Friedrich-Alexander-Universität Erlangen-Nürnberg,
Lehrstuhl für Technische Elektronik,
Prof. Dr.-Ing. Dr.-Ing habil. Robert Weigel

Studienarbeit im Fach

Elektrotechnik, Elektronik und Informationstechnik

Ausarbeitung eines Praktikums zur Vorlesung Architekturen Digitaler Signalverarbeitung

- Hardwareentwurf, VHDL-Programmierung, Simulation und Inbetriebnahme -

Bearbeiter:
Daniel Glaser

Betreuer:
Prof. Dr.techn. Mario Huemer
Dipl.-Ing. Alexander Kölpin

Beginn: 22.05.2006
Abgabe: 22.11.2006

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe, und dass die Arbeit in gleicher oder in ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, den 22. November 2006 Daniel Glaser

Zusammenfassung

In der Vorlesung „Architekturen der Digitalen Signalverarbeitung“ am Lehrstuhl für Technische Elektronik der Friedrich-Alexander-Universität Erlangen-Nürnberg werden Architekturen behandelt, mittels derer, früher analog durchgeführte Signalverarbeitungsschritte, effizient in digitaler Hardware bewältigt werden können. Dabei kommen zur digitalen Verarbeitung immer häufiger FPGAs zum Einsatz.

Die Studierenden sollen in Form eines Praktikums die Schritte von der Konzeption über die MATLAB-Simulation, der VHDL-Beschreibung bis hin zur Erprobung in der Praxis mittels FPGA durchlaufen.

Die vorliegende Studienarbeit beschäftigt sich mit dem Entwurf einer flexiblen Hardware, die für die Praktikumsversuche notwendig ist und der Vermittlung des Wissens, um ausgehend von der Konzeption und der Simulation in MATLAB zu einer funktionierenden Hardware zu gelangen. Weiterhin werden die Probleme beim Entwurf der Hardware, potenzielle Probleme im Verlauf des Praktikums, Abweichungen von der MATLAB-Simulation und mögliche Lösungen hierzu diskutiert.

Der Teil der Simulation in MATLAB ist Aufgabe einer anderen Studienarbeit.

Inhaltsverzeichnis

Teil I

Ausarbeitung

1 Einleitung und Motivation

1.1 Einleitung

Seit vielen Jahren drängt die Digitaltechnik immer mehr in die früher den analogen Komponenten vorbehaltenen Bereiche. Dies hat zum Einen den Grund, dass ein digitales Signal, entsprechende Wortbreite vorausgesetzt, nicht bei der Verarbeitung degradiert oder verfälscht wird. Zumindest sind die entstehenden Fehler exakt kalkulierbar. Andererseits sind analoge Schaltungen nur sehr schwer zu integrieren, da im Zuge der Miniaturisierung die Bauteileigenschaften immer größeren Toleranzen ausgesetzt sind. Daher ist es meist billiger, die Verarbeitung digital durchzuführen, obwohl der Schaltungsaufwand in Bezug auf die Anzahl der Transistoren bedeutend größer ist. Der Grund hierfür ist im aufwändigen Entwurf aufgrund der großen Toleranzen zu suchen, die bei digitalen Komponenten wesentlich einfacher zu handhaben sind.

Mit steigender Integrationsdichte und schnelleren Transistoren kam aus vorgenannten Gründen immer mehr der Wunsch auf, analoge Konzepte in digitale Algorithmen umzusetzen. Heute gibt es praktisch keine analogen Schaltungskonzepte (z.B. Filter, Signalgeneratoren, Mischer,...), denen kein numerischer Algorithmus gegenübersteht. Einige dieser Algorithmen werden in der Vorlesung „Architekturen der Digitalen Signalverarbeitung“ von Prof. M. Huemer behandelt (siehe [2]).

1.2 Motivation

Um die erlernten Algorithmen praktisch umzusetzen, genügt nicht allein das Verständnis der mathematischen Hintergründe. Eine Simulation des Algorithmus an sich und eine Möglichkeit, die Hardware zu beschreiben werden in der Vorlesung nicht behandelt. Auch die Auswirkungen auf die Hardware, die sich aus den Algorithmen ergeben, werden während der Vorlesung erwähnt, aber von den Studenten nicht notwendigerweise in der Tragweite verstanden, da die Erfahrung mit den Systemen fehlt.

Für die Implementierung gibt es zahlreiche Möglichkeiten. Die gebräuchlichsten sind: Software für digitale Signalprozessoren (DSP), Programmierbare Logikbausteine (PLD,

FPGA) und der Entwurf einer anwendungsspezifischen integrierten Schaltung (ASIC). Die Entscheidung für eine Variante ist nicht zuletzt stückzahlabhängig. In der Praxis wird oft die Entwicklung der Algorithmen mittels programmierbarer Logikbausteine durchgeführt, bevor eine integrierte Schaltung entworfen wird.

Im Entwurf sind programmierbare Logikbausteine und integrierte Schaltungen sehr ähnlich, da die Beschreibung mittels Hardware-Beschreibungssprachen (VHDL, Verilog) erfolgt. Diese Beschreibungssprachen sind relativ einfach zu erlernen und können mit Einschränkungen der verfügbaren Konstrukte und Anweisungen beispielsweise in ein FPGA programmiert werden.

Um die Algorithmen evaluieren zu können dient häufig die Software MATLAB. Mit Hilfe dieser können bereits die Auswirkungen bei Verwendung von realer Hardware (endliche Wortbreiten) analysiert werden und Entscheidungen bezüglich grundlegender Gesichtspunkte der Implementierung getroffen werden. Dieser Teil wurde parallel in der Studienarbeit von Andreas Schedel bearbeitet und soll der vorliegenden Studienarbeit nur in ihrem Ergebnis beitragen.

Um in der Praxis einerseits einen gewissen Überblick über die Vor- und Nachteile einzelner Konzepte zu besitzen und andererseits den Studenten einen Einblick in die Praxis zu gewähren, bedarf es eines Praktikums, das die Theorie der Vorlesung ergänzt und zu fundiertem, praktischem Wissen führt.

2 Zielsetzung

In dieser Arbeit soll ein Praktikum zur Vermittlung der nötigen Kenntnisse entworfen werden, das ausgehend von den Konzepten den gesamten Weg bis zur funktionsfähigen Schaltung durchläuft. Um dieses Ziel zu erreichen, sind folgende Teilaufgaben zu erfüllen:

- Vorgabenanalyse
- Konzeptauswahl
- Plattformauswahl
- Didaktische Aufbereitung
- Problemanalyse

Um zu dem gewünschten Ergebnis der Teilaufgaben zu gelangen, wurden im Vorfeld einige Eckdaten festgelegt und ein bereits bestehendes Praktikum der Fachhochschule Hagenberg vorgestellt. Es diente dieser Studienarbeit lediglich als Anlehnung für die Gliederung des Praktikumsskripts und als Grundlage für den MATLAB-Quellcode.

3 Vorgaben

Eine der entscheidendsten Vorgaben bestand im geforderten Vorwissen der Praktikumsteilnehmer. Diese sollen, sofern sie das Grundstudium der Elektrotechnik durchlaufen haben, ohne weiteres Vorwissen an dem Praktikum teilnehmen können.

Um die Versuche anschaulich zu gestalten und besser verstehen zu können und um die Signalverarbeitung bei relativ niedrigen Datenraten durchführen zu können, wurde die Bandbreite der zu verarbeitenden Signale auf den Audio-Bereich (20Hz bis 20kHz) festgelegt. Zur weiteren Veranschaulichung fiel die Wahl auf eine Signalübertragungsstrecke mit FSK-Modulation(Frequency Shift Keying: Zwei unterschiedliche Frequenzen für logisch 0 bzw. 1) von digitalen Daten und der Luft als Übertragungsmedium. Die Verifikation kann so durch das Gehör erfolgen und bedarf keiner aufwändigen Messung. Dies trägt auch zur Einhaltung der engen zeitlichen Grenze von einer Woche Praktikumsdauer bei.

Die finanziellen Vorgaben von 200 Euro/Praktikumsplatz schränken die Möglichkeiten in Bezug auf die Hardware stark ein. So geht die möglichst gute Nutzung der in den Praktikumsräumen bereits vorhanden Ressourcen damit direkt einher. Der Wunsch, FPGAs (Field Programmable Gate Array) von Lattice zu verwenden, unterstützt diese Vorgabe aufgrund des niedrigen Preises dieser programmierbaren Logikbausteine. Die Prototypen der Leiterkarte durften die Kostenvorgabe jedoch geringfügig übersteigen.

Ein Wunsch des Bearbeiters der parallelen Studienarbeit war es je einen Ein- und Ausgang, an die Soundkarte eines PC anschließen zu können. Mittels dieser Verbindung wäre es möglich, von MATLAB generierte Signale der Hardware zuzuführen bzw. Signale von der Hardware mittels PC zu analysieren.

Im Verlauf der Studienarbeit kam es zu weiteren, nicht unmittelbar vorhersehbaren Einschränkungen bezüglich der Fertigung der Prototypen. Hier musste aus Kostengründen der günstigste Hersteller gewählt werden. Dies hatte Auswirkungen auf das Design (nur zwei Entflechtungslagen) und die Qualität der Leiterplatte (starker Versatz der Lötstoppmaske). Auch die Beschaffung der Bauteile stieß auf unerwartete Probleme, da der gewählte Zulieferer die dem Lehrstuhl zur Verfügung stehenden Zahlungsarten nich anbot. Darum musste teilweise auf Ersatztypen ausgewichen und ein Zulieferer gefunden werden, der die gewünschten Bauteile führte. Für manche der Bauteile war es unumgänglich, auf Muster des Herstellers zurückzugreifen, um den Prototypen realisieren zu können.

4 Konzeptauswahl

Da nicht alle im Skript beschriebenen Konzepte zur digitalen Signalverarbeitung behandelt werden können und die geforderte Signalübertragung bestimmte Konzepte voraussetzt, musste eine Auswahl getroffen werden. Die folgenden grundlegenden Module sind in jedem Fall notwendig:

- Signalgenerator
- Modulator
- Demodulator
- Filter
- Datenquelle

Aus diesen ist jeweils die jenige Realisierungsmöglichkeit, die am geeignetsten für den Anwendungsfall erscheint auszuwählen. Dabei sind die Einschränkungen der Hardware eines der Hauptkriterien bei der Entscheidungsfindung.

4.1 Signalgenerator

Für den Signalgenerator kommen zwei Konzepte in Frage: Die **D**irekte **D**igitale **S**ynthese (DDS) und der **CO**ordinate **R**otation **DI**gital **C**omputer (CORDIC).

4.2 Modulator

Auch hier bestehen mehrere Möglichkeiten, wenn FSK als Modulation zum Einsatz kommt: Multiplizierer, Multiplexer und Umschaltung der Generatorfrequenz.

4.3 Demodulator

Als Demodulator bei einer FSK-Modulation kommen mehrere Varianten in Betracht: Spektrumanalyse mittels (**F**ast) **F**ourier **T**ransformation (FFT), Hüllkurven-Demodulator, Synchronisation auf Träger mittels **P**hase **L**ock **L**oop (PLL), synchroner Demodulator, ...

4.4 Filter

Es existieren zwei grundlegende Verfahren zur Realisierung von digitalen Filtern: **F**inite **I**mpulse **R**esponse (FIR) und **I**nfinite **I**mpulse **R**esponse (IIR) Filter. Eine mögliche Implementierung ist in Abb. 4.1

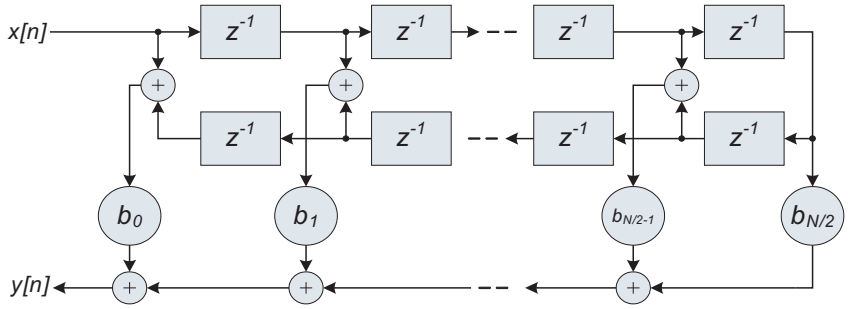


Abbildung 4.1: FIR-Filter in Linearphasenstruktur

4.5 Datenquelle

Als Signalquelle bietet sich eine definierte Folge von Werten an. Mögliche Konzepte sind das Auslesen zuvor festgelegter Werte aus einem Speicher und die Erzeugung einer Pseudozufallszahl mittels eines **P**seudo **R**andom **N**oise (PRN, vgl. 4.2) Schieberegisters.

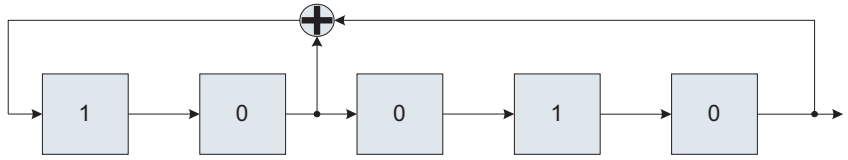


Abbildung 4.2: Schematische Darstellung eines PRN-Schieberegisters

5 Auswahl einer Entwicklungsplattform

5.1 Rahmenbedingungen

Beim Entwicklungssystem lag, wie in den Vorgaben bereits erwähnt, das Hauptaugenmerk auf dem Preis. Die Hardware muss bestimmten Mindestanforderungen genügen, um die Konzepte umsetzen zu können. Hierzu eignen sich praktisch alle FPGAs die zusätzliche Hardwareeinheiten bereitstellen, sogenannte DSP-Einheiten, welche die Signalverarbeitung unterstützen. Diese DSP-Einheiten bestehen zumeist aus Multiplizierern mit Wortbreiten bis 36 Bit und zusätzlichen Addierern. Von diesen Einheiten muss eine gewisse Anzahl im FPGA zur Verfügung stehen, um die geforderten Aufgaben implementieren zu können. Erste Schätzungen beliefen sich hier auf eine Untergrenze von zwanzig Multiplizierern, um beispielsweise ein Filter ohne mehrfache Nutzung der Hardware implementieren zu können.

Ein weiterer Gesichtspunkt ist die Anzahl der implementierbaren Logikfunktionen. Diese wird häufig durch die Anzahl der **L**ook-**U**p **T**ables (LUT) angegeben und sollte, ausgehend von Erfahrungswerten und einer Reserve um lange Synthesezeiten zu vermeiden, bei etwa 20.000 (20k) LUTs liegen.

5.2 Kernkomponente: FPGA

Bei der Wahl des FPGA kamen aufgrund der Vorgaben bei Preis und Hersteller und der benötigten Ressourcen nur die Familien ECP [5] und ECP2 [3] in Betracht. Leider waren die ECP2-FPGAs, da es sich um eine neue Familie handelt, nicht im Zeitraum der Studienarbeit verfügbar. Unter Beachtung der vorgenannten Bedingungen bleiben lediglich der ECP20 und der ECP33. Aus Kostengründen wurde der ECP20 im 484fpBGA[1] gewählt.

[1] fp: fine pitch $\widehat{=}1,0$mm; BGA: **B**all **G**rid **A**rray

5.3 Entwicklungssystem

Grundsätzlich kann man zwischen dem Kauf und der eigenen Entwicklung eines Systems wählen. Bei geringen Stückzahlen oder Einzelstücken ist es meist finanziell vorteilhaft, ein bestehendes System zu erwerben und gegebenenfalls mit den erwünschten Komponenten zu erweitern, sofern nur wenige Erweiterungen nötig sind und das System dies unterstützt.

5.3.1 Verfügbare Systeme

Das einzig derzeit verfügbare Entwicklungssystem (LFECP20E-H-EV) für das angestrebte FPGA LFECP20E bewegt sich im Preissegment bei 1300$ (bei deutschen Distributoren konnte der Preis nicht ermittelt werden) und scheidet daher von vorne herein aus. Weiterhin handelt es sich bei diesem System um eine PCI-Einsteckkarte mit wenigen zusätzlichen Funktionen, die für das gewünschte Ziel nützlich sind. Die Hardware müsste um viele Komponenten erweitert werden.

5.3.2 Entwicklung einer Plattform

Eine andere Möglichkeit ist die eigene Entwicklung eines Systems. Dies ist mit erheblichem Arbeitsaufwand verbunden, angesichts der wenigen Alternativen, allerdings die einzig gangbare. Der geschätzte finanzielle Aufwand bei Fertigung zweier Prototypen liegt bei etwa 300-400 EURO und die Fertigung von 15 Stück, im Folgenden Kleinserie genannt, für das Praktikum bei etwa 200 EURO.

5.4 Schaltungsentwurf

Einige der wichtigsten Aspekte beim Schaltungsentwurf sind die Auswahl der Bauelemente, die Prüfung der Verfügbarkeit und die richtige Verschaltung der Komponenten. Um eine ausreichende Reserve für die spätere Schaltung und eine Funktion des Prototyps möglichst sicherzustellen, fiel die Auswahl immer auf Bauelemente, die die erforderlichen Daten auch bei schlechten Bedingungen einhalten. Die Verwendung von günstigeren Bauteilen in der Kleinserie zur Kosteneinsparung kann nach Verifikation der Schaltung erfolgen, da so unnötige Iterationsstufen beim Prototypen vermieden werden können.

5.4.1 FPGA

Anhand der Angaben im Datenblatt der ECP-Familie [4] ist ein Teil der Beschaltung des FPGA bereits vorgegeben. Besonders in Bezug auf die Konfiguration existieren feste Bezie-

hungen zu Anschlüssen des Gehäuses (vgl. S. 66). Weiterhin sind die frei konfigurierbaren Anschlüsse in Bänke aufgeteilt, die eine unterschiedliche Versorgungsspannung ermöglichen. Die Konzentration von funktional zusammengehörigen Signalen auf eine Bank bietet auch bei der späteren Synthese (genauer dem Place and Route) Vorteile. Durch die freie Belegung der meisten Anschlüsse vereinfacht sich meist das Layout erheblich.

Konfiguration

Die Konfiguration des FPGA kann immer durch den nach IEEE1149.1 (JTAG[2]) standardisierten Anschluss erfolgen. Wahlweise stehen noch weitere Möglichkeiten der Konfiguration zur Verfügung, wovon SPI-Slave[3] optional verfügbar ist, um ein Testprogramm implementieren zu können, mit dessen Hilfe die Studenten bei Beginn des Praktikums die Funktion der Hardware überprüfen können. Der Flash-Speicher muss mindestens 8MBit Speicher besitzen, um unkomprimierte Konfigurationsdaten aufnehmen zu können. Er ist vorgesehen, wird aber für den Entwurf noch nicht bestückt.

Spannungsversorgung

Für die Funktion des FPGA müssen bestimmte Spannungen zur Verfügung gestellt werden, andere sind optional und für die Grundfunktion nicht notwendig. Zu den benötigten zählen die Core-Spannung[4], deren Nominalwert $1,2V$ beträgt und deren enge Toleranz eine genaue Planung und Kalkulation. Die PLL hängt ebenfalls von dieser Versorgung ab. $3,3V$ müssen auch zwingend zur Verfügung gestellt werden, um die Schnittstellen zur Konfiguration zu betreiben. Auch die Versorgung wichtiger (VCC_{aux}), und später benötigter Komponenten (Versorgung einiger IO-Bänke) des Bausteins hängen von dieser Versorgung ab. Die Strombelastung der Core-Spannung hängt stark von der Taktfrequenz und der Menge der Gatter, sowie deren Schalthäufigkeit ab und kann im Vorfeld nur geschätzt werden. Aus Erfahrungen mit SpartanIII-FPGAs der Firma Xilinx, die dem ECP am nächsten kommen, kann von einer maximalen Stromaufnahme im Betrieb von 3A ausgegangen werden. Die Ruhestromaufnahme ohne Konfigurationsdaten beträgt nur wenige mA, während der Konfiguration typisch 60mA. Die Belastung der $3,3V$-Versorgung kann mit wenigen mA in Ruhe und 40-100mA im Vergleich zu der Belastung durch weitere Systemkomponenten, die im Folgenden behandelt werden, vernachlässigt werden.

[2] Joint Test Action Gorup
[3] FPGA liest aktiv die Konfiguration aus einem Flash-Speicher mit SPI-Schnittstelle (Serial Peripherial Interface)
[4] Kern des FPGA

5.4.2 Audio-Eingang

Analog-Digital-Umsetzer

Der Audio-Eingang gliedert sich in einen analogen und einen digitalen Teil, deren Trennung durch einen AD-Umsetzer erfolgt. Durch den Einsatz eines Umsetzers aus dem sogenannten Consumer[5]-Bereich, kann eine sehr kosteneffiziente und dennoch hochwertige Lösung erzielt werden. Höherwertige Umsetzer dieser Klasse bieten Wandlungsraten (f_r) von 96kHz und Auflösungen (N) von 24 Bit. Die Schwierigkeit lag in der Beschaffung eines Umsetzers mit $3,3V$-Schnittstelle auf der Digitalseite, da das FPGA keine höheren Spannungspegel erlaubt. Lediglich Texas Instruments bot noch wenige günstige Schaltkreise an, die die gewünschten Eigenschaften besitzen. Nach Elimination der AD-Umsetzer, die ungeeignete Schnittstellen oder ungünstige Wandlungsraten besitzen, bleiben lediglich drei in der engeren Auswahl (vgl. Tab. 5.1).

Bezeichnung	**ADCs**	**N**	**SNR**	**Eingänge**	f_r	**Interface**	**€** (1k)
PCM1802	2	24	105	2	96	I^2S	3.35
PCM1803(A)	2	24	103	2	96	I^2S	1.10
PCM1808	2	24	101	2	96	I^2S	1.00

Tabelle 5.1: Auswahl von geeigneten AD-Umsetzern

Aufgrund des geringfügig besseren Signal-Rausch-Abstandes (SNR) und der besseren Verfügbarkeit, findet der PCM1803(A) [15] Verwendung. Die Prüfung der Verfügbarkeit erfolgte durch Digi-Key (möglicher Distributor), der gute Preise und eine große Auswahl bietet.

Die externe Beschaltung und die Dimensionierung der Elemente ist im Datenblatt bereits vorgegeben. Die Grenzfrequenz des Anti-Aliasing-Filters beträgt rund $45kHz$. Die Wandlungsrate liegt bei 96kHz. Da der Wandler nach dem Sigma-Delta-Verfahren arbeitet und vom Noise-Shaping-Verfahren Gebrauch macht, ist die Grenzfrequenz des Filters ausreichend. Entsprechend einem 64-fach Oversampling[6] muss dieses Filter erst bei 3 MHz eine genügend hohe Dämpfung aufweisen. Ein RC-Filter erster Ordnung genügt diesen Anforderungen. Weiterhin ist noch ein Hochpass vor den AD-Umsetzer zu schalten, um Gleichspannungsanteile zu blockieren. Der Eingangspegel für Vollaussteuerung beträgt $0,506dbV$. Der zugehörige Schaltplan ist auf S. 62 zu finden.

[5] Nicht-professioneller Anwendungsbereich
[6] Die Abtastung geschieht mit einem Vielfachen der Wandlungsrate

Mikrofon

Durch die Vorgabe, die Luft als Übertragungsmedium zu verwenden, ergibt sich zwangsweise die Wandlung des Schalldrucks in ein elektrisches Signal mittels eines Mikrofons. Aus Kostengründen wurde hier eine Elektret-Mikrofonkapsel von Conrad Electronic gewählt, die die erforderliche Bandbreite besitzt, um die oberen Frequenzen des hörbaren Spektrums noch nutzen zu können. Ein Hersteller ist leider nicht angegeben, der Austausch gegen andere Kapseln sollte jedoch ohne Weiteres möglich sein. Die Schätzung der erwarteten Schallpegel und eine Umrechnung in Schalldruck, der laut Datenblatt [1] einen Spannungspegel von $-40dBV \pm 20dB$ erwarten lässt, gibt einen groben Anhaltspunkt für die Dimensionierung. Ausgehend von dieser Betrachtung und der enormen Dynamik des Signals ist ein regelbarer Vorverstärker unerläßlich.

Mikrofonvorverstärker

Aufgrund der schon genannten Dynamik ist eine Regelung der Verstärkung wünschenswert. Idealerweise erfolgt die Regelung automatisch. Da allerdings nur eine grobe Abschätzung der Pegel exisitiert, wird für den Prototyp der digital einstellbare Mikrofonvorverstärker PGA2500 von Texas Instruments verwendet, einer der wenigen digital einstellbaren Mikrofonvorverstärker, die eine $3,3V$-tolerante Schnittstelle besitzen. Der hohe Preis von 9,95$ (bei 1k) ist einer der entscheidenden Nachteile dieser Lösung. Die Regelung der Verstärkung kann bei dieser Lösung im FPGA erfolgen, die in einem weiten Bereich von 10 bis $65dB$ eingestellt oder auf $0dB$ festgelegt werden kann.

Pegelanpassung

Die erforderlichen Pegel an den Ein- und Ausgängen der verwendeten Komponenten und der Anschlüsse müssen aufeinander abgestimmt werden. Teilweise ist eine Verstärkung vorzunehmen, teilweise eine Abschwächung. Der Pegelverlauf entlang des Signalweges ist dem Pegelplan in Abb. 5.1 zu entnehmen. Der externe Standard-Pegel im Consumer-Audio-Bereich beträgt $-10dBV$, was einem Pegel von $3dBm$ bei einer Last von 50Ω entspricht.

Die zur aktiven Pegelanpassung verwendeten Operationsverstärker OPA2134 [11] sind ebenfalls von Texas Instruments und besitzen ausgezeichnete technische Daten. Speziell für den professionellen Audio-Bereich entwickelt, zeichnet sich das Bauteil durch sehr geringe Verzerrungen und niedriges Rauschen im Vergleich zu anderen Operationsverstärkern aus. Da der Standardtyp TL082 [6] Pinkompatibel zum OPA2134 ist, kann in einer späteren Version auf diesen gewechselt werden um die Kosten zu minimieren, falls die Störunempfindlichkeit des gesamten Entwurfs dies zulässt.

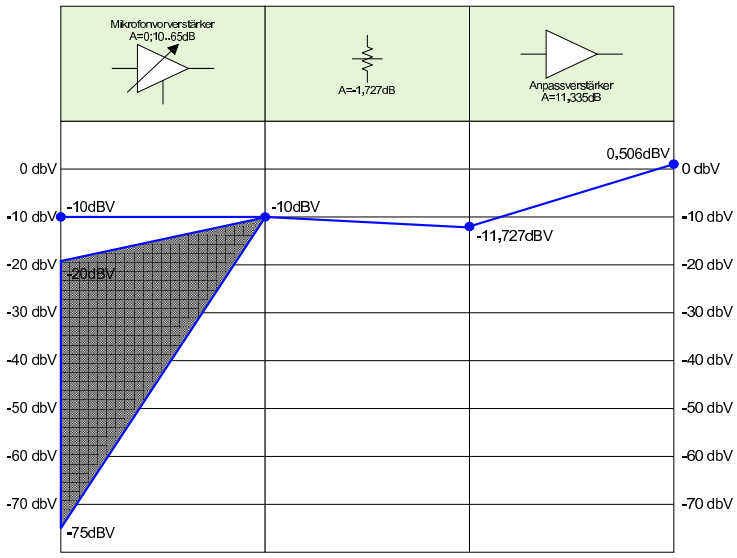


Abbildung 5.1: Pegelplan des Audio-Eingangs

Das Einstecken eines externen Mikrofons oder eines Gerätes an den Line-In entkoppelt den jeweils vorigen Schaltungsteil mittels eines in die Klinkenbuchse integrierten Schalters.

Spannungsversorgung

Die Versorgung des analogen Eingangs gestaltet sich wie gewohnt mittels einer symmetrischen Spannung von $\pm 5V$. Der AD-Umsetzer setzt diese Spannung voraus, die Operationsverstärker tolerieren diese Spannung mit Einschränkungen bei der maximalen Amplitude, die allerdings oberhalb des Aussteuerungsbereichs des AD-Umsetzers liegt. Sollten diese maximalen Pegel erreicht werden, befindet sich der AD-Umsetzer ohnehin in der Übersteuerung. Werden die genormten Pegel an den Eingängen eingehalten, sollte dieser Fall nie eintreten. Wegen der Widerstände und der begrenzenden Z-Dioden an den Eingängen, die von außerhalb des Systems kommen, sollte es nur schwer möglich sein, die Baugruppen zu schädigen.

5.4.3 Audio-Ausgang

Digital-Analog-Umsetzer

Der Audio-Ausgang beinhaltet die DA-Umsetzung und die Leistungs-Verstärkung zur Ansteuerung eines Lautsprechers (siehe S. 63). Die erforderlichen digitalen Pegel von $3,3V$ und die gewünschte Umsetzrate von $192kHz$ schränken die Auswahl der verfügbaren DA-Umsetzer erneut ein. Unter Beachtung finanzieller Aspekte bleiben einige Modelle von Texas Instruments übrig. Diese Auswahl findet sich in Tabelle 5.2.

Bezeichnung	**ADCs**	**N**	**SNR**	**Ausgänge**	f_r	**Interface**	**Preis (1k)**
PCM1730	2	24	117	2	192	I2S	5.00
PCM1739	2	24	106	2	192	I2S	3.70
PCM1793	2	24	113	2	192	I2S	2.10
PCM1798	2	24	123	2	192	I2S	6.50

Tabelle 5.2: Auswahl von geeigneten DA-Umsetzern

Nach einer Verfügbarkeitsprüfung wurde die Entscheidung zugunsten des PCM1730 [13] gefällt, der hinsichtlich des Preises einen guten Kompromiss darstellt. Durch Verwendung eines DA-Umsetzers von TI mit gleicher Schnittstelle kann eine einfache Überprüfung der Funktion erfolgen, indem der Signalweg im FPGA durchgeschleift und ein Takt zugeführt wird. Der Umsetzer besitzt zwei differentielle Stromausgänge, die eine hervorragende Störresistenz aufweisen.

I-U-Wandlung und Anti-Alias-Filterung Um die Stromsignale weiterverarbeiten zu können, müssen diese in eine proportionale Spannung umgesetzt und einer Anti-Alias-Filterung unterzogen werden. Dies geschieht pro Kanal mittels dreier Operationsverstärker. Je zwei pro Kanal für die Strom-Spannungs-Wandlung und einer für die Umsetzung des symmetrischen Signals in ein unsymmetrisches. Der OPA2134 bzw. der TL082 sind für diesen Zweck auch sehr gut geeignet. Das Anti-Alias-Filter ist in der Empfehlung des Datenblattes bereits integriert. Die Grenzfrequenz beträgt laut Datenblatt 45kHz. Da nur 8-fach Oversampling vorliegt, muss das Filter geringfügig bessere Steilheit der Dämpfung aufweisen als dies beim AD-Umsetzer nötig war. Um den Umsetzer zu verifizieren wurde eine Simulation in PSpice durchgeführt. Die Schaltung der Simulation findet sich in Abb. 5.2, der berechnete Frequenzgang in Abb. 5.3.

Leistungsverstärker

Um nicht auf Lautsprecher mit integriertem Verstärker und einem zusätzlichen Netzteil angewiesen zu sein, ist der Leistungsverstärker auf dem System integriert. Auch hier gibt es viele Alternativen. Sowohl ein Mono-Lautsprecher, als auch der Anschluss eines Stereo-Lautsprechers mittels 3,5mm-Klinkenstecker sollen möglich sein.

Ein aufgrund der Versorgungsspannung, des Preises, der einfachen externen Beschaltung und der Möglichkeit, sowohl einen Mono- als auch Stereo-Lautsprecher betreiben zu können, geeigneter Verstärker ist der TPA0213 [12] von Texas Instrument. Der Verstärker liefert bis zu $2W$ an 4Ω und kann mittels externer Beschaltung automatisch beim Einstecken eines Stereo-Lautsprechers in den Stereo-Betrieb umgeschatet werden.

Durch die gemeinsame Masse bei Stereo-Lautsprechern kann nur unsymmetrische An-

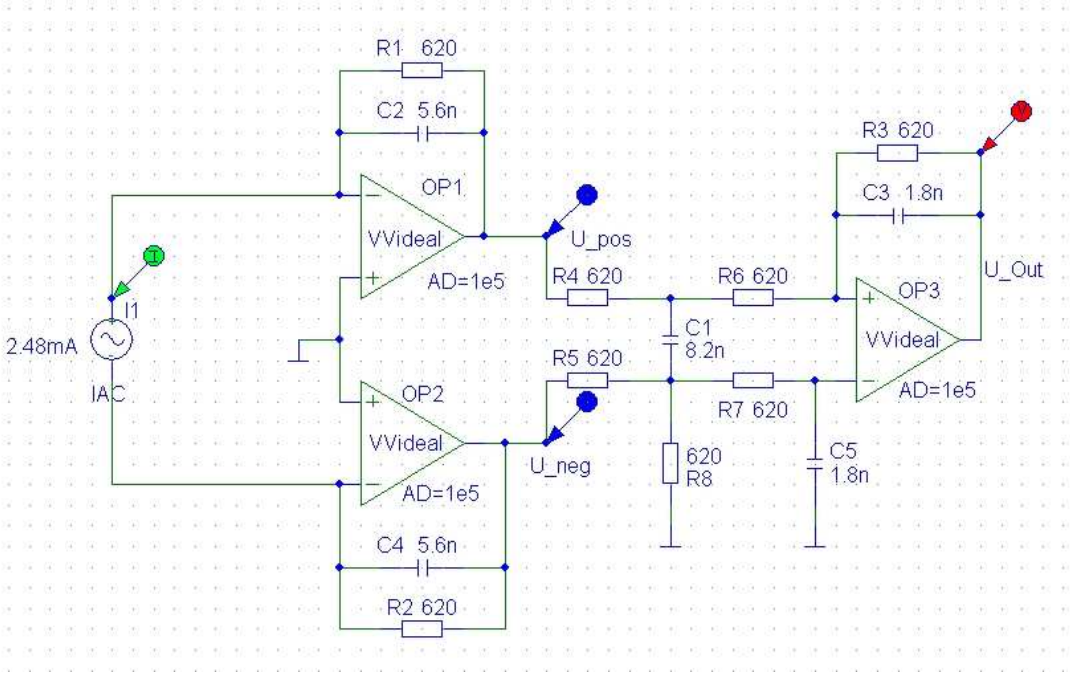


Abbildung 5.2: Schaltung des I/U-Wandlers mit Anit-Alias-Filter

steuerung erfolgen. Im Mono-Betrieb erfolgt gegenphasige Aussteuerung der beiden Lautsprecheranschlüsse. Damit kann die vierfache Leistung ($P \sim V^2$) eines Stereo-Kanals zur Verfügung gestellt werden.

Pegelanpassung

Der Ausgang des Strom-Spannungs-Wandlers liefert eine Ausgangsspannung von $6,2V_{SS}$, Consumer-Audio liegt bei $1V_{SS}$. Eine Pegelanpassung findet hierfür nicht statt, da der Leistungsverstärker nur eine sehr geringe Spannungsverstärkung ($2,5V/V$ Mono, $1,25V/V$ Stereo) besitzt und auf große Eingangspegel angewiesen ist. Falls Consumer-Pegel gewünscht sind, erfolgt dies durch Skalierung auf digitaler Seite. Eine Übersteuerung des Ausganges zu Versuchszwecken ist somit gezielt möglich. Durch Berechnung der erforderlichen Werte und Ersetzen der Widerstände und Kapazitäten im Strom-Spannungswandler kann eine Anpassung auf den Consumer-Pegel erfolgen. Ist sowohl der Consumer-Pegel, als auch die maximale Ausgangsleistung des Leistungsverstärkers gewünscht, so kann dies mittels zweier weiterer Operationsverstärker erfolgen.

Der Pegel für den Leistungsverstärker kann mittels eines Stereo-Potentiometers von Hand eingestellt werden. Dies ist sinnvoll, um die Lautstärke schnell den Bedürfnissen anpassen zu können.

