

    o_sig_inc <= sig_inc;
    o_zero_cross <= zero_cross;

end arch;
```

Quelltext B.21: smart_comp.vhd – Implementierung des Smart Comparator.

C Encounter RTL Compiler-Skripte

Mithilfe dieser Skripte ist es möglich, die digitale Logik des Protokollgenerators und der digitalen Signalverarbeitung mit dem Enocounter RTL Compiler zu synthetisieren.

C.1 Protokollgenerator (abs_manchester_encoder)

```
##############################################################################
## Title       : Encounter(R) RTL Compiler - Script
## Project     : ESZ-ABS
##############################################################################
## File        : vhdlRTL.tcl
## Author      : Daniel Sabotta
## Company     : HAW Hamburg
## Created     : 2012-05-23
## Last update: 2012-05-23
##############################################################################
## Description: Load this file into Encounter(R) RTL Compiler - Script to
##              synthesize a vhdl file
##
##              Usage: "rc -f vhdlRTL.tcl"
##
##              Original file from Erik Brunvand, 2008
##############################################################################
## Revisions  :
## Date        Version  Author  Description
## 2012-05-23  1.0      sabotta_d   Created
##############################################################################

## Set the search paths to the libraries and HDL
## Renenber that "." yout current directory
set_attribute hdl_search_path   {.}                                 ;# Search Path for VHDL and Verilog files
set_attribute lib_search_path   {/home/cdsmgr/ams/liberty/c35_3.3V} ;# Search Path for library files
set_attribute library           [list c35_CORELIB.lib]              ;# Target library
set_attribute information_level 6                                   ;# See a lot of warnings


set hdlFiles        [list abs_manchester_encoder.vhd ]  ;# All HDL files
set basename        abs_manchester_encoder              ;# name of toplevel module
set reportPath      {./synthesis_reports/}              ;# Save created Reports here
set hdlSavePath     {./}                                ;# save generatet HDL file here
set runName         synthesized                         ;# name appended to output files

set clkName1        i_sysClk    ;# clock name
set clkName2        internalClk ;# clock name

set clkPeriod1_ps   62500       ;# clock periode in ps
set clkPeriod2_ps   25000000    ;# clock periode in ps

set clkInDelay_ps   250         ;# delay from clock to input valid
set clkOutDelay_ps  250         ;# delay from clock to output valid

##############################################################################
##             below here shouldn't need to be changed...                   ##
##############################################################################

## Analyze and Elaborate the HDL files
read_hdl    -vhdl   ${hdlFiles}
elaborate   ${basename}

## Apply Constraints and generate clocks
#set clock [define_clock -period ${clkPeriod_ps} -name ${clkName} [clock_ports] define_clock -period ${clkPeriod_ps}
      -name ${clkName} [clock_ports]]
set clock1 [define_clock -period ${clkPeriod1_ps} -name ${clkName1} [clock_ports]]
set clock2 [define_clock -period ${clkPeriod2_ps} -name ${clkName2} [clock_ports]]

external_delay -input   $clkInDelay_ps  -clock ${clkName1} [find / -port ports_in/*]
external_delay -output  $clkOutDelay_ps -clock ${clkName1} [find / -port ports_out/*]
```

```
external_delay -input   $clkInDelay_ps  -clock ${clkName2} [find / -port ports_in/*]
external_delay -output  $clkOutDelay_ps -clock ${clkName2} [find / -port ports_out/*]

## Sets transition to default values for Synopsys SDC format,
## fall/rise 400ps
dc::set_clock_transition .4 $clkName1
dc::set_clock_transition .4 $clkName2

## check that the design is OK so far
check_design -unresolved
report timing -lint

## Synthesize the design to the target library
synthesize -to_mapped

## Write out the reports
report timing > ${reportPath}${basename}_${runName}_timing.rep
report gates  > ${reportPath}${basename}_${runName}_cell.rep
report power  > ${reportPath}${basename}_${runName}_power.rep

## Write out the structural Verilog and sdc files
write_hdl -mapped >  ${hdlSavePath}${basename}_${runName}.v
write_sdc >  ${hdlSavePath}${basename}_${runName}.sdc

# write sdf-timing file with
#   -prec 3             : Use 3 digits floating point in delay calculation
#   -edges check_edge   : Check which edge delays are defined in technology file and calculate only those edges.
write_sdf -prec 3 -edges check_edge > ${hdlSavePath}${basename}_${runName}.sdf

# log actual time in logfile
date

## exit rc (usefull, when the command "rc -f <thisFileName>" is used)
#quit
```

Quelltext C.1: abs_manchester_encoder_rc-compiler.tcl – Tcl-Skript zum Synthetisieren des Protokollgenerators mit dem dem Enocounter RTL Compiler.

C.2 Digitale Signalverarbeitung (diagnosis_top)

```
################################################################################
## Title      : Encounter(R) RTL Compiler - Script
## Project    : ESZ-ABS
################################################################################
## File       : vhdlRTL.tcl
## Author     : Daniel Sabotta
## Company    : HAW Hamburg
## Created    : 2012-05-23
## Last update: 2012-05-23
################################################################################
## Description: Load this file into Encounter(R) RTL Compiler - Script to
##              synthesize a vhdl file
##
##              Usage: "rc -f vhdlRTL.tcl"
##
##              Original file from Erik Brunvand, 2008
################################################################################
## Revisions  :
## Date        Version  Author  Description
################################################################################

## Set the search paths to the libraries and HDL
## Renenber that "." yout current directory
set_attribute hdl_search_path   {../}           ;# Search Path for VHDL and Verilog files
set_attribute lib_search_path   {/home/cdsmgr/ams/liberty/c35_3.3V}     ;# Search Path for library files
set_attribute library           [list c35_CORELIB.lib]                  ;# Target library
set_attribute information_level 6                                       ;# See a lot of warnings


set hdlFiles        [list global_pkg.vhd analog_mux.vhd smart_comp.vhd period_timer.vhd sample_timer.vhd sample.vhd
     lut.vhd mul_para.vhd add.vhd fsm_calc_harm.vhd period_time_chk.vhd calc_hd_emu.vhd calc_hd_square.vhd
     calc_hd_add_samples.vhd calc_hd_div.vhd calc_hd_fsm.vhd calc_hd_sub.vhd calc_hd_harm_serilizer.vhd
     calc_hd_in_reg.vhd calc_hd_sqrt.vhd hdSerilizer.vhd calc_hd_TL.vhd calc_hd.vhd diagnosis_top.vhd]  ;# All HDL files
set basename        diagnosis_top                               ;# name of toplevel module
set savePath        {./}            ;# Save created Files here
set runName         syn                     ;# name appended to output files
```

```
34
set clkName          i_clock      ;# clock name
set clkPeriod_ps     62500        ;# clock periode in ps
set clkInDelay_ps    250          ;# delay from clock to input valid
set clkOutDelay_ps   250          ;# delay from clock to output valid
39



################################################################################
##              below here shouldn't need to be changed...                    ##
44 ################################################################################

## Analyze and Elaborate the HDL files
read_hdl     -vhdl    ${hdlFiles}
elaborate    ${basename}
49
## Apply Constraints and generate clocks
set clock [define_clock -period ${clkPeriod_ps} -name ${clkName} [clock_ports]]
external_delay -input $clkInDelay_ps -clock ${clkName} [find / -port ports_in/*]
external_delay -output $clkOutDelay_ps -clock ${clkName} [find / -port ports_out/*]
54
## Sets transition to default values for Synopsys SDC format,
## fall/rise 400ps
dc::set_clock_transition .4 $clkName

59 ## check that the design is OK so far
check_design -unresolved
report timing -lint

## Synthesize the design to the target library
64 synthesize -to_mapped

## Write out the reports
report timing > ${savePath}${basename}_${runName}_timing.rep
report gates  > ${savePath}${basename}_${runName}_cell.rep
69 report power  > ${savePath}${basename}_${runName}_power.rep

## Write out the structural Verilog and sdc files
write_hdl -mapped >  ${savePath}${basename}_${runName}.v
write_sdc >  ${savePath}${basename}_${runName}.sdc
74
# write sdf-timing file with
#    -prec 3              : Use 3 digits floating point in delay calculation
#    -edges check_edge    : Check which edge delays are defined in technology file and calculate only those edges.
write_sdf -prec 3 -edges check_edge > ${savePath}${basename}_${runName}.sdf
79
# log actual time in logfile
date

## exit rc (usefull, when the command "rc -f <thisFileName>" is used)
84 #quit
```

Quelltext C.2: diagnosis_top_rc-compiler.tcl – Tcl-Skript zum Synthetisieren der digitalen Signalverarbeitung mit dem Enocounter RTL Compiler.

D Skripte zum Auslesen der HDI- und HD5-Werte

D.1 hd_ser.py

```
"""
Created on Tue Sep 25 14:46:27 2012

@author: sabotta_d


Testen des CMOS-Testchips:

Der CMOS-Testchips muss wie folgt an ein MSO3034 Oszilloscope
angeschlossen werden:

Scope | CMOS-Testchip
------+--------------
D3    | hd_ser_act
D4    | hd_ser
D7    | clk

IP des Scopes = 192.168.0.12

Dann liefert dieses Skript den HDI- und HD5 wert, den der
Chip ausgibt. (natuerlich nur, wenn das Skript gestartet wird
waerend die Daten auf dem Scope angezeigt werden)...

"""

import tek_aufnahmen
import mso3034
import matplotlib.pyplot as plt
import pickle
import deserializer



wave = tek_aufnahmen.get_plot_waveform(ip="192.168.0.12",
                                       channels=["D7", "D4", "D3"],
                                       labels=["clk", "hd_ser", "hd_ser_act"],
                                       title="hd_after-fib-1",
                                       xscale="m",
                                       plot = True)


ser_bits = "".join([str(x) for x in deserializer.deserializer(clk=wave[0][1],ser=wave[1][1],ser_act=wave[2][1])])

hdi = int(ser_bits[ 0:16],2)
hd5 = int(ser_bits[16:32],2)
print("HDI: {0:016b}b = {0:04x}h => {1:2.3f}%".format(hdi, 100*hdi/2.0**16))
print("HD5: {0:016b}b = {0:04x}h => {1:2.3f}%".format(hd5, 100*hd5/2.0**16))
```

Quelltext D.1: hd_ser.py – HDI- und HD5-Werte über ein MSO3034 Oszilloskop auslesen und in Prozent darstellen.

D.2 deserializer.py

```
"""
Created on Mon Oct  8 15:02:00 2012

@author: sabotta_d

"""

def deserializer(clk, ser, ser_act, thres = 0.5):
    """
    Waerend die Signalleitung ser_act einen High-Pegel liefert,
    werden die Daten (ser), eingelsen.
    Dabei werden die Daten synchron zum Systemtakt (clk),
    bei fallender Taktflanke eingelesen.
    Dabei werden alle Signalpegel über thres als logisch High
    erkannt.
    """
    serial = list()


    # quantisieren
    clk = [1 if c >= thres else 0 for c in clk]
    ser = [1 if c >= thres else 0 for c in ser]
    ser_act = [1 if c >= thres else 0 for c in ser_act]


    for t in xrange(1,len(clk)):
        # negative flanke finden
        if clk[t-1] == 1 and clk[t] == 0:
            if ser_act[t] > thres:
                serial.append(ser[t])

    return serial
```

Quelltext D.2: deserializer.py – Serielle Daten parallelisieren.

D.3 tek_aufnahmen.py

```
"""
Created on Thu Oct  4 08:37:29 2012

@author: sabotta_d
"""

import mso3034
import matplotlib.pyplot as plt
import pickle


__scalings = {u"":1, u"m":1e3, unichr(0x03BC):1e6, u"n":1e9}

def get_plot_waveform(ip="192.168.0.14", channels = ["CH1"], labels=["CH1"],
                      title = None, xscale=u"m", yscale="",
                      plot = True):
    """
    plots all channels and saves it in as a pdf (title.pdf)
    saves wavforms in a pickle-file (title.picle)

    xsacle and yscale can be "m" "u" "n" ""
    """

    if xscale == "u":
        xscale = unichr(0x03BC)

    if yscale == "u":
        yscale = unichr(0x03BC)

    if len(channels) != len(labels):
        raise ValueError("labels and channels must have same length")

    tek = mso3034.mso3034(ip)

    waveforms = list()
```

```
    for ch in channels:
        header, wf = tek.get_waveform(ch)
        waveforms.append(wf)

    for ii in xrange(len(channels)):
        plt.plot(waveforms[ii][0], waveforms[ii][1], label=labels[ii])


    # xticks
    locs,labels = plt.xticks()
    plt.xticks(locs, map(lambda x: "%g" % x, locs * __scalings[xscale]))
    plt.xlabel(u"Time in " + xscale + "s")
    plt.xlim(min(waveforms[0][0]), max(waveforms[0][0]))

    # ytikcs
    locs,labels = plt.yticks()
    #plt.yticks(locs, map(lambda x: "%.1f" % x, locs*1e9))
    plt.yticks(locs, map(lambda x: "%g" % x, locs * __scalings[yscale]))
    plt.ylabel(u'Voltage in ' + yscale + 'V')

    plt.legend()
    plt.grid(True)

    if title != None:
        with open(title+".pickle", "wb") as file:
            pickle.dump(waveforms, file)

        plt.savefig(title+".pdf", format="pdf")
    if plot:
        plt.show()

    return waveforms

def plot_from_pickle(labels=["CH1"], title = None, xscale=u"m", yscale="", pdf = None):
    """
    get wavforms from a pickle-file (title.picle)
    and plots it

    xsacle and yscale can be "m" "u" "n" ""
    """

    if xscale == "u":
        xscale = unichr(0x03BC)

    if yscale == "u":
        yscale = unichr(0x03BC)

    waveforms = list()
    with open(title+".pickle", "rb") as file:
        waveforms = pickle.load(file)

    for ii in xrange(len(waveforms)):
        plt.plot(waveforms[ii][0], waveforms[ii][1], label=labels[ii])

    # xticks
    locs,labels = plt.xticks()
    plt.xticks(locs, map(lambda x: "%g" % x, locs * __scalings[xscale]))
    plt.xlabel(u"Time in " + xscale + "s")
    plt.xlim(min(waveforms[0][0]), max(waveforms[0][0]))

    # ytikcs
    locs,labels = plt.yticks()
    #plt.yticks(locs, map(lambda x: "%.1f" % x, locs*1e9))
    plt.yticks(locs, map(lambda x: "%g" % x, locs * __scalings[yscale]))
    plt.ylabel(u'Voltage in ' + yscale + 'V')

    plt.legend()
    plt.grid(True)

    if pdf != None:
        plt.savefig(title+".pdf", format="pdf")

    plt.show()

    return waveforms
```

Quelltext D.3: tek_aufnahmen.py – Oszilloskop über Ethernet auslesen und Daten ausgeben oder abspeichern.

D.4 mso3034.py

```
"""
tds3014b.py - Script to provide control and data acquisition from the
Tektronix TDS3014B oscilloscope via Ethernet.

Example Usage:
>>> import tds3014b
>>> t = tds3014b.tds3014b('192.168.0.14')

Send a GPIB command to the scope - the return value is the response from the scope.
>>> t.gpib_cmd('*IDN?')
'TEKTRONIX,TDS 3014B,0,CF:91.1CT FV:v3.27 TDS3GV:v1.00 TDS3FFT:v1.00 TDS3TRG:v1.00\n'

Fetch the current data for a waveform on the specified channel using
get_waveform.  get_waveform returns a header, containing metadata
about the waveform, and the data as a list of points.

>>> header, data = t.get_waveform('CH1') # returns header (dict) and converted data points (array)
>>> import matplotlib.pyplot as plt
>>> plt.plot(meas[0], meas[1]) # plot CH1
>>> plt.show()

"""
import httplib2 # for Http
import urllib # for urlencode


class mso3034:
    def __init__(self, ip_addr):
        self.ip_addr = ip_addr
        self.base_url = "http://%s" % (ip_addr)
        self.h = httplib2.Http()

    def gpib_cmd(self, cmd):
        """
        Send a command to the scope and gets the response, if any.
        Commands are anything from the programming menu that you would
        typically send over GPIB, such as 'BUSY?' or 'FPANEL:PRESS
        clearmenu'.  The commands are sent by encoding them into a URL
        like this: http://192.168.0.106/?COMMAND=:BUSY?
        """
        # Commands that start with '*' (like *IDN?) shouldn't get a
        # ':' prefix.  Not exactly sure what the spec is, but this
        # works.
        if(cmd.startswith('*')):
            prefix = ''
        else:
            prefix = ':'

        cmd_url = "?COMMAND=%s%s" % (prefix, urllib.quote_plus(cmd))
        url = '/'.join([self.base_url, cmd_url])
        f = urllib.urlopen(url)
        content = f.read()
        f.close()
        return content

    def force_trig(self):
        """ If the scope is in a state where the force trig button
        would trigger it, this would do the same thing.
        """
        self.gpib_cmd('FPANEL:PRESS FORCETRIG')

    def get_waveform(self, channel="CH1"):
        """
        Returns the data points that make up the currently displayed
        waveform. There will be 10,000 points spanning the X-axis time, and 7
        bits spanning the Y-axis voltage. The time and voltage range depend on
        how the scope is currently configured, and this configuration is
        returned as metadata depending on the value of format.

        Arguments:
         channel: 'CH1', 'CH2', 'CH3', 'CH4', 'D0' ... 'D16'

        Returns:
         Data representing the currently displayed waveform from the
         oscilloscope in the format described above.

        E.g.:
         get_waveform('CH1')

        Notes:
        """
```

```
        csv_data = self.fetch_data(channel).strip().split("\n")

        header_str = csv_data[:15]
        data_str = csv_data[15:]

        header = {}
        for s in header_str:
            try:
                [key, val] = s.split(",")
                header[key.strip()] = val.strip()
            except ValueError:
                pass

        data = []
        time = []
        for s in data_str:
            [t, x] = s.split(",")
            data.append(float(x))
            time.append(float(t))

        return header, [time, data]