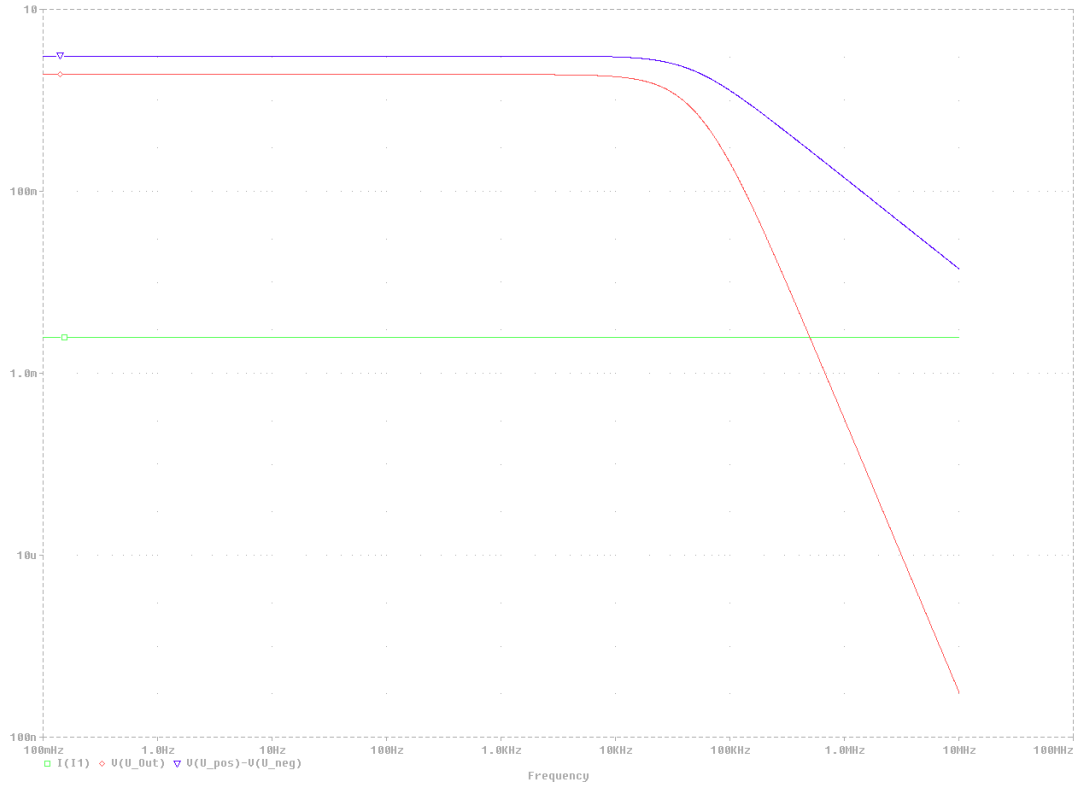


Abbildung 5.3: Frequenzgang des I/U-Wandlers mit Anit-Alias-Filter

5.4.4 Bedienelemente

Als Bedienelemente kommen Taster und Schalter zum Einsatz (vgl. S. 65). Um nicht unnötig Platz auf der Platine zu verschwenden und den Aufwand des Montierens zu minimieren, werden SMD-Bauteile verwendet. Als Taster finden Standard-SMD-Taster Verwendung. Weiter kommt ein 8-fach DIP-Schalter zum Einsatz. Die Bedienelemente schalten gegen Masse, haben also einen invertierenden Charakter.

Da Taster sehr stark zum Prellen neigen und so für viele hochfrequente Impulse und zusätzliche Störungen sorgen, wird den Tastern ein einfacher RC-Tiefpass mit $\tau = R \cdot C = 4,7ms$ nachgeschaltet.

Als Anzeigeelemente wurden zwei Pegel-, drei Status- und vier 7-Segment-Anzeigen gewählt. Die 7-Segment-Anzeigen dienen der einfachen Darstellung von Zahlen, die Statusanzeigen bestehen aus je einer grünen, gelben und roten LED zur Anzeige von Betriebszuständen durch den Nutzer, die Pegelanzeigen enthalten je eine rote, zwei gelbe und fünf grüne LEDs, auf denen die Audio-Pegel anschaulich dargestellt werden können.

5.4.5 PC-Schnittstelle

Die Anbindung an einen PC kann am Einfachsten durch die serielle Schnittstelle erfolgen. Diese kann im Verlauf des Praktikums als digitale Signalquelle für die Modulation verwendet werden, wenn die Übertragung in der Praxis getestet werden soll. Da die verwendete Schnittstelle des PC nach EIA-232, auch bekannt als RS232, Pegel von $\pm 5 - 15V$ verwendet, muss ein Pegelwandler eingesetzt werden. Hierfür gibt er eine unüberschaubare Menge an Derivaten von verschiedensten Herstellern, die ihrerseits eine Wandlung und die Erzeugung der benötigten Spannung aus einer einzelnen durchführen. Einer der bekanntesten Vertreter ist der MAX232 ($5V$-Versorgung). Ein weiterer Vertreter, der für eine Versorgung von $3,3V$ ausgelegt ist und mit kleineren Kondensatoren ($0,1\mu F$ statt $1\mu F$) zur Hilfsspannungserzeugung auskommt, ist der MAX3232 und baugleiche Typen anderer Hersteller. Auf Grund des Preises und der Verfügbarkeit wird der SP3232 [8] von Sipex verwendet. Die genaue Beschaltung kann dem Datenblatt entnommen werden und findet sich auch auf S. 66 wieder.

5.4.6 Erweiterungen

Um Signale aus dem FPGA nach Außen führen, sie mit dem Oszilloskop messen und später eine einfache Möglichkeit zu haben, andere Komponenten an das FPGA anbinden zu können, wurden zwei Erweiterungssteckplätze integriert. Diese sind jeweils an eine Bank angebunden und erlauben die Auswahl verschiedener Logikpegel. (vgl. S. 66)

5.4.7 Spannungsversorgung

Die Versorgung des Systems erfolgt durch die vorhandenen Labornetzgeräte, die zwei Spannungen von $15V$ mit je $1A$ liefern können. Die geschätzte Leistungsaufnahme der gesamten Schaltung, basierend auf der Summe aller Einzelleistungen und der Wirkungsgrade der Schaltregler, sollte 30W nicht übersteigen. Indem man die Ausgangsspannungen in Reihe schaltet, können die Netzgeräte die erforderliche Leistung bereit stellen. Daraus folgt die Auslegung der Eingangsspannung auf $35V$.

Nach der Festlegung der einzelnen Komponenten des Systems steht fest, welche Spannungen und Strombelastbarkeiten benötigt werden. Indem schon frühzeitig bedacht wurde, welche Spannungen unabdingbar sind, konnte der Aufwand für die Versorgung minimiert werden. Tabelle 5.3 gibt die Strombelastbarkeiten der benötigten Spannungsversorgungsebenen wieder. Die Werte beziehen sich auf die tatsächliche Belastbarkeit der Versorgungen. Zuvor wurde ermittelt, welche Werte sich im ungünstigsten Fall ergeben und danach die Komponenten ausgesucht. Unter Beachtung der Störempfindlichkeit der verschiedenen

Spannung	Verwendung	Belastbarkeit	Quelle
$+12V$	Vers. des $+1,2V$- und $+2,5V$-Reglers	2,7A	extern
$+3,3V$	Vers. der digitalen Peripherie	2,1A	extern
$+2,5V$	Für Erweiterungen (optional)	6A	$+12V$
$+1,2V$	VCore des FPGA	3A (6A)	$+12V$
$+5,2V$	Vers. der analogen Komponenten	1,7A	extern
$-7V$	Negative Hilfsspannung	300mA	$+5,2V$
$+5V_{MA}$	Versorgung des Mikrofon-Vorverstärkers	100mA	$+5,2V$
-5_{VD}	Neg. Digitalhilfssp. für Mikrofonvorverst.	100mA	$-7V$
$+5V_{AD}$	Positive Vers. des AD-Wandler-Teils	100mA	$+5,2V$
$-5V_{AD}$	Negative Vers. des AD-Wandler-Teils	100mA	$-7V$
$+5V_{DA}$	Positive Vers. des DA-Wandler-Teils	100mA	$+5,2V$
$-5V_{DA}$	Negative Vers. des DA-Wandler-Teils	100mA	$-7V$

Tabelle 5.3: Versorgungsspannungen

Schaltungsteile wurde eine Auswahl zwischen Schalt- und Linearregler getroffen. (vgl. S. 64)

Aufgrund positiver Erfahrungen mit den Schaltregler-Modulen PTH12000W [16] von Texas Instruments kam dieser für die Versorgung des FPGA-Kerns (VCore=$1,2V$) zum Einsatz. Die Eingangsspannung dieser Schaltregler beträgt $12V$. Die Erzeugung dieser obliegt dem nachfolgenden Konzept. Durch Beschaltung mit einem Widerstand kann die Ausgangsspannung in einem weiten Bereich eingestellt werden. Im Falle der $1,2V$ entfällt dieser Widerstand. Optional kann noch eine weitere Spannung ($2,5V$) zur Verfügung gestellt werden, allerdings wird das Modul vorerst nicht bestückt. Leider sind diese Schaltregler derzeit sehr begehrt und haben lange Lieferzeiten. Der Distributor Farnell konnte noch in genügender Anzahl liefern. Der Preis des Moduls in kleinen Stückzahlen ist mit etwa 12 EURO vergleichsweise teuer.

Die Spannungen $3,3V$, $5,2V$ und $12V$ werden durch eine eigene Lösung erzeugt. Zum Einsatz kommt der universelle Schaltregler TPS5430 [17] ebenfalls von Texas Instrument, da dieser preisgünstig ist, wenig externe Bauteile erfordert (integrierter MOSFET) und eine Software zur Verfügung gestellt wird, mit der man alle nötigen Dimensionierungen durchführen kann. Die Schaltfrequenz von 500kHz ist ein guter Kompromiss zwischen möglicher Störausstrahlung und Größe der induktiven und kapazitiven Elemente. Die Software erfordert die Eingabe der Eckdaten wie Eingangsspannungsbereich, maximale Belastung und erforderliche maximale Restwelligkeit. Es erfolgt die Ausgabe eines Schaltplanes mit allen Bauteilwerten, ein Referenzlayout und die simulierte Welligkeit der Ausgangsspannung im Zeitbereich. Entgegen der Empfehlung der Software kam die preisgünstige Schottky-Diode STPS340U [9] von STMicroelectronics zum Einsatz.

Die negative Versorgungsspannung, die für die analogen Komponenten nötig ist, wird

zweistufig erzeugt. In der ersten Stufe erfolgt die Invertierung der $5,2V$ zu $-7V$ mittels des Schaltreglers TSP6755 [10] ebenfalls von Texas Instruments. In der zweiten Stufe erfolgt die Regelung mittels linearer Standard-Regler 79L05 [7] mit 100mA Belastbarkeit, die für eine bessere Qualität der Spannung sorgen. Sowohl der analoge Eingang, als auch der analoge Ausgang erhalten eigene Linearregler für eine bessere Trennung der analogen Signale und eine ausreichende Versorgung. Ein Regler kann den erforderlichen Strom für beide Teile nicht zur Verfügung stellen. Der Mikrofonvorverstärker benötigt ebenfalls noch eine negative Hilfsspannung, die auch durch einen 79L05 zur Verfügung gestellt wird.

Die positive Versorgung der analogen Komponenten erfolgt ebenfalls getrennt nach Ein- und Ausgang. Der Leistungsverstärker erhält hingegen eine direkte Versorgung durch die $5,2V$, da dieser einen relativ großen Strombedarf hat und keine besonderen Ansprüche an die Qualität stellt. Die empfindlicheren Komponenten werden nochmals durch Linearregler mit besonders geringem Spannungsabfall von der $5,2V$-Versorgung entkoppelt. Dabei stehen zwei pinkompatible Schaltkreise zur Verfügung. Der TPS73250 [18] von Texas Instruments und der XC6201P502MRN [20] von Torex Semiconductor Ltd., deren Daten nahezu identisch sind.

Mit den Spannungsversorgungen sind alle Aufgaben des Systementwurfs auf Schaltungsebene abgeschlossen. Die übrigen Komponenten wie Steckverbinder und andere mechanische Komponenten sollen hier nicht ausgeführt werden.

5.5 Platzierung und Entflechtung

Aufgrund der Verwendung des genannten FPGAs, welches nur im **B**all **G**rid **A**rray (BGA) verfügbar ist, war klar, dass das Layout sehr aufwändig werden würde. Aus Kostengründen entschied man sich für eine Pltine mit zwei Kupferlagen. Nach Rücksprache mit ILFA Feinstleitertechnik und weiterer Unterredung mit dem Betreuer schied auch diese Alternative aus, die die Möglichkeit gefüllter Durchkontaktierungen bedeutet hätte. Diese können unter den Balls verwendet werden und erhöhen somit die Qualität der Verbindungen und reduzieren die Wahrscheinlichkeit von Kurzschlüssen deutlich. Die Wahl fiel zugunsten des günstigsten Leiterplattenherstellers PCB-Pool, der nur Standard-Durchkontaktierungen mit Bohrloch und relativ ungenaue Strukturen von $150\mu m$ anbietet. Beim Layout waren mehrere kritische Bereiche von Bedeutung. Die Platzierung der Bauteile erfolgte parallel zur Entflechtung, um ein möglichst effizientes Layout zu erreichen.

5.5.1 Spannungsversorgung

Die Spannungsversorgung des FPGA erforderte die intensivste Betrachtung. Aus dem Datenblatt kann der Betriebsspannungsbereich der Kern-Spannung entnommen werden. Diese sollte für einen zuverlässigen Betrieb im Bereich $1,14 - 1,26V$ liegen. Dabei ist jedoch der Spannungsabfall auf den Zuleitungen (VCore und Masse) nicht zu vernachlässigen, da erhebliche Ströme auftreten können. Ausgehend von einer geschätzten, maximalen Stromaufnahme von $3A$ bei $60mV$ erlaubtem Spannungsabfall darf der maximale Widerstand der Zuleitung $20m\Omega$ nicht überschreiten. Bei $35\mu m$ Kupferdicke und $1mm$ Leiterbahnbreite entspricht dies einer maximalen Länge von $40mm$, Übergangswiderstände der Anschlüsse nicht eingerechnet. Des weiteren sind hohe Stromspitzen weit über $3A$ zu erwarten, die obige Abschätzung nicht beinhaltet. Um diese abzufangen bedarf es einiger niederohmiger Kondensatoren, die diesen kurzzeitigen Strom liefern können. Diese sollten in umittelbarer Nähe der Versorgungsanschlüsse des FPGA liegen.

Die Versorgung aller weiteren Komponenten ist unkritisch und lediglich mit breiteren Leiterbahnen ausgeführt, um den Spannungsabfall gering zu halten.

Die Masse bedarf wiederum genauerer Betrachtung, da diese unmittelbaren Einfluss auf die Störemfindlichkeit und die Störaussendung hat. Um Störungen von den analogen Signalleitungen fern zu halten, ist eine möglichst flächendeckende Verteilung der Masse notwendig, die Störungen einfängt und abführt. Im digitalen Teil ist aufgrund der hohen Stromspitzen und der Verringerung der Störaussendung die Verwendung einer Masseflächе empfehlenswert. Aufgrund dieser gegensätzlichen Anforderungen wird die Masse von analogem und digitalem Teil strikt getrennt und erst im Bereich der Systemspannungsversorgung zusammengeführt. Dazu ist die Leiterplatte in verschiedene Bereiche aufgeteilt. Da eine zweiseitige Leiterplatte Verwendung findet, konnte keine eigene Lage für die Massen verwendet werden. Daher ist jede freie Fläche, die zugänglich und groß genug für eine Durchkontaktierung ist, mit der entsprechenden Masse verbunden. Signale wurden sofern möglich auf der Oberseite geführt, um eine durchgängige Masse zumindest auf der Rückseite zu erreichen. Die gesamten Masseflächen wurden an ihren Rändern mit Durchkontaktierungen versehen, um einen geringen Widerstand zu garantieren.

Das Layout im Bereich der Schaltregler entspricht den Empfehlungen der Datenblätter.

5.5.2 Analoge Signale

Das Layout der analogen Signale folgt den Regeln möglichst hoher Störresistenz. Dies bedeutet kurze Signalwege und ausreichenden Abstand zu digitalen Schaltungsteilen. Signale mit sehr niedrigem Pegel (Mikrofon) sind möglichst separat geführt und mit möglichst niederohmiger Masse umgeben.

5.5.3 Digitale Signale

Die digitalen Signale bedürfen bezüglich des Layouts nur weniger Überlegungen. Für Signale mit Informationscharakter wurde die kleinste Leiterbahnbreite gewählt. Taktleitungen sind nach Möglichkeit mit Masse umgeben, um deren Störabstrahlung in den analogen Bereich zu minimieren. Signale deren Anschluss am FPGA lediglich einer Bank zugeordnet sind und deren Belegung innerhalb der Bank frei wählbar ist, wurden auf direktem Weg an die Bank herangeführt, im Schaltplan den günstigsten Anschlüssen zugeordnet und anschließend im Layout fertiggestellt. Ohne diesen Freiheitsgrad wäre eine Entflechtung auf zwei Ebenen unmöglich gewesen.

Nach Fertigstellung des Layouts und der Festlegung der einzelnen Komponenten konnte mit der Programmierung in VHDL begonnen werden. Das Ergebnis der Platzierung und Verdrahtung is in Abb. 5.4 zu sehen.

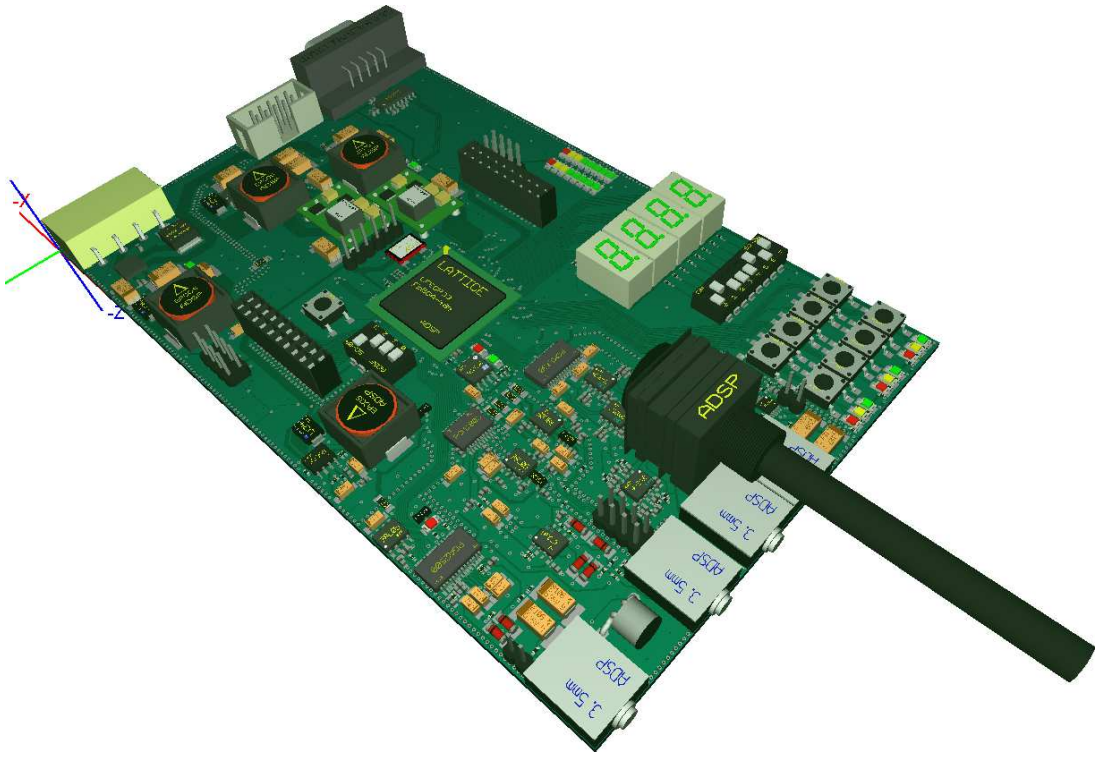


Abbildung 5.4: 3D-Ansicht des Modells aus Target3001!

6 Praktikumskonzept

Da eine komplette Implementierung aller benötigten Funktionen durch die Praktikumsteilnehmer aufgrund der kurzen Praktikumsdauer von einer Woche ausscheidet, wurde nach einer effizienten Möglichkeit gesucht, die Kenntnisse anhand praktischer Versuche zu vermitteln. Das Ziel des Praktikums stellt ein Übertragungssystem dar, auf das Schritt für Schritt hingearbeitet wird. Die einzelnen Versuche unterscheiden sich stark in der Implementierung und sorgen für Abwechslung durch unterschiedliche Probleme, die gelöst werden müssen. Trotz dieser Aufteilung ist jeder Versuch separat mit der Hardware testbar und nicht nur auf die Simulation begrenzt.

6.1 Aufbau des Praktikums

Um die Lernphase zu verkürzen und schnell zu ersten praktischen Ergebnissen zu kommen, wird im ersten Versuch fast der komplette VHDL-Code vorgegeben und hauptsächlich auf die Bedienung der Software eingegangen. Lediglich ein kurzer Teil geht auf wenige Eigenschaften der Sprache VHDL ein, der gerade ausreicht, um den Aufbau eines VHDL-Moduls zu verstehen. Trotz der Einfachheit ist der Abschluss des ersten Versuchs eine funktionsfähige Hardware.

Durch den hoffentlich schnellen Erfolg des ersten Versuchs bleibt die Motivation erhalten, die bei der vorbereitenden Durcharbeitung eines Tutorials schnell verloren ginge. Die MATLAB-Teile der einzelnen Versuche des Empfängers stehen den VHDL-Teilen direkt voran. Unmittelbar bevor die noch unbekannten Konstrukte Verwendung finden, werden diese in den „VHDL-Basics“ genannten Abschnitten bekannt gemacht und erklärt. Gegebenenfalls wird noch auf Besonderheiten und häufige Fehlerquellen hingewiesen. Durch die kurze zeitliche Differenz von Lerninhalt und Anwendung sollte ein Nachschlagen bei der Programmierung unnötig sein und sich durch die Anwendung selbst besser einprägen.

Die Notwendigkeit, dem Teilnehmer zusätzliche Hilfestellungen zu geben, entfällt zum Großteil auch, da die Nutzung der im gleichen Kapitel erlernten Inhalte dem Teilnehmer logisch erscheint.

In den weiteren Versuchen steigt der Programmieraufwand der Teilnehmer über die Implementierung eines kompletten Moduls bis hin zur selbstständigen Partitionierung in

Teilmodule und deren komplette Programmierung.

Um auch die Unterschiede der verschiedenen Konzepte herauszustellen, wird am Beispiel des Signalgenerators die Variante der direkten digitalen Synthese vorgegeben und den Teilnehmern die Implementierung des eleganteren, jedoch schwieriger zu implementierenden Algorithmus überlassen.

Gegen Ende der Versuche wird es aufgrund der wachsenden Anzahl der zur Verfügung stehenden Module in Bezug auf die Konzepte und deren Umsetzung wieder einfacher. Allerdings wird das Gesamtsystem komplexer und die Schwierigkeit der Verschaltung der einzelnen Module untereinander drängt in den Vordergrund. Hier kommt eine unangenehme Eigenschaft von VHDL zum Tragen, die bei großen Projekten dazu führt, dass der Deklarationsteil und die Instanziierung der Komponenten im Verhältnis zum eigentlichen Code immer mehr überwiegen.

6.2 Konzeptauswahl und Reihenfolge

Aus den Vorgaben zum Gesamtsystem ergeben sich viele der zu implementierenden Konzepte. Der vorangegangene Abschnitt erfordert gründliche Überlegungen bezüglich der Reihenfolge. Hier bietet es sich an, mit einfachen Grundkonzepten zu beginnen, die im weiteren Verlauf häufig benötigt werden. Die hierfür gewählten Versuche sind der Multiplexer und ein Pseudo-Zufallsfolgen-Generator, der als Quelle digitaler Daten bei der Simulation eingesetzt werden kann. Diese Versuche dienen hauptsächlich dazu, den Vorgaben im Hinblick auf das Vorwissen der Teilnehmer Sorge zu tragen.

6.2.1 Sender

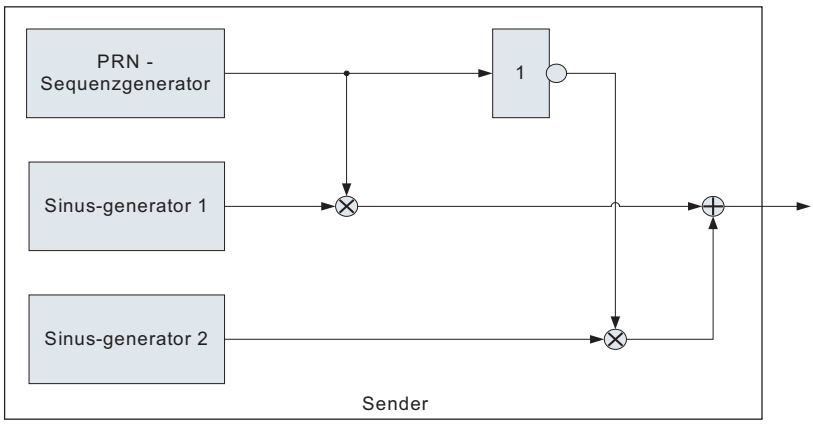


Abbildung 6.1: Blockschaltbild des Senders

Es erscheint auch sinnvoll, mit der Signalquelle der Übertragungsstrecke zu beginnen, da von ihr die Verifikation des Empfängers abhängt. Der Sender selbst (vgl. Abb. 6.1 besteht aus einfachen Modulen, die sich gut dazu eignen, in die Sprache VHDL einzuführen, die Umsetzung der Code-Konstrukte in Hardware zu erläutern und Unterschiede zur Simulation in MATLAB aufzuzeigen. Der Sender selbst besteht aus den Teilen Signalgenerator, Pegelanzeige und Modulator.

Der Signalgenerator behandelt den CORDIC-Algorithmus, einen der elegantesten Algorithmen der digitalen Signalverarbeitung, dessen Behandlung in diesem Praktikum praktisch unumgänglich ist. Durch die Vereinfachung, die sich durch den Einsatz als Signalgenerator ergeben, entfallen viele problematische Teile. Indem man keine Einschränkungen bezüglich des Hardwareaufwandes macht und die Pipeline-Architektur zulässt, entfällt auch das, aufgrund der variablen Schiebeweite, schwer zu implementierende Schieberegister. Will der Teilnehmer den CORDIC in Bit-Parallel-Architektur aufbauen, muss er hierzu eine Lösung finden. Da die Simulation in MATLAB nur einen begrenzten Zeitabschnitt betrachtet, wirken sich Fehler, die sich aufsummieren, nur sehr wenig aus. Bei der derzeitigen Implementierung des Oszillators fällt auf, dass der Ausgangspunkt der folgenden Drehung immer das Ergebnis der letzten ist. Was bei Verwendung von Fließkommazahlen noch über sehr viele Perioden funktioniert, endet bei Verwendung von Fixed-Point Arithmetik bereits nach wenigen Zyklen, je nach Rundungsverfahren, entweder in einem Überlauf oder in einer verschwindend kleinen Amplitude. In Hardware ist daher, ausgehend von einem festen Verktor, der Winkel mit jedem Schritt zu vergrößern.

Die Pegelanzeige dient dem Zweck, die Verifikation zu erleichtern. Das lautstarke Testen kann damit meist entfallen. Anhand dieses einfachen Beispiels wird die Multiplikation und deren Verwendung eingeführt.

Der Modulator gestaltet sich sehr einfach und kann in mehreren Varianten implementiert werden. Eine Variante der Multiplikation des aktiven Signals mit 1, des inaktivn mit 0 bzw. eines Multiplexers besitzt den Vorteil der einfachen Implementierung in VHDL. Die zweite Möglichkeit der Umschaltung des Phaseninkrements erfordert mehr Programmieraufwand, reduziert die Hardware jedoch erheblich. Durch die Implementierung beider Varianten durch die Teilnehmer soll aufgezeigt werden, dass einfache Programmierung nicht immer die beste Umsetzung in Hardware bedeutet und immer überlegt werden muss, ob es möglich ist, durch mehrfache Verwendung oder geschickte Programmierung Hardware einzusparen.

6.2.2 Empfänger

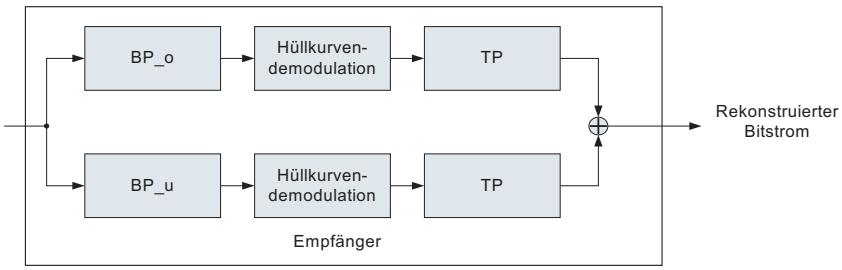


Abbildung 6.2: Blockschaltbild des Empfängers

Nachdem der Signalgenerator fertig gestellt ist und als Quelle für Simulationen dienen kann, erfolgt die Implementierung des Empfängers (vgl. Abb. 6.2). Hierzu ist eine Einführung in die Grundlagen der digitalen Filter notwendig (abgedeckt von der Studienarbeit von Andreas Schedel). Der nachfolgende MATLAB-Teil behandelt das komplette Empfängerdesign.

Im Empfänger finden hauptsächlich verschiedene Filter Einsatz, deren Koeffizienten im MATLAB-Teil bereits berechnet wurden. In einigen Fällen ist es notwendig diese nochmals zu ermitteln, um bestimmte Eigenschaften der verwendeten Hardware, beispielsweise der 4-fach Multiplizierer, zu nutzen. Die Konzepte, die zur Anwendung kommen sind Bandpassfilter, Filteroptimierung, Hüllkurvendemodulation, Tiefpassfilterung und Signalregeneration.

Das Filter besteht im ersten Anlauf aus einem direkt in Hardware umgesetzen FIR-Filter in Linear-Phasen-Struktur. Der Aufwand für das Filter ist jedoch zu hoch, um die übrigen Module implementieren zu können.

Die Filteroptimierung durch Mehrfachnutzung der Hardware beseitigt das vorhergehende Problem und besitzt großes Einsparpotenzial, erfordert jedoch einen erheblichen Aufwand in der Programmierung. Diese gestaltet sich wegen der komplexen Kontrollstruktur sehr fehleranfällig.

Die Hüllkurvendemodulation ist lediglich eine andere Anwendung der Multiplikation, wie sie schon bei der Pegelanzeige verwendet wurde, ist an dieser Stelle aber für den Empfänger notwendig.

Die Tiefpassfilterung erfolgt hier erneut durch einen FIR-Filter, sollte aber wegen der niedrigen Grenzfrequenz durch Abtastratenumsetzung erfolgen. Andererseit hat die

Filteroptimierung das Potenzial, selbst Filter mit Ordnungen größer 500 mit einem DSP-Block zu realisieren. Würde man die RAM-Blöcke noch geschickt verwenden, wären kaum Logik-Ressourcen notwendig. Dies würde aber im Rahmen dieses Praktikums deutlich zu weit führen.

6.2.3 Gesamtsystem

Nachdem nun auch die Module des Empfängers komplett umgesetzt wurden, erfolgt die Erprobung des Gesamtsystem. Dabei wird in zwei Schritten vorgegangen, der direkten elektrischen Verbindung und der realen Übertragung über die Luft.

Die elektrische Verbindung ermöglicht die Kopplung von Sender und Empfänger (Loop) fast ohne Störeinflüsse und ermöglicht die Verifikation der Algorithmen. Als Informationsquelle für die Modulation kommt die serielle Schnittstelle zum Einsatz, da sie eine einfache Überprüfung der Korrektheit bei höheren Datenraten (110 Baud) erlaubt. In dieser Phase sollten noch alle Zeichen korrekt übertragen werden.

Die reale Übertragung in Form einer Unterhaltung mittels Terminal (Chat) schließt alle störenden Einflüsse wie Mehrwegeausbreitung, Störer und Dämpfung mit ein. Damit kann die Robustheit der Algorithmen erprobt werden. Die Fehler in der Übertragung werden direkt am Bildschirm sichtbar. Ein Test aller Gruppen zur gleichen Zeit soll die Grenzen des Modulationsverfahrens und der verwendeten Algorithmen aufzeigen.

6.3 Abstraktion der Schnittstellen

Die Anbindung einiger wichtiger Komponenten geschieht durch serielle Schnittstellen, der Informationsfluss in den Algorithmen ist jedoch parallel. Um die Teilnehmer nicht unnötig zu belasten, wurde eine Abstraktionsebene eingeführt, die diese Teile verbirgt. Auch Teile die wenig Lernerfolg versprechen und für die Anbindung notwendig sind, beispielsweise die Entprellung der Schalter, werden vorgegeben. Durch eine zusätzliche Abstraktionsebene (vgl. Abb. 6.3) mittels des toplevels, der dazwischen liegenden Module und des stud_toplevels wird erreicht, dass den Teilnehmern ein unberührter toplevel zur Verfügung steht. Ein angenehmer Nebeneffekt der Abstraktion besteht in der einfachen Anpassung an Hardwareänderungen.

Die in die Abstraktion einfließenden Module umfassen die im Folgenden erläuterten Bestandteile.

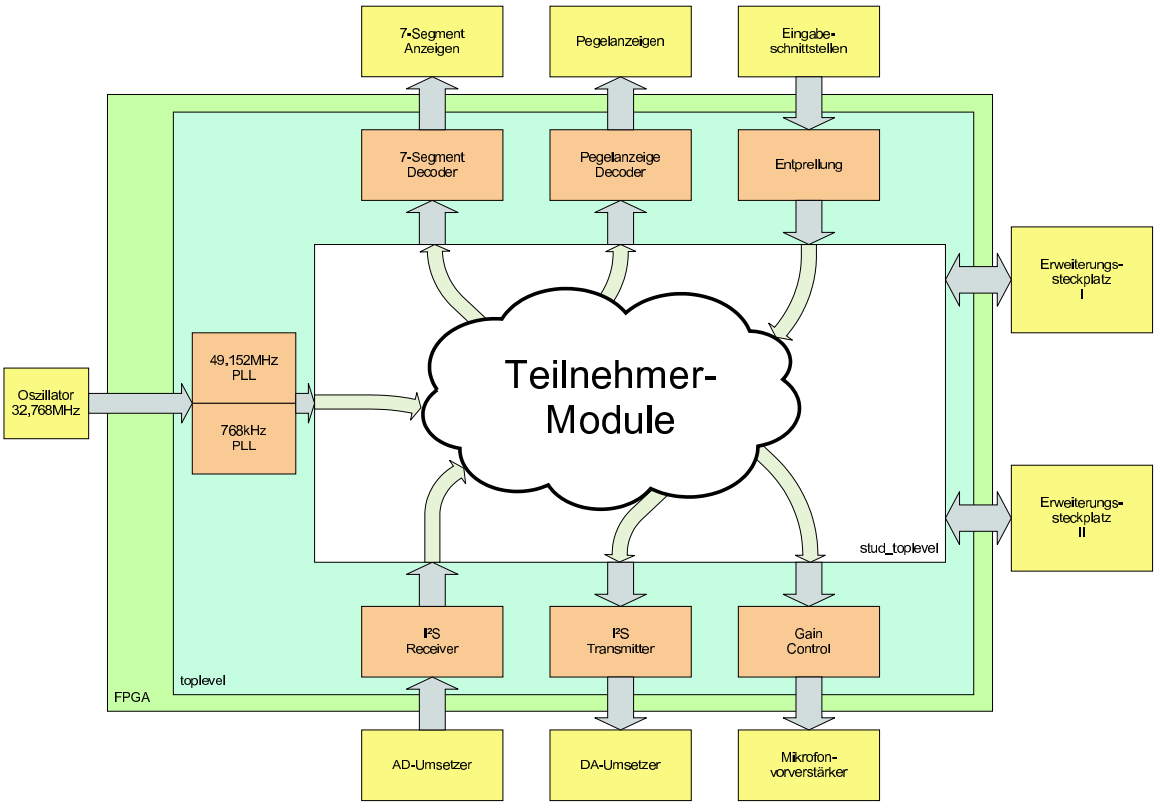


Abbildung 6.3: Abstraktion der Hardware

6.3.1 Taktversorgung

Die verschiedenen Takte im FPGA werden von zwei Modulen erzeugt, die mittels IP[1]Express[2] generiert wurden. Sie stellen einen langsamen ($768kHz$) und einen schnellen ($49,152MHz$) Takt zur Verfügung. Der langsame Takt kann der Verarbeitung der Audiodaten dienen, hauptsächlich bei einfacher Nutzung von Hardware. Der schnelle Takt wird für den Betrieb des AD- und DA-Wandlers benötigt, eignet sich aber auch gut bei Mehrfachnutzung von Hardware. Als Zwischenstufe jeder der beiden PLLs wird ein Takt mit $98,304MHz$ generiert, der grundsätzlich auch verwendet werden könnte.

6.3.2 I2S-Interface

Sowohl der AD- (vgl. Abb. 6.4), als auch DA-Umsetzer (vgl. Abb. 6.5) besitzen ein serielles I^2S-Interface. Hierfür stehen zwei getrennte Module zur Verfügung. Der Receiver implementiert den Slave-Modus und empfängt die Daten des AD-Umsetzers. Der Trans-

[1]**IP**: Intellectual Property (dt. Geistiges Eigentum) bezeichnet vorgefertigte Module, teilweise von Drittanbietern und evtl. kostenpflichtig

[2]**IPExpress**: Mittels dieser in ispLEVER integrierten Software können diese IP-Module konfiguriert werden

mitter schickt die Daten im Master-Modus zum DA-Umsetzer. Das verwendete Protokoll und die zugehörigen Einstellungen mittels der Konfigurationspins folgen aus den Datenblättern der Schaltkreise.