    def fetch_data(self, channel):
        """
        Communicate with the scope and request a readout of the current
        data.
        """
        data = urllib.urlencode([('WFMFILENAME', channel),
                                 ('WFMFILEEXT', 'csv'),
                                 ('command', 'select:control %s' % (channel)),
                                 ('command1', 'save:waveform:fileformat spreadsheet'),
                                 ('wfmsend', 'Get')])
        url = '/'.join([self.base_url, "getwfm.isf"])
        response, content = self.h.request(url, "POST", data)
        content = content[:-3] # remove '\n\r\n' from content
        return content
```

Quelltext D.4: mso3034.py – Kommunikation mit einem MSO3034 Oszilloskop von Tektronix.

E Schaltpläne

E.1 Schaltplan der analogen Signalverarbeitung

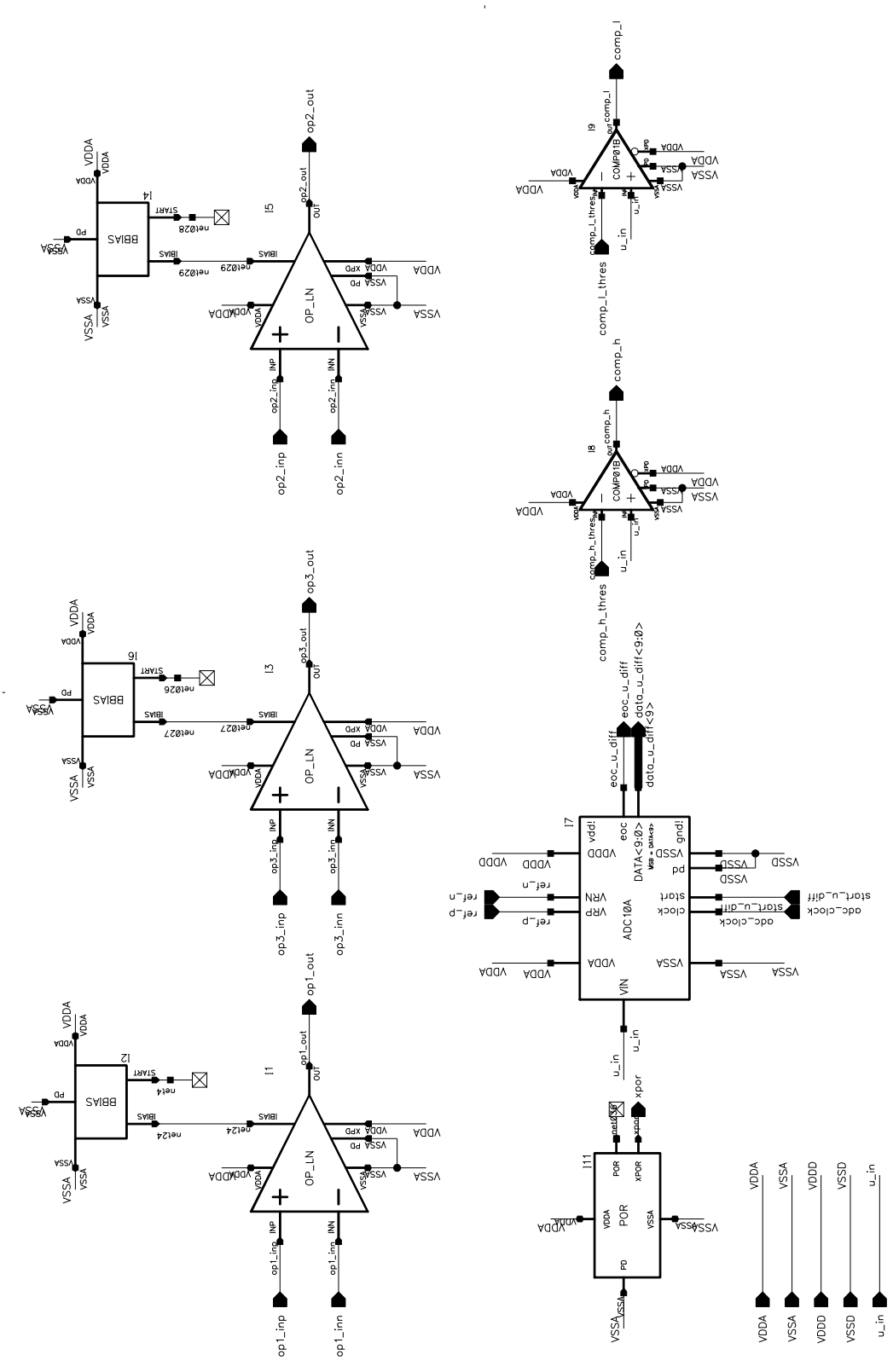


Abbildung E.1: Schaltplan der analogen Signalverarbeitung.

E.2 Schaltplan der Testschaltung für die Simulation der analogen Signalverarbeitung

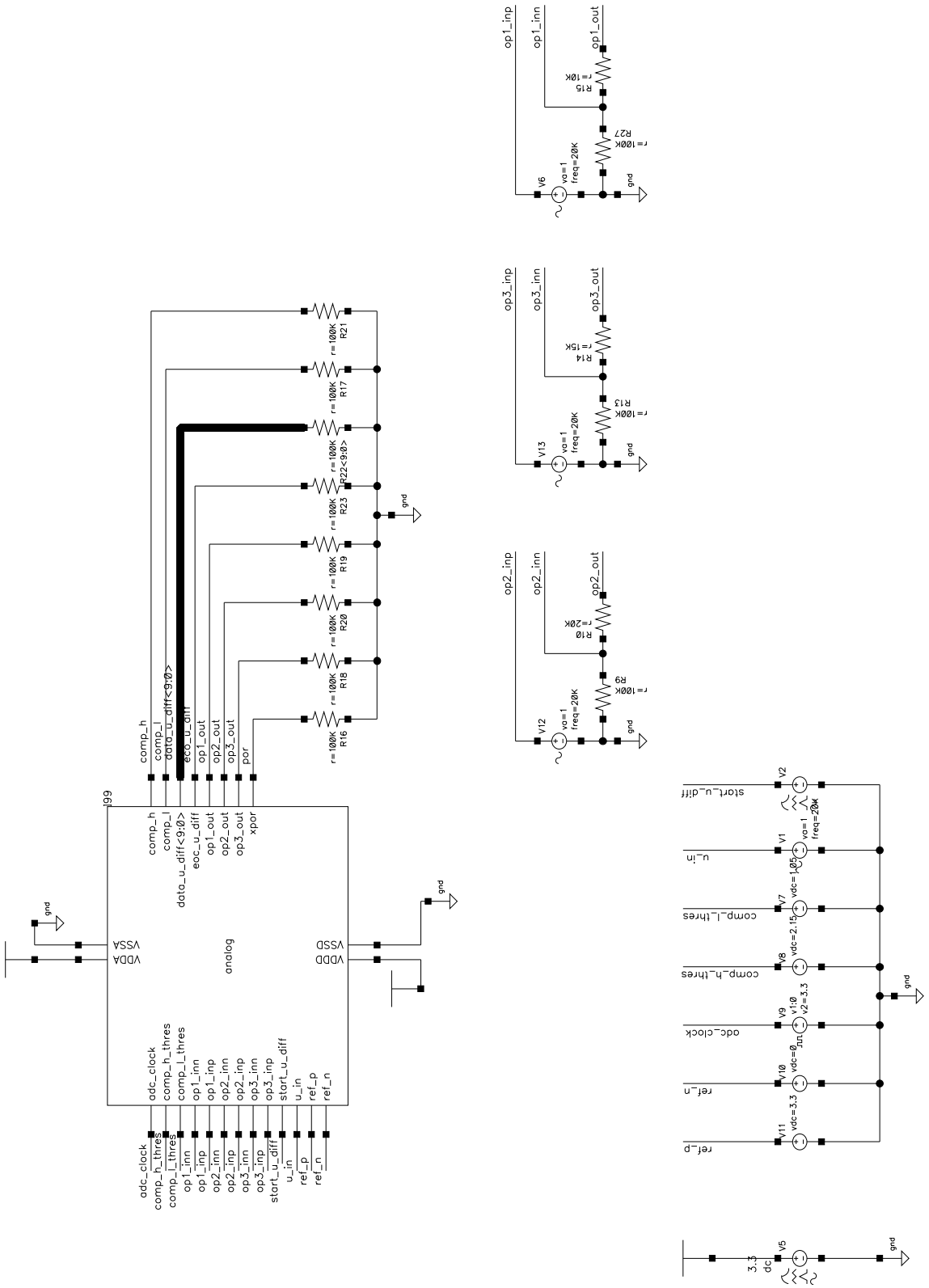


Abbildung E.2: Schaltplan der Testschaltung für die Simulation der analogen Signalverarbeitung. – Der hier eingebundene „analog“-Block ist in Abbildung E.1 zu sehen.

F Testprotokoll für den CMOS-Testchip (Erstinbetriebnahme)

1. Spannung (3.3 V) anlegen und die Stromaufnahme überwachen. Dabei die Strombegrenzung des Netzteils aktivieren.
   a) Digitalteil:
      - VDD (3.3 V) an Pins 15, 43, 78 und
      - GND (0 V) an Pins 16, 42, 79 anlegen.
      - Die Stromaufnahme muss nach einem kurzen Impuls beim Einschalten 0 mA betragen.
   b) Analogteil:
      - VDDA (3.3 V) an Pin 31 und
      - VSSA (0 V) an Pins 32 anlegen.
      - Die Stromaufnahme sollte bei ungefähr 1.0462 mA (maximal 2.02 mA) liegen.
2. Überprüfen des `abs_manchester_encoder`:
   a) Interner Taktteiler:
      - Pin 20 (`i_test_nInternalResetSysClk`) auf VDD legen.
      - Einen Takt von 16 MHz (3.3 V) an Pin 19 (`i_sysClk`) anlegen und
      - internen Takt messen (Pin 18, `o_test_internalClk`).
      - Der Systemtakt sollte um 400 heruntergeteilt sein. Ist bei 16 MHz Systemtakt also ein Rechtecksignal mit einer Frequenz von $f = 40\,\text{kHz}$ und einem Tastgrad von $D = 50\,\%$) zu messen, kann Pin 18 mit Pin 17 (`i_test_internalClk`) verbunden werden.
      - Falls das Rechtecksignal nicht zu messen ist, muss an Pin 17 ein Rechtecksignal (Frequenz $f = 40\,\text{kHz}$ und Tastgrad $D = 50\,\%$) eingespeist werden.
   b) Intern synchronisierte Resets:
      i. Reset auf Systemtakt:

- Bei angelegtem Takt ein Reset an Pin 24 (`i_nReset`; low-aktiv) auslösen (High-Low-Wechsel).
- An Pin 21 (`o_test_nInternalResetSysClk`) muss der Pegel sofort von Low- auf High-Pegel wechseln.
- Beim Loslassen des Resets an Pin 24 (wieder High-Pegel anlegen) muss der Pegel an Pin 21 nach zwei Perioden des Systemtakts taksynchron High werden.
- Funktioniert der interne Reset wie erwartet (siehe Abbildung F.1), so können die Pin 21 und 20 (`i_test_nInternalResetSysClk`) miteinander verbunden werden.
- Falls der interne Reset nicht fehlerfrei funktionieren sollte, muss das Reset-Signal direkt über die Pin 20 eingespeist werden.

ii. Reset auf internem Takt:

- Über Pin 20 den internen Reset (auf Systemtaktebene) einspeisen.
- Bei angelegtem Takt ein Reset an Pin 24 (`i_nReset`; low-aktiv) auslösen (High-Low-Wechsel).
- An Pin 23 (`o_test_nInternalReset`) muss der Pegel sofort von Low auf High-Pegel wechseln.
- Beim Loslassen des Resets an Pin 24 (wieder High-Pegel anlegen) muss der Pegel an Pin 23, nach zwei Perioden des internen Takts taktsynchron High werden.
- Funktioniert der interne Reset wie erwartet (siehe Abbildung F.1), so können die Pins 23 und 22 (`i_test_nInternalReset`) miteinander verbunden werden.
- Falls die interne Reset nicht fehlerfrei funktioniert, muss das Reset-Signal direkt über Pin 23 eingespeist werden.

c) Protokoll:

- Reset und Taktsignale wie oben beschrieben anlegen.
- Zu sendene Daten anlegen (siehe hierzu Tabelle F.1)
- Pin 10 (`o_test_manchesterRegOut`) mit Pin 9 (`i_test_manchesterRegOut`) verbinden.
- Pin 7 (`i_test_speedpulseRegOut`) mit Pin 8 (`o_test_speedpulseRegOut`) verbinden.

- An Pin 6 (`i_speedpulse`) für einen Systemtakt ($f = 16\,\text{MHz}$; $T = 62.5\,\mu\text{s}$) High-Pegel anlegen (damit wird das Senden gestartet).
- An Pin 4 (`o_speedpulse_28mA`) und Pin 5 (`o_datapulse_14mA`) die ausgehenden Signalpegel messen. Sie müssen mit den Pegeln aus Abbildung F.2 übereinstimmen.

3. Power-on Reset:
   - Analoge Ausgangsspannung an Pin 25 (`xpor`) messen.
   - Versorgungsspannung an den Analogteil anlegen (Pin 31 (VDDA) und 32 (VSSA))
   - Pin 25 sollte der Betriebsspannung ab 2.26 V (Min:1.24 V; Typ:2.26 V Max:2.82 V; siehe [5]) folgen. Siehe Abbildung F.3
   - Beim Abschalten der Versorgungsspannung sollte Pin 25 der Versorgungsspannung bis 2.12 V (Min:1.24 V; Typ:2.26 V Max:2.82 V; siehe [5]) folgen.
4. Operationsverstärker:
   - Operationsverstärker als nichtinvierende Verstärker aufbauen (siehe Abbildung F.4).
   - Signale mit Abbildung F.5 vergleichen.
5. Analog-Digital-Umsetzer
   - Pin 59 (`i_mux_mode<1>`) muss auf High- und Pin 60 (`i_mux_mode<0>`) auf Low-Pegel gelegt werden, damit der ADC extern angesteuert werden kann.
   - Als Referenzspannung für den ADC sollten 3.3 V zwischen die Pins 27 (`ref_n`) und 26 (`ref_p`) gelegt werden.
   - An Pin 57 (`adc_clock`) kann somit der Takt für den ADC angelegt werden, da dieser nun als Eingang dient. Der Takt muss ein Rechtecksignal mit einer Spitze-Spitze-Spannung von 3.3 V, einem Offset von 1.65 V und einer Frequenz von 1 MHz sein.
   - An den Eingang des ADCs, Pin 28 (`u_in`), wird eine Gleichspannung von 2.30 V angelegt.
   - Jezt wird Pin 58 (`start_u_diff`) auf High-Pegel gesetzt, um die Konvertierung zu starten.

- Das Signal `eoc_u_diff` (Pin 54) zeigt, wann die Konvertierung beendet. An den Pins 44 bis 53 (`data_u_diff`) sollte dann ein Wert von 0x2C9 abgelesen werden können (siehe Tabelle F.2 und Abbildung F.7). Bei 10 Bit (Maximalwert: 0x3FF) ergibt das die Eingangsspannung 2.30 V: $\frac{0x2C9}{0x3FF} \cdot V_{ref} = 0.969697 \cdot 3.3\,V = 2.3\,V$

6. Komparatoren
   - Pin 59 (`i_mux_mode<1>`) muss auf High- und Pin 60 (`i_mux_mode<0>`) auf Low-Pegel gelegt werden, damit die Ausgangswerte der Komparatoren ausgegeben werden.
   - Die Schaltschwellen der Komparatoren werden über die Pins 29 (`comp_h_thres`) und 30 (`comp_l_thres`) mit einer Gleichspannungsquelle auf $V_{\texttt{comp\_h\_thres}} = 2.15\,V$ und $V_{\texttt{comp\_l\_thres}} = 1.05\,V$ eingestellt.
   - An Pin 28 (`u_in`) ist eine sinusförmige Spannung mit 1 V Amplitude und einem Offset von 1.65 V anzulegen.
   - Die Ausgangssignale an Pin 56 (`comp_h`) und 55 (`comp_l`) sollten dann wie in Abbildung F.6 gezeigt aussehen.
7. digitale Signalverarbeitung
   a) **Voraussetzung für diesen Test ist, dass der Analogteil funktionsfähig ist.**
   b) Pin 59 (`i_mux_mode<1>`) und Pin 60 (`i_mux_mode<0>`) müssen auf Low-Pegel gelegt werden, damit das digitale mit dem analogen Modul kommunizieren kann.
   c) Die Schwellenwerte der Komparatoren wie oben beschrieben einstellen.
   d) Über Pin 28 (`u_in`) nacheinander die Signale aus Tabelle F.3 einspeisen.
   e) An Pin 70 (`o_hd_ser`) könnnen die berechneten HDI- und HD5-Werte ausgelesen werden (mithilfe des Python-Skripts aus D).
   f) Die HDI- und HD5-Werte sollten mit den theoretischen aus Tabelle F.3 übereinstimmen.

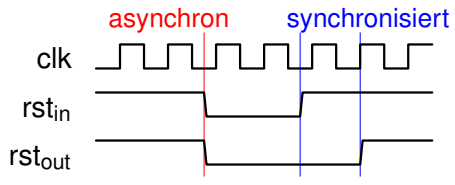


Abbildung F.1: Intern synchronisierte Resets.

Tabelle F.1: Testdaten für Erstinbetriebnahme des `abs_manchester_encoder` Moduls. – High-Pegel entspricht 3.3 V. und Low-Pegel 0 V

Pinnummer	Pegel	Pinbezeichnung
80	High	`i_dataBits <7>`
81	High	`i_dataBits <6>`
82	Low	`i_dataBits <5>`
83	High	`i_dataBits <4>`
84	High	`i_dataBits <3>`
1	Low	`i_dataBits <2>`
2	High	`i_dataBits <1>`
3	Low	`i_dataBits <0>`

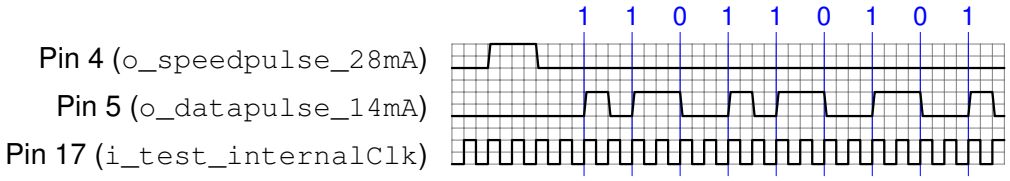


Abbildung F.2: Ausgangssignale für Erstinbetriebnahme des `abs_manchester_encoder` Moduls. – High-Pegel entspricht 3.3 V. und low-Pegel 0 V

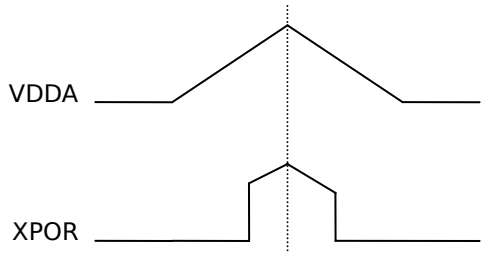


Abbildung F.3: Verhalten des Power on Resets laut Datenblattt. – Der xpor-Ausgang folgt ab einer Spannung der Versorgungsspannung (aus [5])

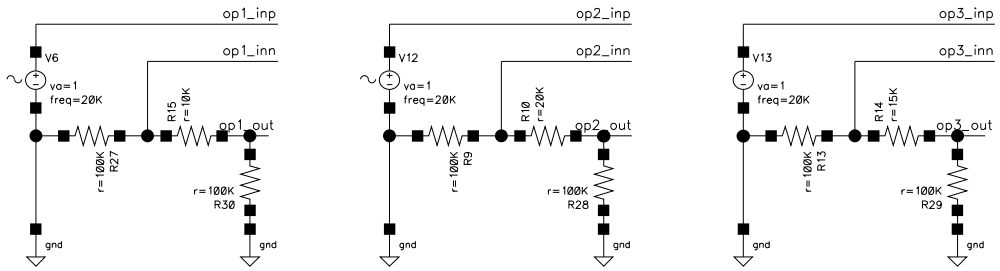


Abbildung F.4: Schaltplan der Testschaltung für die Operationsverstärker.

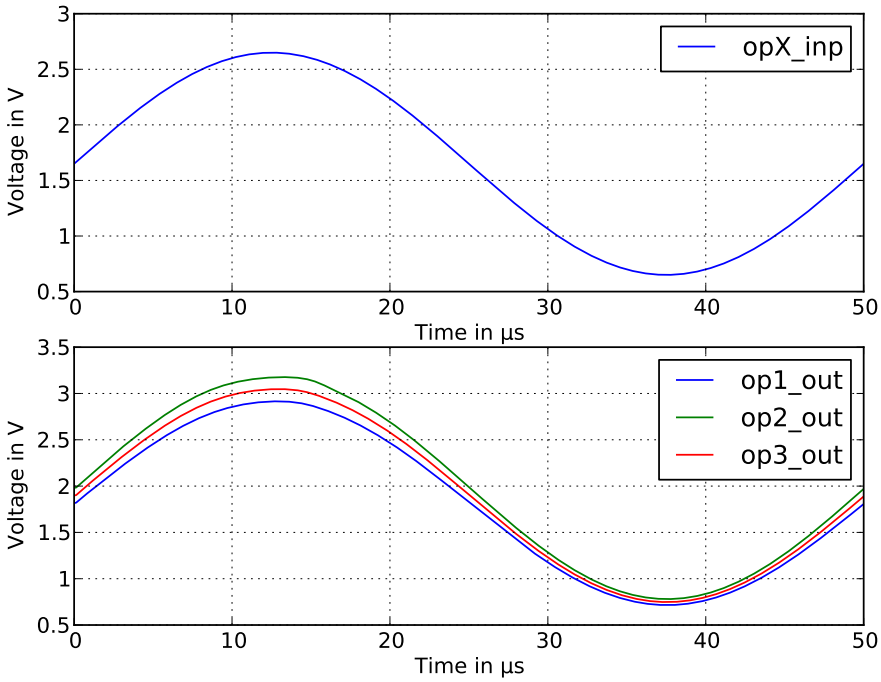


Abbildung F.5: Simulationsergebnis der Operationsverstärker. – OP als nichtinvertierender Verstärker beschaltet (Schaltplan siehe Abbildung F.4); OP1 mit Verstärkungsvaktor $v = 1.10$, OP2 mit Verstärkungsvaktor $v = 1.20$ und OP3 mit Verstärkungsvaktor $v = 1.15$.

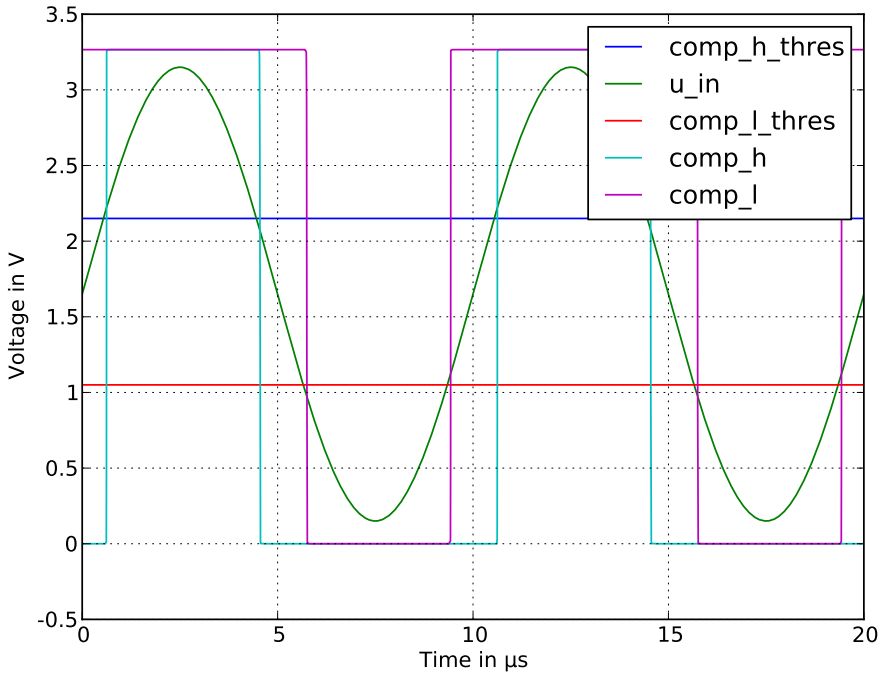


Abbildung F.6: Simulationsergebnis der Komparatoren. – Die Schaltschwellen wurden für diese Simulation auf $V_{\texttt{comp\_h\_thres}} = 2.15\,\text{V}$ und $V_{\texttt{comp\_l\_thres}} = 1.05\,\text{V}$ gelegt. Die Eingangsspannug `u_in` ist ein Sinus mit 1 V Amplitude und einem Offset von 1.65 V. Liegt das Eingangssignal (hier in Grün dargestellt) über der Vergleichsspannung (`comp_l_thres` bzw. `comp_h_thres`) der Komparatoren, so liefern diese einen High-Pegel (`comp_l` bzw. `comp_h`).

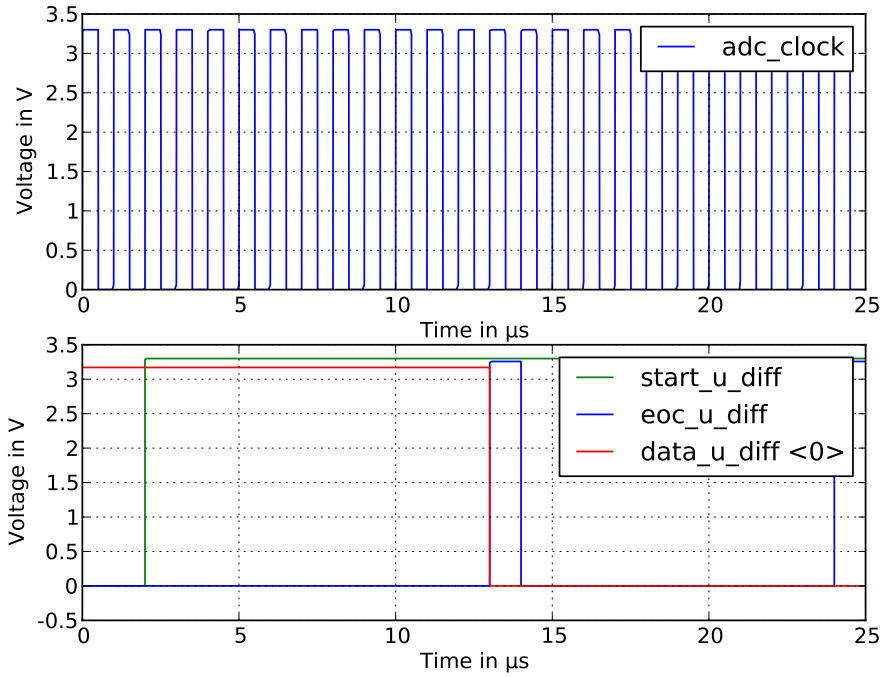


Abbildung F.7: Simulationsergebnis des ADC. – Elf Takte (`adc_clock`) nachdem die Umsetzung mit `start_u_diff` gestartet wurde, erscheint an den Signalen `data_u_diff` (hier exemplarisch `data_u_diff <0>`) der Ausgangswert und das Ende der Umsetzung wird mit `eoc_u_diff` signalisiert. Die Pegel für `start_u_diff` sind in Tabelle F.2 zu finden.

Tabelle F.2: Ausgangssignale des ADCs bei 2.3 V Eingangsspannung. – Diese Signal-Pegel liegen, nachdem das Signal `eoc_u_diff` von Low- auf High-Pegel gewechselt ist, an. Siehe Abbildung F.7. Dabei ist `data_u_diff <0>` das LSB.

Signalname	Pin	Pegel
`data_u_diff <0>`	53	High
`data_u_diff <1>`	52	Low
`data_u_diff <2>`	51	Low
`data_u_diff <3>`	50	High
`data_u_diff <4>`	49	Low
`data_u_diff <5>`	48	Low
`data_u_diff <6>`	47	High
`data_u_diff <7>`	46	High
`data_u_diff <8>`	45	Low
`data_u_diff <9>`	44	High

Tabelle F.3: Einzuspeisende Signale und deren theoretischer HD5. – Alle Signalformen benötigen einen Offset von 1.65 V.

Eingespeistes Signal			Theoretischer HD5
Signalform	Frequenz	Spitzen-spannung	
Sinus	37 Hz	0.90 V	0.0 %
Dreieck	37 Hz	0.90 V	13.1 %

G Cadence Einführung

Cadence Einführung

Digitales Chipdesign mit
Cadence/Austriamicrosystems

HAW Hamburg

Daniel Sabotta

16. Januar 2013

Inhaltsverzeichnis

1 Einleitung

Dieses Dokument soll als Leitfaden zum Erstellen eines ASIC-Layouts mit Standard-Zellen dienen. Die umzusetzende Hardware muss in Form eines VHDL- oder Verilog-Quelltextes vorliegen (synthesefähiger RTL-Code). Hier wird als Beispiel ein 4-Bit-Zähler verwendet (siehe Quelltext 2.1 in Kapitel 3).

Die Dokumente [9] und [10] geben eine kurze Einführung in die Bedienung der Cadence-Toolchain und sollten vorher durchgearbeitet werden.

1.1 Software

Diese Einführung wurde mit *Cadence Custom IC Design Tools IC6.1.4.500.6* mit dem *HIT-Kit v4.00* von Austriamicrosystems (AMS) erstellt.

1.2 Designprozess

1. Simulation des VHDL-RTL-Quelltextes
2. VHDL-RTL-Quelltext in Verilog-Code mit Standard-Zellen von AMS übersetzen (Cadence Encounter RTL Compiler)
3. Simulieren des Verilog-Quelltextes
4. Erstellen des Layouts der digitalen Standard-Zellen (Encounter Digital Implementation System)
   a) Floorplan
   b) ENDCAPs platzieren
   c) Powerplaning
      - Power-Ring
      - Power-Stripe
      - Specialroute (vdd! und gnd!)
   d) Standardzellen platzieren

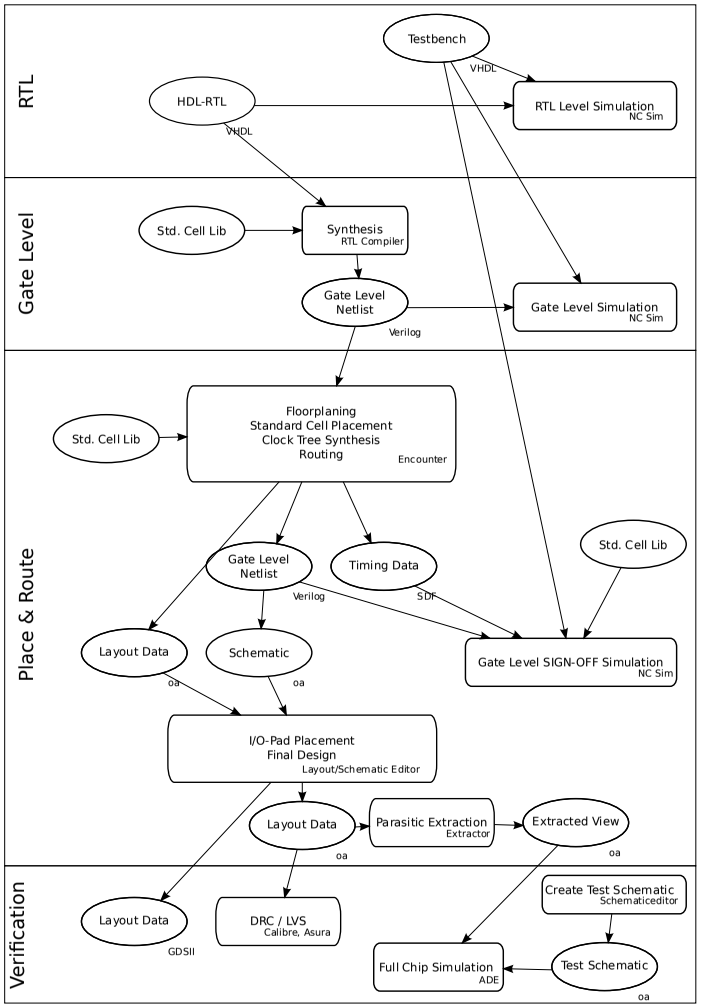


Abbildung 1.1: Workflow des Designprozesses.

e) Thie 1/0 Cells platzieren

f) Clock-Tree Synthese

g) Routing

h) Filler Cells platzieren

5. Simulation der Neuen Netzliste mit Timing Daten
6. Layout nach Virtuoso exportieren
7. Timing Daten und neue Netzliste exportieren
8. Neuen Schaltplan mit I/O-Elementen erstellen (Virtuoso Schematic Editor)
9. Layout erstellen (Virtuoso Layout Suite)
10. Simulation des gesamten Layouts (Virtuoso Analog Design Environment)
11. Design Rule Checks und Vergleich zwischen Schaltplan und Layout (LVS)
12. GDSII erzeugen

1.3 Starten von Cadence

Zunächst sollte man sich ein Projektverzeichnis anlegen und in dieses wechseln (dies ist notwendig, da Cadence beim Start einige Dateien im aktuellen Verzeichnis anlegt).

```
mkdir Projektverzeichnis && cd Projektverzeichnis.
```

Bevor Cadence gestartet werden kann, muss noch der Befehl

```
source /home/cdsmgr/cadence_env.sh
```

ausgeführt werden, der alle für Cadence benötigten Umgebungsvariablen setzt. Nun kann Cadence aus dem so eben erstellten Verzeichnis mit

```
ams_cds -tech <TECH_LIB> &
```

gestartet werden. Wobei mit *<TECH_LIB>* die Technologie angegeben wird (hier c35b3, also ein $0.35\,\mu$m-Prozess mit drei Metalllagen). Das Und-Zeichen sorgt dafür, dass Cadence im Hintergrund läuft.

Bei dem ersten Start aus einem Verzeichnis muss der verwendete Prozesss angegeben werden (hier C35B3C3 [7] siehe Abbildung 1.2)

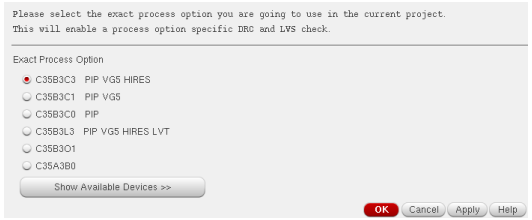


Abbildung 1.2: Select Process Options Fenster. – wird beim ersten Start von Cadence angezeigt. Es muss hier der C35B3C-Prozess gewählt werden.

1.4 Library Manager

Mit Virtuoso wird immer der Library Manager gestartet. Hieraus werden die Schaltungen und Layouts zum Lesen und Editieren geöffnet. Hinter den einzelnen Librarys stecken Verzeichnisse, die wiederum Verzeichnisse der Cells enthalten (worin die einzelnen views enthalten sind). Diese Verzeichnisse sollten auf keinen Fall von Hand verändert werden, sondern nur über den Library Manager.

Die zu ladenden Librarys werden in der Datei `cds.lib` (wird im Startverzeichnis beim ersten Start automatisch erzeugt) festgelegt. Um eine bereits vorhandene Library in den Library Manager einzubinden, muss lediglich die `cds.lib` um eine Zeile ergänzt werden, die den Namen und den Pfad zu der Library enthält:

```
DEFINE eingebundene_lib /pfad/zur/lib
```

2 HDL-Simulation

Für die HDL-Simulation werden eine Reihe von Programmen benötigt, die über die graphische Oberfläche *NCLaunch*, über die Konsole mittels *irun* oder per Hand ausgeführt werden können.

Vor dem Ausführen der Simulation müssen zwei Schritte durchgeführt werden:

1. Die benötigten HDL-Dateien (in der richtigen Reihenfolge) **kompilieren** (`ncvlog / ncvhdl`) und
2. die Testbench **elaborieren** (`ncelab`).

Um eine genauere Beschreibung der Fehler und Warnungen zu bekommen dient der Befehl

```
nchelp <tool> <error>.
```

Wobei `<tool>` das Tool angibt, in dem die Fehlermeldung auftritt und `<error>` der Fehler, zu dem man mehr Informationen bekommen möchte. Z.B.:

```
nchelp ncvhdl_p ASILDR
```

In dieser Einleitung wird der Quelltext aus 2.1 (counter_4bit.vhd) und 2.2 (counter_4bit_TB.vhd) verwendet.

```
1  ----------------------------------------------------
   -- Title      : counter_4bit
   ----------------------------------------------------
   -- File       : counter_4bit.vhd
   -- Author     : Daniel Sabotta
6  -- Company    : HAW
   -- Created    : 2012-03-07
   -- Last update: 2012-04-27
   ----------------------------------------------------
   -- Description: Counts up to 15
11 ----------------------------------------------------
   library ieee;
   use ieee.std_logic_1164.all;
   use ieee.std_logic_arith.all;
   use ieee.std_logic_unsigned.all;
16
```

```
entity counter_4bit is
  port(
    clk: in std_logic;
    nRst: in std_logic;
    Cout: out std_logic_vector(3 downto 0)
  );
end counter_4bit;

architecture arc of counter_4bit is
  signal cInt: std_logic_vector(3 downto 0);
begin
  process (clk)
  begin
    if (clk'event and clk = '1') then
      if (nRst = '0') then
        cInt <= "0000";
      else
        cInt <= cInt + "0001";
      end if;
    end if;
  end process;

  cOut <= cInt;
end arc;
```

Quelltext 2.1: counter_4bit.vhd – 4-Bit-Zähler in VHDL.