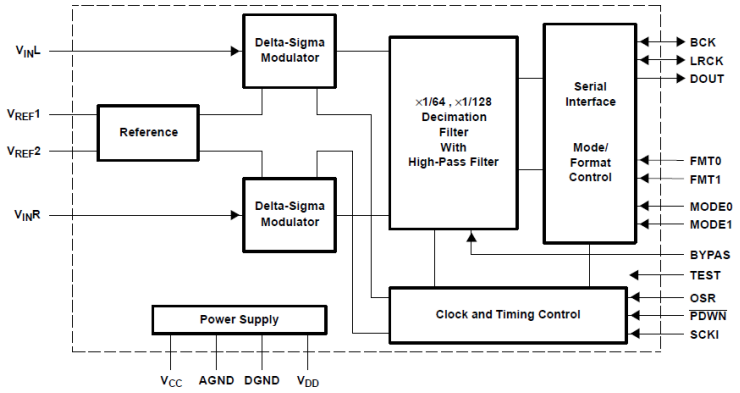


Abbildung 6.4: Blockschaltbild des AD-Umsetzers PCM1803

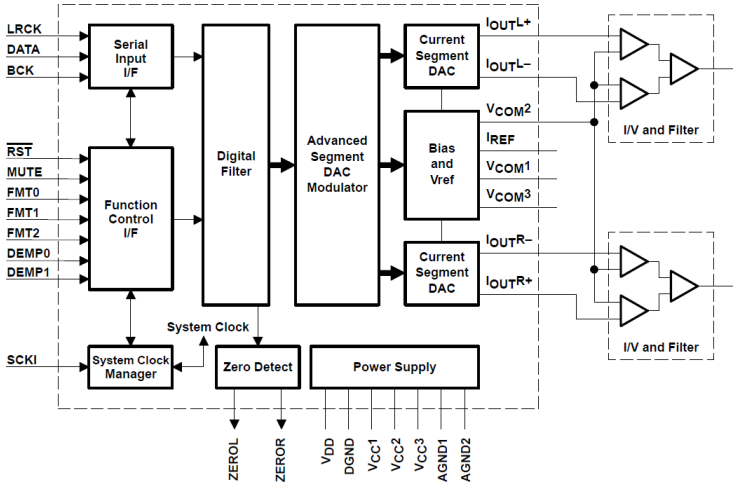


Abbildung 6.5: Blockschaltbild des DA-Umsetzers PCM1730

6.3.3 Mikrofon Vorverstärkung

Im Mikrofonvorverstärker ist ein synchrones, serielles Protokoll zur Einstellung der Parameter implementiert (vgl. Abb. 6.6. Das VHDL-Modul besitzt eine parallele Schnittstelle als Eingang, über welche die Verstärkung und einige weitere Parameter kommuniziert werden. Bei Änderungen an der FPGA-internen, parallelen Schnittstelle werden die neuen Einstellungen an den Vorverstärker übertragen und erlangen beim nächsten Nulldurchgang des analogen Signals Gültigkeit. Gegebenenfalls kann hier noch ein Modul programmiert werden, das die Verstärkung automatisch regelt.

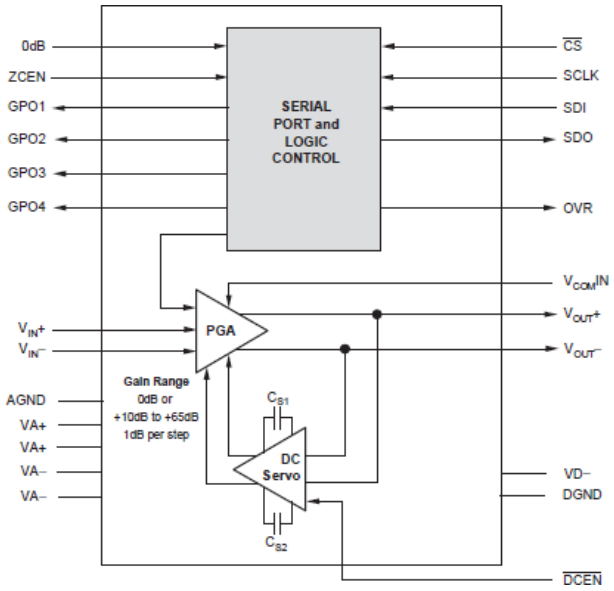


Abbildung 6.6: Blockschaltbild des PGA2500

6.3.4 Anzeigendecoder

Die einzelnen 7-Segment-Anzeigen des Systems stellen sich dem Praktikumsteilnehmer als 4-Bit-Schnittstelle dar. In dem Abstraktionsmodul werden dann die 4-Bit-Daten in eine hexadezimale Darstellung umgewandelt.

6.3.5 Entprellung

Die Entprellung der Schalter und eine zusätzliche Entprellung der Taster erfolgt durch einen Algorithmus, der kurzzeitige Pegelwechsel unterdrückt. Die Anzahl der Takte, nachdem eine Änderung als statisch angenommen wird, ist einstellbar.

6.3.6 Pegelanzeige

Die Pegelanzeige kann auf Wunsch auf einen Modus umgeschaltet werden, bei dem lediglich zwei LEDs leuchten. Dies dient hauptsächlich der geringeren Stromaufnahme, kann aber auch der besseren Ablesbarkeit der Anzeige dienen.

7 VHDL-Programmierung

In den folgenden Abschnitten wird systhematisch die Verifikation der VHDL-Module aus B.1 und B.3 ab S. 69 vorgenommen. Im ersten Schritt erfolgt die Beschreibung der notwenidgen Stimuli, um die Module zu testen. Teilweise werden Takt und Initialisierungssignale nicht speziell genannt, da sie meist selbstverständlich sind.

Anschließend wird diskutiert, welche Werte zu erwarten sind. Schließlich erfolgt noch eine kurze analytiche Betrachtung des Simulationswaveforms, wie es von Modelsim generiert wurde. Auf die Einstellungen von Modelsim (z.B. Simulationsdauer) soll nicht genauer eingegangen werden, da dies im Normalfall aus dem Waveform bzw. aus dem Quellcode B.2 S. 119 hervorgeht. In den Abbildungen gilt folgende Vereinbarung für die Farben der einzelnen Signalverläufe:

rot Systembedingte Stimuli (synchrone Schaltung)

blau Eingangsgrößen/-signale

gold-orange Ausgangsgrößen/-signale

grün Hilfssignale

Die Hilfssignale dienen nur der besseren Lesbarkeit der Ergebnisse und werden zumeist nur für die Simulation generiert. Manchmal handelt es sich allerdings auch um interne Zustände des Moduls. Die Funktion der einzelnen Signale kann aus dem Namen im linken Teil des Waveforms abgeleitet werden. Auf diese Namen wird auch in der Analyse Bezug genommen.

7.1 Bereitgestellte Module

7.1.1 Bargraph Decoder

Der Bargraph Decoder dient der einfachen Ansteuerung der Pegelanzeigen. Um Strom zu sparen, kann die Pegelanzeige so konfiguriert werden, dass jeweils nur die obersten Anzeigeelemente leuchten. Diese relativ einfache Schaltung wurde allerdings so optimiert,

dass nur wenige Hardwareressourcen benötigt werden. Auch ein einfaches Abschalten der Anzeige ist damit möglich.

Stimuli

Als Stimuli kommen verschiedene Eingangssignale zum Einsatz, die eine möglichst gute Testabdeckung erreichen sollen. Getestet werden mit der Testbench speziell die Fälle, in denen Fehler zu erwarten sind:

- Kein Bit gesetzt
- Ein Bit gesetzt
- Mehere Bits gesetzt

Weitere denkbare Fälle bedürfen nicht der Beachtung.

Erwartungswerte

Das höchstwertige gesetzte Bit und ein weiteres (`BAR_LIGHT_COUNT=2`) sollen am Ausgang erscheinen. Alle weiteren Bits sollen inaktiv sein.

Simulationsergebnisse

In Abb. 7.1 is das Ergebnis der Simulation zu sehen. Die Verzögerung von einigen Takten ergibt sich aus der VHDL-Beschreibung und ist beabsichtigt.

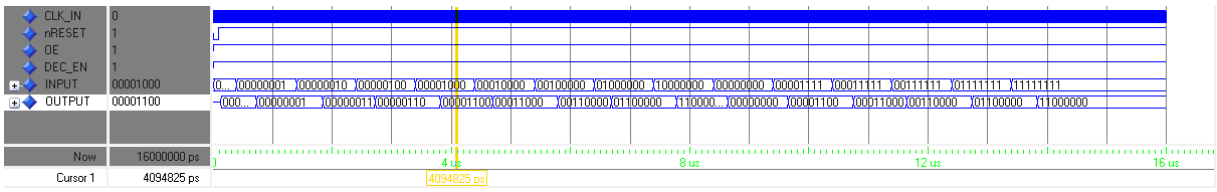


Abbildung 7.1: Simulationswaveform des Bargraph Decoders

7.1.2 Barrel Shifter

Diese Schaltung ermöglicht es, innerhalb eines Taktes ein Wort um mehrere Schritte nach links oder rechts zu verschieben. Mathematisch handelt es sich dabei um eine Multiplikation oder eine Division mit einer Zweierpotenz.

Stimuli

Als Stimuli werden ein Datenwort `INPUT` und die Anzahl der zu verschiebenden Stellen `SHIFT_AMOUNT` verwendet. Das `SIGNED_SHIFT`-Eingangssignal dient der Aktivierung der Sign-Extension.

Erwartungswerte

Zu erwarten ist eine Links- bzw. Rechtsverschiebung des Eingangssignals.

Simulationsergebnisse

In Abb. 7.2 ist zu erkennen, dass die Verschiebung nach links mit positiven und die Verschiebung nach recht mit negativen Werten des Eingangssignals `SHIFT_AMOUNT` korrekt abläuft. In Abb. 7.3 ist der Verlauf mir aktivierter Sign-Extension gezeigt.

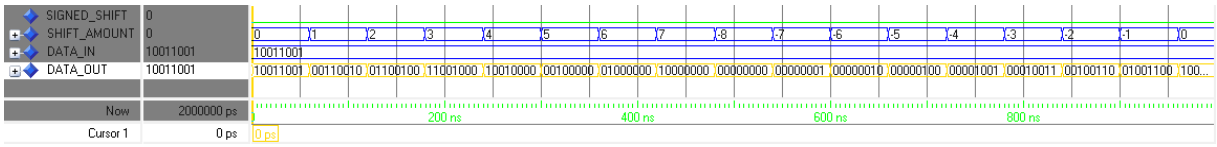


Abbildung 7.2: Simulationswaveform des Barrel Shifters

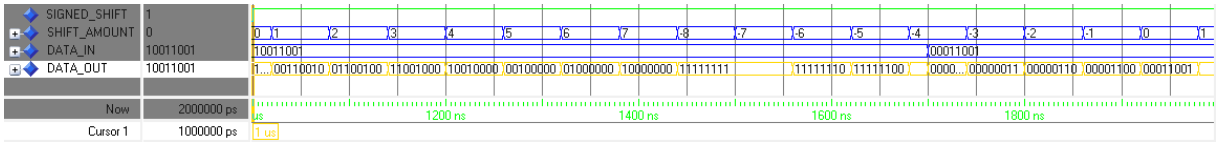


Abbildung 7.3: Simulationswaveform des Barrel Shifters mit aktivierter Sign-Extension

7.1.3 Tasterentprellung

Die Tasterentprellung dient der Entprellung der Schalter und Taster, die beim Öffnen und Schließen aufgrund der federnden Wirkung der Kontakte ein sogenanntes Prellen verursachen (s. Abb. 7.4). Zwar übernimmt dies zum Teil schon ein RC-Tiefpass, es kann aber nicht davon ausgegangen werden, dass dieser korrekt funktioniert, da die Eingänge des FPGA keine Hysterese besitzen.

Stimuli

Als Stimuli kommen Signale zum Einsatz, die möglichst gut einen prellenden Taster darstellen. Da nur digitale Signale zur Verfügung stehen, werden kurze Impulse generiert,

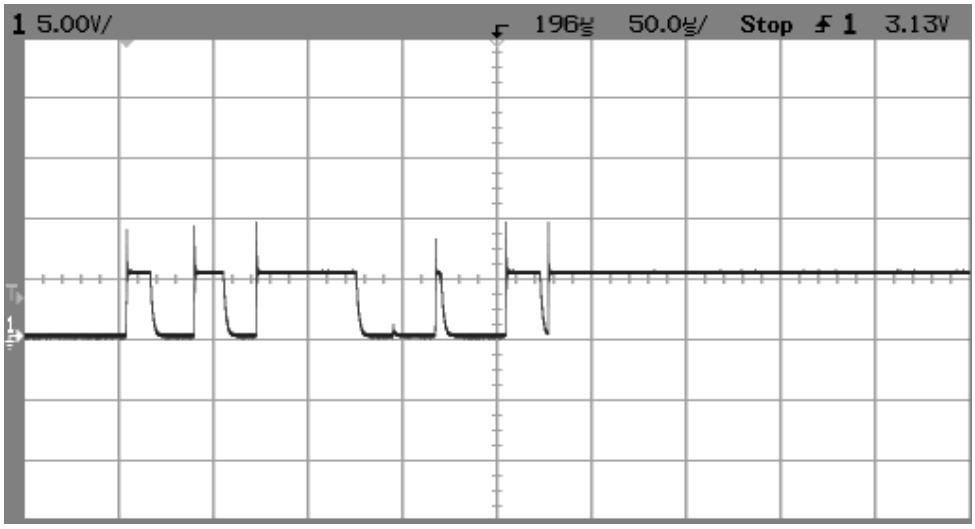


Abbildung 7.4: Oszillogramm eines prellenden Tasters (Quelle: Wikipedia)

bevor der Signalpegel in den stationären Zustand übergeht. Impulse kürzer als die Taktperiode sind in der Simulation nicht sinnvoll, da das Eingangssignal abgetastet wird.

Erwartungswerte

Das Ausgangssignal soll dem Eingangssignal folgen, wenn dieses einen stabilen Zustand über einen einstellbaren Zeitraum eingenommen hat.

Simulationsergebnisse

In Abb. 7.5 ist gut zu erkennen, dass das Ausgangssignal erst nach der eingestellten Zeit von 8 Taktperioden dem Eingangssignal folgt. Kommt es innerhalb dieser 8 Takperdioden zu einer Änderung des Eingangssignals (300 bis 400 ns), so wird diese Totzeit zurück gesetzt und beginnt erneut abzulaufen.

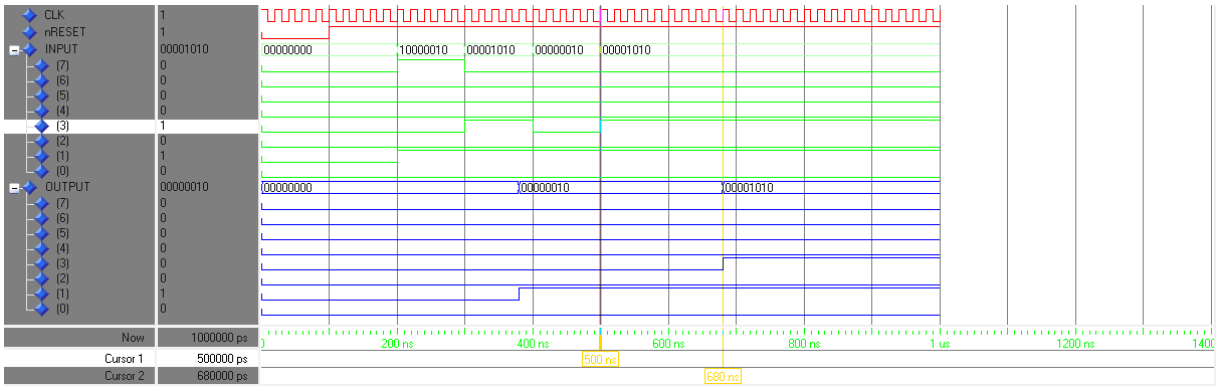


Abbildung 7.5: Simulationswaveform der Tasterentprellung

7.1.4 DDS-Signalgenerator

Der DDS-Signalgenerator erzeugt mit Hilfe einer Look-Up-Table eine angenäherte Sinusfunktion.

Stimuli

Als Stimuli kommt lediglich ein Taktsignal und das gewünschte Phaseninkrement zur Anwendung. Ein Phaseninkrement entspricht in der Standrardeinstellung etwa $200Hz$. In der Simulation wird ein Phaseninkrement von 100 und 50 verwendet.

Erwartungswerte

Das Ausgangssignal soll mögichst einer Sinusfunktion mit der gewünschten Frequenz, die abhängig vom Phaseninkrement ist, entsprechen. Es sollten sich Sinusschwingungen mit etwa $20kHz$ und $10kHz$ zeigen.

Simulationsergebnisse

Im Bereich von 0 bis 1 ms weist das Signal in Waveform 7.6 die erwartete Frequenz von $20kHz$ auf. Nach dem Zeitpunkt 1 ms die erwarteten $10kHz$.

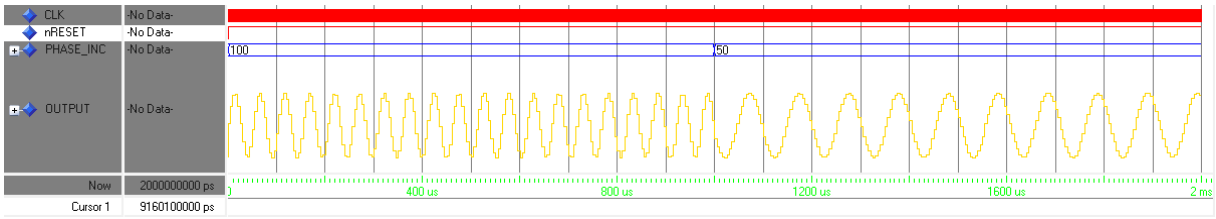


Abbildung 7.6: Simulationswaveform des DDS-Sinus-Signalgenerators

7.1.5 Mikrofonvorverstärkungseinstellung

Stimuli

Die Eingangssignale sind die Verstärkung und einige weitere Einstellungen. Nach kurzer Zeit wird ein Eingangssignal geändert, um zu überprüfen, ob bei neuen Eingangswerten eine erneute Übertragung stattfindet.

Erwartungswerte

Bei einer Änderung der Eingangswerte sollen die Eingangszustände als Einstellungen seriell an den Mikrofonvorverstärker übertragen werden. Die genauen Spezifikationen finden

sich in [14], Seite 10, Abb. 5.

Simulationsergebnisse

Wie erwartet zeigt ich in Abb. 7.7, dass die Einstellungen seriell in der richtigen Reihenfolge übertragen werden. Bei Änderung eines Eingangssignals erfolgt die erneute Übertragung.

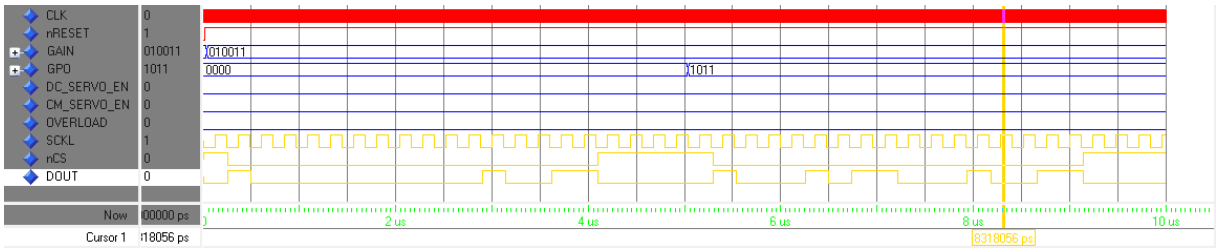


Abbildung 7.7: Simulationswaveform der Mikrofonvorverstärkungseinstellung

7.1.6 I2S Empfänger

Stimuli

Als Eingangssignal wird ein zufällig gewählter Wert jeweils für den linken und den rechten Audio-Kanal in einen seriellen Bitstrom gewandelt, der den Spezifikationen in [15], Seite 13, Format 1 entspricht. Die Signale `LEFT_IN` und `RIGHT_IN` stellen die zu übertragenden Daten dar und sind nur in der Testbench vorhanden.

Erwartungswerte

Nach erfolgter serieller Übertragung sollte das empfangene Datenwort parallel an dem entsprechenden Ausgang `DOUTL` bzw. `DOUTR` erscheinen und eine Invertierung von `DVAL` erfolgen, um die Gültigkeit der neuen Daten zu signalisieren.

Simulationsergebnisse

Aus Abb. 7.8 geht hervor, dass die Erwartungen erfüllt werden und nach erfolgter Übertragung und einer kurzen Latenzzeit die Daten am Ausgang zur Verfügun stehen. Kurz danach wird auch deren Gültigkeit signlisiert. Die Pegel von `DVAL` haben gleiche Bedeutung wie die von `LRCK`, `'0'` für den linken, `'1'` für den rechten Kanal.

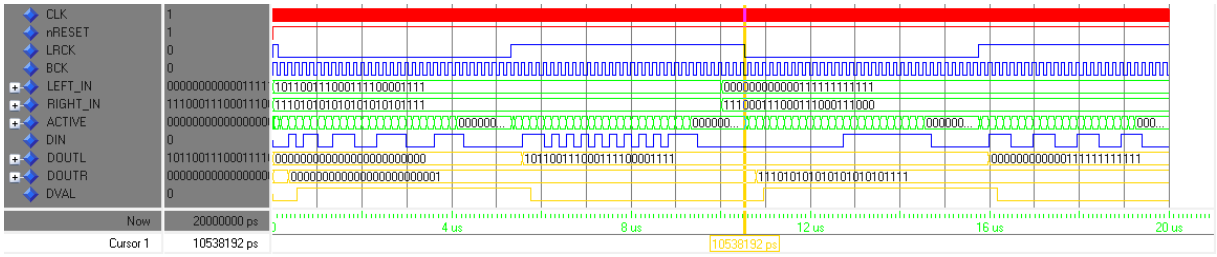


Abbildung 7.8: Simulationswaveform des I^2S-Receivers

7.1.7 I2S Sender

Stimuli

Beim Sender müssen lediglich die parallelen Daten und ein Taktsignal zur Verfügung gestellt werden.

Erwartungswerte

Gemäß der Spezifikation [13], Seite 10, (3) ist die Funktion des Moduls leicht zu überprüfen. Ab der zweiten steigenden Flanke von `BCK` nach einer Flanke von `LRCK` muss mit dem MSB des ersten Datenwortes eines Kanals begonnen werden. Nach erfolgter Übertragung und einem geeigneten zeitlichen Abtand erfolgt die nächste Flanke von `LRCK` und leitet die Datenübertragung des anderen Kanals ein.

Simulationsergebnisse

Wie in Abb. 7.9 zu erkennen folgt die Simulation den Erwartungen und serialisiert die parallel angelegten Daten nach dem I^2S-Format.

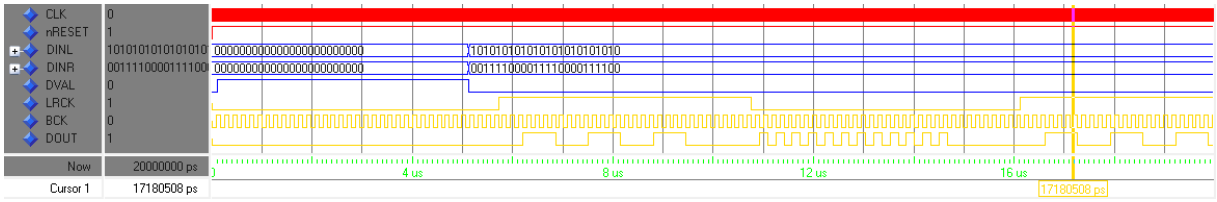


Abbildung 7.9: Simulationswaveform des I^2S-Transmitters

7.1.8 Initialisierung

Stimuli

Das Initialisierungsmodul benötigt keine Stimuli. Lediglich ein Takt muss zur Verfügung gestellt werden.

Erwartungswerte

Zu Beginn soll das `RESET`- respektive das `nRESET`-Signal unmittelbar in den aktiven Zustand übergehen, nach einer einstellbaren Zeit (50 ms) in den inaktiven Zustand wechseln und die Funktion der übrigen Hardare, hier nicht enthalten, freigeben.

Simulationsergebnisse

Gleich zu Anfang wechselt das `RESET`-Signal in den aktivn Zustand. Nach der eingestellten Zeit von etwa 50 ms geht `RESET` auf LOW-Pegel. Das Komplementäre Signal `nRESET` verhält sich erwartungsgemäß gleich mit komplementären Pegeln.

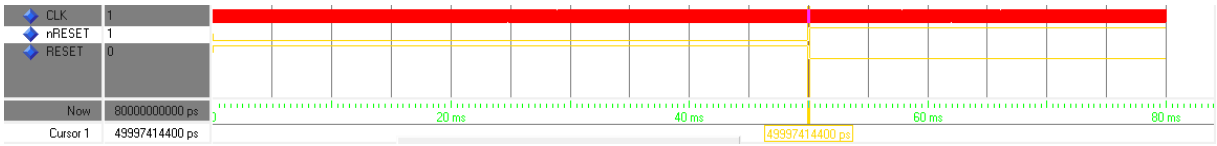


Abbildung 7.10: Simulationswaveform des Initialisierungsmoduls

7.1.9 7-Segment-Anzeigen-Decoder

Stimuli

Als Stimuli kommt eine ansteigende Folge von 0 bis 15 zur Anwendung. Weiterhin ein `CE`-Signal, das die Ausgänge aktiviert bzw. deaktiviert.

Erwartungswerte

Zu erwarten ist die Ansteuerung der Anzeige mit LOW-Aktiven Signalen, jeweils um einen Takt verzögert. Die Codierung entspricht der einer gewöhnlichen 7-Segment-Anzeige (vgl. 7.11). Der Dezimalpunkt wird durch das LSB des Ausgangssignals angestuert und ist bei Ziffern aktiv, die nicht im dezimalen Bereich liegen, also bei Werten von A(10) bis F(15).

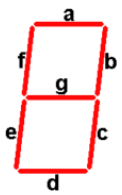


Abbildung 7.11: 7-Segment-Anzeige

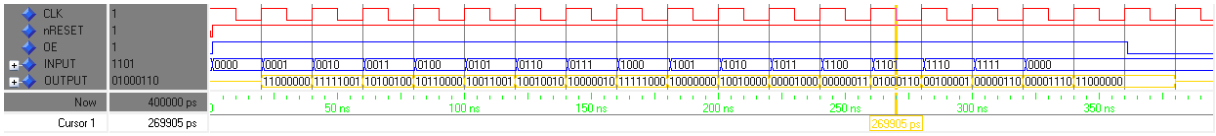


Abbildung 7.12: Simulationswaveform des Anzeigen-Decoders

Simulationsergebnisse

In Abb. 7.12 ist zu erkennen, dass die Dekodierung einen Taktzyklus in Anspruch nimmt. Zu beachten ist, dass die Ausgänge LOW-Aktiv sind und `'0'` für ein leuchtendes Segment steht.

7.2 Musterimplementierungen

Nicht zuletzt um den Betreuern die Möglichkeit zu geben, den Studenten bei Bedarf eine funktionierende Implementierung zur Hand geben zu können, sollten diese die Aufgabe nicht korrekt lösen können, sondern auch, um die Machbarkeit der Aufgabe unter Beweis zu stellen, wurde jedes benötigte Modul komplett und funktionsfähig implementiert. Zwar konnte die Funktion nicht in der realen Hardware nachgewiesen werden, aufgrund der Erfahrungen sollten allenfalls kleine Anpassungen notwendig sein. Die Simulation mit Hilfe von Modelsim garantiert zumindest die Korrektheit der implementierten Algorithmen.

7.2.1 Multiplexer

Stimuli

Um den Aufwand in der Testbench zu reduzieren, wurde auf sinusförmige Stimuli verzichtet, stattdessen werden einfache Rampen mit verschiedenen Anstiegsgeschwindigkeiten erzeugt und als Signale verwendet. Damit kann die Funktion anschaulich dargestellt werden. Auf die Verwendung des Taktsignals wird ebenfalls verzichtet, da es sich beim Multiplexer um eine asynchrone Schaltung handelt.

Erwartungswerte

Die Eingangssignale `sine_5kHz`, `sine_7kHz`, `adc_data` sollen in Abhängigkeit von `sel_s1` und `sel_s2` am Ausgang erscheinen.

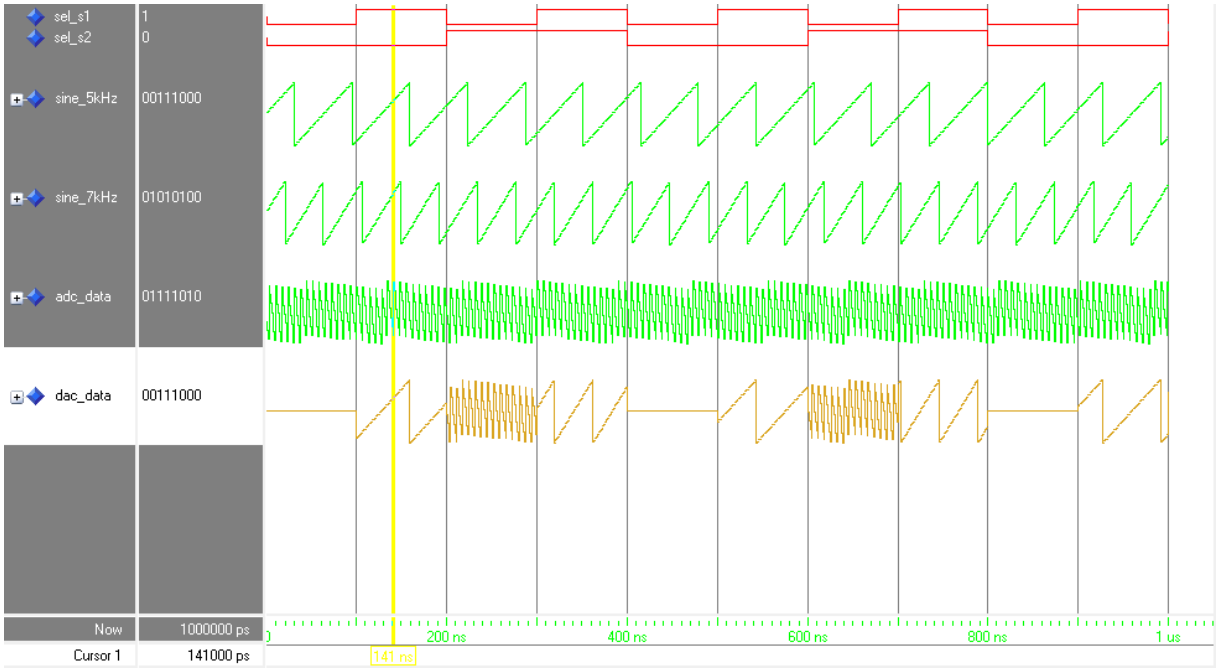


Abbildung 7.13: Simulationswaveform des Multiplexers

Simulationsergebnisse

Wie in der Signaldarstellung (Abb. 7.13) zu sehen, schaltet der Multipexer wie erwartet in Abhängigkeit der Eingangssignale `sel_s1` und `sel_s2` den entsprechenden Eingang auf den Ausgang durch.

7.2.2 Zufallsfolgengenerator

Stimuli

Der Zufallsfolgengenerator benötigt nur ein Takt- und ein Aktivierungssignal.

Erwartungswerte

Bei Aktivität, gesteuert durch `CE` soll eine Pseudo-Zufallsfolge am Ausgang zur Verfügung stehen.

Simulationsergebnisse

Aus Abb. 7.14 geht hervor, dass am Ausgang eine beinahe unvorhersagbare Folge erscheint.

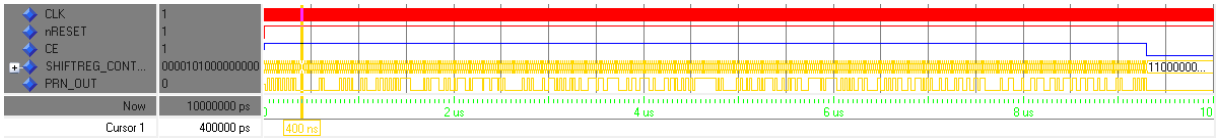


Abbildung 7.14: Simulationswaveform des Zufallsfolgengenerators

7.2.3 CORDIC-Signalgenerator

Stimuli

Für den Signalgenerator sind nur wenige Stimuli nötig. Hierbei handelt es sich um folgende Signale:

CLK Dient dem Signalgenerator als Zeitbasis

nRESET Als Initilisierungssignal

PHASE_INC Steht für den Drehwinkel pro Zeitschritt (Vielfache des Taktes)

Weiterhin sind einige systembedingte Vorgaben zu treffen, welche die Anzahl der Iterationen (`ITERATIONS`) und die Verarbeitungswortbreite (`DATA_WIDTH`) betreffen.

Erwartungswerte

Da am Ausgang des Signalgenerators die Werte eines Sinussignals mit festgelegter Frequenz zur Verfügung gestellt werden sollen, wird am Ausgang des Moduls ein Wert erwartet, der dem Sinus entspricht, der mit Ablauf der Zeit $1/CLK*ITERATIONS$ jeweils um den Winkel `PHASE_INC` fortschreitet. Um die Simulation besser nachvollziehen zu können, sei noch erwähnt, dass im Quellcode Integer-Typen verwendet werden, deren Wertebereich umdefiniert wurde, da VHDL keine Festkommazahlen unterstützt. Die Skalierungskonstanten sind cn_scale_factor $= 2^{\text{DATA\_WIDTH}-2} = 16384$ sowie cn_angle_scale_factor $= 2^{\text{DATA\_WIDTH}-3} = 8192$, wodurch sich ein Wertebereich von $[-2; 2]$ bzw. $[-4; 4]$ ergibt.

Simulationsergebnisse

Abb. 7.15 zeigt die Simulation der Look-Up-Tabelle des CORDIC. Abb. 7.16 zeigt den Basis-Algorithmus, Abb. 7.17 den auf 360°erweiterten, mit dem es möglich ist, um ±180° zu drehen. Genau genommen ist eine Drehung um ±229° entsrechend dem Wertebereich $[-4; 4]$ möglich. Zur Veranschaulichung werden in der Testbench die Integer-Werte in reelle Zahlen bzw. in Grad umgerechnet.

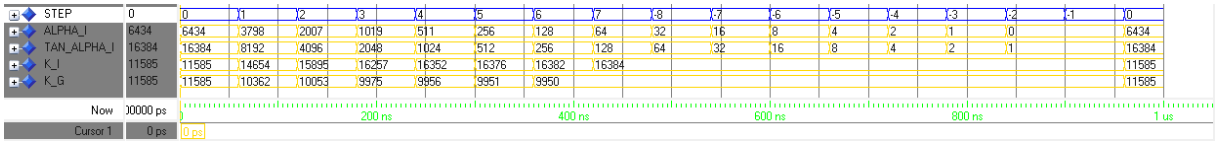


Abbildung 7.15: Simulationswaveform der Cordic-Look-Up-Tabelle

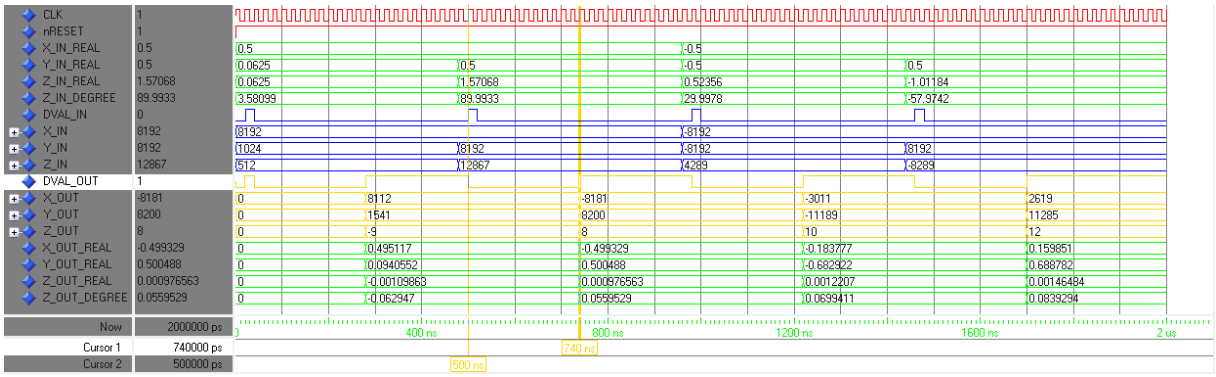


Abbildung 7.16: Simulationswaveform des Cordic-Basis-Algorithmus

7.2.4 Pegelanzeige

Stimuli

Für die Pegelanzeige kommt im ersten Simulationsabschnitt ein Pseudozufallssignal zum Einsatz. Im zweiten Abschnitt werden statische Werte verwendet.