```
-------------------------------------------------------
-- Title      : counter_4bit Testbench
-------------------------------------------------------
-- File       : counter_4bit_TB.vhd
-- Author     : Daniel Sabotta
-- Company    : HAW
-- Created    : 2012-05-10
-- Last update: 2012-05-11
-------------------------------------------------------
-- Description: Testbench for counter_4bit
-------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;


entity counter_4bit_TB is
```



```
end entity counter_4bit_TB;

architecture bhv of counter_4bit_TB is

  -- Moduldeklaration
  component counter_4bit is
    port (
      clk:  in  std_logic;
      nRst: in  std_logic;
      Cout: out std_logic_vector(3 downto 0)
    );
  end component;

  -- input
  signal clk  : std_logic := '0';
  signal nRst : std_logic;

  -- output
  signal Cout : std_logic_vector(3 downto 0);

begin
-- Modulinstatziierung
  dut : counter_4bit
    port map (
      clk   => clk,
      nRst  => nRst,

      Cout  => Cout
  );


  clk   <= not clk  after 20 ns;  -- 25 MHz Taktfrequenz
  nRst <= '0', '1' after 100 ns; -- erzeugt Resetsignal: __--


end bhv;
```

Quelltext 2.2: counter_4bit_TB.vhd – Testbench für den 4-Bit-Zähler in VHDL.

2.1 NCLaunch

Das Dokument [11] gibt eine Einführung in die HDL Simulation mit *NCLaunch*.

NCLaunch wird mit

```
nclaunch &
```

gestartet.

Als Erstes muss mit

```
File → Set Design Directory...
```

ein neuer Workspace namens *work* angelegt werden.

Falls in dem Startverzeichnis die Datei *cds.lib* nicht vorhanden ist, muss diese aus diesem Dialog heraus erstellt werden. Ist sie allerdings vorhanden, dann muss sie um die Zeile

```
include $CDS_INST_DIR/tools/inca/files/cds.lib
```

ergänzt werden, damit die Standardbibliotheken gefunden werden.

Das NCLaunch Fenster ist in drei Teile aufgeteilt. Das linke Teilfenster ist ein Dateibrowser (der sich nicht besonders gut bedienen lässt), im rechten Teilfenster sieht man die bereits kompilierten entyties in den jeweiligen Bibliotheken und im unterem Fenster ist die Ausgabe der Programme zu sehen (Fehlermeldungen, Warnungen etc.).

Um den HDL-Code zu kompilieren müssen die Dateien im linken Teilfenster markiert werden und mit

```
Tools → VHDL bzw. Verilog Compiler...
```

der Compiler gestartet werden (bei VHDL muss hier *Enable VHDL 93 features* ausgewählt sein). Anschließend ist die entity im rechten Fenster wiederzufinden (in dem entsprechendem Workspace).

Jetzt muss die Testbench noch elaboriert werden. Dazu muss diese (im rechten Fenster) markiert werden und mit

```
Tools → Elaborator...
```

der Elaborator gestartet werden. Dieser ist mit den Standardeinstellungen auszuführen (es sei denn, es soll eine Simulation mit Timing-Files durchgeführt werden, siehe weiter unten).

Der Simulator wird mit


```
Tools → Simulator...
```

gestartet. Hier muss die Testbench als *Snapshot* ausgewählt und mit *OK* bestätigt werden.

Es öffnet sich daraufhin der *Design Browser* aus dem mit einem Klick auf den *Waveform*-Knopf das eigentliche Programm zum Anzeigen der Waveform gestartet werden kann (*SimVision*). *SimVision* kann auch direkt gestartet werden; dazu muss die Umgebungsvariable *SIMVISIONOPTS=waves"* gesetzt werden
(z. B. in der *.bashrc*: `export SIMVISIONOPTS="-waves"` eintragen).

Wurde etwas an den Quelldateien verändert, so können mit einem Klick auf

```
Simulation → Reinvoke Simulator...
```

die Quelldateien neu kompiliert und simuliert werden, ohne den Simulator von Hand neu starten zu müssen. Hierfür kann auch eine Tastenkombination definiert werden: Im Menü

```
Edit → Preferences...
```

unter

```
General Options → Keyboard Shortcuts →
    Simulation → ReinvokeSimulator.
```

Man kann wie folgt die Einstellungen (Fensterpositionen, welche Signale angezeigt werden ...) von SimVision abspeichern:

```
File → Save Command Script...
```

2.2 Das Komandozeilentool *irun*

Mithilfe von *irun* kann der Simulator mit vorherigem Kompilieren und Elaborieren (falls erforderlich) per Konsole gestartet werden. Dazu muss der Befehl

```
irun <HDL-Dateien> -v93 -gui -access +r
 -top <worklib>.<TB>:<TB_architecture>
```

aufgerufen werden. Um das zu vereinfachen kann auch das Skript aus Quelltext 2.3 benutzt werden. Hier werden auch vorher gespeicherte Einstellungen von SimVision eingelesen.

```
#!/bin/bash

# HDL-Files for simulation (*.vhd *.v)
SOURCES="HDL/counter_4bit.vhd HDL/counter_4bit_TB.vhd"

# Working Library (lib)
WORKLIB="worklib"

# Testbench (cell)
TESTBENCH="counter_4bit_TB"

# TB-Arschitecture (view)
TBARC="bhv"

# TCL-File with Sim Vision Settings
SVSETTINGS="restore.tcl"

irun $SOURCES -v93 -gui -access +r -top $WORKLIB.$TESTBENCH:$TBARC
    -input $SVSETTINGS
```

Quelltext 2.3: irun_skript.sh – Einfaches kompilieren und Starten von SimVision.

2.3 Simulation mit Standard Zellen von AMS

In Kapitel 3) wird beschrieben, wie der VHDL-Quelltext aus Standardzellen synthetisiert wird. Damit die Standardzellen der Simulation zu Verfügung stehen, müssen diese vorher kompiliert werden. Dazu legt man ein neuen Workspace mit dem Namen

`c35_CORELIB`

an und setzt ihn als aktuelles Workspace. Dann navigiert man zu dem Ordner

`/home/cdsmgr/ams/verilog/c35b3.`

Aus diesem Ordner müssen die Dateien

- `35_CORELIB.v`,
- `c35_IOLIB_3M.v`,
- `FILLANT.v` und

- `c35_UDP.v` kompiliert werden.

Des Weiteren muss beim Elaborieren unter `General` die Option `Default timescale` aktiviert und Zeiten in der Form von `10ps/1ps` eingeben (dabei bedeutet die erste Zahl die Schrittweite der Simulation und die Zweite Zahl die Auflösung) werden.

Jetzt können die synthetisierten Verilog Dateien kompiliert und anschließend simuliert werden.

2.4 Simulation mit SDF-Timing-File

Die aus den Tools extrahierten Timings werden in SDF-Dateien (Standard Delay Format) gespeichert. Diese können in die Simulation eingebunden werden, um die Schaltzeiten der Standardzellen und die parasitären Effekte (nach Layout) zu berücksichtigen.

Um eine SDF-Datei einzubinden muss diese zuerst kompiliert werden. Dazu muss die SDF-Datei im linken Fenster ausgewählt werden (Hierzu ggf. die Filtereinstellung des Fensters, unter dem Fenster, um **.sdf* erweitern) und der *SDF Compiler* unter

`Tools → SDF Compiler...`

ausgeführt werden. Jetzt kann die *SDF*-Datei mit den Standardeinstellungen kompiliert werden. Der Compiler erzeugt eine *.sdf.X*-Datei, die in die Simulation eingebunden werden kann.

Dazu müssen im `Elaborator` zusätzlich folgende Einstellungen gemacht werden: Unter `Advanced Options...`:
Bei `Annotation` die Auswahl `Use SDF command file` aktivieren und auf `Edit...` klicken. Hier gewünschte `.sdf.X`-Datei mit `Add...` auswählen. Und unter `SDF Command File Name` eine vorher erstellte, **leere** `.cmd` Datei auswählen. Unter `Scope:` muss der Name der Instanz der, zu testenden Logik, mit führendem Doppelpunkt (`:`) angegeben werden (hier `:dut`). Man kann hier noch ein Log-File angeben. Mit `OK` bestätigen (siehe Abbildung 2.1). Eine beispielhafte SDF-Command-Datei ist in Quelltext 2.2 zu sehen. Wie auch bei der Simulation mit Standard Zellen von AMS aus Kapitel 2.3, muss hier die `Default timescale` gesetzt werden.

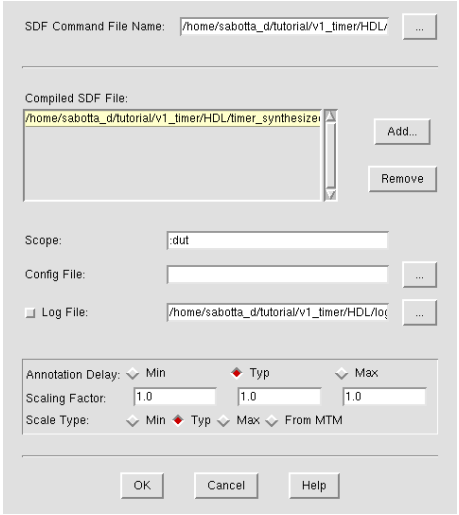


Abbildung 2.1: Elaborate Einstellungen – SDF Command File.


```
1  // SDF command file /home/sabotta_d/tutorial/v1_timer/HDL/
       timer_synthesized_from_encounter_SDF/SDFCommandFile.cmd
2
3  COMPILED_SDF_FILE = "/home/sabotta_d/tutorial/v1_timer/HDL/
       timer_synthesized_from_encounter_SDF/none_all.sdf.X",
4  SCOPE = :dut,
5  LOG_FILE = "/home/sabotta_d/tutorial/v1_timer/HDL/logFile.sdf.log"
       ,
6  MTM_CONTROL = "TYPICAL",
7  SCALE_FACTORS = "1.0:1.0:1.0",
8  SCALE_TYPE = "FROM_TYPICAL";
9
10
11 // END OF FILE: /home/sabotta_d/tutorial/v1_timer/HDL/
       timer_synthesized_from_encounter_SDF/SDFCommandFile.cmd
```

Abbildung 2.2: SDFCommandFile.cmd.

3 Synthetisieren des VHDL/Verilog-Quelltextes

Um aus dem HDL-Quelltext zu synthetisieren, wird der Cadence Encounter RTL Compiler verwendet. Dieser wird mit

```
rc -gui
```

mit grafischer Benutzeroberfläche gestartet. Hier ist es unbedingt notwendig den Prozess im Vordergrund laufen zu lassen (ohne &), da das Terminal für die Eingabe von Befehlen dient. Es empfiehlt sich, den Cadence Encounter RTL Compiler aus einem Unterverzeichnis (z. B. `./rtl_compiler`) zu starten, da dieser viele Dateien anlegt.

Über den Menüpunkt *File* → *Source Script...* kann nun ein Skript (z.B. das Skript aus Quelltext 3.1) eingelesen werden, das die Befehle ausführt, die benötigt werden um die HDL-Datei zu kompilieren. Alternativ können diese Befehle auch in die Konsole eingegeben werden.

```
################################################################
## Title      : Encounter(R) RTL Compiler - Script
## Project    : ESZ-ABS
################################################################
## File       : vhdlRTL.tcl
## Author     : Daniel Sabotta
## Company    : HAW Hamburg
## Created    : 2012-05-23
## Last update: 2012-05-23
################################################################
## Description: Load this file into Encounter(R) RTL Compiler
##                  - Script to synthesize a vhdl file
##
##              Usage: "rc -f vhdlRTL.tcl"
##
##              Original file from Erik Brunvand, 2008
################################################################
## Revisions  :
## Date        Version  Author  Description
################################################################

## Set the search paths to the libraries and HDL
```



```
## Remember that "." yout current directory

## Search Path for VHDL and Verilog files
set_attribute hdl_search_path   {./HDL};
## Search Path for library files
set_attribute lib_search_path   {/home/cdsmgr/ams/liberty/↩
   c35_3.3V};
## Target library
set_attribute library           [list c35_CORELIB.lib];
## See a lot of warnings
set_attribute information_level 6;


set hdlFiles        [list counter_4bit.vhd] ;# All HDL files
set basename        counter_4bit    ;# name of toplevel module
set savePath        {./}    ;# Save created Files here
set runName         syn     ;# name appended to output files

set clkName         clk     ;# clock name
set clkPeriod_ps    62500   ;# clock periode in ps
set clkInDelay_ps   250     ;# delay from clock to input valid
set clkOutDelay_ps  250     ;# delay from clock to output valid



################################################################
##      below here shouldn't need to be changed...              ↩
   ##
################################################################

## Analyze and Elaborate the HDL files
read_hdl    -vhdl   ${hdlFiles}
elaborate   ${basename}

## Apply Constraints and generate clocks
set clock [define_clock -period ${clkPeriod_ps} -name ${clkName}↩
    [clock_ports]]
external_delay -input $clkInDelay_ps -clock ${clkName} [find / ↩
   -port ports_in/*]
external_delay -output $clkOutDelay_ps -clock ${clkName} [find /↩
    -port ports_out/*]

## Sets transition to default values for Synopsys SDC format,
## fall/rise 400ps
```

```
dc::set_clock_transition .4 $clkName

## check that the design is OK so far
check_design -unresolved
report timing -lint

## Synthesize the design to the target library
synthesize -to_mapped

## Write out the reports
report timing > ${savePath}${basename}_${runName}_timing.rep
report gates  > ${savePath}${basename}_${runName}_cell.rep
report power  > ${savePath}${basename}_${runName}_power.rep

## Write out the structural Verilog and sdc files
write_hdl -mapped >  ${savePath}${basename}_${runName}.v
write_sdc >  ${savePath}${basename}_${runName}.sdc

## write sdf-timing file with
##  -prec 3             : Use 3 digits floating point
##                        in delay calculation
##  -edges check_edge   : Check which edge delays are defined in
##                        technology file and calculate only
##                        those edges.
write_sdf -prec 3 -edges check_edge > ${savePath}${basename}_${↩
   runName}.sdf

## log actual time in logfile
date

## exit rc (usefull, for "rc -f <thisFileName>")
#quit
```

Quelltext 3.1: TCL-Skript für die Synthese des Timers.

Als Ergebnis bekommt man ein Verilog-Quelltext (hier timer_syn.v, siehe Quelltext 3.2), indem die Funktion des VHDL-Quelltextes mit Standardzellen von AMS gebildet wird. Diese Standardzellen sind auch in dem Library Manager zu finden.