Erwartungswerte

Im ersten Abschnitt sollte sich bei einem gut statistisch verteilten Signal etwa der halbe Pegel, also ein Viertel der Ausgangsleistung einstellen. Bei statischen Signalen entsprechend das Quadrat des Eingangswertes, gewichtet mit einem Skalierungsfaktor.

Simulationsergebnisse

Wie man in Abb. 7.19 erkennen kann, stellt sich ein relativ stabiler Wert im Bereich des Rauschsignals ein. Im Bereich der statischen Werte $\frac{(DATA\_IN^2)-1}{64}$, wenn man von den Übergangsbereichen absieht. Mit dem Skalierungsfaktor ist sichergestellt, dass das Ausgangssignal die maximale Dynamik bei gleicher Wortbreite aufweist, ohne den erlaubten Wertebereich zu verlassen.

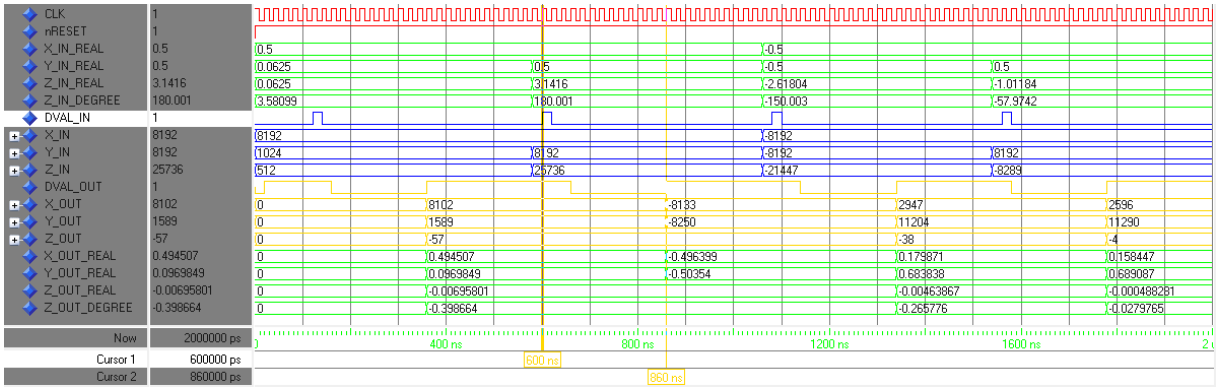


Abbildung 7.17: Simulationswaveform des 360°-Cordic-Algorithmus

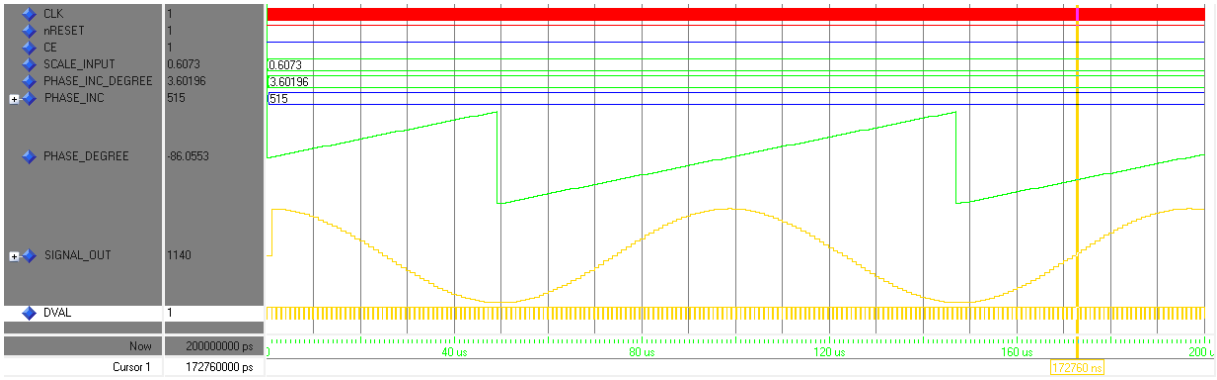


Abbildung 7.18: Simulationswaveform des Cordic-Signalgenerators

7.2.5 Filter in Linearphasenstruktur

Stimuli

Als Stimulus werden vier Sinussignale unterschiedlicher Frequenzen verwendet. Um den Tiefpasscharakter besser darstellen zu können, wurde noch ein Rechteckimpuls generiert. Ein Taktsignal ist ebenfalls nötig. Das `CE`-Signal aktiviert das Pipeline-Register des Filters nur, wenn ein neues Datenwort zur Verfügung steht. Dies erfolgt mit $96kHz$, wie es auch später vom AD-Umsetzer geliefert wird. Zwar soll im Versuch ein Bandpass implementiert werden, die Koeffizienten entsprechen jedoch einem Tiefpassfilter, da dies in der Simulation leichter zu verifizieren ist. Diese wurden [19] (Tiefpass, $F_g = 0,1f_s$, $N = 15$) entnommen:

$c0 = c15$	$c1 = c14$	$c2 = c13$	$c3 = c12$
-0.00101	-0.00521	-0.01269	-0.01214
$c4 = c11$	$c4 = c10$	$c6 = c9$	$c7 = c8$
$+0.01830$	$+0.08914$	$+0.17962$	$+0.24399$

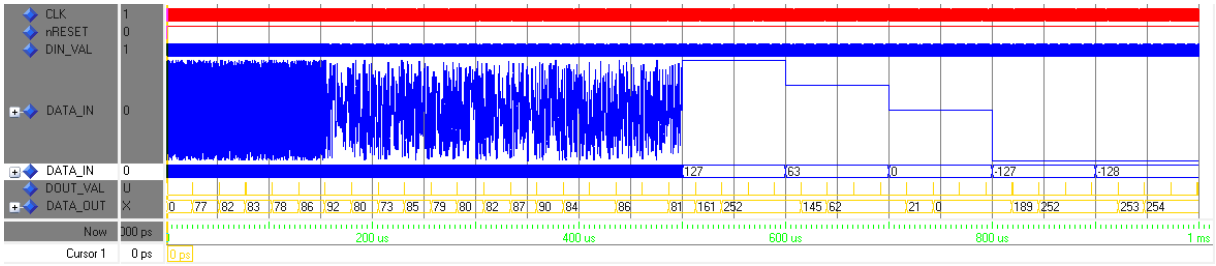


Abbildung 7.19: Simulationswaveform der Pegelanzeige

Erwartungswerte

Die höherfrequenten Anteile sollten unterdrückt werden, die niederfrequenten hingegen nur sehr schwach. Eine Verzögerung ist auch zu erwarten.

Simulationsergebnisse

Wie in Abb. 7.20 gut zu erkennen, hat die Struktur einen Tiefpasscharakter. Speziell der Rechteck-Impuls gegen Ende der Simulationszeit, dessen Flanken verschliffen sind, zeigt das Verhalten deutlich. Die hochfrequenten Spitzen im Signal werden ebenfalls reduziert.

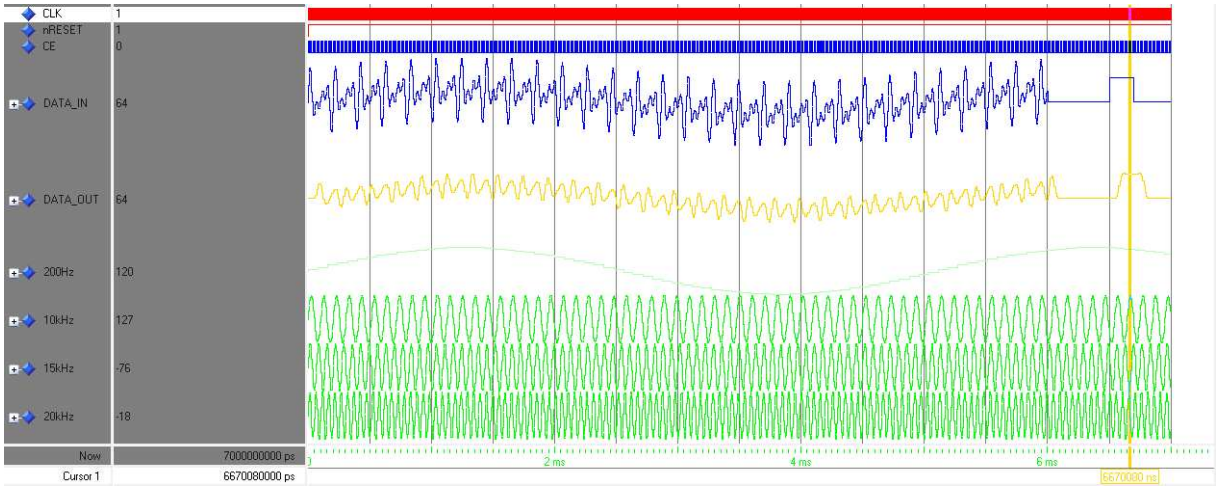


Abbildung 7.20: Simulationswaveform der Linearphasenstruktur

7.2.6 Filteroptimierung

Stimuli

Auch hier kommen wieder 4 Sinussignale unterschiedlicher Frequenz zum Einsatz. Ein Rechteckimpuls mit der Dauer einer Halbperiode der Mittenfrequenz soll den Charakter des Bandpasses weiter verdeutlichen. In allen nachfolgenden Simulationen wird ein

Impulsignal erzeugt, welches die Länge einer Taktperiode besitzt und mit $\approx 96kHz$ wiederholt wird. Folgende Koeffizienten, die aus dem MATLAB-Quellen von Andreas Schedel hervorgehen, wurden für den Filter verwendet:

$c0 = c36$	$c1 = c35$	$c2 = c34$	$c3 = c33$	$c4 = c32$	$c4 = c31$	$c6 = c30$
-0.00128	-0.04028	-0.02579	-0.02100	-0.00510	$+0.01782$	$+0.04010$
$c7 = c29$	$c8 = c28$	$c9 = c27$	$c10 = c26$	$c11 = c25$	$c12 = c24$	$c13 = c23$
$+0.05267$	$+0.04843$	$+0.02582$	-0.00974	-0.04694	-0.07233	-0.07553
$c14 = c22$	$c15 = c21$	$c16 = c20$	$c17 = c19$	$c18$		
-0.05338	-0.01187	$+0.03558$	$+0.07288$	$+0.08694$		

Die gleichen Abhängigkeiten wie im Skript von Herrn Schedel sind nicht zu erwarten, da dieser offenbar eine Abtastfrequenz von $44kHz$ zu Grunde gelegt hat, hier allerdings $96kHZ$ verwendet werden. Dies entspricht einer Skalierung der Mittenfrequenz mit Faktor ≈ 0.5.

Erwartungswerte

Zu erwarten ist aufgrund der relativ hohen Ordnung eine sehr gute Selektion. Der Rechteckimpuls sollte ein ungefähres Abbild der Filterkoeffizienten sein.

Simulationsergebnisse

Im Abb. 7.21 ist gut die erwartete Impulsantwort des Filters zu erkennen. In Abb. 7.22 erkennt man die Selektionseigenschaft des Filters, dessen Mittenfrequenz im Bereich von $4.5kHz$ liegt und eine Bandbreite von etwa 1 kHz besitzt. Die Unterdrückung der restlichen Anteile ist nicht besonders gut und auf die geringe Genauigkeit von 9 Bit und die relativ niedrige Ordnung zurückzuführen.

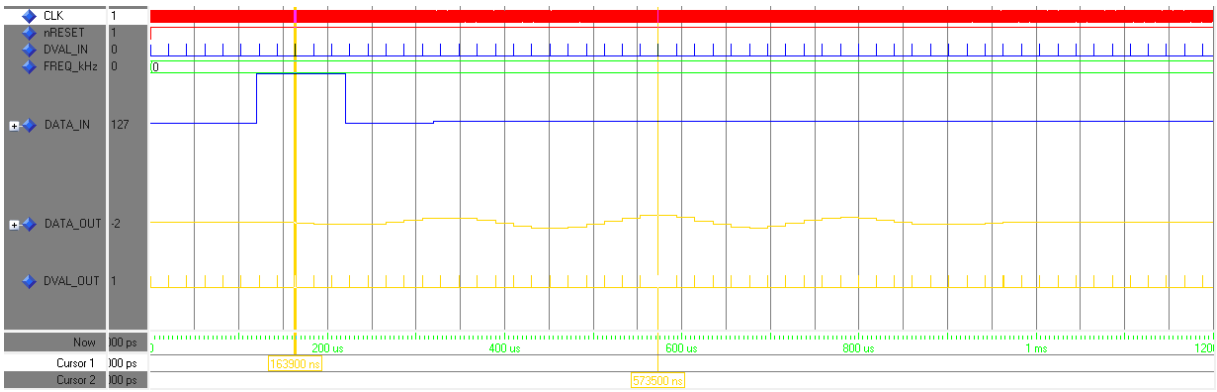


Abbildung 7.21: Simulationswaveform des Bandpases mit Recheckimpulsanregung

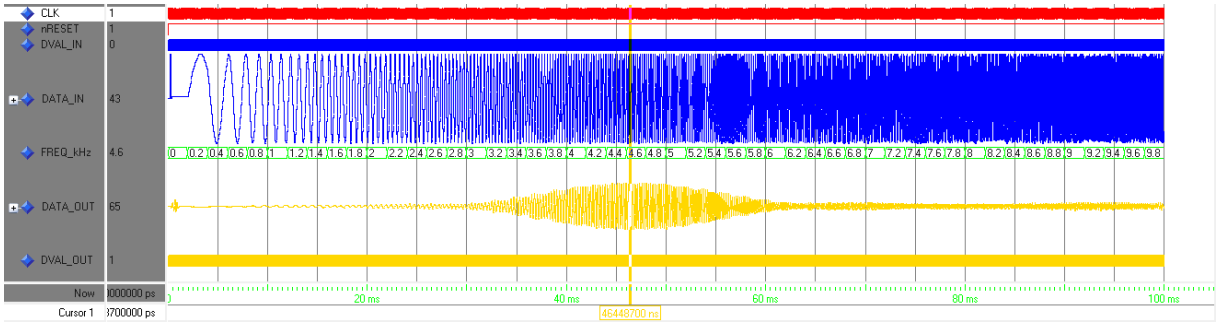


Abbildung 7.22: Simulationswaveform des Bandpasses mit unterschiedlichen Eingangsfrequenzen

7.2.7 Demodulator

Stimuli

Als Stimuli kommen zwei unterschiedliche Eingangsfrequenzen zum Einsatz, die mit Hilfe eines Multiplexers moduliert werden. Weiterhin sind die üblichen Signale notwendig.

Erwartungswerte

Am Ausgang sollte ein zeitlich verzögertes Abbild der Eingangssignale anliegen, allerdings verzerrt.

Simulationsergebnisse

Aus der Abbildung 7.23 geht nicht unmittelbar hervor, dass es sich beim Ausangssignal um ein vorzeichenloses Signal handelt. Auch schlecht darzustellen ist die quadratische Abhängigkeit von `DATA_IN_A/B` und `DATA_OUT_A/B`. Allerdings erkennt man an der Periodendauer/Frequenz z.B. von `DATA_IN_A` ($10kHz$) und `DATA_OUT_A` ($20kHz$), dass eine Quadrierung erfolgt sein muss.

7.2.8 Tiefpassfilter

Stimuli

Die Stimuli des Tiefpassfilters entsprechen denen des Bandpassfilters, allerdings mit Koeffizienten eines Tiefpasses. Folgende Filterkoeffizienten, wieder aus [19] entnommen (Tiefpass, $F_g = 0,025 f_s$, $N = 31$), wurden verwendet:

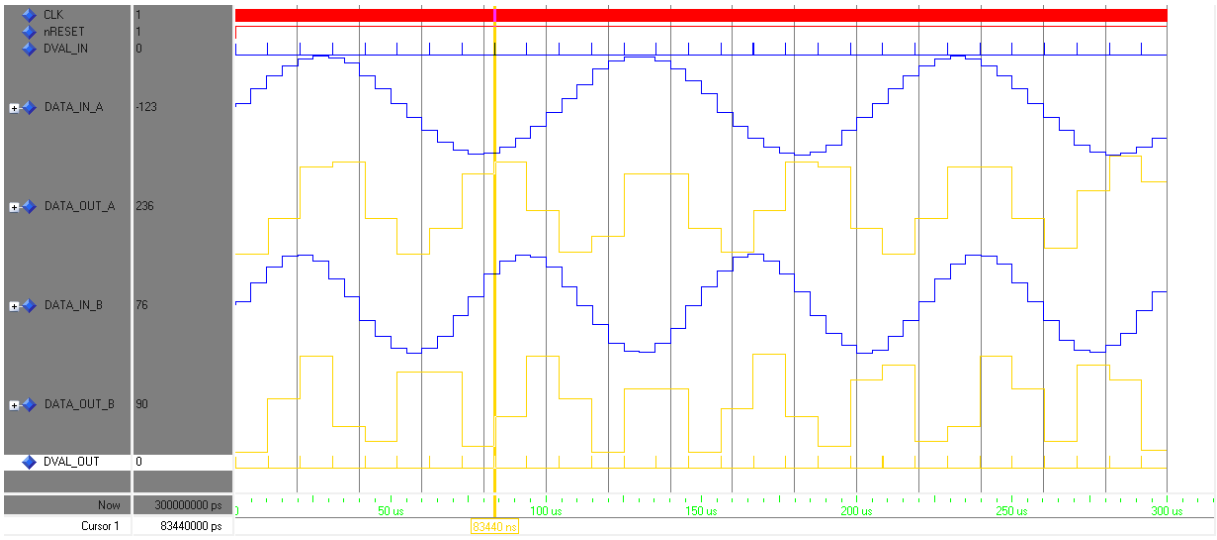


Abbildung 7.23: Simulationswaveform des Demodulators

$c0 = c31$	$c1 = c30$	$c2 = c29$	$c3 = c28$	$c4 = c27$	$c4 = c26$
$+0.00077$	-0.00132	$+0.00236$	$+0.00417$	$+0.00698$	$+0.01095$
$c6 = c25$	$c7 = c24$	$c8 = c23$	$c9 = c22$	$c10 = c21$	$c11 = c20$
$+0.01613$	$+0.02244$	$+0.02968$	$+0.03754$	$+0.04559$	$+0.05335$
$c12 = c19$	$c13 = c18$	$c14 = c17$	$c15 = c16$		
$+0.06033$	$+0.06606$	$+0.07012$	$+0.07222$		

Erwartungswerte

Zu erwarten ist ein Einbrechen der Amplitude ab $\approx 2,4kHz$.

Simulationsergebnisse

Im Simulationswaveform Abb. 7.24 ist deutlich der Tiefpasscharakter zu erkennen. Bei welcher Frequenz die Amplitude auf $-3dB$ abgefallen ist, kann nur schwer genau ermittelt werden.

7.2.9 Signalrückgewinnung

Stimuli

Als Stimuli dienen zwei Sinusschwingungen mit 10 und $14kHz$, die mit einem Wert `AMP_A` bzw. `AMP_B` skaliert werden. Diese werden dem Modul zugeführt. Weiterhin sind die üblichen Signale `DVAL_IN` und `CLK` nötig. Die Hysterese wurde auf 2 eingestellt.

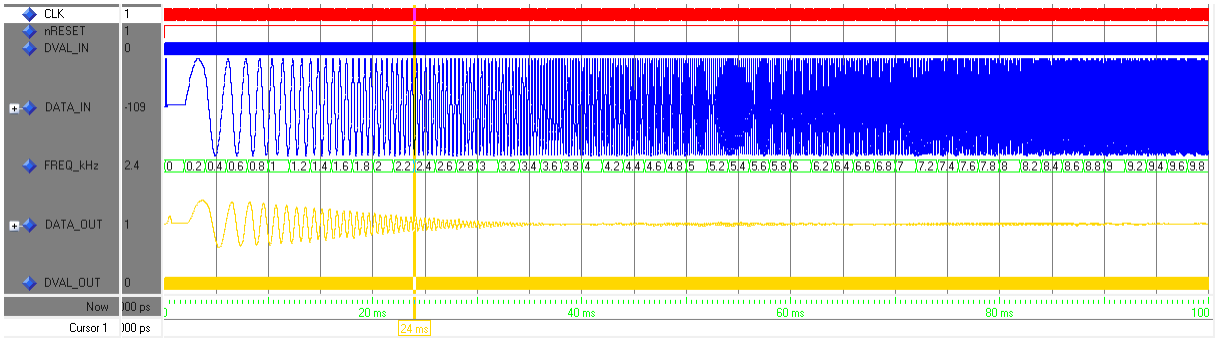


Abbildung 7.24: Simulationswaveform des Tiefpassfilters

Erwartungswerte

Mit einer Verzögerung entsprechend den Verzögerungen der Filter sollte sich das Eingangssignal `DATA_IN` am Ausgang einstellen.

Simulationsergebnisse

Die Simulation in Abb. 7.25 lässt darauf hoffen, dass die Signalrückgewinnung auch in der Praxis gut funktioniert. Selbst bei einem nur sehr kleinen Unterschied der beiden Signale wird trotz der Hystere das Vorherrschende gut erkannt (im Bereich der beiden Cursor) und auf den richtigen Pegel gewechselt. Allerdings muss ein relativ großes Signal anliegen, um eine Erkennung zu ermöglichen. Dies ist auf die vorherige Quadrierung zurückzuführen, die kleine Signale weiter unterdrückt.

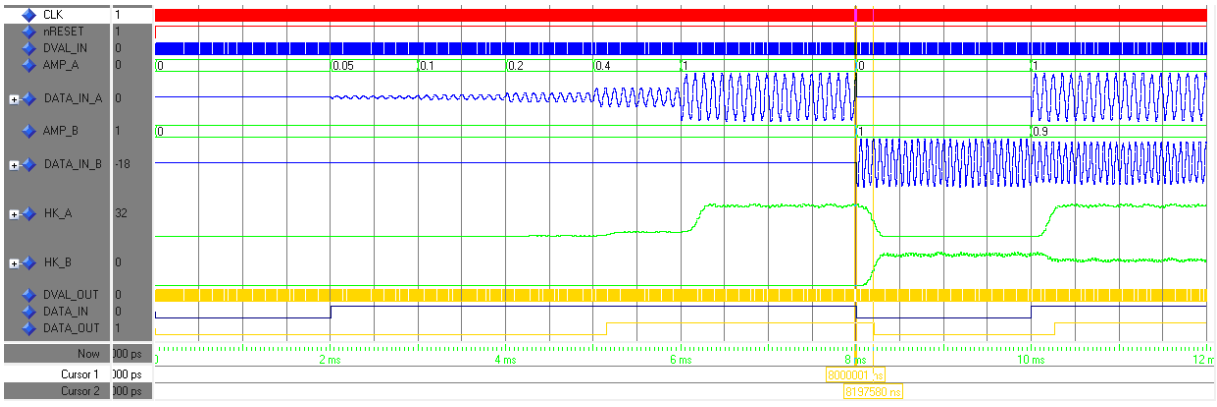


Abbildung 7.25: Simulationswaveform des Signalregenerierungsmoduls

8 Probleme

In diesem Kapitel sollen die Probleme, die beim Entwurf des Praktikums aufgetreten und die bei der Durchführung des Praktikums zu erwarten sind, diskutiert werden.

8.1 Entwurf

8.1.1 Organisation

Durch die Vergabe einer weiteren Studienarbeit, was im Zuge der Suche eines Entwicklungssystems auch diskutiert wurde, die lediglich den Entwurf des Entwicklungssystem als Aufgabe gehabt hätte, wären keine zeitlichen Engpässe entstanden. Auch die Verifikation der Hardware hätte besser duchgeführt werden können. Die Entscheidung, das FPGA zuerst bestücken zu lassen wäre sicher nicht getroffen, sondern im Vorfeld die Funktion der Schaltnetzteile sichergestellt worden, was ein Überleben der FPGAs zur Folge gehabt hätte.

Die nachträgliche Einbeziehung von unvorhersehbaren Einschränkungen behinderte das Vorankommen des Systementwurfs. Diese Vorgaben erforderten nach Fertigstellung der Platine noch erheblichen Zeitaufwand, um die Verfügbarkeitsprüfung die im Vorfeld stattgefunden hatte, nochmals mit anderen Rahmenbedingungen durchzuführen. Die Wahl des Distributors fiel im ersten Anlauf aufgrund des Preises, der parametrischen Suchfunktion und der Verfügbarkeit der meisten Bauteile auf Digi-Key. Anfangs war nicht klar, dass die Bestellung bei diesem Distributor Schwierigkeiten bereiten würde. Die Bezahlung per Kreditkarte oder eine Liquiditätsprüfung die von Digi-Key vorausgesetzt werden, waren schließlich die Kriterien, die diesen Distributor ausschlossen. Da die gesamte Verfügbarkeitsprüfung darauf beruhte verzögerte sich die Bearbeitung der Studienarbeit erheblich.

8.1.2 Hardware

Als negativ erwies sich die frühzeitige Bestückung des FPGAs, welches gleich zu Anfang von einer Fremdfirma aufgebracht wurde. Dies erschwerte die Inbetriebnahme der Schaltnetzteile und hatte schließlich die Zerstörung des FPGA zur Folge. Die Qualität

des Lötstopplacks der einen Platine, der einen Versatz von etwa $100\mu m$ aufweist, kann auch ein Grund für die Zerstörung des ersten FPGA gewesen sein. Das Problem der möglichen Kurzschlüsse unter dem FPGA hätte durch die Wahl eines besseren Herstellers sicher umgangen werden können. Die Verwendung von Kupfergefüllten Durchkontaktierungen bietet sowohl unter dem FPGA, als auch bei der Verwendung von Schaltkreisen mit thermischer Kopplung zur Platine (z.B. Schaltregler) erhebliche Vorteile. Trotz dieser Nachteile kann ich die Beweggründe nachvollziehen, warum diese Einsparungen gemacht wurden.

Der schon genannte Schaltregler für die Bereitstellung der $3,3V$-Versorgung sollte bei einem Redesign auch durch ein fertiges Modul ersetzt werden. Hier bietet sich der, bereits für die Kernspannung des FPGA verwendete, PTH12000W an.

Da die Hardware aus vorgenannten Gründen nicht zur Verfügung steht, war eine Verifikation der VHDL-Quellen nur mittels Simulation möglich und eine Funktion in der Hardware ist nicht sichergestellt. Besonders die Interaktion mit der übrigen Hardware ist dadurch betroffen.

8.1.3 Pflege des Quellcodes

Es wurde versucht, den Quellcode ausreichend zu dokumentieren, um eine schnelle Einarbeitung zu ermöglichen. Da sich die Wartung wegen der unterschiedlichen Versionen für den pflegenden Programmierer, den Betreuer und den Studenten schwierig gestaltet und eine Änderung in bis zu drei verschiedenen Quellen propagiert werden muss, wurde ein Skript erstellt, das diese Arbeit erleichtert. Hierzu ist ActivePerl, welches im Entwicklungs-Verzeichnis abgelegt ist, zu installieren. Das Skript analysiert die Quellcodes im Verzeichnis `ispLEVER` und reagiert auf wenige Steuerkommandos, die als Kommentare in den Quellcode eingefügt werden. Sind keine Steuerkommandos in der Datei vorhanden, so wird diese in das Studenten- und in das Betreuer-Verzeichnis kopiert, taucht der Kommentar `--NOCOPY` auf, so findet sich die Datei im Betreuer-, nicht aber im Studenten-Verzeichnis. Mit den beiden Kommentaren `--STARTL` und `--STOPL` kann ein Bereich markiert werden, der in der Studentenversion des Quellcodes fehlt und mit `--WRITE HERE` die Stelle markiert, die der Student ergänzen soll. Beispiele finden sich im `ispLEVER`-Verzeichnis praktisch in jedem Unterverzeichnis, in dem die VHDL-Dateien für die Versuche liegen und sind auch im Anhang zu finden.

8.2 Praktikumsverlauf

Im Verlauf des Praktikums sind einige Probleme zu erwarten, die hier aufgrund der fehlenden Erfahrung nur abgeschätzt werden können.

Ein unumgängliches Problem wird sicher die lange Synthesezeit darstellen, die je nach Leistungsfähigkeit der zur Verfügung stehenden PCs und der Komplexität des Codes bis zu 20 Minuten in Anspruch nehmen kann. Eine Möglichkeit bestünde darin, für jeden Versuch unterschiedliche toplevel zu verwenden, die bei den ersten Versuchen nur die nötigen Module der Abstraktionsschicht einbinden. Der entscheidenste Geschwindigkeitsfaktor bleibt allerdings der Code der Studenten. Da das FPGA sehr viele Ressource bereitstellt, kann bei ungeschickter Programmierung erheblich Logik erzeugt werden, den die Synthese versucht, mit allen Mitteln in das FPGA zu integrieren. Meist erkennt man den Fehler erst, wenn die Synthese nach einer längeren, erfolglosen Phase den Versuch abbricht. Die im Normalfall zu erwartenden Synthesezeiten auf den Praktikumsrechnern, die zum Zeitpunkt der Studienarbeit zur Verfügung stehen, sollte allerdings 5 Minuten nicht übersteigen, im Regelfall etwa bei 3 Minuten liegen. tark abhängig ist die vom Hauptspeicher (mindestens 768 MB, besser 1 GB).

Unangenehm fiel teilweise auch die Entwicklungssoftware ispLEVER auf, die manche Module nicht kompilieren konnte und mit einer gravierenden Fehlermeldung endete, die auf einen Softwarefehler schließen lässt, nicht zuletzt, weil gebeten wird die ausgegebene Fehlermeldung an den Hersteller weiterzuleiten. Abhilfe schaffte in einen Fall die Verwendung einer Variablen anstatt eines Signals. In einem anderen Fall sorgte die Vertauschung einiger Programmzeilen, ebenfalls ohne Auswirkung auf die Funktion, für einen erfolgreichen Syntheselauf. Diese Tatsache sollte den Betreuern in jedem Fall mitgeteilt werden. Die Betreuer wiederum sollten dies an Lattice weitergeben.

VHDL an sich bietet auch genügend Potenzial, Beschreibungen zu erzeugen, die nicht synthetisierbar sind. Letztlich ist nur eine kleine Untermenge der möglichen Konstrukte derzeit synthetisierbar. Die bisherige Erfahrung mit VHDL-Neulingen zeigt, dass das Verständnis der Hardware sich erst im Laufe der Zeit einstellt und Fehler aufgrund der Verwendung beispielsweise von `wait for 10 ns` nicht verstanden werden. Zwar wird im Skriptum bereits auf sehr viele dieser Fallen hingewiesen, ganz ausschließen lassen sie sich aber nicht.

Auch die Verwendung von

- Vergleichen auf ganze Wertebereiche,
- von `for`-Schleifen im falschen Kontext,
- zu wenigen Registern über Hirarchieebenen hinweg und

- Integer-Typen im Port-Teil einer Entity

um nur einige zu nennen, führen nicht unmittelbar zu einem Fehler. Oft bemerkt man diese nur bei genauer Analyse der Ausgabe der Synthese und sie können nie ganz ausgeschlossen werden. Selbst erfahrene Programmierer bemerken oft erst an der Angabe der verbrauchten Ressourcen, dass ein Fehler dieses Typs vorliegt. Eben diese Konstrukte können die Synthesezeit (genauer die Place-and-Route-Zeit) erhöhen bzw. die zum Abbruch führen.

Auch mit der Hardware könnten im Verlauf des Praktikums Probleme entstehen, sofern der Entwurf dieser Arbeit weiter verfolgt wird. Bei Teilnehmern des Praktikums ist praktisch mit allen Enventualitäten zu rechnen und man kann nicht ausschließen, dass Kurzschlüsse provoziert werden, die Versorgungsspannung verpolt oder der Lausprecher-Ausgang des Leistungsverstärkers mit dem Mikrofoneingang verbunden wird. Schaltungstechnisch wurde dem mit unterschiedlichen Maßnahmen Rechnung getragen. So ist in den Versorgungsspannungseingang eine Leistungsdiode integriert, die den Stromfluss bei falscher Polung unterbindet. Die Eingänge der Schaltung sind mit Schutzbeschaltungen versehen, die den Wert der Spannung auf erträgliche Werte begrenzen. Lediglich eine Übersteuerung wäre die Folge, eine Zerstörung ist beinahe unmöglich. Die Verwendung des BGA-Gehäuses bietet bei dieser Betrachtungsweise sogar Vorteile, da die Anschlüsse nur sehr schwer zugänglich sind. Auch die wenigen Bauteile an der Unterseite und die Verwendung von SMD-Bauelementen, bis auf wenige Ausnahmen, tragen zur Robustheit der Hardware bei. Beschädigungen sollte selbst bei unsachgemäßer Handhabung die Ausnahme bleiben.

Die Liste der zu erwartenden Probleme ist sicher endlos fortsetzbar, die wichtigsten und häufigsten dürften allerdings enthalten sein.

9 Ergebnisse

Am Ende der Studienarbeit ist die Umsetzung der Konzepte der Vorlesung Architekturen der Digitalen Signalverarbeitung in einen VHDL-Quellcode erfolgt. Dieser funktioniert in der Simulation und die dazu notwendigen Kenntnisse zur Umsetzung liegen in Form eines Skriptums vor.

Das Skriptum ist so aufgebaut, dass es in kürzester Zeit die notwendigen Kenntnisse zur Programmierung von FPGAs und Grundlagen der digitalen Signalverarbeitung schon beim Erlernen der Sprache VHDL vermittelt. Auch hier stehen noch einige Iterationsstufen aus, die einige reale Praktikumsdurchläufe erfordern.

Leider konnte das untergeordnete Ziel, eine Entwicklungsplattform zur Verfügung zu stellen, nicht umgesetzt werden, da die entstandenen Probleme dies im zeitlichen Rahmen der Studienarbeit nicht erlaubten. Dennoch war der Aufwand nicht umsonst, da genügend Erfahrungen gesammelt werden konnten, um in der nachfolgenden Iterationsstufe der Hardware bereits ein fehlerfreies Design wahrscheinlch zu machen. Die nötigen Änderungen an der Hardware werden im Kapitel 8 und in den Schaltplan-Errata ausreichend diskutiert. Eine schnelle Lösung bestünde im Entfernen des BGA, einer genauen Einstellung und Messung der Spannungen und eine erneute Bestückung des FPGAs.

10 Schlussbemerkungen

Letztendlich ist festzustellen, dass die Studienarbeit zu umfangreich war und in zwei Arbeiten hätte aufgeteilt werden können.

Die Bandbreite und die Anzahl der bearbeiteten Aufgaben sorgte allerdings für viel Abwechslung und einen großen Schatz an Erfahrungen. Besonders die Programmierung in VHDL und die Auseinandersetzung mit der Didaktik eröffneten dem Verfasser neue Möglichkeiten der Beschreibung und sorgten für einen sichereren Umgang mit der Sprache. Auch die Verwendung der bis zur Studienarbeit unbekannten Entwicklungsumgebung ispLEVER schärft den Sinn für die Vor- und Nachteile der bereits bekannten Alternativen. Die gesammelten Erfahrungen mit den FPGAs von Lattice dienten auch schon der Entscheidungshilfe in der Bildsensorik des Fraunhofer Instituts.

Eine Lockerung der finanziellen Vorgaben hätten zu einer funktionierenden Hardware einen großen Teil beigetragen. Ein Zugang bei Digi-Key wäre für die Zukunft sicher eine sinnvolle Investition, da nicht davon ausgegangen werden kann, dass die hiesigen Distributoren die Auswahl an aktueller Hardware dem der amerikanischen Distributoren anpassen werden. Zwar ist es möglich, beispielsweise bei Spoerle ein Angebot zu einem schlecht verfügbaren Bauteil einzuholen, die Mindestabnahmemengen sind allerdings sehr hoch.