```
// Generated by Cadence Encounter(R) RTL Compiler RC9.1.203 - ↩
   v09.10-s242_1

module counter_4bit(clk, nRst, Cout);
  input clk, nRst;
```



```
  output [3:0] Cout;
  wire clk, nRst;
  wire [3:0] Cout;
  wire n_0, n_1, n_2, n_3, n_4, n_5, n_6, n_7;
  wire n_8, n_9, n_10, n_11;
  DF3 \cInt_reg[3] (.C (clk), .D (n_11), .Q (Cout[3]), .QN ());
  NOR21 g64(.A (n_8), .B (n_10), .Q (n_11));
  DF3 \cInt_reg[2] (.C (clk), .D (n_9), .Q (Cout[2]), .QN ());
  XNR21 g66(.A (Cout[3]), .B (n_5), .Q (n_10));
  NOR21 g67(.A (n_8), .B (n_7), .Q (n_9));
  INV3 g68(.A (n_6), .Q (n_7));
  DF3 \cInt_reg[1] (.C (clk), .D (n_4), .Q (Cout[1]), .QN ());
  ADD22 g69(.A (n_1), .B (Cout[2]), .CO (n_5), .S (n_6));
  NOR21 g71(.A (n_8), .B (n_3), .Q (n_4));
  DF3 \cInt_reg[0] (.C (clk), .D (n_0), .Q (Cout[0]), .QN ());
  INV3 g72(.A (n_2), .Q (n_3));
  ADD22 g73(.A (Cout[0]), .B (Cout[1]), .CO (n_1), .S (n_2));
  NOR21 g75(.A (n_8), .B (Cout[0]), .Q (n_0));
  CLKIN3 g76(.A (nRst), .Q (n_8));
endmodule
```

Quelltext 3.2: timer_syn.v – 4-Bit-Zähler in Verilog mit AMS-Standardzellen.

4 Erstellen des Layouts mit Standardzellen

Mithilfe des Programms `Encounter Digital Implementation System` kann, aus dem in Kapitel 3 erzeugten Verilog-Quelltext, ein Layout erzeugt werden. Hier werden die Standardzellen von AMS platziert, mit der Betriebsspannung verbunden und die Verbindungsleitungen zwischen den einzelnen Zellen gelegt.

Die Dokumente [1] und [3] geben eine Einführung in die Bedienung des Encounters.

Bevor der encounter gestartet werden kann, ist es notwendig den Befehl

```
ams_encounter -tech c35b3
```

auszuführen. Dadurch werden von dem encounter benötigte Dateien angelegt. Das Encounter Digital Implementation System wird mit dem Befehl

```
encounter
```

aus der Konsole heraus gestartet. Auch hier ist es wichtig, dass der Prozess im Vordergrund läuft (also ohne & gestartet wird).

Nach dem Start des encounters muss in der Konsole (jetzt steht dort die Zeichenkette `encounter>`) der Befehl

```
source amsSetup.tcl
```

eingegeben werden, um spezielle Befehle von Austriamicrosystems benutzen zu können, siehe [3].

Es empfiehlt sich den Entwurf zwischendurch als Encounter-Datei (`.enc`) zu speichern:

```
File → Save Design...
```

Über den Menüpunkt

```
File → Restore Design...
```


können diese Dateien wieder eingelesen werden.

Der encounter erstellt bei jedem Start zwei Logdateien (steigend nummeriert). In der `encounter.log` wird die Konsolenausgabe des encounter gespeichert. Jeden eingegebenen Befehl schreibt der encounter in die `encounter.cmd` (auch die über die GUI eingegebenen). Ist der Designprozess einmal durchlaufen, so kann dieser jederzeit mit der Ausführung der `encounter.cmd` wiederholt werden (`source encounter.cmd`). Und somit können auch kleine Änderungen einfach durchgeführt werden.

4.1 Design einlesen

Der Verilog-Quelltext wird mithilfe der Eingabemaske `Design Import` eingelesen; diese wird aus dem Menü

```
File → Import Design...
```

geöffnet. Über den Button `Load...` können Voreinstellungen von Austriamicrosystems geladen werden (z. B. `c35b3_std.conf` in dem Ausführungsordner).

In dem `Basic`-Tab des `Design Import`-Fensters muss in das Feld `Files` die Verilog-Quelldatei eingetragen werden und in dem darunterliegenden Feld die entsprechende `Top Cell` angegeben werden (es kann auch `Auto Assign` gewählt werden). Siehe Abbildung 4.1.

In dem `Advanced`-Tab unter `Power` sollten, wenn man (wie hier) ausschließlich digitale Blöcke und keine I/O-Zellen bearbeiten möchte die Einträge `vdd3r1!`, `vdd3r2!`, `vdd3o!`, `gnd3r!` und `gnd3o!` gelöscht werden (siehe Abbildung 4.2 und [3]).

Nach einem Klick auf `OK` wird die Verilog-Quelldatei eingelesen und falls kein Fehler auftritt erscheint in der encounter GUI ein Floorplan.

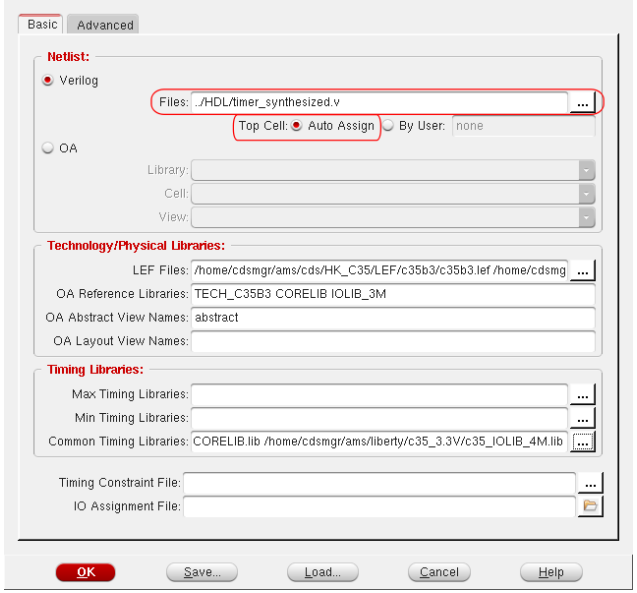


Abbildung 4.1: Design Import Fenster 1. – Die rot markierten Felder müssen angepasst werden.

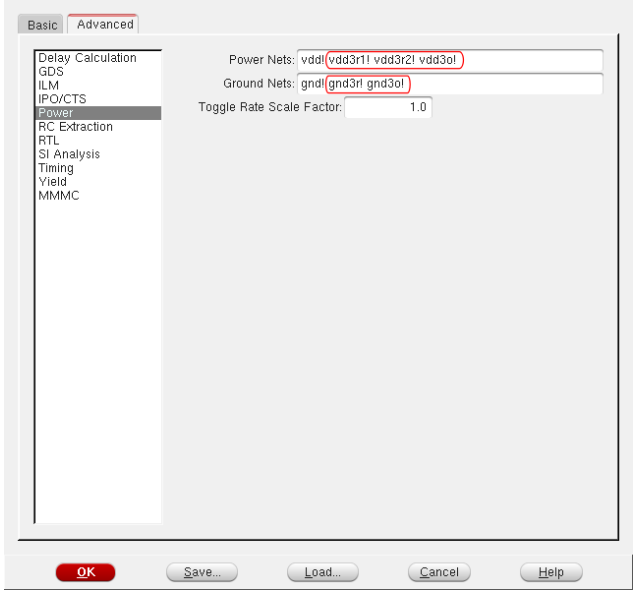


Abbildung 4.2: Design Import Fenster 2. – Die rot umrandeten Einträge können entfernt werden, wenn nur digital Blöcke geroutet werden sollen.

4.2 Floorplan editieren

Beim Floorplan wird in diesem Fall die benötigte Fläche für die Platzierung der Logikzellen, die Verdrahtung und dem Power-Ring angegeben. Abbildung 4.3 zeigt den Floorplan nach Einlesen des Verilog-Quelltextes (allerdings wurde der Hintergrund weiß eingefärbt). Die äußere, gestrichelte Linie begrenzt die komplette Fläche des digitalen Teils. Die innere, durchgezogene Linie umschließt die Fläche in der die Logikzellen gelegt und verdrahtet werden. Der Zwischenraum (zwischen den Linien) wird dabei für die Spannungsversorgung genutzt.

Der Floorplan kann unter

`Floorplan → Specify Floorplan...`

angepasst werden. Hier können Einstellungen, wie z. B. die Größe der zu benutzenden Fläche oder das Seitenverhältnis eingestellt werden können. Laut [3] sollte die `Core Utilisation` bei ungefähr 60 % liegen, was je nach Schaltung auch optimiert werden kann. Im Reiter `Advanced` kann man angeben, wie die Reihen von Standardzellen zueinander angeordnet werden. Mit der Option `Row Spacing` können leere Reihen eingefügt werden, um das Routen der Verbindungen zu vereinfachen.

Der Platz für die Spannungsversorgung ist abhängig von der Größe des Layout (Berechnung siehe [6]). Laut [5] ist `Core to left/right/topbottom` aber auf 30 zu setzen.

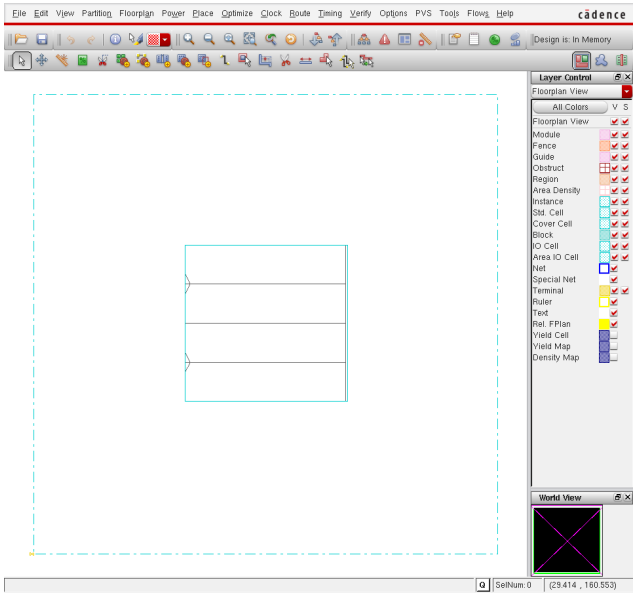


Abbildung 4.3: Floorplan im Encounter – der schwarze Hintergrund wurde in diesem Bild weiß gefärbt.

4.3 End Cap-Zellen einfügen

End Caps (englisch für Abschlussstück) bilden den Abschluss einer Reihe von Standard-Zellen. Um diese zu platzieren kann der Befehl

```
amsAddEndCaps
```

im Encounter-Terminal ausgeführt werden. Alternativ kann man unter

```
Place → Physical Cells → Add End Caps...
```

die gewünschten End Caps auswählen und platzieren lassen. [3]

4.4 Powerplaning

Hier werden die Standard-Zellen mit der Spannungsversorgung verbunden. Da normalerweise die höheren Metalllagen dicker sind als die niedrigeren, sind diese besser für die Spannungsversorgung geeignet. [4, Seite 68]

Als Erstes müssen die globalen Versorgungsspannungen miteinander verbunden werden. Dazu öffnet man