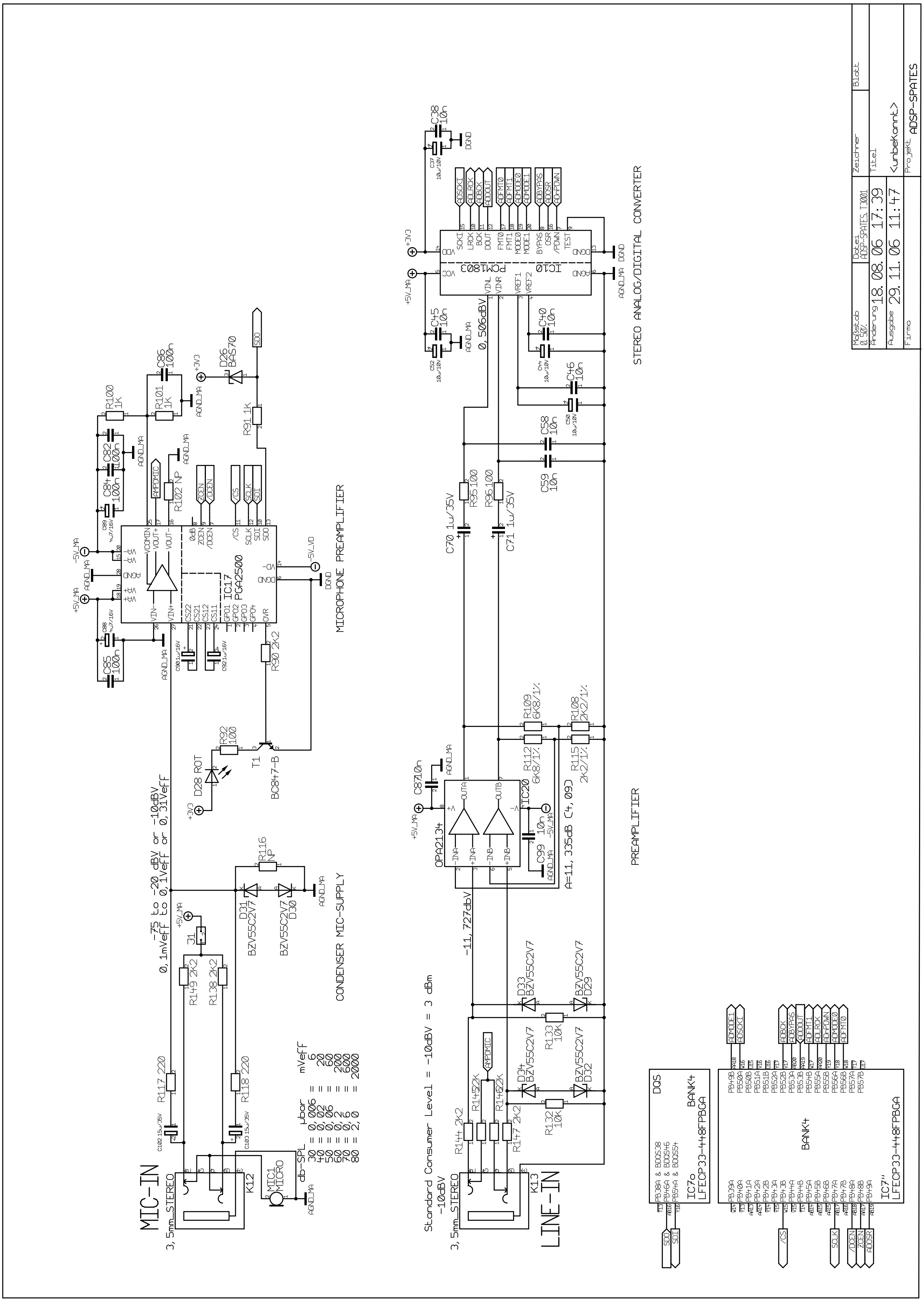
Dieses Praktikum hat in jedem Fall einige Alleinstellungsmerkmale, die es von den übrigen Praktika der Technischen Fakultät abheben. Wegen des komplexen Themas bedarf es Betreuern, die schon eine gewisse Erfahrung im Umgang mit VHDL besitzen, im besten Fall schon einige Zeit damit gearbeitet haben.

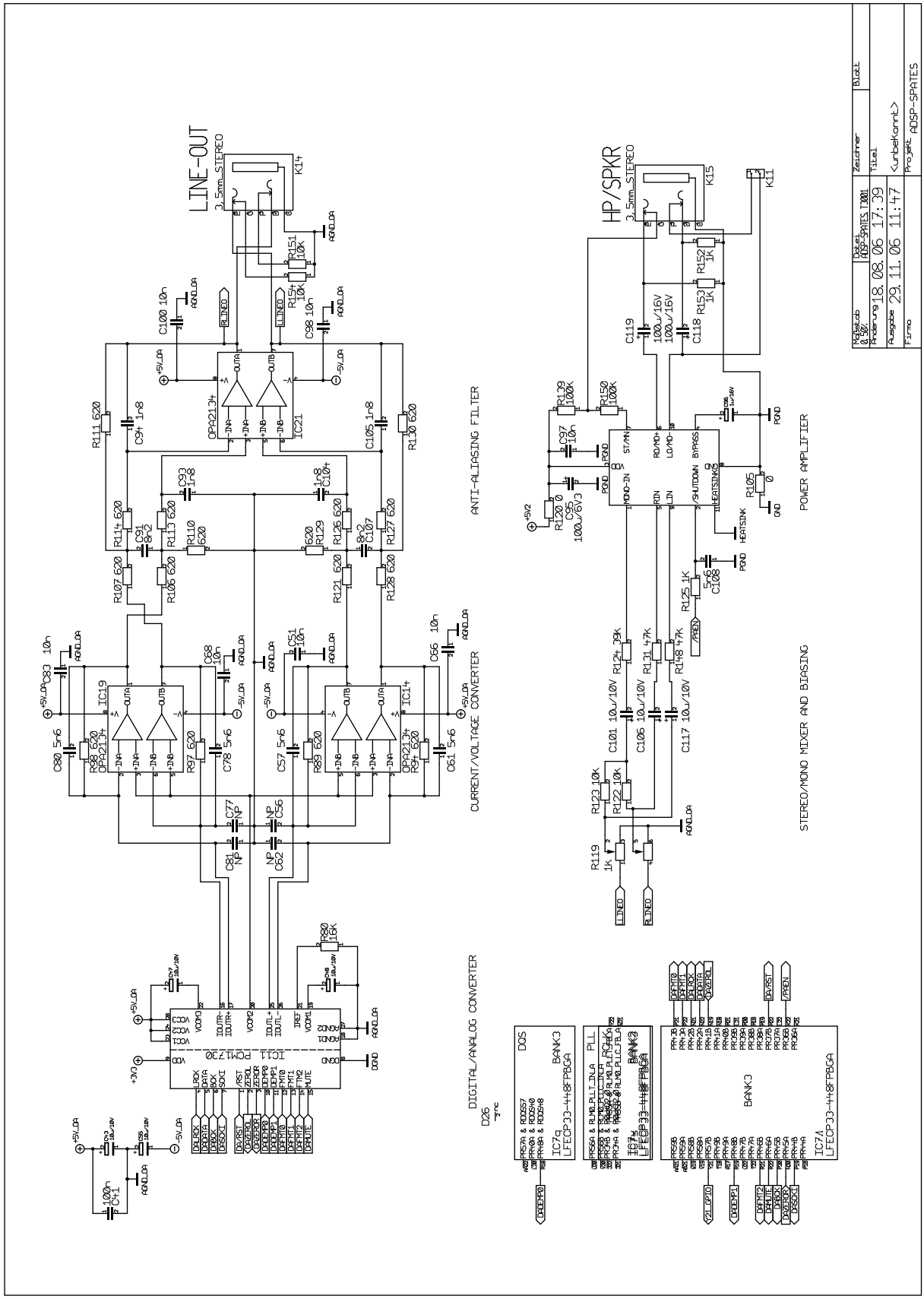
Der MATLAB-Teil sollte noch besser auf die Hardware abgestimmt werden. Viele Konzepte, die darin Verwendung finden, sind zwar auch in Hardware zu implementieren, haben allerdings eher akademischen Charakter und die Lösung in der Realität beschreibt wesentlich elegantere Wege, die einen Bruchteil der Hardware zur Folge haben. Als Beispiel sei hier der Modulator erwähnt.

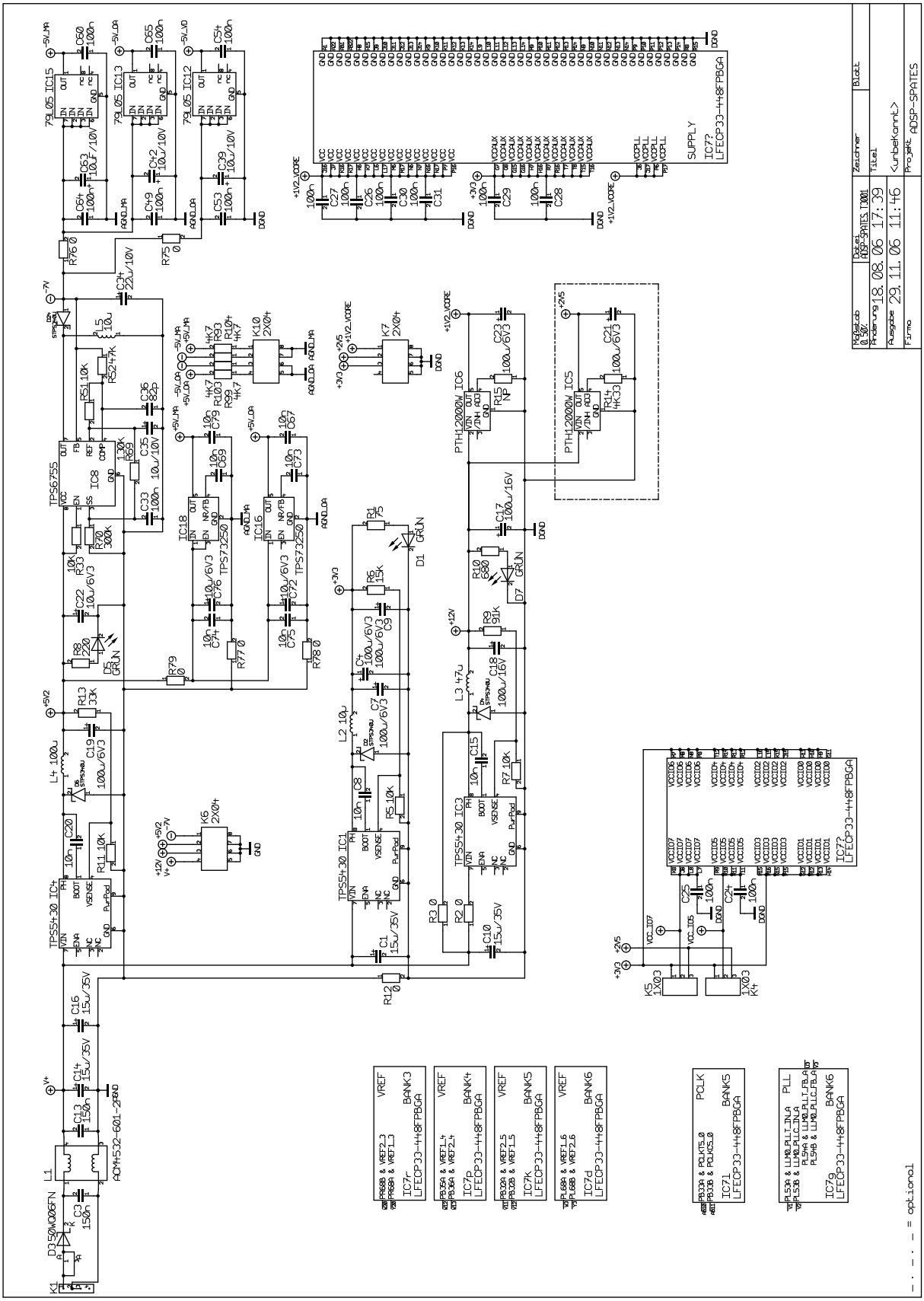
Teil II

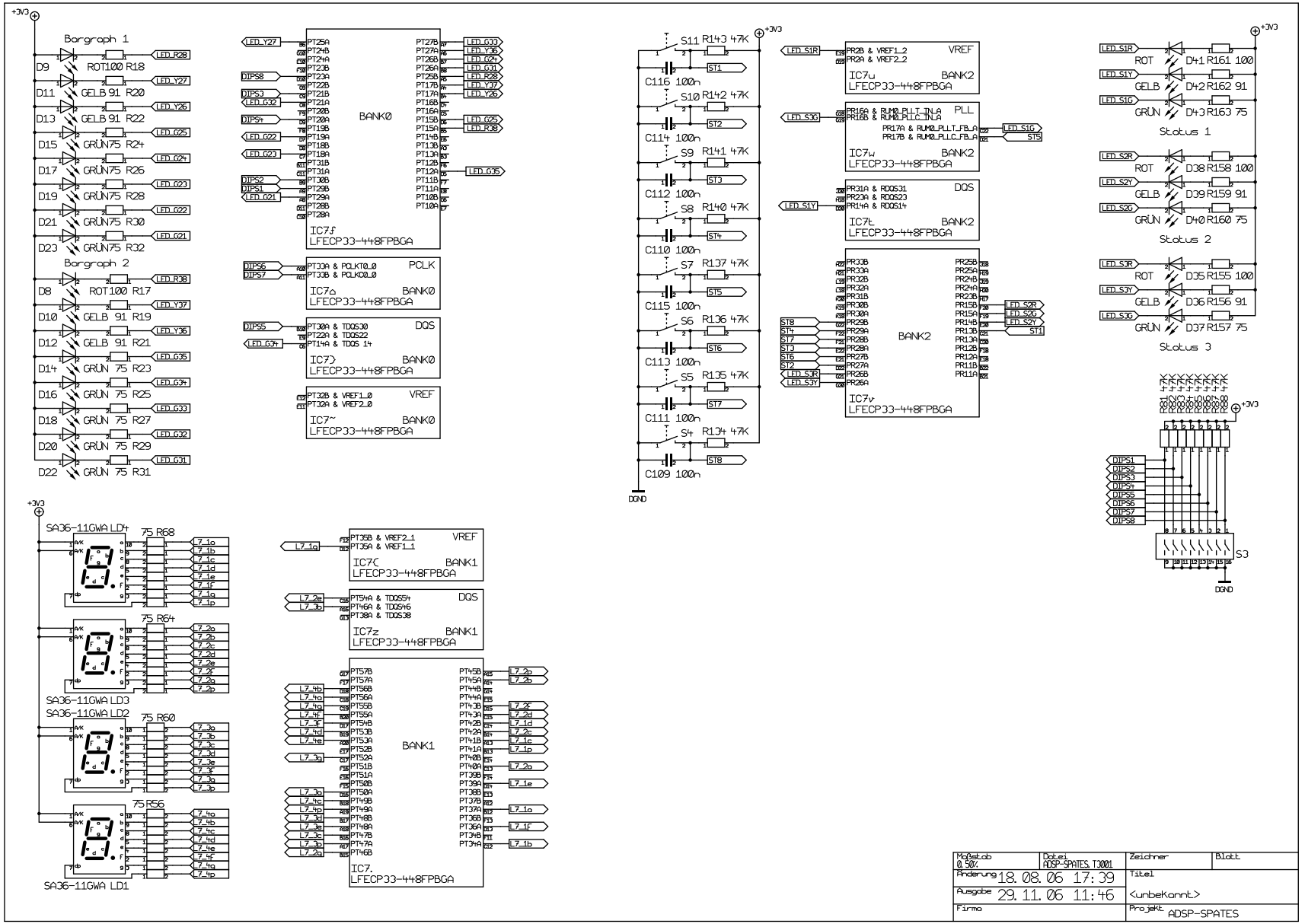
Anhang

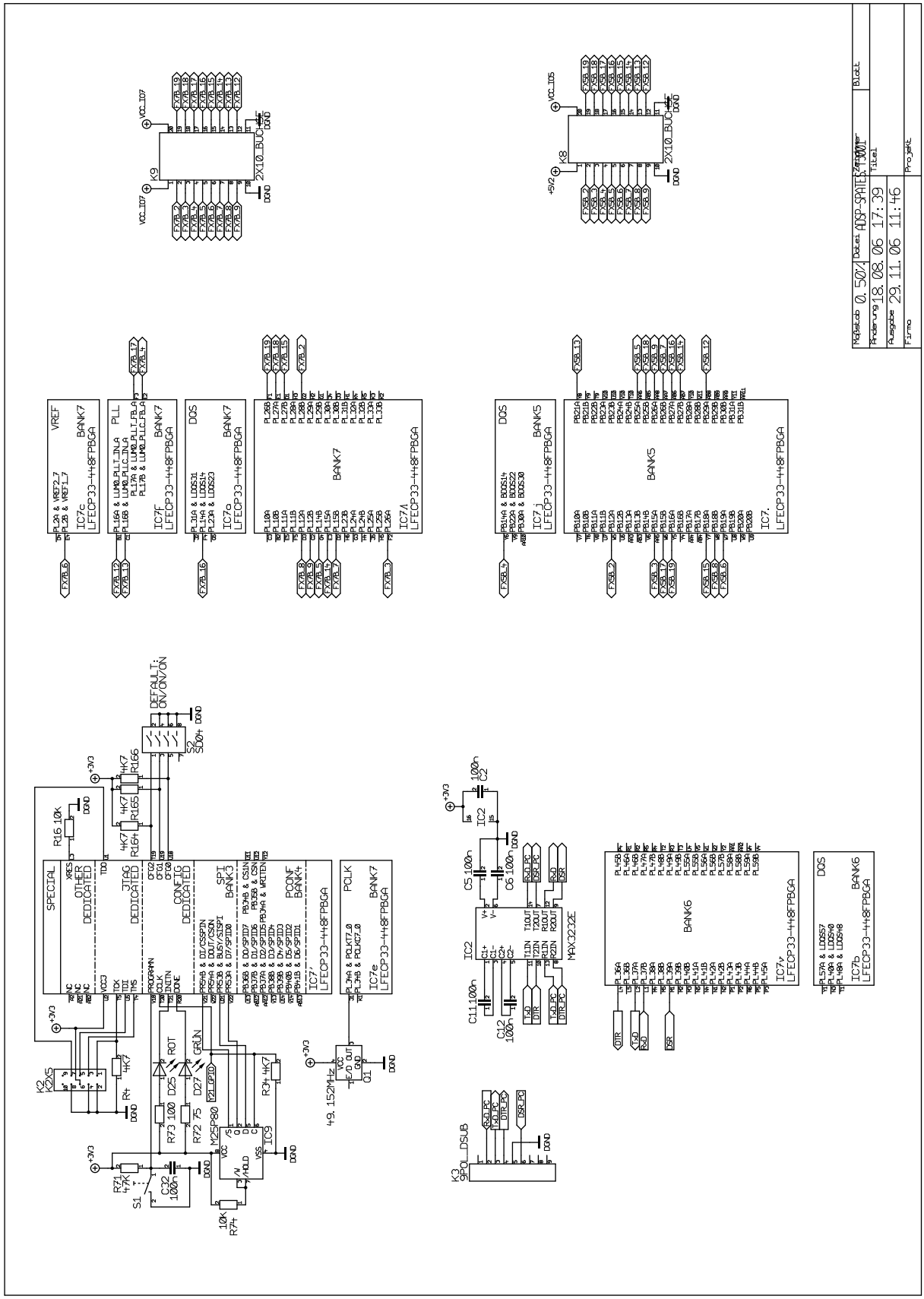
A Schaltpläne


MICROPHONE PREAMPLIFIER
STEREO ANALOG/DIGITAL CONVERTER
PREAMPLIFIER
CONDENSER MIC-SUPPLY
MIC-IN
LINE-IN
ADSP-SPATES


LINE-OUT
3.5mm_STEREO
ANTI-ALIASING FILTER
CURRENT/VOLTAGE CONVERTER
DIGITAL/ANALOG CONVERTER
HP/SPKR
3.5mm_STEREO
POWER AMPLIFIER
STEREO/MONO MIXER AND BIASING
Zeichner
Titel
<unbekannt>
Projekt ADSP-SPATES




Bargraph 1
D9 ROT100 R18
D11 GELB 91 R20
D13 GELB 91 R22
D15 GRÜN75 R24
D17 GRÜN75 R26
D19 GRÜN75 R28
D21 GRÜN75 R30
D23 GRÜN75 R32
Bargraph 2
D8 ROT100 R17
D10 GELB 91 R19
D12 GELB 91 R21
D14 GRÜN 75 R23
D16 GRÜN 75 R25
D18 GRÜN 75 R27
D20 GRÜN 75 R29
D22 GRÜN 75 R31
BANK0
IC7f
LFECP33-448FPBGA
PCLK
IC7a
DQS
IC7)
VREF
IC7~
S11 R143 47K
S10 R142 47K
S9 R141 47K
S8 R140 47K
S7 R137 47K
S6 R136 47K
S5 R135 47K
S4 R134 47K
C116 100n
C114 100n
C112 100n
C110 100n
C115 100n
C113 100n
C111 100n
C109 100n
DGND
BANK2
IC7u
IC7w
IC7t
IC7v
PLL
Status 1
Status 2
Status 3
SA36-11GWA LD4
SA36-11GWA LD3
SA36-11GWA LD2
SA36-11GWA LD1
75 R68
75 R64
75 R60
75 R56
BANK1
IC7c
IC7z
IC7.
S3
Maßstab 0.50:
Datei ADSP-SPATES_T3001
Änderung 18.08.06 17:39
Ausgabe 29.11.06 11:46
Firma
Zeichner
Blatt
Titel <unbekannt>
Projekt ADSP-SPATES



A.1 Errata

Folgenden Fehler und Änderungen sind bisher bekannt:

C34 Polung falsch

C89 Polung falsch

R100 muss gegen +5V gehen

Q1 32,768MHz wegen besserer Verfügbarkeit

IC2 SP3232 anstatt MAX3232

B VHDL-Quellcode

Der anhängende VHDL-Quellcode beinhaltet alle letztendlich im Praktikum zur Anwendung kommenden Module. Ebenfalls im Entwicklungs-Verzeichnis befindliche VHDL-Dateien sind bestenfalls alternative Lösungswege, meist jedoch verworfene und unfertige Entwürfe. Sie könnten allerdings für die Betreuer von Nutzen sein. Übrige Dateinen mit anderen Endungen sind von ispLever oder anderen beteiligten Anwendungen erzeugte Dateien. Manche könnten bedenkenlos gelösch werden, andere sind wichtig, um das Projekt in ispLEVER mit dem letzten Stand zu öffnen.

B.1 Hauptmodule

B.1.1 Toplevel

Listing B.1: toplevel.vhd

```
-------------------------------------------------------------------------------
-- Title      : Toplevel
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : toplevel.vhd
-- Author     : Daniel Glaser
-- Company    : LfTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-11-14
-- Last update: 2007-01-09
-- Platform   : LFECP20E
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description: This module is the toplevel of every exercise in this
--              laboratory.
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LfTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-11-14  1.0      sidaglas        Created
```

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

entity toplevel is

  generic (
    DATA_WIDTH : positive := 8);

  port (
    ------------------------------------------------------------------------
    -- Generic signals
    ------------------------------------------------------------------------
    CLK : in std_logic;

    ------------------------------------------------------------------------
    -- Human device interface (HDI)
    ------------------------------------------------------------------------

    -- Inputs
    SWITCHES : in std_logic_vector(7 downto 0);
    BUTTONS  : in std_logic_vector(7 downto 0);

    -- Outputs
    STATUS_RED      : out std_logic_vector(2 downto 0);
    STATUS_YELLOW   : out std_logic_vector(2 downto 0);
    STATUS_GREEN    : out std_logic_vector(2 downto 0);
    SEGMENT_LED1    : out std_logic_vector(7 downto 0);
    SEGMENT_LED2    : out std_logic_vector(7 downto 0);
    SEGMENT_LED3    : out std_logic_vector(7 downto 0);
    SEGMENT_LED4    : out std_logic_vector(7 downto 0);
    BARGRAPH_LEFT   : out std_logic_vector(7 downto 0);
    BARGRAPH_RIGHT  : out std_logic_vector(7 downto 0);

    ------------------------------------------------------------------------
    -- PC communications
    ------------------------------------------------------------------------
    RS232_TX   : out std_logic;
    RS232_RX   : in  std_logic;
    RS232_DTR  : out std_logic;
    RS232_DSR  : in  std_logic;
```

```
-- Board Extensions
EXTENDER_B7 : inout std_logic_vector(15 downto 0);
EXTENDER_B5 : inout std_logic_vector(15 downto 0);

-- Peripherial connections

-- Digitally controlled microphone preamplifier
MA_ZCEN  : out std_logic;
MA_nDCEN : out std_logic;
MA_nCS   : out std_logic;
MA_SCLK  : out std_logic;
MA_SDI   : out std_logic;
MA_SDO   : in  std_logic;

-- Analog to digital converter
AD_SCKI  : out   std_logic;
AD_LRCK  : inout std_logic;
AD_BCK   : inout std_logic;
AD_DOUT  : in    std_logic;
AD_FMT   : out   std_logic_vector(1 downto 0);
AD_MODE  : out   std_logic_vector(1 downto 0);
AD_BYPAS : out   std_logic;
AD_OSR   : out   std_logic;
AD_nPDWN : out   std_logic;

-- Digital to analog converter
DA_SCKI  : out   std_logic;
DA_LRCK  : inout std_logic;
DA_BCK   : inout std_logic;
DA_DIN   : out   std_logic;
DA_nRST  : out   std_logic;
DA_ZEROL : in    std_logic;
DA_ZEROR : in    std_logic;
DA_DEMP  : out   std_logic_vector(1 downto 0);
DA_FMT   : out   std_logic_vector(2 downto 0);
DA_MUTE  : out   std_logic;

-- Speaker power amplifier
PA_nEN : out std_logic);
```

```
attribute LOC      : string;
attribute IO_TYPES : string;
attribute SLEW     : string;

-- LOC attributes
attribute LOC of CLK             : signal is "J1";
attribute LOC of SWITCHES        : signal is "A9,B9,C9,D9,B10,A10,A11,D10";
attribute LOC of BUTTONS         : signal is "C21,D22,E22,F22,D21,E21,F21,G22";
attribute LOC of STATUS_RED      : signal is "E19,F20,G21";
attribute LOC of STATUS_YELLOW   : signal is "S20,E20,G20";
attribute LOC of STATUS_GREEN    : signal is "C22,F19,G19";
attribute LOC of SEGMENT_LED1    : signal is "B12,C12,A13,C14,D14,D13,D12,B13";
attribute LOC of SEGMENT_LED2    : signal is "C13,A14,B14,C15,C16,D15,B15,A15";
attribute LOC of SEGMENT_LED3    : signal is "D16,A16,B16,B17,A18,D17,C17,A17";
attribute LOC of SEGMENT_LED4    : signal is "C18,D18,B18,B19,A20,B20,C19,A19";
attribute LOC of BARGRAPH_LEFT   : signal is "B5,A4,A6,D5,C6,A7,C8,B8";
attribute LOC of BARGRAPH_RIGHT  : signal is "A5,B6,B4,D6,B7,C7,D7,A8";
attribute LOC of RS232_TX        : signal is "L2";
attribute LOC of RS232_RX        : signal is "L1";
attribute LOC of RS232_DTR       : signal is "L4";
attribute LOC of RS232_DSR       : signal is "M1";
attribute LOC of EXTENDER_B5     : signal is "Y5,AB5,W6,AB6,Y7,AB7,Y8,AB8,AA8,W8,AA7,W7
attribute LOC of EXTENDER_B7     : signal is "F1,E1,F3,F4,D1,E3,C1,B1,C2,D3,D2,E4,G4,E2
attribute LOC of MA_ZCEN         : signal is "AA17";
attribute LOC of MA_nDCEN        : signal is "AB18";
attribute LOC of MA_nCS          : signal is "W15";
attribute LOC of MA_SCLK         : signal is "AB17";
attribute LOC of MA_SDI          : signal is "Y16";
attribute LOC of MA_SDO          : signal is "AB16";
attribute LOC of AD_SCKI         : signal is "W16";
attribute LOC of AD_LRCK         : signal is "AA20";
attribute LOC of AD_BCK          : signal is "V17";
attribute LOC of AD_DOUT         : signal is "AA19";
attribute LOC of AD_FMT          : signal is "W17,W18";
attribute LOC of AD_MODE         : signal is "AA18,Y18";
attribute LOC of AD_BYPAS        : signal is "AB20";
attribute LOC of AD_OSR          : signal is "AB19";
attribute LOC of AD_nPDWN        : signal is "Y19";
attribute LOC of DA_SCKI         : signal is "P19";
attribute LOC of DA_LRCK         : signal is "N21";
attribute LOC of DA_BCK          : signal is "P20";
attribute LOC of DA_DIN          : signal is "N22";
attribute LOC of DA_nRST         : signal is "M22";
attribute LOC of DA_ZEROL        : signal is "N19";
```

```
attribute LOC of DA_ZEROR       : signal is "N20";
attribute LOC of DA_DEMP        : signal is "R19,R18";
attribute LOC of DA_FMT         : signal is "R21,P22,P21";
attribute LOC of DA_MUTE        : signal is "R22";
attribute LOC of PA_nEN         : signal is "K22";

-- IO-Type attributes
attribute IO_TYPES of CLK              : signal is "LVCMOS33, -";   -- NO OUT
attribute IO_TYPES of SWITCHES         : signal is "LVCMOS33, -";   -- NO OUT
attribute IO_TYPES of BUTTONS          : signal is "LVCMOS33, -";   -- NO OUT
attribute IO_TYPES of STATUS_RED       : signal is "LVCMOS33, 20";  -- 20mA
attribute IO_TYPES of STATUS_YELLOW    : signal is "LVCMOS33, 20";  -- 20mA
attribute IO_TYPES of STATUS_GREEN     : signal is "LVCMOS33, 20";  -- 20mA
attribute IO_TYPES of SEGMENT_LED1     : signal is "LVCMOS33, 20";  -- 20mA
attribute IO_TYPES of SEGMENT_LED2     : signal is "LVCMOS33, 20";  -- 20mA
attribute IO_TYPES of SEGMENT_LED3     : signal is "LVCMOS33, 20";  -- 20mA
attribute IO_TYPES of SEGMENT_LED4     : signal is "LVCMOS33, 20";  -- 20mA
attribute IO_TYPES of BARGRAPH_LEFT    : signal is "LVCMOS33, 20";  -- 20mA
attribute IO_TYPES of BARGRAPH_RIGHT   : signal is "LVCMOS33, 20";  -- 20mA
attribute IO_TYPES of RS232_TX         : signal is "LVCMOS33, 20";  -- 20mA
attribute IO_TYPES of RS232_RX         : signal is "LVCMOS33, -";   -- NO OUT
attribute IO_TYPES of RS232_DTR        : signal is "LVCMOS33, 20";  -- 20mA
attribute IO_TYPES of RS232_DSR        : signal is "LVCMOS33, -";   -- NO OUT
attribute IO_TYPES of EXTENDER_B5      : signal is "LVCMOS33, 20";  -- 20mA
attribute IO_TYPES of EXTENDER_B7      : signal is "LVCMOS33, 20";  -- 20mA
attribute IO_TYPES of MA_ZCEN          : signal is "LVCMOS33, 8";   -- 8mA
attribute IO_TYPES of MA_nDCEN         : signal is "LVCMOS33, 8";   -- 8mA
attribute IO_TYPES of MA_nCS           : signal is "LVCMOS33, 8";   -- 8mA
attribute IO_TYPES of MA_SCLK          : signal is "LVCMOS33, 8";   -- 8mA
attribute IO_TYPES of MA_SDI           : signal is "LVCMOS33, 8";   -- 8mA
attribute IO_TYPES of MA_SDO           : signal is "LVCMOS33, -";   -- NO OUT
attribute IO_TYPES of AD_SCKI          : signal is "LVCMOS33, 8";   -- 8mA
attribute IO_TYPES of AD_LRCK          : signal is "LVCMOS33, 8";   -- 8mA
attribute IO_TYPES of AD_BCK           : signal is "LVCMOS33, 8";   -- 8mA
attribute IO_TYPES of AD_DOUT          : signal is "LVCMOS33, -";   -- NO OUT
attribute IO_TYPES of AD_FMT           : signal is "LVCMOS33, 8";   -- 8mA
attribute IO_TYPES of AD_MODE          : signal is "LVCMOS33, 8";   -- 8mA
attribute IO_TYPES of AD_BYPAS         : signal is "LVCMOS33, 8";   -- 8mA
attribute IO_TYPES of AD_OSR           : signal is "LVCMOS33, 8";   -- 8mA
attribute IO_TYPES of AD_nPDWN         : signal is "LVCMOS33, 8";   -- 8mA
attribute IO_TYPES of DA_SCKI          : signal is "LVCMOS33, 8";   -- 8mA
attribute IO_TYPES of DA_LRCK          : signal is "LVCMOS33, 8";   -- 8mA
attribute IO_TYPES of DA_BCK           : signal is "LVCMOS33, 8";   -- 8mA
attribute IO_TYPES of DA_DIN           : signal is "LVCMOS33, 8";   -- 8mA
```

```
attribute IO_TYPES of DA_nRST        : signal is "LVCMOS33, 8";   -- 8mA
attribute IO_TYPES of DA_ZEROL       : signal is "LVCMOS33, -";   -- 8mA
attribute IO_TYPES of DA_ZEROR       : signal is "LVCMOS33, -";   -- 8mA
attribute IO_TYPES of DA_DEMP        : signal is "LVCMOS33, 8";   -- 8mA
attribute IO_TYPES of DA_FMT         : signal is "LVCMOS33, 8";   -- 8mA
attribute IO_TYPES of DA_MUTE        : signal is "LVCMOS33, 8";   -- 8mA
attribute IO_TYPES of PA_nEN         : signal is "LVCMOS33, 8";   -- 8mA

-- Slew rate attributes
attribute SLEW of STATUS_RED      : signal is "SLOW";
attribute SLEW of STATUS_YELLOW   : signal is "SLOW";
attribute SLEW of STATUS_GREEN    : signal is "SLOW";
attribute SLEW of SEGMENT_LED1    : signal is "SLOW";
attribute SLEW of SEGMENT_LED2    : signal is "SLOW";
attribute SLEW of SEGMENT_LED3    : signal is "SLOW";
attribute SLEW of SEGMENT_LED4    : signal is "SLOW";
attribute SLEW of BARGRAPH_LEFT   : signal is "SLOW";
attribute SLEW of BARGRAPH_RIGHT  : signal is "SLOW";
attribute SLEW of RS232_TX        : signal is "FAST";
attribute SLEW of RS232_DTR       : signal is "FAST";
attribute SLEW of EXTENDER_B5     : signal is "FAST";
attribute SLEW of EXTENDER_B7     : signal is "FAST";
attribute SLEW of MA_ZCEN         : signal is "FAST";
attribute SLEW of MA_nDCEN        : signal is "FAST";
attribute SLEW of MA_nCS          : signal is "FAST";
attribute SLEW of MA_SCLK         : signal is "FAST";
attribute SLEW of MA_SDI          : signal is "FAST";
attribute SLEW of AD_SCKI         : signal is "FAST";
attribute SLEW of AD_LRCK         : signal is "FAST";
attribute SLEW of AD_BCK          : signal is "FAST";
attribute SLEW of AD_FMT          : signal is "SLOW";
attribute SLEW of AD_MODE         : signal is "SLOW";
attribute SLEW of AD_BYPAS        : signal is "SLOW";
attribute SLEW of AD_OSR          : signal is "SLOW";
attribute SLEW of AD_nPDWN        : signal is "SLOW";
attribute SLEW of DA_SCKI         : signal is "FAST";
attribute SLEW of DA_LRCK         : signal is "FAST";
attribute SLEW of DA_BCK          : signal is "FAST";
attribute SLEW of DA_DIN          : signal is "FAST";
attribute SLEW of DA_nRST         : signal is "FAST";
attribute SLEW of DA_DEMP         : signal is "SLOW";
attribute SLEW of DA_FMT          : signal is "SLOW";
attribute SLEW of DA_MUTE         : signal is "SLOW";
attribute SLEW of PA_nEN          : signal is "SLOW";
```

```
end toplevel;

architecture behavioral of toplevel is

  constant cn_data_width : natural := 8;

  component clockgen
    port (
      CLK    : in  std_logic;
      RESET  : in  std_logic;
      CLKOP  : out std_logic;
      CLKOK  : out std_logic;
      LOCK   : out std_logic);
  end component;

  component clockgen_main
    port (
      CLK    : in  std_logic;
      RESET  : in  std_logic;
      CLKOP  : out std_logic;
      CLKOK  : out std_logic;
      LOCK   : out std_logic);
  end component;

  signal sl_clk_98m304, sl_clk_49m152, sl_clk_768k : std_logic;
  signal sl_clk_lock                               : std_logic;

  component reset_gen
    generic (
      CLK_FREQ        : integer;
      RESET_HOLD_TIME : integer);
    port (
      CLK     : in  std_logic;
      nRESET  : out std_logic;
      RESET   : out std_logic);
  end component;

  signal sl_reset, sl_nreset : std_logic;

  component i2s_receiver
    generic (
      DATA_WIDTH : positive);
    port (
```

```
    CLK_IN : in  std_logic;
    nRESET : in  std_logic;
    LRCK   : in  std_logic;
    BCK    : in  std_logic;
    DIN    : in  std_logic;
    DOUTL  : out std_logic_vector(DATA_WIDTH-1 downto 0);
    DOUTR  : out std_logic_vector(DATA_WIDTH-1 downto 0);
    DVAL   : out std_logic);
end component;

signal sv_ad_doutl, sv_ad_doutr : std_logic_vector(23 downto 0);
signal sl_ad_dval               : std_logic;

component i2s_transmitter
  generic (
    DATA_WIDTH     : positive;
    CLK_IN_PER_BCK : positive;
    BCK_PER_LRCK   : positive);
  port (
    CLK_IN : in  std_logic;
    nRESET : in  std_logic;
    LRCK   : out std_logic;
    BCK    : out std_logic;
    DOUT   : out std_logic;
    DINL   : in  std_logic_vector(DATA_WIDTH-1 downto 0);
    DINR   : in  std_logic_vector(DATA_WIDTH-1 downto 0);
    DVAL   : in  std_logic);
end component;

signal sv_da_doutl, sv_da_doutr : std_logic_vector(23 downto 0);
signal sl_da_dval               : std_logic;

component gain_control
  generic (
    CLK_FREQ : positive;
    CLK_MAX  : positive);
  port (
    CLK_IN       : in  std_logic;
    nRESET       : in  std_logic;
    GAIN         : in  std_logic_vector(5 downto 0);
    GPO          : in  std_logic_vector(3 downto 0);
    DC_SERVO_EN  : in  std_logic;
    CM_SERVO_EN  : in  std_logic;
    OVERLOAD     : in  std_logic;
```

```
    SCKL        : out std_logic;
    nCS         : out std_logic;
    DOUT        : out std_logic;
    DIN         : in  std_logic);
end component;

signal sv_ma_gain : std_logic_vector(5 downto 0);

component chatter_suppress
  generic (
    BUTTON_COUNT    : positive;
    SUPPRESS_CLOCKS : positive);
  port (
    CLK_IN : in  std_logic;
    nRESET : in  std_logic;
    INPUT  : in  std_logic_vector(BUTTON_COUNT-1 downto 0);
    OUTPUT : out std_logic_vector(BUTTON_COUNT-1 downto 0));
end component;

signal sv_buttons, sv_switches : std_logic_vector(7 downto 0);

component bargraph_decoder
  generic (
    BAR_LIGHT_COUNT   : positive;
    OUTPUT_ACTIVE_LOW : boolean);
  port (
    CLK_IN : in  std_logic;
    nRESET : in  std_logic;
    OE     : in  std_logic;
    DEC_EN : in  std_logic;
    INPUT  : in  std_logic_vector(7 downto 0);
    OUTPUT : out std_logic_vector(7 downto 0));
end component;

signal sv_bargraph_left, sv_bargraph_right : std_logic_vector(BARGRAPH_LEFT'range);
signal sl_bargraph_dec_en                  : std_logic;

component segment_decoder
  generic (
    OUTPUT_ACTIVE_LOW : boolean;
    REGISTER_OUTPUTS  : boolean);
  port (
    CLK_IN : in  std_logic;
    nRESET : in  std_logic;
```

```
    OE      : in  std_logic;
    INPUT   : in  std_logic_vector(3 downto 0);
    OUTPUT  : out std_logic_vector(7 downto 0));
end component;

signal sv_segment_led1, sv_segment_led2, sv_segment_led3, sv_segment_led4 :
  std_logic_vector(3 downto 0);

component stud_toplevel
  generic (
    DATA_WIDTH : positive);
  port (
    CLK_FAST        : in  std_logic;
    CLK_SLOW        : in  std_logic;
    nRESET          : in  std_logic;
    SWITCH_1        : in  std_logic;
    SWITCH_2        : in  std_logic;
    SWITCH_3        : in  std_logic;
    SWITCH_4        : in  std_logic;
    SWITCH_5        : in  std_logic;
    SWITCH_6        : in  std_logic;
    SWITCH_7        : in  std_logic;
    SWITCH_8        : in  std_logic;
    BUTTON_1        : in  std_logic;
    BUTTON_2        : in  std_logic;
    BUTTON_3        : in  std_logic;
    BUTTON_4        : in  std_logic;
    BUTTON_5        : in  std_logic;
    BUTTON_6        : in  std_logic;
    BUTTON_7        : in  std_logic;
    BUTTON_8        : in  std_logic;
    STATUS_L_RED    : out std_logic;
    STATUS_L_YEL    : out std_logic;
    STATUS_L_GRE    : out std_logic;
    STATUS_M_RED    : out std_logic;
    STATUS_M_YEL    : out std_logic;
    STATUS_M_GRE    : out std_logic;
    STATUS_R_RED    : out std_logic;
    STATUS_R_YEL    : out std_logic;
    STATUS_R_GRE    : out std_logic;
    SEGMENT_LED1    : out std_logic_vector(3 downto 0);
    SEGMENT_LED2    : out std_logic_vector(3 downto 0);
    SEGMENT_LED3    : out std_logic_vector(3 downto 0);
    SEGMENT_LED4    : out std_logic_vector(3 downto 0);
```

```
      BARGRAPH_LEFT    : out std_logic_vector(7 downto 0);
      BARGRAPH_RIGHT   : out std_logic_vector(7 downto 0);
      BARGRAPH_DEC_EN  : out std_logic;
      PC_SDIN          : in  std_logic;
      PC_SDOUT         : out std_logic;
      MA_GAIN          : out std_logic_vector(5 downto 0);
      AD_PDIN_L        : in  std_logic_vector(23 downto 0);
      AD_PDIN_R        : in  std_logic_vector(23 downto 0);
      AD_DVAL          : in  std_logic;
      DA_PDOUT_L       : out std_logic_vector(23 downto 0);
      DA_PDOUT_R       : out std_logic_vector(23 downto 0);
      DA_DVAL          : out std_logic);
  end component;

begin  -- behavioral

  clockgen_1 : clockgen
    port map (
      CLK    => CLK,
      RESET => '0',
      CLKOP => open,
      CLKOK => sl_clk_768k,
      LOCK  => open);

  clockgen_main_1 : clockgen_main
    port map (
      CLK    => CLK,
      RESET => '0',
      CLKOP => sl_clk_98m304,
      CLKOK => sl_clk_49m152,
      LOCK  => sl_clk_lock);

  reset_gen_1 : reset_gen
    generic map (
      CLK_FREQ        => 32768000,
      RESET_HOLD_TIME => 50)
    port map (
      CLK     => CLK,
      nRESET => sl_nreset,
      RESET  => sl_reset);

  DA_SCKI <= sl_clk_49m152;
  AD_SCKI <= sl_clk_49m152;
```

```
i2s_receiver_1 : i2s_receiver
  generic map (
    DATA_WIDTH => 24)
  port map (
    CLK_IN => sl_clk_49m152,
    nRESET => sl_nreset,
    LRCK   => AD_LRCK,
    BCK    => AD_BCK,
    DIN    => AD_DOUT,
    DOUTL  => sv_ad_doutl,
    DOUTR  => sv_ad_doutr,
    DVAL   => sl_ad_dval);

i2s_transmitter_1 : i2s_transmitter
  generic map (
    DATA_WIDTH      => 24,
    CLK_IN_PER_BCK => 32,
    BCK_PER_LRCK    => 8)
  port map (
    CLK_IN => sl_clk_49m152,
    nRESET => sl_nreset,
    LRCK   => DA_LRCK,
    BCK    => DA_BCK,
    DOUT   => DA_DIN,
    DINL   => sv_da_doutl,
    DINR   => sv_da_doutr,
    DVAL   => sl_da_dval);

gain_control_1 : gain_control
  generic map (
    CLK_FREQ => 49152000,
    CLK_MAX  => 6250000)
  port map (
    CLK_IN       => sl_clk_49m152,
    nRESET       => sl_nreset,
    GAIN         => sv_ma_gain,
    GPO          => "0000",
    DC_SERVO_EN => '1',
    CM_SERVO_EN => '1',
    OVERLOAD     => '1',
    SCKL         => MA_SCLK,
    nCS          => MA_nCS,
    DOUT         => MA_SDI,
    DIN          => MA_SDO);
```

```
MA_ZCEN  <= '1';
MA_nDCEN <= '0';

chatter_suppress_1 : chatter_suppress
  generic map (
    BUTTON_COUNT    => 8,
    SUPPRESS_CLOCKS => 64)
  port map (
    CLK_IN => sl_clk_49m152,
    nRESET => sl_nreset,
    INPUT  => BUTTONS,
    OUTPUT => sv_buttons);

chatter_suppress_2 : chatter_suppress
  generic map (
    BUTTON_COUNT    => 8,
    SUPPRESS_CLOCKS => 64)
  port map (
    CLK_IN => sl_clk_49m152,
    nRESET => sl_nreset,
    INPUT  => SWITCHES,
    OUTPUT => sv_switches);

bargraph_decoder_1 : bargraph_decoder
  generic map (
    BAR_LIGHT_COUNT   => 1,
    OUTPUT_ACTIVE_LOW => true)
  port map (
    CLK_IN => sl_clk_49m152,
    nRESET => sl_nreset,
    OE     => '1',
    DEC_EN => sl_bargraph_dec_en,
    INPUT  => sv_bargraph_left,
    OUTPUT => BARGRAPH_LEFT);

bargraph_decoder_2 : bargraph_decoder
  generic map (
    BAR_LIGHT_COUNT   => 1,
    OUTPUT_ACTIVE_LOW => true)
  port map (
    CLK_IN => sl_clk_49m152,
    nRESET => sl_nreset,
    OE     => '1',
```

```
      DEC_EN => sl_bargraph_dec_en,
      INPUT  => sv_bargraph_right,
      OUTPUT => BARGRAPH_RIGHT);

  segment_decoder_1 : segment_decoder
    generic map (
      OUTPUT_ACTIVE_LOW => true,
      REGISTER_OUTPUTS  => true)
    port map (
      CLK_IN => sl_clk_49m152,
      nRESET => sl_nreset,
      OE     => '1',
      INPUT  => sv_segment_led1,
      OUTPUT => SEGMENT_LED1);

  segment_decoder_2 : segment_decoder
    generic map (
      OUTPUT_ACTIVE_LOW => true,
      REGISTER_OUTPUTS  => true)
    port map (
      CLK_IN => sl_clk_49m152,
      nRESET => sl_nreset,
      OE     => '1',
      INPUT  => sv_segment_led2,
      OUTPUT => SEGMENT_LED2);

  segment_decoder_3 : segment_decoder
    generic map (
      OUTPUT_ACTIVE_LOW => true,
      REGISTER_OUTPUTS  => true)
    port map (
      CLK_IN => sl_clk_49m152,
      nRESET => sl_nreset,
      OE     => '1',
      INPUT  => sv_segment_led3,
      OUTPUT => SEGMENT_LED3);

  segment_decoder_4 : segment_decoder
    generic map (
      OUTPUT_ACTIVE_LOW => true,
      REGISTER_OUTPUTS  => true)
    port map (
      CLK_IN => sl_clk_49m152,
      nRESET => sl_nreset,
```

```
    OE      => '1',
    INPUT   => sv_segment_led4,
    OUTPUT => SEGMENT_LED4);

stud_toplevel_1 : stud_toplevel
  generic map (
    DATA_WIDTH => cn_data_width)
  port map (
    CLK_FAST         => sl_clk_49m152,
    CLK_SLOW         => sl_clk_768k,
    nRESET           => sl_nreset,
    SWITCH_1         => sv_switches(0),
    SWITCH_2         => sv_switches(1),
    SWITCH_3         => sv_switches(2),
    SWITCH_4         => sv_switches(3),
    SWITCH_5         => sv_switches(4),
    SWITCH_6         => sv_switches(5),
    SWITCH_7         => sv_switches(6),
    SWITCH_8         => sv_switches(7),
    BUTTON_1         => sv_buttons(0),
    BUTTON_2         => sv_buttons(1),
    BUTTON_3         => sv_buttons(2),
    BUTTON_4         => sv_buttons(3),
    BUTTON_5         => sv_buttons(4),
    BUTTON_6         => sv_buttons(5),
    BUTTON_7         => sv_buttons(6),
    BUTTON_8         => sv_buttons(7),
    STATUS_L_RED     => STATUS_RED(0),
    STATUS_L_YEL     => STATUS_YELLOW(0),
    STATUS_L_GRE     => STATUS_GREEN(0),
    STATUS_M_RED     => STATUS_RED(1),
    STATUS_M_YEL     => STATUS_YELLOW(1),
    STATUS_M_GRE     => STATUS_GREEN(1),
    STATUS_R_RED     => STATUS_RED(2),
    STATUS_R_YEL     => STATUS_YELLOW(2),
    STATUS_R_GRE     => STATUS_GREEN(2),
    SEGMENT_LED1     => sv_segment_led1,
    SEGMENT_LED2     => sv_segment_led2,
    SEGMENT_LED3     => sv_segment_led3,
    SEGMENT_LED4     => sv_segment_led4,
    BARGRAPH_LEFT    => sv_bargraph_left,
    BARGRAPH_RIGHT   => sv_bargraph_right,
    BARGRAPH_DEC_EN => sl_bargraph_dec_en,
    PC_SDIN          => RS232_RX,
```

```
        PC_SDOUT        => RS232_TX,
        MA_GAIN         => sv_ma_gain,
        AD_PDIN_L       => sv_ad_doutl,
        AD_PDIN_R       => sv_ad_doutr,
        AD_DVAL         => sl_ad_dval,
        DA_PDOUT_L      => sv_da_doutl,
        DA_PDOUT_R      => sv_da_doutr,
        DA_DVAL         => sl_da_dval);

end behavioral;
```

B.1.2 Barrel Shifter

Listing B.2: barrel_shifter.vhd

```
-------------------------------------------------------------------------------
-- Title      : Barrel Shifter
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : barrel_shifter.vhd
-- Author     : Daniel Glaser
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-11-01
-- Last update: 2007-01-22
-- Platform   : LFECP20E
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description: This module is for multiplications and divisions by powers of
--              two. Left side shifting is done by positive, right side by
--              negative values of SHIFT_AMOUNT.
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-11-01  1.0      sidaglas        Created
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
use ieee.numeric_std.all;

entity barrel_shifter is
```

```
  generic (
    USE_MULTIPLIER : boolean  := true;
    AMOUNT_WIDTH   : positive := 4;
    DATA_WIDTH     : positive := 8);

  port (
    SIGNED_SHIFT : in  std_logic;
    SHIFT_AMOUNT : in  std_logic_vector(AMOUNT_WIDTH-1 downto 0);
    DATA_IN      : in  std_logic_vector(DATA_WIDTH-1 downto 0);
    DATA_OUT     : out std_logic_vector(DATA_WIDTH-1 downto 0));

end barrel_shifter;

architecture behavioral of barrel_shifter is

  signal si_amount : integer range -DATA_WIDTH to DATA_WIDTH-1;
--  signal sv_amount_pow : std_logic_vector((2*DATA_WIDTH)-1 downto 0);
  type   tva_pow is array (0 to (2*DATA_WIDTH)-1)
    of std_logic_vector((2*DATA_WIDTH)-1 downto 0);
  signal sva_amount_pows : tva_pow;
  signal sv_multi        : std_logic_vector((3*DATA_WIDTH)-2 downto 0);
  signal sv_multi_datab  : std_logic_vector((2*DATA_WIDTH)-1 downto 0);
  signal si_mult1, si_mult2 : integer;

begin  -- behavioral
  si_amount <= conv_integer(SHIFT_AMOUNT);

  gen_no_multiplier : if not USE_MULTIPLIER generate
    gen_bits : for i in DATA_WIDTH downto 0 generate
      DATA_OUT(i) <=
        DATA_IN(i - si_amount)      when ((i-si_amount >= 0) and (i-si_amount <= DATA_WID
        else DATA_IN(DATA_IN'left) when SIGNED_SHIFT = '1'
        else '0';
    end generate gen_bits;
  end generate gen_no_multiplier;

  gen_with_multiplier : if USE_MULTIPLIER generate

    gen_pows : for i in 0 to (2*DATA_WIDTH)-1 generate
      sva_amount_pows(i) <= conv_std_logic_vector(2**i, 2*DATA_WIDTH);
    end generate gen_pows;

    si_mult1 <= conv_integer((SIGNED_SHIFT and DATA_IN(DATA_WIDTH-1)) & DATA_IN);
    si_mult2 <= conv_integer(sva_amount_pows(si_amount + DATA_WIDTH));
```

```
    sv_multi <= conv_std_logic_vector(si_mult1 * si_mult2,(3*DATA_WIDTH)-1);

    DATA_OUT <= sv_multi((2*DATA_WIDTH)-1 downto DATA_WIDTH);
  end generate gen_with_multiplier;

end behavioral;
```

B.1.3 Initialisierung

Listing B.3: reset_gen.vhd

```
-------------------------------------------------------------------------------
-- Title      : reset generator
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
--              This Project aims at applicating the knowledge, collected in
--              the lecture ADS. Students should learn, how to use MATLAB for
--              dimensioning digital signal processing architectures and how
--              to implement this algorithms and architectures into hardware,
--              in this case some Lattice ECP20-FPGA.
-------------------------------------------------------------------------------
-- File       : reset_gen.vhd
-- Author     : Daniel Glaser
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-09-01
-- Last update: 2007-01-14
-- Platform   : LFECP20E
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description: Generates a reset pulse at the beginning right after power is
--              applied and configuration is done. It's implemented as a
--              simple counter that counts up some time and then simply stops
--              counting while taking back reset. This module expects an
--              initial value of the reset counter beeing (others => '0'). This
--              is given e.g. for Xilinx Virtex-Family.
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-09-01  1.0      sidaglas        Created
-- 2006-11-23  1.1      sidaglas        Fixed some comments
-------------------------------------------------------------------------------

library ieee;
```

```
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity reset_gen is

  generic (
    CLK_FREQ         : integer := 49152000;  -- The clock frequency in Hz
    RESET_HOLD_TIME : integer := 50);        -- Time in milliseconds

  port (
    CLK     : in  std_logic;
    nRESET : out std_logic;
    RESET   : out std_logic);

end reset_gen;

architecture behavioral of reset_gen is

begin  -- behavioral

  -- purpose: This process is intended to count the clocks right after
  --          configuration and then releases the reset signal to inactive state
  -- type   : sequential
  -- inputs : CLK
  -- outputs:
  reset_count : process (CLK)
    variable vn_count : natural range 0 to (RESET_HOLD_TIME*(CLK_FREQ/1000) - 1):= 0;
  begin  -- process reset_count
    if rising_edge(CLK) then            -- rising clock edge
      if vn_count = (RESET_HOLD_TIME*(CLK_FREQ/1000)-1) then
        RESET <= '0';
        nRESET <= '1';
      else
        RESET <= '1';
        nRESET  <= '0';
        vn_count := vn_count + 1;
      end if;
    end if;
  end process reset_count;

end behavioral;
```

B.1.4 Tasterentprellung

Listing B.4: chatter_suppress.vhd

```
-------------------------------------------------------------------------------
-- Title      : Chatter suppress
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : chatter_suppress.vhd
-- Author     : Daniel Glaser
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-09-04
-- Last update: 2007-01-11
-- Platform   : LFECP20E
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description: This module suppresses the chatter from switches and buttons.
--              It is configurable in swing suppress time.
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-09-04  1.0      sidaglas        Created
-- 2006-11-23  1.1      sidaglas        Fixed some comments
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;

entity chatter_suppress is

  generic (
    BUTTON_COUNT    : positive := 8;
    SUPPRESS_CLOCKS : positive := 64);

  port (
    CLK_IN : in  std_logic;
    nRESET : in  std_logic;
    INPUT  : in  std_logic_vector(BUTTON_COUNT-1 downto 0);
    OUTPUT : out std_logic_vector(BUTTON_COUNT-1 downto 0));

end chatter_suppress;

architecture behavioral of chatter_suppress is
```

```
begin  -- behavioral

  gen_proc_button : for i in 0 to BUTTON_COUNT-1 generate

    -- purpose: This process suppresses the chatter of one button
    -- type   : sequential
    -- inputs : CLK_IN, nRESET
    -- outputs:
    chatter_suppress : process (CLK_IN, nRESET, INPUT)
      variable vn_count : natural range 0 to SUPPRESS_CLOCKS-1 := SUPPRESS_CLOCKS-1;
      variable vl_prev  : std_logic                            := '0';
    begin  -- process chatter_suppress
      if nRESET = '0' then               -- asynchronous reset (active low)
        OUTPUT(i) <= '0';
        vl_prev   := INPUT(i);
        vn_count  := SUPPRESS_CLOCKS-1;
      elsif rising_edge(CLK_IN) then     -- rising clock edge

        if vl_prev /= INPUT(i) then
          vl_prev  := INPUT(i);
          vn_count := SUPPRESS_CLOCKS-1;
        elsif vn_count = 0 then
          OUTPUT(i) <= INPUT(i);
        else
          vn_count := vn_count - 1;
        end if;

      end if;
    end process chatter_suppress;

  end generate gen_proc_button;

end behavioral;
```

B.1.5 I2S-Empfänger

Listing B.5: i2s_receiver.vhd

```
-------------------------------------------------------------------------------
-- Title      : i2c receiver
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : i2s_receiver.vhd
-- Author     : Daniel Glaser
```

```
-- Company     : LTE, FAU Erlangen-Nuremberg, Germany
-- Created     : 2006-09-01
-- Last update: 2007-01-13
-- Platform    : LFECP20E
-- Standard    : VHDL'87
-------------------------------------------------------------------------------
-- Description: The task of this module is to receive serial adc data and
--              present it in parallel to the toplevel
--
-- Information: Give this module some I2S circuit in master mode and it will
--              receive the stereo audio information from it. CLK_IN must be at
--              least twice BCK frequency for proper function. The DVAL output
--              toogles each time, new data is incoming. It has the same
--              meaning as LRCK. If low LRCK means left channel data from audio
--              circuit, left parallel data is just valid, when dval goes down.
--              Left channel parallel data is valid as long as DVAL stays low.
--              It is valid until short before (3 BCK cycles) next high
--              to low edge. This allows to sample left and right data as well
--              at high to low edge of DVAL and vice versa.
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author         Description
-- 2006-09-01  1.0      sidaglas       Created
-- 2006-11-23  1.1      sidaglas       Fixed some comments
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;

entity i2s_receiver is

  generic (
    DATA_WIDTH : positive := 24);

  port (
    CLK_IN : in  std_logic;
    nRESET : in  std_logic;
    LRCK   : in  std_logic;
    BCK    : in  std_logic;
    DIN    : in  std_logic;
```

```
    DOUTL   : out std_logic_vector(DATA_WIDTH-1 downto 0);
    DOUTR   : out std_logic_vector(DATA_WIDTH-1 downto 0);
    DVAL    : out std_logic);

end i2s_receiver;

architecture behavioral of i2s_receiver is

  signal sl_val_l, sl_val_r : std_logic;

  -- pragma translate_off
  signal sn_count_l, sn_count_r       : natural;
  signal sv_shiftreg_l, sv_shiftreg_r : std_logic_vector(DATA_WIDTH-1 downto 0);
  -- pragma translate_on

begin  -- behavioral

  -- purpose: This process handles the receiption of the left channel data
  -- type   : sequential
  -- inputs : CLK_IN, nRESET
  -- outputs:
  rec_left : process (CLK_IN, nRESET)
    variable vv_shiftreg            : std_logic_vector(DATA_WIDTH-1 downto 0);
    variable vn_count               : natural range 0 to DATA_WIDTH;
    variable vl_bck, vl_lrck, vl_val : std_logic;

  begin  -- process rec_left
    if nRESET = '0' then                -- asynchronous reset (active low)

      vv_shiftreg := (others => '0');
      sl_val_l    <= '0';
      vl_val      := '0';
      vl_bck      := '0';
      vl_lrck     := '0';
      DOUTL       <= (others => '0');
      vn_count    := 0;

    elsif rising_edge(CLK_IN) then      -- rising clock edge

      -- This defines the rising clock edge of BCK
      if vl_bck = '0' and BCK = '1' then

        if vn_count /= 0 then
          vv_shiftreg(DATA_WIDTH-1 downto 1) := vv_shiftreg(DATA_WIDTH-2 downto 0);
```

```
        vv_shiftreg(0)                    := DIN;
      end if;

      -- The sequence LRCK '1' => '0' defines the left channel
      if vn_count = 0 and LRCK = '0' and vl_lrck = '1' then
        vn_count := vn_count + 1;
      elsif vn_count = DATA_WIDTH then
        vn_count := 0;
      elsif vn_count /= 0 then
        vn_count := vn_count + 1;
      end if;

      if LRCK = '1' and vl_lrck = '0' then
        sl_val_l <= '0';
      elsif sl_val_l = '0' and vl_val = '1' then
        DOUTL <= vv_shiftreg;
      elsif vl_val = '0' then
        sl_val_l <= '1';
      end if;

      vl_val := sl_val_l;

      -- pragma translate_off
      sn_count_l    <= vn_count;
      sv_shiftreg_l <= vv_shiftreg;
      -- pragma translate_on

      vl_lrck := LRCK;

    end if;

    vl_bck := BCK;

  end if;

end process rec_left;

-- purpose: This process handles the receiption of the right channel data
-- type   : sequential
-- inputs : CLK_IN, nRESET
-- outputs:
rec_right : process (CLK_IN, nRESET)
  variable vv_shiftreg            : std_logic_vector(DATA_WIDTH-1 downto 0);
  variable vn_count               : natural range 0 to DATA_WIDTH;
```

```
  variable vl_bck, vl_lrck, vl_val : std_logic;

begin  -- process rec_left
  if nRESET = '0' then                  -- asynchronous reset (active low)

    vv_shiftreg := (others => '0');
    sl_val_r    <= '0';
    vl_bck      := '0';
    vl_val      := '0';
    vl_lrck     := '0';
    DOUTR       <= (others => '0');
    vn_count    := 0;

  elsif rising_edge(CLK_IN) then        -- rising clock edge

    -- This defines the rising clock edge of BCK
    if vl_bck = '0' and BCK = '1' then

      if vn_count /= 0 then
        vv_shiftreg(DATA_WIDTH-1 downto 1) := vv_shiftreg(DATA_WIDTH-2 downto 0);
        vv_shiftreg(0)                     := DIN;
      end if;

      -- The sequence LRCK '0' => '1' defines the right channel
      if vn_count = 0 and LRCK = '1' and vl_lrck = '0' then
        vn_count := vn_count + 1;
      elsif vn_count = DATA_WIDTH then
        vn_count := 0;
      elsif vn_count /= 0 then
        vn_count := vn_count + 1;
      end if;

      if LRCK = '0' and vl_lrck = '1' then
        sl_val_r <= '0';
      elsif sl_val_r = '0' and vl_val = '1' then
        DOUTR <= vv_shiftreg;
      elsif vl_val = '0' then
        sl_val_r <= '1';
      end if;

      vl_val := sl_val_r;

      -- pragma translate_off
      sn_count_r    <= vn_count;
```

```
        sv_shiftreg_r <= vv_shiftreg;
        -- pragma translate_on

        vl_lrck := LRCK;

      end if;

      vl_bck := BCK;

    end if;

  end process rec_right;

  -- purpose: Makes the valid signal DVAL changing level each time a channel
  --          has new data according to the finished data (e.g. left means low)
  -- type   : sequential
  -- inputs : CLK, nRESET
  -- outputs:
  valid_toggle : process (CLK_IN, nRESET)
    variable vl_dval : std_logic;
  begin  -- process valid_toggle
    if nRESET = '0' then                -- asynchronous reset (active low)
      DVAL    <= '0';
      vl_dval := '0';
    elsif rising_edge(CLK_IN) then      -- rising clock edge
      if (sl_val_l and sl_val_r) = '1' and vl_dval = '0' then
        DVAL <= not LRCK;
      end if;
      vl_dval := sl_val_l and sl_val_r;
    end if;
  end process valid_toggle;

end behavioral;
```

B.1.6 I2S-Sender

Listing B.6: i2s_transmitter.vhd

```
-------------------------------------------------------------------------------
-- Title      : i2s transmitter
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : i2s_transmitter.vhd
-- Author     : Daniel Glaser
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
```

```
-- Created    : 2006-09-04
-- Last update: 2007-01-13
-- Platform   : LFECP20E
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description: The task of this module is to receive serial adc data and
--              present it in parallel to the toplevel
--
-- Information: Give this module some I2S circuit in slave mode and it will
--              transmit the stereo audio information to it. CLK_IN must be at
--              least twice BCK frequency for proper function. If DVAL input
--              toogles, new data is accepted. It has the same meaning as LRCK.
--              If low LRCK means left channel data from audio circuit, left
--              parallel data is just valid, when dval goes down.
--              If no new data arrives, before next transmission cycle,
--              transmission continues with old data to not disturb proper
--              protocol function.
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author         Description
-- 2006-09-01  1.0      sidaglas       Created
-- 2006-11-23  1.1      sidaglas       Fixed some comments
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;

entity i2s_transmitter is

  generic (
    DATA_WIDTH      : positive := 24;
    CLK_IN_PER_BCK : positive := 8;
    BCK_PER_LRCK    : positive := 64);

  port (
    CLK_IN : in  std_logic;
    nRESET : in  std_logic;
    LRCK   : out std_logic;
    BCK    : out std_logic;
    DOUT   : out std_logic;
```

```
    DINL    : in  std_logic_vector(DATA_WIDTH-1 downto 0);
    DINR    : in  std_logic_vector(DATA_WIDTH-1 downto 0);
    DVAL    : in  std_logic);

end i2s_transmitter;

architecture behavioral of i2s_transmitter is

  signal sl_dout                  : std_logic;
  signal sl_bck, sl_lrck, sl_sync : std_logic;
  signal sv_inreg_l, sv_inreg_r   : std_logic_vector(DATA_WIDTH-1 downto 0);

  -- pragma translate_off
  signal sn_count_bck, sn_count_lrck, sn_count_transmit : natural;
  signal sv_shiftreg                                    : std_logic_vector(DATA_WIDTH-1
  -- pragma translate_on

begin  -- behavioral

  -- purpose: This process stores the incoming data in some register
  -- type   : sequential
  -- inputs : CLK_IN, nRESET
  -- outputs:
  reg_inputs : process (CLK_IN, nRESET)
    variable vl_dval : std_logic := '0';
  begin  -- process reg_inputs
    if nRESET = '0' then                -- asynchronous reset (active low)

      sv_inreg_l <= (others => '0');
      sv_inreg_r <= (others => '0');
      vl_dval := '0';

    elsif rising_edge(CLK_IN) then      -- rising clock edge

      if DVAL /= vl_dval then
        sv_inreg_l <= DINL;
        sv_inreg_r <= DINR;
      end if;

      vl_dval := DVAL;

    end if;
  end process reg_inputs;
```

```
-- purpose: This process generates the BCK signal
-- type   : sequential
-- inputs : CLK_IN, nRESET
-- outputs: sl_bck    This is the bitclock for DAC
--          sl_sync   This toggles one CLK_IN before bck falling edge
bck_gen : process (CLK_IN, nRESET)
  variable vn_count : natural range 0 to CLK_IN_PER_BCK-1 := 0;
begin  -- process bck_gen
  if nRESET = '0' then                  -- asynchronous reset (active low)

    sl_bck   <= '0';
    sl_sync  <= '0';
    vn_count := 0;

  elsif rising_edge(CLK_IN) then        -- rising clock edge

    if vn_count = 0 then
      sl_bck <= '0';
    elsif vn_count = 1 then
      sl_sync <= not sl_sync;
    elsif vn_count = (CLK_IN_PER_BCK/2) then
      sl_bck <= '1';
    end if;

    if vn_count = 0 then
      vn_count := CLK_IN_PER_BCK-1;
    else
      vn_count := vn_count - 1;
    end if;

    -- pragma translate_off
    sn_count_bck <= vn_count;
    -- pragma translate_on

  end if;
end process bck_gen;

BCK <= sl_bck;

-- purpose: This process generates the lrck signal
-- type   : sequential
-- inputs : CLK_IN, nRESET
--          sl_sync
-- outputs: sl_lrck
```

```
lrck_gen : process (CLK_IN, nRESET)
  variable vn_count : natural range 0 to BCK_PER_LRCK-1;
  variable vl_sync  : std_logic := '0';
begin  -- process lrck_gen
  if nRESET = '0' then                  -- asynchronous reset (active low)

    sl_lrck  <= '0';
    vl_sync  := '0';
    vn_count := 0;
    LRCK     <= '0';

  elsif rising_edge(CLK_IN) then        -- rising clock edge

    if (sl_sync xor vl_sync) = '1' then

      if vn_count = 0 then
        sl_lrck <= '0';                 -- '1'->'0' identifies left channel
      elsif vn_count = (BCK_PER_LRCK/2 - 1) then
        sl_lrck <= '1';                 -- '0'->'1' identifies right channel
      end if;

      if vn_count = 0 then
        vn_count := BCK_PER_LRCK-1;
      else
        vn_count := vn_count - 1;
      end if;

      vl_sync := sl_sync;

      LRCK <= sl_lrck;                  -- This will be delayed one bck cycle

    end if;

    -- pragma translate_off
    sn_count_lrck <= vn_count;
    -- pragma translate_on

  end if;
end process lrck_gen;

-- purpose: This process shifts the registers and puts out the serial data
-- type   : sequential
-- inputs : CLK_IN, nRESET,
--          sv_shiftreg_l, sv_shiftreg_r, sl_sync, sl_lrck
```

```
-- outputs: sl_dout
shift_regs_and_output : process (CLK_IN, nRESET)
  variable vv_shiftreg      : std_logic_vector(DATA_WIDTH-1 downto 0);
  variable vl_sync, vl_lrck : std_logic                      := '0';
  variable vn_count         : natural range 0 to DATA_WIDTH-1 := 0;
begin  -- process shift_regs
  if nRESET = '0' then                  -- asynchronous reset (active low)

    vv_shiftreg := (others => '0');
    vl_sync     := '0';
    vn_count    := 0;
    vl_lrck     := '0';
    sl_dout     <= '0';

  elsif rising_edge(CLK_IN) then        -- rising clock edge

    if (sl_sync xor vl_sync) = '1' then
      -- We are only active when BCK does '1' -> '0'


      if vn_count = 0 then
        -- We have finished transmission and return signal to '0' waiting
        -- for another transfer cycle
        if sl_lrck = '0' and vl_lrck = '1' then
          -- We will copy the left regin to shiftreg

          vv_shiftreg := sv_inreg_l;
          vn_count    := DATA_WIDTH-1;

        elsif sl_lrck = '1' and vl_lrck = '0' then
          -- We will copy the right regin to shiftreg

          vv_shiftreg := sv_inreg_r;
          vn_count    := DATA_WIDTH-1;

        end if;

        -- pragma translate_off
        sv_shiftreg <= vv_shiftreg;
        -- pragma translate_on

        sl_dout <= '0';

      elsif vn_count /= 0 then
```

```
          -- We put out the topmost bit and shift the register

          sl_dout                                <= vv_shiftreg(DATA_WIDTH-1);
          vv_shiftreg(DATA_WIDTH-1 downto 0) := vv_shiftreg(DATA_WIDTH-2 downto 0) & '0
          vn_count                               := vn_count - 1;

        end if;

        vl_lrck := sl_lrck;
        vl_sync := sl_sync;

      end if;

      -- pragma translate_off
      sn_count_transmit <= vn_count;
      -- pragma translate_on

    end if;
  end process shift_regs_and_output;

  DOUT <= sl_dout;

end behavioral;
```

B.1.7 Mikrofonvorvrstärkungseinstellung

Listing B.7: gain_control.vhd

```
-------------------------------------------------------------------------------
-- Title      : Gain Control
-- Project    :  Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : gain_control.vhd
-- Author     : Daniel Glaser
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-11-22
-- Last update: 2007-01-13
-- Platform   : LFECP30E
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description: This module compares the inputs to it's previous values and if
--              they change, it configures the Mic PreAmp with the new values.
--              It also generates the apropriate clock and nCS-signal.
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
```