```
Power → Connect Global Nets
```

(siehe Abbildung 4.4). Hier muss einmal als `Pin Name(s):` und `To Global Net:` der Wert `vdd!` und einmal `gnd!` eingetragen werden. Dabei muss jeweils der Schalter `Scope` auf `Apply All` stehen und mit `Add to List` übernommen werden. Wichtig sind dabei die Ausrufezeichen hinter `vdd!` und `gnd!`, siehe [1].

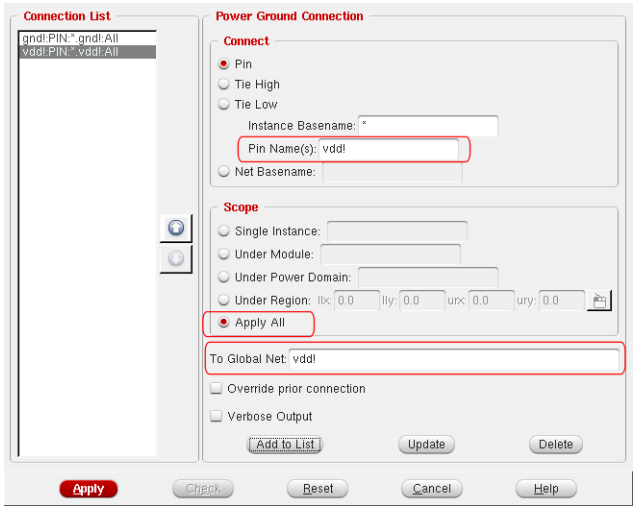


Abbildung 4.4: Global Net Connections.

4.4.1 Power-Ring platzieren

Um den Kern aus Standard-Zellen wird ein Ring mit Spannungsversorgung gelegt. Dieser Ring kann unter dem Menüeintrag

`Power → Power Planning → Add Ring...`

generiert werden. In dem Fenster können die Eigenschaften der Ringe verändert werden, wie z. B. Breite, Abstand oder Position, siehe [1].

In diesem Beispiel wurde für die horizontale (Top und Bottom) Leiterbahn die Metallage 3 und für die vertikale (Left und Right) die Metallage 2 ausgewählt (siehe Abbildung 4.5).

4.4.2 Power-Stripes

Um den Stromfluss besser aufzuteilen, können jetzt noch vertikale oder horizontale Streifen in das Layout gelegt werden. Unter

`Power → Power Planning → Add Stripes...`

können diese generiert werden, siehe [1].

Hier wurde mit den Einstellungen aus Abbildung 4.6 ein Stripe ungefähr in die mitte des Cores gelegt. Nach diesem Schritt sollte das Digitallayout wie in Abbildung 4.7 aussehen.

4.4.3 Specialroute

Mithilfe der Funktion

`Route → Special Route...`

werden die Bahnen für die Spannungsversorgung der einzelnen Logikreihen gelegt, siehe [1].

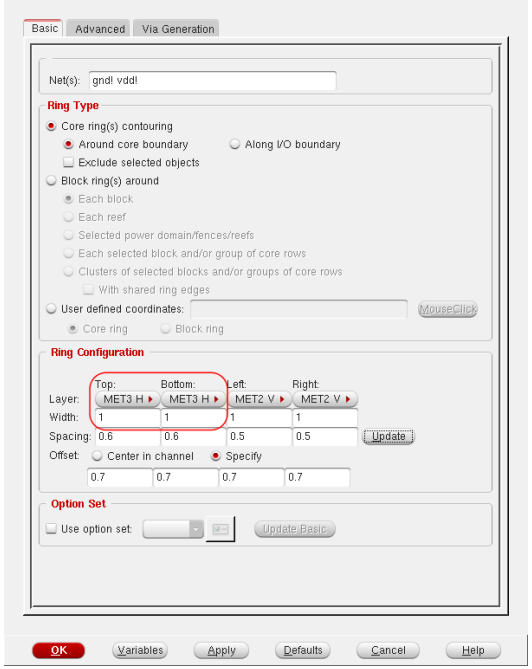


Abbildung 4.5: Encounter – Add Rings. Die hier veränderten Einstellungen wurden in dieser Abbildung rot umrahmt.

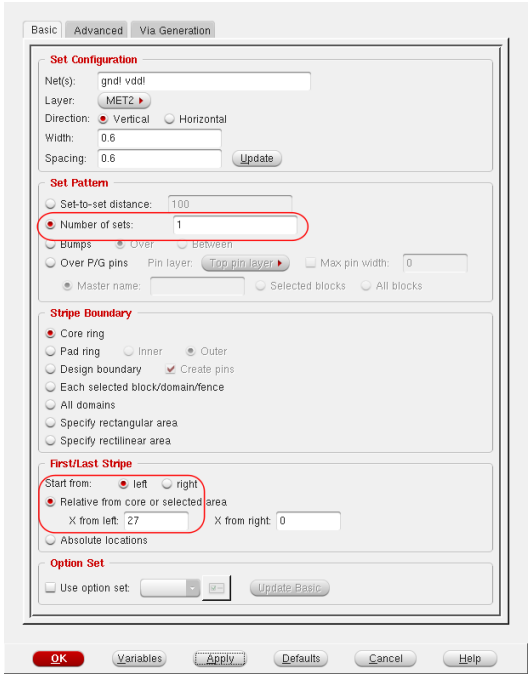


Abbildung 4.6: Encounter – Add Stripes. Die hier veränderten Einstellungen wurden in dieser Abbildung rot umrahmt.

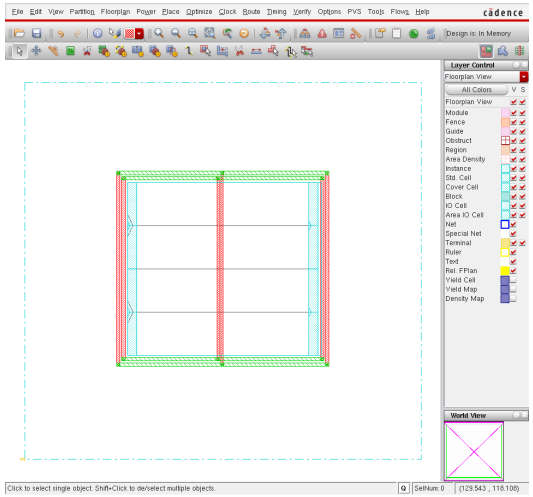


Abbildung 4.7: Digitallayout nach Powerring und -stripe Platzierung – der schwarze Hintergrund wurde aus Gründen der Darstellung eingefärbt.

4.5 Standardzellen platzieren

Unter

`Place → Place Standard Cell...`

können die Standardzellen platziert werden. Um die platzierten Zellen sehen zu können, muss man in die `Physical View` des Encounters wechseln. Dies kann man über die rechts oben in der Ecke liegenden Symbole (Abbildung 4.8) einstellen (und nicht über das Layer Control Fenster!).

Es kann beim Platzieren der Standardzellen dazu kommen, das nicht genügend Platz vorhanden ist und sich die Zellen überlappen. Dann muss wie in Kapitel 4.2 das Verhältnis der Logikkernausnutzung (`Core Utilisation`) angepasst werden. Anschließend müssen alle Schritte wiederholt werden (hierbei ist es sinnvoll die `encounter.cmd` anzupassen und auszuführen).

Abbildung 4.8: Encounter – Symbole zum Wechseln der Ansicht

Jetzt kann eine erste Timinganalyse vorgenommen werden

`Timing → Report Timing.`

Da noch kein Taktbaum generiert wurde, muss unter `design Stage` die Option `Pre-CTS` ausgewählt werden. Nach einem Klick auf `OK` erscheinen in der Konsole die Analyseergebnisse. Da noch keine Verbindungen geroutet wurden, erzeugt der encounter vorläufige Verbindugen.

4.6 Thie Cells

Signale, die auf die Potentiale der Spannungsversorgung gelegt werden erzeugen DRC-Fehler. Um logisch 1 und 0 zu erreichen, müssen deshalb Thie Cells verwendet werden. Die Zellen die dazu benötigt werden, heißen TIE0 und TIE1 und können mit dem Befehl

```
addTieHiLo -cell "TIE1 TIE0"
```

eingefügt werden (in diesem Beispiel werden allerdings keine platziert, da sie nicht gebraucht werden).

4.7 Taktbaum generieren

Da der Eingangspin nicht alle Takteingänge versorgen kann, werden hier Verstärker eingefügt und möglichst so platziert, dass kein Abweichungen in der Laufzeit auftreten.

Um den Taktbaum zu generieren muss das Skript aus Quelltext 4.1 modifiziert und ausgeführt (`source createClockTree.tcl`) werden. Als `buffer_list` können ein oder mehrere (durch Leerzeichen getrennt) der Buffer und Inverter aus Tabelle 4.1 genommen werden.

```
# Create a clock object
create_clock -period 83 -name clk -add [get_ports clk]

# Extract and translate clock-related timing constraint
# information from the Encounter timing constraints file
# and adds that data to the clock tree specification file.
createClockTreeSpec -file clockTest.ctstch -bufferList CLKBU2 ↩
   CLKBU4 CLKBU6 CLKBU8 CLKBU12 CLKBU15

# Load the clock tree specification file.
specifyClockTree -clkfile clockTest.ctstch

# Automate the Clock Tree Synthesize portion of the Encounter
# timing closure flow.
clockDesign -specFile clockTest.ctstch -clk clk

# delete Trial Route
deleteTrialRoute
```

Quelltext 4.1: createClockTree.tcl – TCL-Script für Encounter zum Erstellen des Taktbaumes.

Tabelle 4.1: Auflistung der Takttreiber – aus [3].

Clock-Inverters:	CLKIN0 CLKIN1 CLKIN2 CLKIN3 CLKIN4 CLKIN6 CLKIN8 CLKIN10 CLKIN12 CLKIN15
Clock-Buffers:	CLKBU2 CLKBU4 CLKBU6 CLKBU8 CLKBU12 CLKBU15

4.8 Routing

Es gibt Verschiedene Autorouter, die in den Encounter eingebunden sind. Am besten geeignet ist laut [3] `wroute`, das mit dem Befehl

```
amsRoute wroute
```

gestartet werden kann. Dabei sollten Signalleitungen bei einem Verfahren mit vier Metalllagen nicht auf der Obersten geroutet werden. [3]

Es kann bei dem Autorouten vorkommen, dass für manche Leitungen unvorteilhafte Leitungsführungen zustande kommen (die teilweise sogar DRC-Fehler verursachen). Es ist jedoch möglich `wroute` von Hand in eine Richtung zu lenken, indem man nach dem Ersten Durchlauf die Leitungen auf die gewünschte Position zieht (dabei muss dass nicht genau passen) und dann `wroute` erneut ausführt. Am einfachsten lässt sich eine Leitung mit dem `Move Wire` Befehl (Shortcut: `M`) versetzen (Hier hilft ein wenig Geduld und „Herumspielen“).

4.9 Einfügen der Filler Cells

Um DRC-Fehler (Design Rule Check) zu vermeiden, müssen die Lücken zwischen den Standardzellen geschlossen werden. Über

```
Place → Physical Cells → Add Filler...
```

wird eine Eingabemaske geöffnet, in der mit `Select` (das Obere) die zu benutzenden Filler Cells ausgewählt werden können. Tabelle 4.2 zeigt die Filler Cells, die benutzt werden können, wobei die Zahlen die Größe der Zelle angeben. Standardmäßig sollten die Zellen mit Kapazität (FILLx) genutzt werden [3].

Alternativ kann auch das TCL-Skript

```
amsFillcore
```

ausgeführt werden. Filler Cells mit Kapazität können nicht überall gesetzt werden (sie ragen in die erste Metalllage). Deshalb platziert das Skript erst die Filler Cells mit Kapazität wo sie passen und anschließend, die ohne Kapazität in den restlichen Lücken. [3]

Tabelle 4.2: Auflistung der Filler Cells – aus [3].

Cell with coupling caps:	FILL25 FILL10 FILL5 FILL2 FILL1
Cell without coupling caps:	FILLRT25 FILLRT10 FILLRT5 FILLRT2 FILLRT1

4.10 Exportieren der Timing-Daten, Netzliste und des Layouts

4.10.1 Netzliste

Da von dem Encounter neue Zellen hinzugefügt werden (z. B. durch den Taktbaum), muss eine neue Netzliste exportiert werden um das Layout auch korrekt simulieren zu können. Die Netzliste lässt sich über den Menüeintrag

```
File → Save → Netlist...
```

speichern (siehe Abbildung 4.9).

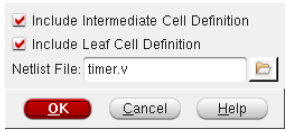


Abbildung 4.9: Encounter – Speichern der Netzliste.

4.10.2 Timing-Daten

Die parasitären Effekte können mithilfe von SDF-Dateien (Standard Delay Format) mit in die Simulation einbezogen werden. Hier empfiehlt sich wieder eine HDL-Simulation wie in Kapitel 2 beschrieben.

Für die Timing-Daten muss zuerst die Arbeitsbedingung festgelegt werden. Das geschieht mit dem Befehl

```
amsOpCond <amsOpCond Option>
```

, wobei der Parameter `<amsOpCond Option>` der Tabelle 4.3 entnommen werden kann. Anschließend können die Timing-Daten mit dem Befehl

```
amsWriteSDF
```

in den Unterordner `SDF/` geschrieben werden, siehe [3].

Tabelle 4.3: Optionen für den `amsOpCond`-Befehl – aus [3].

`amsOpCond` option	min-Timing	max-Timing
`min`	BEST-MIL	BEST-MIL
`typ`	TYPICAL	TYPICAL
`max`	WORST-MIL	WORST-MIL
`minmax`	BEST-MIL	WORST-MIL

4.10.3 Layout

Um das Layout in Virtuoso einzubinden, muss das Layout erst aus dem encounter exportiert werden:

```
oaOut <ZielLib> <ZielCell> <layout> -leafViewNames {layout}
```

In dem Beispiel des Zählers würde es folgendermaßen aussehen:

```
oaOut Counter_4Bit_LIB counter_4bit layout
        -leafViewNames {layout}
```

Mit diesem Befehl wird ein Verzeichnis im aktuellen Arbeitsverzeichnis erstellt, das in Virtuoso eingebunden werden kann (siehe Kapitel 1.4).

4.10.4 Zusammenfassendes Script

Quelltext 4.2 zeigt ein Script, indem das exportieren zusammengefasst ist.

```
# export layout and SDF-Timingfile and netlist

set targetLib   Counter_4Bit_LIB
set targetCell  counter_4bit

set targetNetlistfile counter_4bit_syn_placed.v

set opCondition minmax
#
# opCondition | min-Timing | max-Timing
# ------------+------------+-----------
# min         | BEST-MIL   | BEST-MIL
# typ         | TYPICAL    | TYPICAL
```



```
# max         | WORST-MIL  | WORST-MIL
# minmax      | BEST-MIL   | WORST-MIL
#


############################################################
############################################################


# export layout
oaOut $targetLib $targetCell layout -leafViewNames {layout}

# exoport sdf-timing
amsOpCond $opCondition
amsWriteSDF

# save netlist
saveNetlist $targetNetlistfile
```

Quelltext 4.2: encounter_export.tcl – TCL-Script für Encounter zum Exportieren des Layouts, der Timing-Datei und der Netzliste.

5 Importieren nach Cadence

5.1 Importieren des Verilog-Codes

Cadence benötigt neben dem Layout auch den Schaltplan (schematic) und ein Symbol (symbol) der einzelnen Komponenten, damit sie zu einem größeren Block (z. B. Chip mit Pads) zusammengefügt werden können.

Bevor der Verilog-Code importiert werden kann, muss eine Bibliothek erstellt werden. Das kann zwar auch direkt von Verilog In gemacht werden, doch dann treten später Probleme auf. Um eine neue Bibliothek zu erstellen, muss im Library Manager das Menü

`File → New → Library..`

aufgerufen werden. Hier müssen Speicherort und Name (Counter_4Bit_LIB) eingetragen und mit `OK` bestätigt werden. Im nächsten Fenster `Attach to an existing technology library` auswählen und ebenfalls mit `OK` bestätigen. Als Technology Library ist dann `TECH_C35B3` zu wählen.

Um die von dem Cadence Encounter RTL Compiler kompilierten Quelltext als Schaltplan in Cadence einzubinden, wird das Programm *Verilog In* benötigt. Dieses lässt sich aus dem Viruoso Fenster im Menü

`File → Import → Verilog...`

starten.

Es erscheint das in Abbildung 5.1 gezeigte Fenster. Hier müssen die in der Abbildung rot umrahmten Felder angepasst werden. Als `Target Library Name` ist die Bibliothek zu wählen, in der der Schaltplan importiert werden soll. `Reference Libraries` ist in unserem Fall „nur“die `CORELIB` (hier sind die Standardzellen von AMS gespeichert). Unter dem Punkt `Verilog Files To Import` wird der zu importierende Verilog-Quelltext angegeben. Bei `Import Structural Modues As` ist `schematic and functional` und bei `Verilog Cell Modules` der Wert `Import` zu wählen.

Nach einem Klick auf `OK` wird die Datei importiert. Anschließend kann die Log-Datei des Importvorganges eingesehen werden. Ist kein Fehler aufgetreten, kann der Schaltplan in der angegebenen Bibliothek gefunden werden. Des Weiteren taucht auch ein Symbol für die Komponente in der Bibliothek auf.

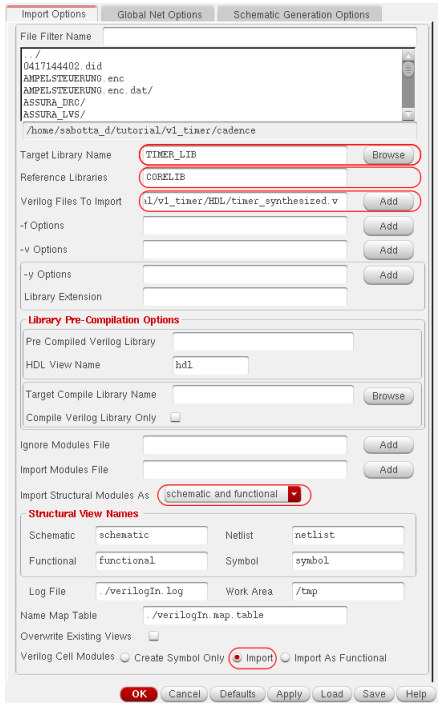


Abbildung 5.1: Verilog In – Eingabemaske

6 Chip-Top-Level erstellen

6.1 Schematic

Um aus den einzelnen (digitalen) Blöcken ein Chip-Layout zu bekommen, müssen die Ausgänge der Blöcke noch an Ein- bzw. Ausgangs-Zellen angeschlossen werden. Diese i/o-Zellen enthalten Treiber, Pads zum anschließen der Bonding-Drähte und einen ESD-Ring. Die i/o-Zellen sind in der IOLIB_3M (bei einem 3 Metalllagen Prozess) zu finden. Auf der Seite der Pads müssen hier noch Pins angeschlossen werden. Die Datenblätter der i/o-Zellen sind auf der Hompage von Austriamicrosystems zu finden. Die Spannungsversorgung wird über spezielle Pins angeschlossen, siehe [8].

Als Erstes muss, wie oben beschrieben, das von dem Encounter erzeugte Layout in Virtuouso eingebunden werden. Am besten in eine extra Library, wie z.B. Counter_4Bit_LIB_encounter, dann kann im Library Manager das Layout einfach in die Cellwiev des Top-Levels kopiert werden. Jetzt muss eine neue Cellview und eine darin erhaltene Schematic erstellt werden. Diese dient als Top-Level des Chips. Hier wird der Digitalcore mit den Pads verbunden (siehe Abbildung 6.1).

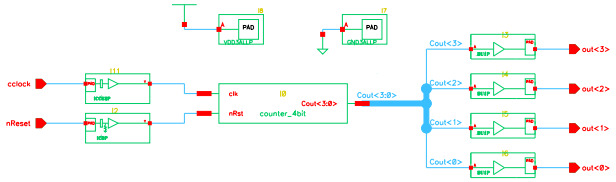


Abbildung 6.1: Schematic des Top-Levels mit Pads

6.2 Layout

Die hier beschriebene Methode bedient sich nicht den von Cadence bereitgestellten Funktionen, die es ermöglichen die folgenden Schritte zu automatisieren.

Das Layout wird mit dem Layout Editor erstellt. Diesen kann man am besten aus dem Schematic Editor starten:

`Launch → Layout GXL`

Um alle in der Schematic platzierten Komponenten in das Layout einzufügen, muss der Menüpunkt

`connectivity > Generate > All From Source...`

aufgerufen werden. Damit die zusammengeschlossenen i/o-Zellen einen Ring ergeben, werden für die Ecken die `COORNERP` benötigt.

Hat man die Größe der benötigten Fläche auf dem Chip ermittelt (bei kleiner Logik mit relativ vielen Ausgängen „Padbestimmt“), so kann man den Floorplan anpassen (auch hier gibt es einige Tools, die diesen Vorgang übernehmen können). Dazu muss als Erstes der `Task Assistant` gestartet werden

`Window → Assistants → Task Assisant`

Nachdem hier die Option `Floorplaning` ausgewählt wurde, erscheinen neue Menüs und Shortcuts. Jetzt kann die Chipgröße unter

`Floorplan → Reinitialize Floorplan`

eingestellt werden. Wurde im Vorfeld die benötigte Chipgröße festgelegt, empfiehlt sich die Größe mit der Option

`Enter Design Size`

einzugeben. Dabei ist zu beachten, dass die tatsächliche Größe des Floorplans von den eingegebenen Werten abweichen kann.

In Abbildung 6.2 ist der angepasste Floorplan und alle Komponenten aus der Schematic und die CORNERCELLS zu sehen.

Jetzt können die i/o-Zellen und die Eckzellen als Ring entlang der Floorplanbegrenzung gelegt werden. Lässt sich die Floorplanbegrenzung nicht komplett mit den i/o-Zellen füllen (es bleiben Lücken), müssen noch Füllzellen (`PERI_SPACER_X_P`) eingefügt werden, um den Ring zu schließen. Ein relativ einfacher Weg die Zellen manuell zu platzieren ist:

1. In die äußere Ecke der Zelle zoomen,

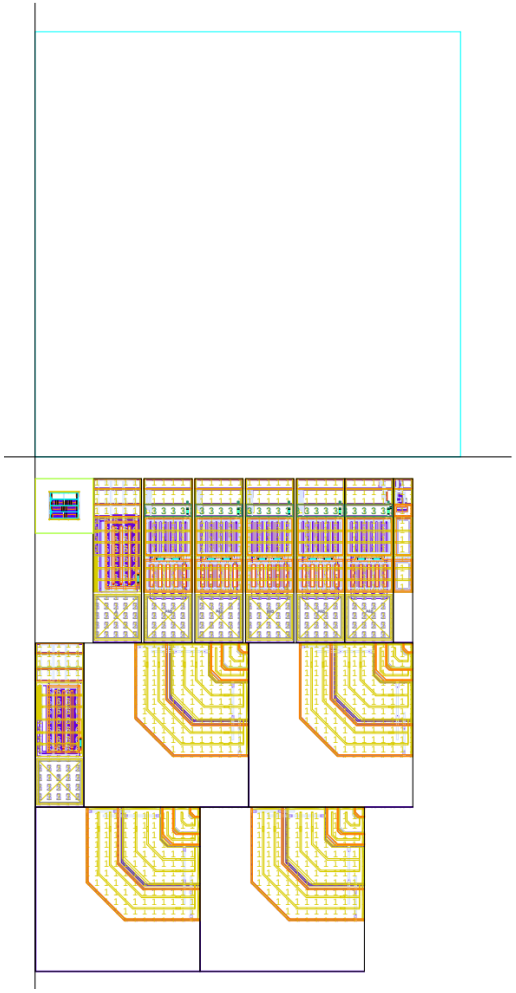


Abbildung 6.2: Layout vor der Platzierung

2. auf die Ecke klicken und diese mit dem `move`-Befehl (`m`) bewegen (zum schnelleren Bewegen herauszoomen),
3. in die gewünschte Stelle zoomen und die Zellen direkt aneinander platzieren.

Eventuell muss die Zelle vorher noch rotiert werden (`SHIFT + o`). Beim Rotieren kann es vorkommen, dass die Zellen aus dem Raster rutschen. Um sie wieder in das Raster zu legen, wählt man den `move`-Befehl und öffnet das dazu gehörige Konfigurationsmenü (`F3`). In dem Konfigurationsmenü kann die Option `snap to grid` aktivieren werden. Bewegt man nun die Zelle, liegt sie danach wieder im Raster.

Wurde der Ring fertig gestellt, können die Zellen in die Mitte des Ringes platziert werden.

Jetzt müssen nur noch die Anschlüsse der einzelnen Zellen mit einander verbunden werden. Dazu kann man einen der vielen Autorouter verwenden oder sie manuell ziehen. Der einzige Autorouter der bei diesem Versuch einigermaßen erfolgreich durchgelaufen ist, kann man über

```
Route > Automatic Routing...
```

gestartet werden. Hier hat allerdings auch nur die Einstellung `Device Level` bei `Style` funktioniert (und dass nicht einmal korrekt). Wichtig ist hier die Option `Bottom Layer` auf `MET1` zu stellen, damit nicht mit dem Poly geroutet wird. Wie schon erwähnt, läuft der Router nicht ganz korrekt, es kam hier vor, dass die Leitungen nicht ganz an die i/O-Zellen herangeführt wurden. Deshalb mussten alle Leitungen noch zu den i/o-Zellen manuell angeschlossen werden (Hierfür gibt es mit Sicherheit noch einfachere Lösungen). Des Weiteren ist zu beachten, dass der Autorouter Leitungen auch mitten durch die Digitalzellen legt (allerdings ohne Fehler zu produzieren).

In Abbildung 6.3 ist das komplette Layout zu sehen.

6.3 DRC

Ist das Layout fertig, muss noch ein DRC durchgeführt werden um eventuell vorhandene Fehler zu finden. Für die Überprüfung steht das Programm `Asura` zur Verfügung, das über

```
Asura → Run DRC...
```

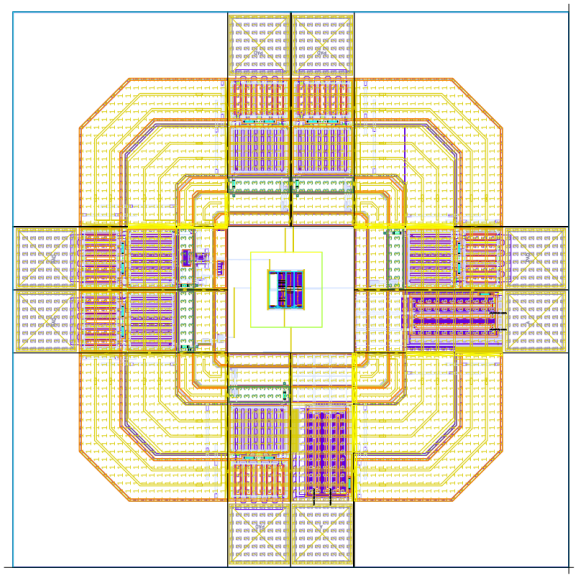


Abbildung 6.3: Fertiges Layout.

gestartet wird. Mit einem Klick auf `OK` startet der DRC (zu Design Rules siehe [2]). Einige der Fehlermeldungen treten in den Standardzellen auf und können ignoriert werden. Antanna-Fehler sind einfach zu beheben, indem man möglichst dicht vor dem Gate, das den Fehler auslöst, auf die höchste Metalllage wechselt.

Das Layout sollte mit der Schematic verglichen werden. Dies geschieht mit dem Layout versus Schematic Tool, dass mit

`Asura` → `Run LVS...`

gestartet werden kann. Hierbei tauchen unvollständige Verbindungen und fehlende bzw. überflüssige Bauteile in dem Layout auf (hierfür müssen alle Schematics der verwendeten Komponenten in den Bibliotheken vorhanden sein).

Literaturverzeichnis

[1] *Cadence SOC Encounter Tutorial.* : *Cadence SOC Encounter Tutorial*, http://eeweb.poly.edu/labs/nanovlsi/tutorials/soctutorials/Tutorial_Encounter.html

[2] AG austriamicrosystems: *0.35 um CMOS C35 Design Rules.* Rev.: 9.0. austriamicrosystems AG, 2011. http://asic.ams.com/cgi-sbin/agreement?ENG-183_rev9.pdf

[3] AUSTRIAMICROSYSTEMS AG (Hrsg.): *First Encounter - P&R Flow.* : austriamicrosystems AG, http://asic.austriamicrosystems.com/hitkit/hk400/fe/index.html

[4] BAKER, R. J.: *CMOS Circuit Design, Layout, and Simulation.* 3. Wiley-IEEE Press, 2010 http://amazon.com/o/ASIN/0470881321/. – ISBN 9780470881323

[5] BRUNVAND, Erik: *Digital VLSI Chip Design with Cadence and Synopsys CAD Tools.* 1. Addison Wesley, 2009 http://amazon.com/o/ASIN/0321547993/. – ISBN 9781926567556

[6] CADENCE (Hrsg.): *Designing Power Mesh Structures Application Note.* : Cadence, July 2002. http://support.cadence.com

[7] EUROPRACTICE: *AMS C35 cmos +opto technology overview (MPW).* http://www.europractice-ic.com/technologies_AMS.php?tech_id=cmos

[8] GOSCH, Karl: *Application Note – Power Supply Concept.* Draft v1.0, 2012. http://asic.ams.com/appnotes/digital/Power_Supply_Concept_Draft1.0.pdf

[9] MÄDER, Andreas: Cadence Grundlagen / MIN-Fakultät der Universität Hamburg. Version: 2009. http://tams-www.informatik.uni-hamburg.de/research/vlsi/edaTools/doc/09/cadence.pdf. 2009. – Forschungsbericht

[10] MÄDER, Andreas: Full-Custom Design - Cadence IC v6.1.31 + AMS HitKit v4.00 / MIN-Fakultät der Universität Hamburg. Version: nov 2009. http://tams-www.informatik.uni-hamburg.de/research/vlsi/edaTools/doc/09/fcDesign.pdf. 2009. – Forschungsbericht

[11] STAN, Mircea R.: *VHDL/Verilog Simulation Tutorial.* : ECE Department, University of Virginia, sep 2006. http://www.ee.virginia.edu/~mrs8n/soc/sim_tutorial.html

Versicherung über die Selbstständigkeit

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

Hamburg, 17. Januar 2013
Ort, Datum

Unterschrift