```
-- Revisions   :
-- Date        Version  Author          Description
-- 2006-11-23  1.0      sidaglas        Created
--                                        SCLK=CLK_IN not fully implemented yet
```

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

entity gain_control is

  generic (
    CLK_FREQ : positive := 49152000;
    CLK_MAX  : positive := 6250000);
  port (
    -- System Inputs
    CLK_IN       : in  std_logic;
    nRESET       : in  std_logic;
    -- Preferences Input
    GAIN         : in  std_logic_vector(5 downto 0);
    GPO          : in  std_logic_vector(3 downto 0);
    DC_SERVO_EN  : in  std_logic;
    CM_SERVO_EN  : in  std_logic;
    OVERLOAD     : in  std_logic;
    -- IC-Connections
    SCKL         : out std_logic;
    nCS          : out std_logic;
    DOUT         : out std_logic;
    DIN          : in  std_logic);

end gain_control;

architecture behavioral of gain_control is

  constant cn_clock_divider   : natural := (CLK_FREQ/(CLK_MAX+1))+1;
  constant cn_clock_count_max : natural := (cn_clock_divider/2)+1;

  signal sl_toggle_sig              : std_logic := '0';
  signal sl_changed, sl_changed_ack : std_logic := '0';

begin  -- behavioral
```

```
proc_toggle_sig : process (CLK_IN, nRESET)
  variable cn_toggle_count : natural range 0 to cn_clock_count_max := cn_clock_count_m
begin  -- process proc_toggle_sig
  if nRESET = '0' then                  -- asynchronous reset (active low)
    sl_toggle_sig   <= '0';
    cn_toggle_count := cn_clock_count_max;
  elsif rising_edge(CLK_IN) then        -- rising clock edge
    if cn_clock_divider = 2 then
      sl_toggle_sig <= not sl_toggle_sig;
    elsif cn_clock_divider > 2 then
      if cn_toggle_count = 0 then
        cn_toggle_count := cn_clock_count_max;
        sl_toggle_sig   <= not sl_toggle_sig;
      else
        cn_toggle_count := cn_toggle_count - 1;
      end if;
    end if;
  end if;
end process proc_toggle_sig;

-- Output the clock for the Mic PreAmp
SCKL <= sl_toggle_sig when cn_clock_divider > 2 else CLK_IN;

proc_change_detect : process (CLK_IN, nRESET)
  variable vv_last_gain                       : std_logic_vector(GAIN'range);
  variable vv_last_gpo                        : std_logic_vector(GPO'range);
  variable vl_last_dc_servo, vl_last_cm_servo : std_logic;
  variable vl_last_overload                   : std_logic;

  variable vb_gain_changed     : boolean;
  variable vb_gpo_changed      : boolean;
  variable vb_servo_changed    : boolean;
  variable vb_overload_changed : boolean;
begin  -- process proc_change_detect
  if nRESET = '0' then                  -- asynchronous reset (active low)
    vv_last_gain     := (others => '0');
    vv_last_gpo      := (others => '0');
    vl_last_dc_servo := '0';
    vl_last_cm_servo := '0';
    vl_last_overload := '0';
  elsif rising_edge(CLK_IN) then        -- rising clock edge

    if vb_gain_changed or vb_gpo_changed or vb_servo_changed or vb_overload_changed th
```

```
      sl_changed <= '1';
    elsif sl_changed_ack = '1' then
      sl_changed <= '0';
    end if;

    -- Compare actual with past values
    if vv_last_gain /= GAIN then
      vb_gain_changed := true;
    else
      vb_gain_changed := false;
    end if;

    if vv_last_gpo /= GPO then
      vb_gpo_changed := true;
    else
      vb_gpo_changed := false;
    end if;

    if vl_last_dc_servo /= DC_SERVO_EN
      or vl_last_cm_servo /= CM_SERVO_EN then
      vb_servo_changed := true;
    else
      vb_servo_changed := false;
    end if;

    if vl_last_overload /= OVERLOAD then
      vb_overload_changed := true;
    else
      vb_overload_changed := false;
    end if;

    -- Assign last values to compare with next cycle
    vv_last_gain     := GAIN;
    vv_last_gpo      := GPO;
    vl_last_dc_servo := DC_SERVO_EN;
    vl_last_cm_servo := CM_SERVO_EN;
    vl_last_overload := OVERLOAD;
  end if;
end process proc_change_detect;

proc_transmit : process (CLK_IN, nRESET)
  variable vn_counter     : natural range 0 to 20;
  variable vv_snapshot    : std_logic_vector(15 downto 0);
  variable vl_last_toggle : std_logic;
```

```
  variable vl_shift_en     : std_logic;
begin  -- process transmit
  if nRESET = '0' then                  -- asynchronous reset (active low)
    DOUT         <= '0';
    nCS          <= '1';
    vv_snapshot := (others => '0');
  elsif rising_edge(CLK_IN) then        -- rising clock edge
    if vl_last_toggle = '1' and sl_toggle_sig = '0' then
      if sl_changed = '1' then

        if vl_shift_en = '1' then
          vv_snapshot(15 downto 0) := vv_snapshot(14 downto 0) & '0';
        else
          vv_snapshot :=
            (not DC_SERVO_EN) & CM_SERVO_EN & '0' & OVERLOAD &
            GPO & "00" & GAIN;
        end if;

        DOUT <= vv_snapshot(15);

        if vn_counter = 0 then
          -- Set data enable and put first data
          nCS          <= '0';
          vl_shift_en := '1';
        elsif vn_counter = 16 then
          nCS            <= '1';
          vl_shift_en    := '0';
          sl_changed_ack <= '1';
        end if;

        if vn_counter = 20 then
          sl_changed_ack <= '1';
        else
          vn_counter := vn_counter + 1;
        end if;

      else

        vn_counter     := 0;
        sl_changed_ack <= '0';

      end if;
    end if;
```

```
      vl_last_toggle := sl_toggle_sig;

    end if;
  end process proc_transmit;

end behavioral;
```

B.1.8 DDS-Signalgenerator

Listing B.8: dds_sinus.vhd

```
-------------------------------------------------------------------------------
-- Title      : dds sinus generator
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : dds_sinus.vhd
-- Author     : Daniel Glaser
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-09-05
-- Last update: 2007-01-15
-- Platform   :
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description: This module generates some sinusodial signal with dds algorithm
--              for use in our first exercises.
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LfTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-09-05  1.0      sidaglas        Created
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
use ieee.numeric_std.all;
use ieee.math_real.all;

entity dds_sinus is

  generic (
    OUTPUT_WIDTH : positive := 8;
    TABLE_WIDTH  : positive := 3;
    PHASE_WIDTH  : positive := 8;
```

```
    CLK_DIV      : positive := 250);

  port (
    CLK_IN     : in  std_logic;
    nRESET     : in  std_logic;
    PHASE_INC  : in  std_logic_vector(PHASE_WIDTH-1 downto 0);
    OUTPUT     : out std_logic_vector(OUTPUT_WIDTH-1 downto 0));

end dds_sinus;

architecture behavioral of dds_sinus is

  constant ci_phase_min     : integer := -(2**PHASE_WIDTH);
  constant ci_phase_max     : integer := (2**PHASE_WIDTH)-1;
  constant ci_table_min     : integer := 0;
  constant ci_table_max     : integer := (2**TABLE_WIDTH)-1;
  constant ci_output_min    : integer := 0;
  constant ci_output_max    : integer := (2**OUTPUT_WIDTH)-1;
--  constant ci_output_offset : integer := (2**(OUTPUT_WIDTH-1))-1;
  constant ci_output_offset : integer := 0;

  signal sl_div_tog : std_logic;

  type   tu_lut is array (0 to ci_table_max) of std_logic_vector(OUTPUT_WIDTH-1 downto 0
  type   tu_fsm_phase is (PHASE_RISING, PHASE_FALLING);
  signal sav_sin_lut : tu_lut;

  signal si_phase_reg : integer range ci_phase_min to ci_phase_max;


  -- pragma translate_off
  signal sn_phase1          : natural range 0 to ci_phase_max;
  signal sn_table_row       : integer;
  signal su_fsm_phase_state : tu_fsm_phase;
  signal si_test1, si_test2 : integer;
  signal sv_test            : std_logic_vector(31 downto 0);
  signal sn_count           : natural;
  -- pragma translate_on

begin  -- behavioral

  gen_sin_lut : for i in 0 to ci_table_max generate
    -- This throws an error in ispLEVER, but Synplify and Modelsim seem to
    -- deal great with this ieee.math_real_package
```

```
    sav_sin_lut(i) <= conv_std_logic_vector(
      integer(ROUND(sin((real(0.5)*MATH_PI*(real(i)+0.5))/real(ci_table_max + 1)) * real
      OUTPUT_WIDTH);
  end generate gen_sin_lut;

  -- pragma translate_off
  -- This is for debugging purposes
  proc_rows : process
  begin  -- process proc_rows
    for k in 0 to 3 loop
      for j in 0 to ci_table_max loop
        if k = 0 then
          sn_table_row <= conv_integer('0' & sav_sin_lut(j));
        elsif k = 1 then
          sn_table_row <= conv_integer('0' & sav_sin_lut(ci_table_max-j));
        elsif k = 2 then
          sn_table_row <= -conv_integer('0' & sav_sin_lut(j));
        elsif k = 3 then
          sn_table_row <= -conv_integer('0' & sav_sin_lut(ci_table_max-j));
        end if;
        wait for 200 ns;
      end loop;  -- j
    end loop;  -- k
  end process proc_rows;
  -- pragma translate_on

  -- purpose: This process divides the clock input
  -- type   : sequential
  -- inputs : CLK_IN, nRESET
  -- outputs:
  div_clk : process (CLK_IN, nRESET)
    variable vn_count : natural range 0 to CLK_DIV-1;
  begin  -- process div_clk
    if nRESET = '0' then                  -- asynchronous reset (active low)
      vn_count := CLK_DIV-1;
      sl_div_tog <= '0';
    elsif rising_edge(CLK_IN) then        -- rising clock edge
      if vn_count = 0 then
        vn_count   := CLK_DIV-1;
        sl_div_tog <= not sl_div_tog;
      else
        vn_count := vn_count - 1;
      end if;
      -- pragma translate_off
```

```
      sn_count <= vn_count;
      -- pragma translate_on
    end if;
  end process div_clk;

  -- purpose: This increments the phase of the wave, that is stored in the table
  -- type   : sequential
  -- inputs : CLK_IN, nRESET
  -- outputs:
  increment_phase : process (CLK_IN, nRESET)
    variable vu_fsm_phase_state : tu_fsm_phase;
    variable vb_second_half     : boolean := false;
    variable vl_div_tog         : std_logic;
  begin  -- process increment_phase
    if nRESET = '0' then                -- asynchronous reset (active low)

      vu_fsm_phase_state := PHASE_RISING;
      si_phase_reg       <= 0;
      vb_second_half     := false;

    elsif rising_edge(CLK_IN) then      -- rising clock edge

      if (sl_div_tog xor vl_div_tog) = '1' then

        case vu_fsm_phase_state is

          when PHASE_RISING =>
            if conv_integer(PHASE_INC) - ci_phase_max + si_phase_reg <= 0 then
              si_phase_reg <= si_phase_reg + conv_integer(PHASE_INC);
            else
              si_phase_reg       <= (2*ci_phase_max) + 1 - si_phase_reg - conv_integer(P
              vu_fsm_phase_state := PHASE_FALLING;
            end if;

          when PHASE_FALLING =>
            if conv_integer(PHASE_INC) - si_phase_reg - ci_phase_max <= 0 then
              si_phase_reg <= si_phase_reg - conv_integer(PHASE_INC);
            else
              si_phase_reg       <= conv_integer(PHASE_INC) + (2*ci_phase_min) + 1 - si_
              vu_fsm_phase_state := PHASE_RISING;
            end if;

          when others => vu_fsm_phase_state := PHASE_FALLING;
```

```
      end case;

      -- pragma translate_off
      su_fsm_phase_state <= vu_fsm_phase_state;
      -- pragma translate_on
    end if;

    vl_div_tog := sl_div_tog;

  end if;
end process increment_phase;

-- purpose: This process get's the data from the table
-- type   : sequential
-- inputs : CLK_IN, nRESET
-- outputs:
get_wave_data : process (CLK_IN, nRESET)
  variable vv_phase         : std_logic_vector(PHASE_WIDTH-1 downto 0);
  variable vv_phase_reduced : std_logic_vector(TABLE_WIDTH-1 downto 0);
  variable vn_phase         : natural range 0 to (2**PHASE_WIDTH)-1;
  variable vb_positive1     : boolean := false;
  variable vb_positive2     : boolean := false;
  variable vv_pre_output    : std_logic_vector(OUTPUT_WIDTH-1 downto 0);
begin  -- process get_wave_data
  if nRESET = '0' then                  -- asynchronous reset (active low)
    OUTPUT   <= (others => '0');
    vv_phase := (others => '0');
    vn_phase := 2**(PHASE_WIDTH-1);
    vb_positive1 := false;
    vb_positive2 := false;
  elsif rising_edge(CLK_IN) then        -- rising clock edge

    if vb_positive2 then
      OUTPUT <= conv_std_logic_vector(ci_output_offset, OUTPUT_WIDTH) + ('0' & vv_pre
    else
      OUTPUT <= conv_std_logic_vector(ci_output_offset, OUTPUT_WIDTH) - ('0' & vv_pre
    end if;

    vv_pre_output := sav_sin_lut(conv_integer('0' & vv_phase(PHASE_WIDTH-1 downto PHAS

    vv_phase := conv_std_logic_vector(abs(si_phase_reg), PHASE_WIDTH);

    -- pragma translate_off
    si_test1 <= conv_integer('0' & vv_phase(PHASE_WIDTH-1 downto PHASE_WIDTH-TABLE_WID
```

```
      si_test2 <= PHASE_WIDTH-TABLE_WIDTH-1;
      sv_test  <= EXT(vv_phase, sv_test'length);

      sn_phase1 <= abs(si_phase_reg);
      -- pragma translate_on

      vb_positive2 := vb_positive1;      -- delay one more cycle

      if si_phase_reg < 0 then
        vb_positive1 := false;
      else
        vb_positive1 := true;
      end if;

    end if;
  end process get_wave_data;

end behavioral;
```

B.1.9 7-Segment-Anzeigen-Decoder

Listing B.9: segment_decoder.vhd

```
-------------------------------------------------------------------------------
-- Title      : segment decoder
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : segment_decoder.vhd
-- Author     : Daniel Glaser
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-09-04
-- Last update: 2006-11-23
-- Platform   : LFECP20E
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description: This module decodes a std_logic_vector 4-bit coded hex number
--              to a 7-segment display, showing hexadecimal numbers
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-09-04  1.0      sidaglas        Created
-- 2006-11-23  1.1      sidaglas        OE added
-------------------------------------------------------------------------------
```

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

entity segment_decoder is

  generic (
    OUTPUT_ACTIVE_LOW : boolean := true;
    REGISTER_OUTPUTS  : boolean := true);
  port (
    CLK_IN : in  std_logic;
    nRESET : in  std_logic;
    OE     : in  std_logic;
    INPUT  : in  std_logic_vector(3 downto 0);
    OUTPUT : out std_logic_vector(7 downto 0));

end segment_decoder;

architecture behavioral of segment_decoder is

  signal sv_output : std_logic_vector(OUTPUT'range);

begin  -- behavioral


  gen_reg_out : if REGISTER_OUTPUTS generate

    -- purpose: This process registers the outputs
    -- type   : sequential
    -- inputs : CLK_IN, nRESET
    -- outputs:
    reg_outputs : process (CLK_IN, nRESET)
    begin  -- process reg_outputs
      if nRESET = '0' then                  -- asynchronous reset (active low)
        OUTPUT <= (others => 'Z');
      elsif rising_edge(CLK_IN) then        -- rising clock edge
        if OE = '0' then
          OUTPUT <= (others => 'Z');
        elsif OUTPUT_ACTIVE_LOW then
          OUTPUT <= not sv_output;
        else
          OUTPUT <= sv_output;
```

```
        end if;
      end if;
    end process reg_outputs;
  end generate gen_reg_out;

  nogen_reg_out : if not REGISTER_OUTPUTS generate
    OUTPUT <= (others => 'Z') when OE = '0'
              else not sv_output when OUTPUT_ACTIVE_LOW
              else sv_output;
  end generate nogen_reg_out;

  sv_output <= "00111111" when INPUT = "0000" else  -- 0
               "00000110" when INPUT = "0001" else  -- 1
               "01011011" when INPUT = "0010" else  -- 2
               "01001111" when INPUT = "0011" else  -- 3
               "01100110" when INPUT = "0100" else  -- 4
               "01101101" when INPUT = "0101" else  -- 5
               "01111101" when INPUT = "0110" else  -- 6
               "00000111" when INPUT = "0111" else  -- 7
               "01111111" when INPUT = "1000" else  -- 8
               "01101111" when INPUT = "1001" else  -- 9
               "11110111" when INPUT = "1010" else  -- A (10)
               "11111100" when INPUT = "1011" else  -- B (11)
               "10111001" when INPUT = "1100" else  -- C (12)
               "11011110" when INPUT = "1101" else  -- D (13)
               "11111001" when INPUT = "1110" else  -- E (14)
               "11110001" when INPUT = "1111" else  -- F (15)
               (others => '0');

end behavioral;
```

B.1.10 Bargraph Decoder

Listing B.10: bargraph_decoder.vhd

```
-------------------------------------------------------------------------------
-- Title      : Bargraph decoder
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : bargraph_decoder.vhd
-- Author     : Daniel Glaser
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-09-04
-- Last update: 2006-11-23
-- Platform   : LFECP20E
```

```
-- Standard    : VHDL'87
-------------------------------------------------------------------------------
-- Description: This decoder accepts a 8-bit-word and keeps only the upper
--              bits, where the generic BAR_LIGHT_COUNT decides how many will
--              stay. A value of 8 or more will result in a full bar from
--              ground to topmost bit. But a bar with 2 lights running up and
--              down looks much cooler and also saves power ;-)
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-09-04  1.0      sidaglas        Created
-- 2006-09-05  1.1      sidaglas        Improved
-- 2006-11-23  1.2      sidaglas        OE added
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
use ieee.numeric_std.all;

entity bargraph_decoder is

  generic (
    BAR_LIGHT_COUNT    : positive := 1;  -- Width of the bar (1 to 8)
    OUTPUT_ACTIVE_LOW : boolean  := true);

  port (
    CLK_IN : in  std_logic;
    nRESET : in  std_logic;
    OE     : in  std_logic;
    DEC_EN : in  std_logic;
    INPUT  : in  std_logic_vector(7 downto 0);
    OUTPUT : out std_logic_vector(7 downto 0));

end bargraph_decoder;

architecture behavioral of bargraph_decoder is

  type   tu_fsm_states is (START, FIND_FIRST, SET_BITS, ZERO_OTHERS, RESULT_OUT);
  signal sv_output : std_logic_vector(7 downto 0);

  -- pragma translate_off
```

```
  signal su_fsm_next_state, su_fsm_active_state : tu_fsm_states;
  signal sn_bitcount, sn_barcount               : natural;
  signal sv_vv_output                           : std_logic_vector(7 downto 0);
  -- pragma translate_on

begin  -- behavioral

  -- purpose: This process calculates the leds to light
  -- type   : sequential
  -- inputs : CLK_IN, nRESET
  -- outputs:
  calc_output : process (CLK_IN, nRESET)
    variable vu_fsm_state : tu_fsm_states;
    variable vn_bitcount  : natural range 0 to 7;
    variable vn_barcount  : natural range 0 to 7;
    variable vv_output    : std_logic_vector(7 downto 0);
  begin  -- process calc_output
    if nRESET = '0' then                -- asynchronous reset (active low)
      OUTPUT      <= (others => 'Z');
      sv_output   <= (others => '0');
      vv_output   := (others => '0');
      vn_bitcount := 0;
      vn_barcount := 0;
    elsif rising_edge(CLK_IN) then      -- rising clock edge

      if DEC_EN = '1' then

        -- pragma translate_off
        su_fsm_active_state <= vu_fsm_state;
        -- pragma translate_on

        case vu_fsm_state is
          when START =>
            vv_output    := INPUT;
            vu_fsm_state := FIND_FIRST;
            vn_barcount  := BAR_LIGHT_COUNT-1;
            vn_bitcount  := 7;

          when FIND_FIRST =>
            if vv_output(vn_bitcount) = '1' then
              vu_fsm_state := SET_BITS;
            elsif vn_bitcount = 0 then
              vu_fsm_state := RESULT_OUT;
            elsif vn_bitcount /= 0 then
```

```
        vn_bitcount := vn_bitcount - 1;
      end if;

    when SET_BITS =>
      vv_output(vn_bitcount) := '1';
      if vn_bitcount /= 0 and vn_barcount /= 0 then
        vn_bitcount := vn_bitcount - 1;
        vn_barcount := vn_barcount - 1;
      elsif vn_bitcount = 0 then
        vu_fsm_state := RESULT_OUT;
      elsif vn_barcount = 0 then
        vu_fsm_state := ZERO_OTHERS;
        vn_bitcount  := vn_bitcount - 1;
      end if;

    when ZERO_OTHERS =>
      vv_output(vn_bitcount) := '0';
      if vn_bitcount /= 0 then
        vn_bitcount := vn_bitcount - 1;
      else
        vu_fsm_state := RESULT_OUT;
      end if;

    when RESULT_OUT =>
      sv_output    <= vv_output;
      vu_fsm_state := START;

    when others =>
      vu_fsm_state := START;

  end case;

  -- pragma translate_off
  su_fsm_next_state <= vu_fsm_state;
  sv_vv_output      <= vv_output;
  sn_bitcount       <= vn_bitcount;
  sn_barcount       <= vn_barcount;
  -- pragma translate_on

  if OE = '0' then
    OUTPUT <= (others => 'Z');
  else
    if OUTPUT_ACTIVE_LOW then
      OUTPUT <= not sv_output;
```

```
          else
            OUTPUT <= sv_output;
          end if;
        end if;

      else
        -- Encoder off

        if OE = '0' then
          OUTPUT <= (others => 'Z');
        else
          if OUTPUT_ACTIVE_LOW then
            OUTPUT <= not INPUT;
          else
            OUTPUT <= INPUT;
          end if;
        end if;
      end if;

    end if;
  end process calc_output;

end behavioral;
```

B.1.11 Templates

Listing B.11: stud_toplevel.vhd (Template)

```
-------------------------------------------------------------------------------
-- Title      : Students toplevel
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : stud_toplevel.vhd
-- Author     : Daniel Glaser
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-09-04
-- Last update: 2006-11-30
-- Platform   : LFECP20E
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description: This Toplevel is for student use. They should start here, as
--              here are already usable signals like parallel data from adc. It
--              simplyfies the work to be done.
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
```

```
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-09-04  1.0      sidaglas        Created
-- 2006-11-22  1.1      sidaglas        Modified
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;

entity stud_toplevel is

  generic (
    DATA_WIDTH : positive := 8);

  port (
    ---------------------------------------------------------------------------
    -- Generic signals
    ---------------------------------------------------------------------------
    CLK_FAST : in std_logic;            -- 49,152 MHz
    CLK_SLOW : in std_logic;            -- 0,768 MHz = 768 kHz

    nRESET : in std_logic;

    ---------------------------------------------------------------------------
    -- Human device interface (HDI)
    ---------------------------------------------------------------------------

    -- Inputs
    SWITCH_1 : in std_logic;
    SWITCH_2 : in std_logic;
    SWITCH_3 : in std_logic;
    SWITCH_4 : in std_logic;
    SWITCH_5 : in std_logic;
    SWITCH_6 : in std_logic;
    SWITCH_7 : in std_logic;
    SWITCH_8 : in std_logic;

    BUTTON_1 : in std_logic;
    BUTTON_2 : in std_logic;
    BUTTON_3 : in std_logic;
    BUTTON_4 : in std_logic;
```

```
BUTTON_5 : in std_logic;
BUTTON_6 : in std_logic;
BUTTON_7 : in std_logic;
BUTTON_8 : in std_logic;

-- Outputs
STATUS_L_RED : out std_logic;
STATUS_L_YEL : out std_logic;
STATUS_L_GRE : out std_logic;
STATUS_M_RED : out std_logic;
STATUS_M_YEL : out std_logic;
STATUS_M_GRE : out std_logic;
STATUS_R_RED : out std_logic;
STATUS_R_YEL : out std_logic;
STATUS_R_GRE : out std_logic;

-- The segment leds take hex numbers and show them
SEGMENT_LED1 : out std_logic_vector(3 downto 0);
SEGMENT_LED2 : out std_logic_vector(3 downto 0);
SEGMENT_LED3 : out std_logic_vector(3 downto 0);
SEGMENT_LED4 : out std_logic_vector(3 downto 0);

-- The Bargraph switches on the count of lights you say
BARGRAPH_LEFT    : out std_logic_vector(7 downto 0);
BARGRAPH_RIGHT   : out std_logic_vector(7 downto 0);
BARGRAPH_DEC_EN : out std_logic;

-----------------------------------------------------------------------
-- PC communications
-----------------------------------------------------------------------
PC_SDIN  : in  std_logic;
PC_SDOUT : out std_logic;

-----------------------------------------------------------------------
-- Peripherial connections
-----------------------------------------------------------------------
-- Digitally controlled microphone preamplifier
MA_GAIN : out std_logic_vector(5 downto 0);

-- Analog to digital converter
AD_PDIN_L : in std_logic_vector(23 downto 0);
AD_PDIN_R : in std_logic_vector(23 downto 0);
AD_DVAL   : in std_logic;
```

```
    -- Digital to analog converter
    DA_PDOUT_L : out std_logic_vector(23 downto 0);
    DA_PDOUT_R : out std_logic_vector(23 downto 0);
    DA_DVAL    : out std_logic);

end stud_toplevel;

architecture behavioral of stud_toplevel is

-- Here's the place for instantiations and code

begin  -- behavioral

-- Here's the place for instantiations and code

end behavioral;
```

Die folgenden Module wurden von IPExpress generiert:

- clockgen.vhd
- clockgen_main.vhd

B.2 Testbenches

Listing B.12: segment_decoder_tb.vhd

```
-------------------------------------------------------------------------------
-- Title      : Testbench for design "segment_decoder"
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : segment_decoder_tb.vhd
-- Author     : Daniel Glaser
-- Company    : LfTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-09-04
-- Last update: 2007-01-14
-- Platform   :
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description:
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LfTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author  Description
```

```
-- 2006-09-04  1.0      student36       Created
-------------------------------------------------------------------------------

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

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

entity segment_decoder_tb is

end segment_decoder_tb;

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

architecture segment_decoder_tb of segment_decoder_tb is

  component segment_decoder
    generic (
      OUTPUT_ACTIVE_LOW : boolean;
      REGISTER_OUTPUTS  : boolean);
    port (
      CLK_IN : in  std_logic;
      nRESET : in  std_logic;
      OE     : in  std_logic;
      INPUT  : in  std_logic_vector(3 downto 0);
      OUTPUT : out std_logic_vector(7 downto 0));
  end component;

  -- component generics
  constant OUTPUT_ACTIVE_LOW : boolean := true;
  constant REGISTER_OUTPUTS  : boolean := true;

  -- component ports
  signal CLK_IN : std_logic;
  signal nRESET : std_logic;
  signal OE     : std_logic := '0';
  signal INPUT  : std_logic_vector(3 downto 0);
  signal OUTPUT : std_logic_vector(7 downto 0);

  -- clock
  signal Clk : std_logic := '1';
```

```
begin  -- segment_decoder_tb

  -- component instantiation
  DUT : segment_decoder
    generic map (
      OUTPUT_ACTIVE_LOW => OUTPUT_ACTIVE_LOW,
      REGISTER_OUTPUTS  => REGISTER_OUTPUTS)
    port map (
      CLK_IN => Clk,
      nRESET => nRESET,
      OE     => OE,
      INPUT  => INPUT,
      OUTPUT => OUTPUT);

  -- clock generation
  Clk    <= not Clk after 10 ns;
  CLK_IN <= Clk;

  -- waveform generation
  WaveGen_Proc : process
  begin
    -- insert signal assignments here
    nRESET <= '0';

    wait for 1 ns;
    nRESET <= '1';
    OE     <= '1';
    INPUT  <= "0000";

    wait for 20 ns;

    for i in 0 to 15 loop
      INPUT <= INPUT + 1;
      wait for 20 ns;
    end loop;  -- i

    wait for 20 ns;
    OE <= '0';

    wait;
  end process WaveGen_Proc;

end segment_decoder_tb;
```

```
configuration segment_decoder_tb_segment_decoder_tb_cfg of segment_decoder_tb is
  for segment_decoder_tb
  end for;
end segment_decoder_tb_segment_decoder_tb_cfg;
```

Listing B.13: bargraph_decoder_tb.vhd

```
-- Title      : Testbench for design "bargraph_decoder"
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : bargraph_decoder_tb.vhd
-- Author     : Daniel Glaser
-- Company    : LfTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-09-04
-- Last update: 2007-01-11
-- Platform   :
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description:
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LfTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author   Description
-- 2006-09-04  1.0      student36      Created
-------------------------------------------------------------------------------

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

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

entity bargraph_decoder_tb is

end bargraph_decoder_tb;
```

```
architecture bargraph_decoder_tb of bargraph_decoder_tb is

  component bargraph_decoder
    generic (
      BAR_LIGHT_COUNT    : positive;
      OUTPUT_ACTIVE_LOW : boolean);
    port (
      CLK_IN : in  std_logic;
      nRESET : in  std_logic;
      OE     : in  std_logic;
      DEC_EN : in  std_logic;
      INPUT  : in  std_logic_vector(7 downto 0);
      OUTPUT : out std_logic_vector(7 downto 0));
  end component;

  -- component generics
  constant BAR_LIGHT_COUNT    : positive := 2;
  constant OUTPUT_ACTIVE_LOW : boolean  := false;

  -- component ports
  signal CLK_IN : std_logic;
  signal nRESET : std_logic;
  signal OE     : std_logic;
  signal DEC_EN : std_logic;
  signal INPUT  : std_logic_vector(7 downto 0);
  signal OUTPUT : std_logic_vector(7 downto 0);

  -- clock
  signal Clk : std_logic := '1';

begin  -- bargraph_decoder_tb

  -- component instantiation
  DUT : bargraph_decoder
    generic map (
      BAR_LIGHT_COUNT    => BAR_LIGHT_COUNT,
      OUTPUT_ACTIVE_LOW => OUTPUT_ACTIVE_LOW)
    port map (
      CLK_IN => Clk,
      nRESET => nRESET,
      OE     => OE,
      DEC_EN => DEC_EN,
```

```
      INPUT  => INPUT,
      OUTPUT => OUTPUT);

  -- clock generation
  Clk <= not Clk after 10 ns;

  -- waveform generation
  WaveGen_Proc : process
  begin
    -- insert signal assignments here
    nRESET <= '0';
    OE <= '1';
    DEC_EN <= '1';
    INPUT  <= (others => '0');
    wait for 100 ns;

    nRESET <= '1';
    wait for 300 ns;

    INPUT <= "00000001";
    wait for 1 us;

    for i in 0 to 7 loop
      INPUT(7 downto 0) <= INPUT(6 downto 0) & '0';
      wait for 1 us;
    end loop;  -- i

    INPUT <= "00001111";
    wait for 1 us;

    for i in 0 to 5 loop
      INPUT(7 downto 1) <= INPUT(6 downto 0);
      wait for 1 us;
    end loop;  -- i

    wait;
  end process WaveGen_Proc;

end bargraph_decoder_tb;

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

configuration bargraph_decoder_tb_bargraph_decoder_tb_cfg of bargraph_decoder_tb is
  for bargraph_decoder_tb
```

```
  end for;
end bargraph_decoder_tb_bargraph_decoder_tb_cfg;
```

Listing B.14: chatter_suppress_tb.vhd

```
-- Title      : Testbench for design "chatter_suppress"
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : chatter_suppress_tb.vhd
-- Author     : Daniel Glaser
-- Company    : LfTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-09-04
-- Last update: 2007-01-11
-- Platform   :
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description:
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LfTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author  Description
-- 2006-09-04  1.0      student36      Created
-------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

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

entity chatter_suppress_tb is

end chatter_suppress_tb;

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

architecture chatter_suppress_tb of chatter_suppress_tb is

  component chatter_suppress
    generic (
      BUTTON_COUNT    : positive;
      SUPPRESS_CLOCKS : positive);
```

```
    port (
      CLK_IN : in  std_logic;
      nRESET : in  std_logic;
      INPUT  : in  std_logic_vector(BUTTON_COUNT-1 downto 0);
      OUTPUT : out std_logic_vector(BUTTON_COUNT-1 downto 0));
  end component;

  -- component generics
  constant BUTTON_COUNT    : positive := 8;
  constant SUPPRESS_CLOCKS : positive := 8;

  -- component ports
  signal nRESET : std_logic;
  signal INPUT  : std_logic_vector(BUTTON_COUNT-1 downto 0);
  signal OUTPUT : std_logic_vector(BUTTON_COUNT-1 downto 0);

  -- clock
  signal Clk : std_logic := '1';

  constant chatter_timebase : time := 10 ps;

begin  -- chatter_suppress_tb

  -- component instantiation
  DUT: chatter_suppress
    generic map (
      BUTTON_COUNT    => BUTTON_COUNT,
      SUPPRESS_CLOCKS => SUPPRESS_CLOCKS)
    port map (
      CLK_IN => Clk,
      nRESET => nRESET,
      INPUT  => INPUT,
      OUTPUT => OUTPUT);

  -- clock generation
  Clk <= not Clk after 10 ns;

  -- waveform generation
  WaveGen_Proc: process
  begin
    -- insert signal assignments here
    INPUT <= (others => '0');
    nRESET <= '0';
```

```
    wait for 100 ns;
    nRESET <= '1';

    wait for 100 ns;

    INPUT <= "10000010";

    wait for 100 ns;

    INPUT <= "00001010";

    wait for 100 ns;

    INPUT <= "00000010";

    wait for 100 ns;

    INPUT <= "00001010";

    wait;

  end process WaveGen_Proc;

end chatter_suppress_tb;
```

```
configuration chatter_suppress_tb_chatter_suppress_tb_cfg of chatter_suppress_tb is
  for chatter_suppress_tb
  end for;
end chatter_suppress_tb_chatter_suppress_tb_cfg;
```

Listing B.15: dds_sinus_tb.vhd

```
-- Title      : Testbench for design "dss_sinus"
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
---------------------------------------------------------------------------
-- File       : dss_sinus_tb.vhd
-- Author     : Daniel Glaser
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-09-05
-- Last update: 2007-01-12
```

```
-- Platform     :
-- Standard     : VHDL'87
------------------------------------------------------------------------
-- Description :
------------------------------------------------------------------------
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
------------------------------------------------------------------------
-- Revisions   :
-- Date        Version  Author   Description
-- 2006-09-05  1.0      student36      Created
------------------------------------------------------------------------

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

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

entity dds_sinus_tb is

end dds_sinus_tb;

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

architecture dds_sinus_tb of dds_sinus_tb is

  component dds_sinus
    generic (
      OUTPUT_WIDTH : positive;
      TABLE_WIDTH  : positive;
      PHASE_WIDTH  : positive;
      CLK_DIV      : positive);
    port (
      CLK_IN     : in  std_logic;
      nRESET     : in  std_logic;
      PHASE_INC  : in  std_logic_vector(PHASE_WIDTH-1 downto 0);
      OUTPUT     : out std_logic_vector(OUTPUT_WIDTH-1 downto 0));
  end component;

  -- component generics
  constant OUTPUT_WIDTH : positive := 8;
  constant TABLE_WIDTH  : positive := 4;
```

```
  constant PHASE_WIDTH  : positive := 8;
  constant CLK_DIV      : positive := 250;

  -- component ports
  signal CLK_IN     : std_logic;
  signal nRESET     : std_logic;
  signal PHASE_INC : std_logic_vector(PHASE_WIDTH-1 downto 0) := (others => '0');
  signal OUTPUT     : std_logic_vector(OUTPUT_WIDTH-1 downto 0);

  -- clock
  signal Clk : std_logic := '1';

begin  -- dss_sinus_tb

  assert PHASE_INC > (2**PHASE_WIDTH)-1 report "PHASE_INC to large, must be smaller than

  -- component instantiation
  DUT : dds_sinus
    generic map (
      OUTPUT_WIDTH => OUTPUT_WIDTH,
      TABLE_WIDTH  => TABLE_WIDTH,
      PHASE_WIDTH  => PHASE_WIDTH,
      CLK_DIV      => CLK_DIV)
    port map (
      CLK_IN     => CLK,
      nRESET     => nRESET,
      PHASE_INC => PHASE_INC,
      OUTPUT     => OUTPUT);

  -- clock generation
  Clk <= not Clk after 10 ns;

  -- waveform generation
  WaveGen_Proc : process
  begin
    -- insert signal assignments here
    nRESET    <= '0';
    wait for 20 ns;
    nRESET <= '1';
    wait for 20 ns;
    PHASE_INC <= conv_std_logic_vector(100, PHASE_WIDTH);
    wait for 1 ms;
    PHASE_INC <= conv_std_logic_vector(50, PHASE_WIDTH);
    wait;
```

```
  end process WaveGen_Proc;


end dds_sinus_tb;
```

```
configuration dds_sinus_tb_dds_sinus_tb_cfg of dds_sinus_tb is
  for dds_sinus_tb
  end for;
end dds_sinus_tb_dds_sinus_tb_cfg;
```

Listing B.16: cordic_lut_tb.vhd

```
-- Title      : Testbench for design "cordic_lut"
-- Project    :
-------------------------------------------------------------------------------
-- File       : cordic_lut_tb.vhd
-- Author     :
-- Company    :
-- Created    : 2006-11-01
-- Last update: 2007-01-22
-- Platform   :
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description:
-------------------------------------------------------------------------------
-- Copyright (c) 2007
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-11-01  1.0      sidaglas        Created
-------------------------------------------------------------------------------

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

```
entity cordic_lut_tb is

end cordic_lut_tb;

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

architecture cordic_lut_tb_behavioral of cordic_lut_tb is

  component cordic_lut
    generic (
      DATA_WIDTH : positive;
      ITER_WIDTH : positive);
    port (
      STEP        : in  std_logic_vector(ITER_WIDTH-1 downto 0);
      ALPHA_I     : out std_logic_vector(DATA_WIDTH-1 downto 0);
      TAN_ALPHA_I : out std_logic_vector(DATA_WIDTH-1 downto 0);
      K_I         : out std_logic_vector(DATA_WIDTH-1 downto 0);
      K_G         : out std_logic_vector(DATA_WIDTH-1 downto 0));
  end component;

  -- component generics
  constant DATA_WIDTH : positive := 16;
  constant ITER_WIDTH : positive := 4;

  -- component ports
  signal STEP        : std_logic_vector(ITER_WIDTH-1 downto 0) := (others => '0');
  signal ALPHA_I     : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal TAN_ALPHA_I : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal K_I         : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal K_G         : std_logic_vector(DATA_WIDTH-1 downto 0);

  -- clock
  signal Clk : std_logic := '1';

begin  -- cordic_lut_tb_behavioral

  -- component instantiation
  DUT: cordic_lut
    generic map (
      DATA_WIDTH => DATA_WIDTH,
      ITER_WIDTH => ITER_WIDTH)
    port map (
      STEP        => STEP,
```

```
      ALPHA_I      => ALPHA_I,
      TAN_ALPHA_I => TAN_ALPHA_I,
      K_I          => K_I,
      K_G          => K_G);

  -- clock generation
  Clk <= not Clk after 10 ns;

  -- waveform generation
  WaveGen_Proc: process
  begin
    -- insert signal assignments here
    wait for 60 ns;
    STEP <= STEP + "0001";
  end process WaveGen_Proc;



end cordic_lut_tb_behavioral;
```

```
configuration cordic_lut_tb_cordic_lut_tb_behavioral_cfg of cordic_lut_tb is
  for cordic_lut_tb_behavioral
  end for;
end cordic_lut_tb_cordic_lut_tb_behavioral_cfg;
```

Listing B.17: cordic_tb.vhd

```
-- Title      : Testbench for design "cordic"
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
---------------------------------------------------------------------------
-- File       : cordic_tb.vhd
-- Author     : Daniel Glaser
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-11-29
-- Last update: 2006-11-30
-- Platform   : LFECP20E
-- Standard   : VHDL'87
---------------------------------------------------------------------------
-- Description:
---------------------------------------------------------------------------
```

```
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
--------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author  Description
-- 2006-11-29  1.0      sidaglas        Created
--------------------------------------------------------------------------------

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

use ieee.math_real.all;

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

entity cordic_tb is

end cordic_tb;

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

architecture cordic_tb_behavioral of cordic_tb is

  constant cn_quart_pi : natural := integer((MATH_PI*real(2**(DATA_WIDTH-3)))/real(4));
  constant cn_half_pi  : natural := integer((MATH_PI*real(2**(DATA_WIDTH-3)))/real(2));

  component cordic
    generic (
      ITER_WIDTH    : positive;
      ITERATIONS    : positive;
      DATA_WIDTH    : positive;
      SCALE_OUTPUT : boolean);
    port (
      CLK_IN    : in  std_logic;
      nRESET    : in  std_logic;
      DVAL_IN   : in  std_logic;
      DVAL_OUT  : out std_logic;
      MODE      : in  std_logic;
      PARAM_M   : in  std_logic_vector(1 downto 0);
      X_IN      : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      Y_IN      : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      Z_IN      : in  std_logic_vector(DATA_WIDTH-1 downto 0);
```

```
      X_OUT     : out std_logic_vector(DATA_WIDTH-1 downto 0);
      Y_OUT     : out std_logic_vector(DATA_WIDTH-1 downto 0);
      Z_OUT     : out std_logic_vector(DATA_WIDTH-1 downto 0));
  end component;

  -- component generics
  constant ITER_WIDTH    : positive := 4;
  constant ITERATIONS    : positive := 10;
  constant DATA_WIDTH    : positive := 16;
  constant SCALE_OUTPUT : boolean  := true;

  -- component ports
  signal CLK_IN    : std_logic;
  signal nRESET    : std_logic;
  signal DVAL_IN   : std_logic;
  signal DVAL_OUT : std_logic;
  signal MODE      : std_logic;
  signal PARAM_M   : std_logic_vector(1 downto 0);
  signal X_IN      : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal Y_IN      : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal Z_IN      : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal X_OUT     : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal Y_OUT     : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal Z_OUT     : std_logic_vector(DATA_WIDTH-1 downto 0);

  -- clock
  signal Clk : std_logic := '1';

begin  -- cordic_tb_behavioral

  -- component instantiation
  DUT: cordic
    generic map (
      ITER_WIDTH    => ITER_WIDTH,
      ITERATIONS    => ITERATIONS,
      DATA_WIDTH    => DATA_WIDTH,
      SCALE_OUTPUT => SCALE_OUTPUT)
    port map (
      CLK_IN    => CLK_IN,
      nRESET    => nRESET,
      DVAL_IN   => DVAL_IN,
      DVAL_OUT => DVAL_OUT,
      MODE      => MODE,
      PARAM_M   => PARAM_M,
```

```
      X_IN      => X_IN,
      Y_IN      => Y_IN,
      Z_IN      => Z_IN,
      X_OUT     => X_OUT,
      Y_OUT     => Y_OUT,
      Z_OUT     => Z_OUT);

  -- clock generation
  Clk <= not Clk after 10 ns;
  CLK_IN <= Clk;

  -- waveform generation
  WaveGen_Proc: process
  begin
    -- insert signal assignments here
    nRESET <= '0';
    X_IN <= (others => '0');
    Y_IN <= (others => '0');
    Z_IN <= (others => '0');
    MODE <= '0';
    PARAM_M <= "01";
    wait for 10 ns;
    DVAL_IN <= '0';
    wait for 10 ns;
    nRESET <= '1';
    wait for 100 ns;
    X_IN <= conv_std_logic_vector(-12456, DATA_WIDTH);
    Y_IN <= conv_std_logic_vector(-1090, DATA_WIDTH);
    Z_IN <= conv_std_logic_vector(cn_half_pi, DATA_WIDTH);
    wait for 50 ns;
    DVAL_IN <= '1';
    wait for 40 ns;
    DVAL_IN <= '0';
    wait;
  end process WaveGen_Proc;



end cordic_tb_behavioral;
```

```
configuration cordic_tb_cordic_tb_behavioral_cfg of cordic_tb is
  for cordic_tb_behavioral
```

```
  end for;
end cordic_tb_cordic_tb_behavioral_cfg;
```

Listing B.18: cordic_base_tb.vhd

```
-- Title      : Testbench for design "cordic_base"
-- Project    :
-------------------------------------------------------------------------------
-- File       : cordic_base_tb.vhd
-- Author     :
-- Company    :
-- Created    : 2006-11-01
-- Last update: 2007-01-22
-- Platform   :
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description:
-------------------------------------------------------------------------------
-- Copyright (c) 2007
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-11-01  1.0      sidaglas        Created
-------------------------------------------------------------------------------

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

-- pragma synthesis_off
use ieee.math_real.all;
-- pragma synthesis_on

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

entity cordic_base_tb is

end cordic_base_tb;

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

```
architecture cordic_base_tb_behavioral of cordic_base_tb is

  component cordic_base
    generic (
      ITER_WIDTH    : positive;
      ITERATIONS    : positive;
      DATA_WIDTH    : positive;
      SCALE_OUTPUT : boolean);
    port (
      CLK_IN    : in  std_logic;
      nRESET    : in  std_logic;
      DVAL_IN   : in  std_logic;
      DVAL_OUT  : out std_logic;
      MODE      : in  std_logic;
      PARAM_M   : in  std_logic_vector(1 downto 0);
      X_IN      : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      Y_IN      : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      Z_IN      : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      X_OUT     : out std_logic_vector(DATA_WIDTH-1 downto 0);
      Y_OUT     : out std_logic_vector(DATA_WIDTH-1 downto 0);
      Z_OUT     : out std_logic_vector(DATA_WIDTH-1 downto 0));
  end component;

  -- component generics
  constant ITER_WIDTH    : positive := 4;
  constant ITERATIONS    : positive := 10;
  constant DATA_WIDTH    : positive := 16;
  constant SCALE_OUTPUT : boolean  := true;

  -- component ports
  signal CLK_IN    : std_logic;
  signal nRESET    : std_logic                                  := '0';
  signal DVAL_IN   : std_logic                                  := '0';
  signal DVAL_OUT  : std_logic;
  signal MODE      : std_logic                                  := '0';
  signal PARAM_M   : std_logic_vector(1 downto 0)               := (others => '0');
  signal X_IN      : std_logic_vector(DATA_WIDTH-1 downto 0) := (others => '0');
  signal Y_IN      : std_logic_vector(DATA_WIDTH-1 downto 0) := (others => '0');
  signal Z_IN      : std_logic_vector(DATA_WIDTH-1 downto 0) := (others => '0');
  signal X_OUT     : std_logic_vector(DATA_WIDTH-1 downto 0) := (others => '0');
  signal Y_OUT     : std_logic_vector(DATA_WIDTH-1 downto 0) := (others => '0');
  signal Z_OUT     : std_logic_vector(DATA_WIDTH-1 downto 0) := (others => '0');
```

```
  -- clock
  signal Clk                    : std_logic := '1';
  constant cn_scale_factor      : natural   := 2**(DATA_WIDTH-2);
  constant cn_angle_scale_factor : natural  := 2**(DATA_WIDTH-3);

  -- pragma synthesis_off
  signal X_IN_REAL : real := 0.0;
  signal Y_IN_REAL : real := 0.0;
  signal Z_IN_REAL : real := 0.0;
  signal Z_IN_DEGREE : real := 0.0;
  signal X_OUT_REAL : real := 0.0;
  signal Y_OUT_REAL : real := 0.0;
  signal Z_OUT_REAL : real := 0.0;
  signal Z_OUT_DEGREE : real := 0.0;
  -- pragma synthesis_on

begin  -- cordic_base_tb_behavioral

  -- component instantiation
  DUT : cordic_base
    generic map (
      ITER_WIDTH    => ITER_WIDTH,
      ITERATIONS    => ITERATIONS,
      DATA_WIDTH    => DATA_WIDTH,
      SCALE_OUTPUT => SCALE_OUTPUT)
    port map (
      CLK_IN    => CLK_IN,
      nRESET    => nRESET,
      DVAL_IN   => DVAL_IN,
      DVAL_OUT  => DVAL_OUT,
      MODE      => MODE,
      PARAM_M   => PARAM_M,
      X_IN      => X_IN,
      Y_IN      => Y_IN,
      Z_IN      => Z_IN,
      X_OUT     => X_OUT,
      Y_OUT     => Y_OUT,
      Z_OUT     => Z_OUT);

  -- clock generation
  Clk    <= not Clk after 10 ns;
  CLK_IN <= Clk;

  -- pragma synthesis_off
```

```
X_IN_REAL <= real(conv_integer(X_IN)) / real(cn_scale_factor);
Y_IN_REAL <= real(conv_integer(Y_IN)) / real(cn_scale_factor);
Z_IN_REAL <= real(conv_integer(Z_IN)) / real(cn_angle_scale_factor);
Z_IN_DEGREE <= (real(180) * Z_IN_REAL) / MATH_PI;
X_OUT_REAL <= real(conv_integer(X_OUT)) / real(cn_scale_factor);
Y_OUT_REAL <= real(conv_integer(Y_OUT)) / real(cn_scale_factor);
Z_OUT_REAL <= real(conv_integer(Z_OUT)) / real(cn_angle_scale_factor);
Z_OUT_DEGREE <= (real(180) * Z_OUT_REAL) / MATH_PI;
-- pragma synthesis_on

-- waveform generation
WaveGen_Proc : process
begin
  -- insert signal assignments here
  nRESET <= '0';
  wait for 1 ns;
  nRESET <= '1';

  X_IN <= conv_std_logic_vector(8192, DATA_WIDTH);  -- 0.5
  Y_IN <= conv_std_logic_vector(1024, DATA_WIDTH);  -- 0.0625
  Z_IN <= conv_std_logic_vector(512, DATA_WIDTH);   -- 0.0625 ~ 3.58°

  wait for 20 ns;

  DVAL_IN <= '1';

  wait for 20 ns;

  DVAL_IN <= '0';

  wait for 439 ns;

  X_IN <= conv_std_logic_vector(8192,DATA_WIDTH);  -- 0.5
  Y_IN <= conv_std_logic_vector(8192,DATA_WIDTH);  -- 0.5
  Z_IN <= conv_std_logic_vector(12867,DATA_WIDTH);  -- 1.5707 ~ 90°

  wait for 20 ns;                    -- We are at 500 ns

  DVAL_IN <= '1';

  wait for 20 ns;

  DVAL_IN <= '0';
```

```
    wait for 440 ns;

    X_IN <= conv_std_logic_vector(-8192,DATA_WIDTH);  -- 0.5
    Y_IN <= conv_std_logic_vector(-8192,DATA_WIDTH);  -- 0.5
    Z_IN <= conv_std_logic_vector(4289,DATA_WIDTH);  -- 1.5707 ~ 90°

    wait for 20 ns;                    -- We are at 500 ns

    DVAL_IN <= '1';

    wait for 20 ns;

    DVAL_IN <= '0';

    wait for 440 ns;

    X_IN <= conv_std_logic_vector(-8192,DATA_WIDTH);  -- 0.5
    Y_IN <= conv_std_logic_vector(8192,DATA_WIDTH);  -- 0.5
    Z_IN <= conv_std_logic_vector(-8289,DATA_WIDTH);  -- 1.5707 ~ 90°

    wait for 20 ns;                    -- We are at 500 ns

    DVAL_IN <= '1';

    wait for 20 ns;

    DVAL_IN <= '0';

    wait;
  end process WaveGen_Proc;



end cordic_base_tb_behavioral;
```

```
configuration cordic_base_tb_cordic_base_tb_behavioral_cfg of cordic_base_tb is
  for cordic_base_tb_behavioral
  end for;
end cordic_base_tb_cordic_base_tb_behavioral_cfg;
```

Listing B.19: cordic_full_tb.vhd

```
-------------------------------------------------------------------------------
-- Title      : Testbench for design "cordic_full"
-- Project    :
-------------------------------------------------------------------------------
-- File       : cordic_full_tb.vhd
-- Author     :
-- Company    :
-- Created    : 2006-11-01
-- Last update: 2007-01-22
-- Platform   :
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description:
-------------------------------------------------------------------------------
-- Copyright (c) 2007
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author  Description
-- 2006-11-01  1.0      sidaglas        Created
-------------------------------------------------------------------------------

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

-- pragma synthesis_off
use ieee.math_real.all;
-- pragma synthesis_on



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

entity cordic_full_tb is

end cordic_full_tb;

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

architecture cordic_full_tb_behavioral of cordic_full_tb is

  component cordic_full
```

```
    generic (
      ITER_WIDTH    : positive;
      ITERATIONS    : positive;
      DATA_WIDTH    : positive;
      SCALE_OUTPUT : boolean);
    port (
      CLK_IN    : in  std_logic;
      nRESET    : in  std_logic;
      DVAL_IN   : in  std_logic;
      DVAL_OUT  : out std_logic;
      MODE      : in  std_logic;
      PARAM_M   : in  std_logic_vector(1 downto 0);
      X_IN      : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      Y_IN      : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      Z_IN      : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      X_OUT     : out std_logic_vector(DATA_WIDTH-1 downto 0);
      Y_OUT     : out std_logic_vector(DATA_WIDTH-1 downto 0);
      Z_OUT     : out std_logic_vector(DATA_WIDTH-1 downto 0));
  end component;

  -- component generics
  constant ITER_WIDTH    : positive := 4;
  constant ITERATIONS    : positive := 10;
  constant DATA_WIDTH    : positive := 16;
  constant SCALE_OUTPUT : boolean  := true;

  -- component ports
  signal CLK_IN    : std_logic;
  signal nRESET    : std_logic := '0';
  signal DVAL_IN   : std_logic := '0';
  signal DVAL_OUT  : std_logic;
  signal MODE      : std_logic  := '0';
  signal PARAM_M   : std_logic_vector(1 downto 0) := (others => '0');
  signal X_IN      : std_logic_vector(DATA_WIDTH-1 downto 0) := (others => '0');
  signal Y_IN      : std_logic_vector(DATA_WIDTH-1 downto 0) := (others => '0');
  signal Z_IN      : std_logic_vector(DATA_WIDTH-1 downto 0) := (others => '0');
  signal X_OUT     : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal Y_OUT     : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal Z_OUT     : std_logic_vector(DATA_WIDTH-1 downto 0);

  -- clock
  signal Clk : std_logic := '1';
  constant cn_scale_factor       : natural   := 2**(DATA_WIDTH-2);
  constant cn_angle_scale_factor : natural   := 2**(DATA_WIDTH-3);
```

```
  -- pragma synthesis_off
  signal X_IN_REAL : real := 0.0;
  signal Y_IN_REAL : real := 0.0;
  signal Z_IN_REAL : real := 0.0;
  signal Z_IN_DEGREE : real := 0.0;
  signal X_OUT_REAL : real := 0.0;
  signal Y_OUT_REAL : real := 0.0;
  signal Z_OUT_REAL : real := 0.0;
  signal Z_OUT_DEGREE : real := 0.0;
  -- pragma synthesis_on

begin  -- cordic_full_tb_behavioral

  -- component instantiation
  DUT: cordic_full
    generic map (
      ITER_WIDTH    => ITER_WIDTH,
      ITERATIONS    => ITERATIONS,
      DATA_WIDTH    => DATA_WIDTH,
      SCALE_OUTPUT => SCALE_OUTPUT)
    port map (
      CLK_IN    => CLK_IN,
      nRESET    => nRESET,
      DVAL_IN   => DVAL_IN,
      DVAL_OUT => DVAL_OUT,
      MODE      => MODE,
      PARAM_M   => PARAM_M,
      X_IN      => X_IN,
      Y_IN      => Y_IN,
      Z_IN      => Z_IN,
      X_OUT     => X_OUT,
      Y_OUT     => Y_OUT,
      Z_OUT     => Z_OUT);

  -- clock generation
  Clk <= not Clk after 10 ns;
  CLK_IN <= Clk;

  -- pragma synthesis_off
  X_IN_REAL <= real(conv_integer(X_IN)) / real(cn_scale_factor);
  Y_IN_REAL <= real(conv_integer(Y_IN)) / real(cn_scale_factor);
  Z_IN_REAL <= real(conv_integer(Z_IN)) / real(cn_angle_scale_factor);
  Z_IN_DEGREE <= (real(180) * Z_IN_REAL) / MATH_PI;
```

```
X_OUT_REAL <= real(conv_integer(X_OUT)) / real(cn_scale_factor);
Y_OUT_REAL <= real(conv_integer(Y_OUT)) / real(cn_scale_factor);
Z_OUT_REAL <= real(conv_integer(Z_OUT)) / real(cn_angle_scale_factor);
Z_OUT_DEGREE <= (real(180) * Z_OUT_REAL) / MATH_PI;
-- pragma synthesis_on

-- waveform generation
WaveGen_Proc : process
begin
  -- insert signal assignments here
  nRESET <= '0';
  wait for 1 ns;
  nRESET <= '1';

  X_IN <= conv_std_logic_vector(8192, DATA_WIDTH);  -- 0.5
  Y_IN <= conv_std_logic_vector(1024, DATA_WIDTH);  -- 0.0625
  Z_IN <= conv_std_logic_vector(512, DATA_WIDTH);   -- 0.0625 ~ 3.58°

  wait for 120 ns;

  DVAL_IN <= '1';

  wait for 20 ns;

  DVAL_IN <= '0';

  wait for 439 ns;

  X_IN <= conv_std_logic_vector(8192,DATA_WIDTH);  -- 0.5
  Y_IN <= conv_std_logic_vector(8192,DATA_WIDTH);  -- 0.5
  Z_IN <= conv_std_logic_vector(25736,DATA_WIDTH);  -- 3.1416 ~ 180°

  wait for 20 ns;                    -- We are at 500 ns

  DVAL_IN <= '1';

  wait for 20 ns;

  DVAL_IN <= '0';

  wait for 440 ns;

  X_IN <= conv_std_logic_vector(-8192,DATA_WIDTH);  -- 0.5
  Y_IN <= conv_std_logic_vector(-8192,DATA_WIDTH);  -- 0.5
```

```
    Z_IN <= conv_std_logic_vector(-21447,DATA_WIDTH);  -- -2.6180 ~ -150°

    wait for 20 ns;                    -- We are at 1000 ns

    DVAL_IN <= '1';

    wait for 20 ns;

    DVAL_IN <= '0';

    wait for 440 ns;

    X_IN <= conv_std_logic_vector(-8192,DATA_WIDTH);  -- 0.5
    Y_IN <= conv_std_logic_vector(8192,DATA_WIDTH);  -- 0.5
    Z_IN <= conv_std_logic_vector(-8289,DATA_WIDTH);  -- -1.5707 ~ -90°

    wait for 20 ns;                    -- We are at 1500 ns

    DVAL_IN <= '1';

    wait for 20 ns;

    DVAL_IN <= '0';

    wait;
  end process WaveGen_Proc;



end cordic_full_tb_behavioral;
```

```
configuration cordic_full_tb_cordic_full_tb_behavioral_cfg of cordic_full_tb is
  for cordic_full_tb_behavioral
  end for;
end cordic_full_tb_cordic_full_tb_behavioral_cfg;
```

Listing B.20: bandpass_tb.vhd

```
-- Title      : Testbench for design "bandpass"
-- Project    :
```

```
-------------------------------------------------------------------------------
-- File       : bandpass_tb.vhd
-- Author     :
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-11-01
-- Last update: 2007-01-22
-- Platform   : LFECP20E
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description:
-------------------------------------------------------------------------------
-- Copyright (c) 2007 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author  Description
-- 2006-11-01  1.0      sidaglas        Created
-------------------------------------------------------------------------------

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

entity bandpass_tb is

end bandpass_tb;

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

architecture bandpass_tb_behavioral of bandpass_tb is

  component bandpass
    generic (
      DATA_WIDTH  : positive;
      ORDER_WIDTH : positive);
    port (
      CLK_IN    : in  std_logic;
      nRESET    : in  std_logic;
      CE        : in  std_logic;
      DATA_IN   : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      DATA_OUT  : out std_logic_vector(DATA_WIDTH-1 downto 0));
  end component;
```

```
  component dds_sinus
    generic (
      OUTPUT_WIDTH : positive;
      TABLE_WIDTH  : positive;
      PHASE_WIDTH  : positive;
      CLK_DIV      : positive);
    port (
      CLK_IN    : in  std_logic;
      nRESET    : in  std_logic;
      PHASE_INC : in  std_logic_vector(PHASE_WIDTH-1 downto 0);
      OUTPUT    : out std_logic_vector(OUTPUT_WIDTH-1 downto 0));
  end component;

  -- component generics
  constant DATA_WIDTH  : positive := 8;
  constant ORDER_WIDTH : positive := 4;

  -- component ports
  signal CLK_IN   : std_logic;
  signal nRESET   : std_logic := '0';
  signal CE       : std_logic := '0';
  signal DATA_IN  : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal DATA_OUT : std_logic_vector(DATA_WIDTH-1 downto 0);

  -- clock
  signal Clk : std_logic := '1';

  signal sv_sine_2_kHz                                  : std_logic_vector(DATA_WIDTH-1
  signal sv_sine_200_Hz                                 : std_logic_vector(DATA_WIDTH-1
  signal sv_sine_10_kHz, sv_sine_15_kHz, sv_sine_20_kHz : std_logic_vector(DATA_WIDTH-1

  signal sl_mixture : std_logic;

  signal sv_rectpulse : std_logic_vector(DATA_WIDTH-1 downto 0);

begin  -- bandpass_tb_behavioral

  -- component instantiation
  DUT : bandpass
    generic map (
      DATA_WIDTH  => DATA_WIDTH,
      ORDER_WIDTH => ORDER_WIDTH)
    port map (
```

```
      CLK_IN    => CLK_IN,
      nRESET    => nRESET,
      CE        => CE,
      DATA_IN   => DATA_IN,
      DATA_OUT => DATA_OUT);

  dds_sinus_1 : dds_sinus
    generic map (
      OUTPUT_WIDTH => DATA_WIDTH,
      TABLE_WIDTH  => 4,
      PHASE_WIDTH  => DATA_WIDTH,
      CLK_DIV      => 250)
    port map (
      CLK_IN     => CLK_IN,
      nRESET     => nRESET,
--      PHASE_INC => "000001010",          -- 10
      PHASE_INC => "00000001",          -- 10
--      OUTPUT     => sv_sine_2_kHz);
      OUTPUT     => sv_sine_200_Hz);

  dds_sinus_2 : dds_sinus
    generic map (
      OUTPUT_WIDTH => DATA_WIDTH,
      TABLE_WIDTH  => 4,
      PHASE_WIDTH  => DATA_WIDTH,
      CLK_DIV      => 250)
    port map (
      CLK_IN     => CLK_IN,
      nRESET     => nRESET,
      PHASE_INC => "00110010",
      OUTPUT     => sv_sine_10_kHz);

  dds_sinus_3 : dds_sinus
    generic map (
      OUTPUT_WIDTH => DATA_WIDTH,
      TABLE_WIDTH  => 4,
      PHASE_WIDTH  => DATA_WIDTH,
      CLK_DIV      => 250)
    port map (
      CLK_IN     => CLK_IN,
      nRESET     => nRESET,
      PHASE_INC => "01001011",
      OUTPUT     => sv_sine_15_kHz);
```

```
dds_sinus_4 : dds_sinus
  generic map (
    OUTPUT_WIDTH => DATA_WIDTH,
    TABLE_WIDTH  => 4,
    PHASE_WIDTH  => DATA_WIDTH,
    CLK_DIV      => 250)
  port map (
    CLK_IN     => CLK_IN,
    nRESET     => nRESET,
    PHASE_INC => "01100100",
    OUTPUT     => sv_sine_20_kHz);

-- clock generation
Clk    <= not Clk after 10 ns;
CLK_IN <= Clk;

gen_ce : process (CLK_IN, nRESET)
  variable vn_clk_count : natural := 0;
begin  -- process gen_ce
  if nRESET = '0' then                  -- asynchronous reset (active low)
    CE <= '0';
  elsif rising_edge(CLK_IN) then        -- rising clock edge
    if vn_clk_count = 0 then
      vn_clk_count := 511;
      CE           <= '1';
    else
      vn_clk_count := vn_clk_count - 1;
      CE           <= '0';
    end if;
  end if;
end process gen_ce;

-- waveform generation
WaveGen_Proc : process
begin
  -- insert signal assignments here
  nRESET <= '0';
  sl_mixture <= '1';
  sv_rectpulse <= (others => '0');
  wait for 1 ns;
  nRESET <= '1';
  wait for 6 ms;
  sl_mixture <= '0';
  wait for 500 us;
```

```
    sv_rectpulse <= conv_std_logic_vector(64, DATA_WIDTH);
    wait for 200 us;
    sv_rectpulse <= (others => '0');
    wait;
  end process WaveGen_Proc;

--  gen_dirac: process
--  begin  -- process gen_dirac
--    DATA_IN <= (others => '0');
--    wait for 100 us;
--    wait until CE = '0';
--    DATA_IN <= conv_std_logic_vector(127,DATA_WIDTH);
--    wait until CE = '1';
--    wait until CE = '0';
--    wait for 150 us;
--    DATA_IN <= conv_std_logic_vector(127, DATA_WIDTH);
--    wait until CE = '1';
--    wait until CE = '0';
--    wait for 5 us;
--    DATA_IN <= conv_std_logic_vector(127,DATA_WIDTH);
--    wait until CE = '1';
--    wait until CE = '0';
--    wait for 5 us;
--    DATA_IN <= (others => '0');
--    wait;
--  end process gen_dirac;

  DATA_IN <= conv_std_logic_vector(
--    (conv_integer(sv_sine_2_kHz) +
    (conv_integer(sv_sine_200_Hz) +
     conv_integer(sv_sine_10_kHz) +
     conv_integer(sv_sine_15_kHz) +
     conv_integer(sv_sine_20_kHz))/4, DATA_WIDTH) when sl_mixture = '1' else sv_rectpuls

end bandpass_tb_behavioral;

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

configuration bandpass_tb_bandpass_tb_behavioral_cfg of bandpass_tb is
  for bandpass_tb_behavioral
  end for;
end bandpass_tb_bandpass_tb_behavioral_cfg;

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

Listing B.21: filteropt_tb.vhd

```
-------------------------------------------------------------------------------
-- Title      : Testbench for design "filteropt"
-- Project    :
-------------------------------------------------------------------------------
-- File       : filteropt_tb.vhd
-- Author     :
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-11-01
-- Last update: 2007-01-22
-- Platform   : LFECP20E
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description:
-------------------------------------------------------------------------------
-- Copyright (c) 2007 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-11-01  1.0      sidaglas        Created
-------------------------------------------------------------------------------

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

use ieee.math_real.all;

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

entity filteropt_tb is

end filteropt_tb;

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

architecture filteropt_tb_behavioral of filteropt_tb is

  component filteropt
    generic (
      FILTER_ORDER : positive;
      DATA_WIDTH   : positive);
```

```
    port (
      CLK_IN    : in  std_logic;
      nRESET    : in  std_logic;
      DVAL_IN   : in  std_logic;
      DVAL_OUT  : out std_logic;
      DATA_IN   : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      DATA_OUT  : out std_logic_vector(DATA_WIDTH-1 downto 0));
  end component;

  component dds_sinus
    generic (
      OUTPUT_WIDTH : positive;
      TABLE_WIDTH  : positive;
      PHASE_WIDTH  : positive;
      CLK_DIV      : positive);
    port (
      CLK_IN    : in  std_logic;
      nRESET    : in  std_logic;
      PHASE_INC : in  std_logic_vector(PHASE_WIDTH-1 downto 0);
      OUTPUT    : out std_logic_vector(OUTPUT_WIDTH-1 downto 0));
  end component;

  -- component generics
--  constant FILTER_ORDER : positive := 37; -- 07_BAndpass
  constant FILTER_ORDER : positive := 32;  -- 10_Tiefpass
  constant DATA_WIDTH   : positive := 8;

  -- component ports
  signal CLK_IN    : std_logic;
  signal nRESET    : std_logic;
  signal DVAL_IN   : std_logic;
  signal DVAL_OUT  : std_logic;
  signal DATA_IN   : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal DATA_OUT  : std_logic_vector(DATA_WIDTH-1 downto 0);

  -- clock
  signal Clk : std_logic := '1';

  signal sv_sine_200_Hz, sv_sine_10_kHz, sv_sine_15_kHz, sv_sine_20_kHz : std_logic_vect
  signal sv_sine_var                                                    : std_logic_vect
  signal sv_testsig                                                     : std_logic_vect
  signal sv_dirac                                                       : std_logic_vect

  signal sv_phase_var : std_logic_vector(DATA_WIDTH-1 downto 0) := (others => '0');
```

```
  signal sn_sine_var  : natural;
  signal sr_sine_var  : real;

  signal sl_dirac : std_logic := '0';

begin  -- filteropt_tb_behavioral

  -- component instantiation
  DUT : filteropt
    generic map (
      FILTER_ORDER => FILTER_ORDER,
      DATA_WIDTH   => DATA_WIDTH)
    port map (
      CLK_IN   => CLK_IN,
      nRESET   => nRESET,
      DVAL_IN  => DVAL_IN,
      DVAL_OUT => DVAL_OUT,
      DATA_IN  => DATA_IN,
      DATA_OUT => DATA_OUT);

  dds_sinus_1 : dds_sinus
    generic map (
      OUTPUT_WIDTH => DATA_WIDTH,
      TABLE_WIDTH  => 4,
      PHASE_WIDTH  => DATA_WIDTH,
      CLK_DIV      => 250)
    port map (
      CLK_IN    => CLK_IN,
      nRESET    => nRESET,
--      PHASE_INC => "000001010",          -- 10
      PHASE_INC => "00000001",          -- 10
--      OUTPUT    => sv_sine_2_kHz);
      OUTPUT    => sv_sine_200_Hz);

  dds_sinus_2 : dds_sinus
    generic map (
      OUTPUT_WIDTH => DATA_WIDTH,
      TABLE_WIDTH  => 4,
      PHASE_WIDTH  => DATA_WIDTH,
      CLK_DIV      => 250)
    port map (
      CLK_IN    => CLK_IN,
      nRESET    => nRESET,
      PHASE_INC => "00110010",
```

```
    OUTPUT     => sv_sine_10_kHz);

dds_sinus_3 : dds_sinus
  generic map (
    OUTPUT_WIDTH => DATA_WIDTH,
    TABLE_WIDTH  => 4,
    PHASE_WIDTH  => DATA_WIDTH,
    CLK_DIV      => 250)
  port map (
    CLK_IN     => CLK_IN,
    nRESET     => nRESET,
    PHASE_INC => "01001011",
    OUTPUT     => sv_sine_15_kHz);

dds_sinus_4 : dds_sinus
  generic map (
    OUTPUT_WIDTH => DATA_WIDTH,
    TABLE_WIDTH  => 4,
    PHASE_WIDTH  => DATA_WIDTH,
    CLK_DIV      => 250)
  port map (
    CLK_IN     => CLK_IN,
    nRESET     => nRESET,
    PHASE_INC => "01100100",
    OUTPUT     => sv_sine_20_kHz);

dds_sinus_5 : dds_sinus
  generic map (
    OUTPUT_WIDTH => DATA_WIDTH,
    TABLE_WIDTH  => 4,
    PHASE_WIDTH  => DATA_WIDTH,
    CLK_DIV      => 250)
  port map (
    CLK_IN     => CLK_IN,
    nRESET     => nRESET,
    PHASE_INC => sv_phase_var,
    OUTPUT     => sv_sine_var);

-- clock generation
Clk    <= not Clk after 10 ns;
CLK_IN <= Clk;

sv_testsig <= conv_std_logic_vector(
  (conv_integer(sv_sine_200_Hz) +
```

```
    conv_integer(sv_sine_10_kHz) +
    conv_integer(sv_sine_15_kHz) +
    conv_integer(sv_sine_20_kHz))/4, DATA_WIDTH) when false
              else sv_sine_var when sl_dirac = '0'
              else sv_dirac;

  DATA_IN <= sv_testsig;

  -- waveform generation
  WaveGen_Proc : process
  begin
    -- insert signal assignments here
    nRESET   <= '0';
    sv_dirac <= (others => '0');
    sl_dirac <= '1';
    wait for 1 ns;
    nRESET   <= '1';
    wait for 120 us;
    sv_dirac <= conv_std_logic_vector(127, DATA_WIDTH);
    wait for 100 us;
    sv_dirac <= (others => '0');
    wait for 100 us;
    sl_dirac <= '0';
    wait;
    wait until Clk = '1';
  end process WaveGen_Proc;

  gen_dvals : process (CLK_IN, nRESET)
    variable vi_dval_count : integer := 1;
  begin  -- process gen_dvals
    if nRESET = '0' then                -- asynchronous reset (active low)
      vi_dval_count := 1;
      DVAL_IN       <= '0';
    elsif rising_edge(CLK_IN) then      -- rising clock edge
      if vi_dval_count = 0 then
--        vi_dval_count := 511;
        vi_dval_count := 1023;
        DVAL_IN       <= '1';
      else
        DVAL_IN       <= '0';
        vi_dval_count := vi_dval_count - 1;
      end if;
    end if;
  end process gen_dvals;
```

```
  proc_phase_inc : process
  begin  -- process proc_phase_inc
    wait for 2 ms;
    if conv_integer(sv_phase_var) < 100 then
      sv_phase_var <= sv_phase_var + "00000001";
    end if;
  end process proc_phase_inc;

  sn_sine_var <= conv_integer('0'&sv_phase_var);
  sr_sine_var <= real(sn_sine_var)/real(5);

end filteropt_tb_behavioral;

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

configuration filteropt_tb_filteropt_tb_behavioral_cfg of filteropt_tb is
  for filteropt_tb_behavioral
  end for;
end filteropt_tb_filteropt_tb_behavioral_cfg;

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

Listing B.22: reset_gen_tb.vhd

```
-------------------------------------------------------------------------------
-- Title      : Testbench for design "reset_gen"
-- Project    :
-------------------------------------------------------------------------------
-- File       : reset_gen_tb.vhd
-- Author     :
-- Company    :
-- Created    : 2006-11-01
-- Last update: 2007-01-22
-- Platform   :
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description:
-------------------------------------------------------------------------------
-- Copyright (c) 2007
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-11-01  1.0      sidaglas        Created
-------------------------------------------------------------------------------
```

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

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

entity reset_gen_tb is

end reset_gen_tb;

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

architecture reset_gen_tb_behavioral of reset_gen_tb is

  component reset_gen
    generic (
      CLK_FREQ        : integer;
      RESET_HOLD_TIME : integer);
    port (
      CLK    : in  std_logic;
      nRESET : out std_logic;
      RESET  : out std_logic);
  end component;

  -- component generics
  constant CLK_FREQ        : integer := 49152000;
  constant RESET_HOLD_TIME : integer := 50;

  -- component ports
  signal nRESET : std_logic;
  signal RESET  : std_logic;

  -- clock
  signal Clk : std_logic := '1';

  constant clk_period      : time := 10172 ps;
  constant clk_full_period : time := 2 * clk_period;

begin  -- reset_gen_tb_behavioral

  -- component instantiation
  DUT : reset_gen
    generic map (
      CLK_FREQ        => CLK_FREQ,
```

```
      RESET_HOLD_TIME => RESET_HOLD_TIME)
    port map (
      CLK     => CLK,
      nRESET => nRESET,
      RESET   => RESET);

  -- clock generation
  Clk <= not Clk after clk_period;

  -- waveform generation
  WaveGen_Proc : process
  begin
    -- insert signal assignments here

    wait;
  end process WaveGen_Proc;



end reset_gen_tb_behavioral;

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

configuration reset_gen_tb_reset_gen_tb_behavioral_cfg of reset_gen_tb is
  for reset_gen_tb_behavioral
  end for;
end reset_gen_tb_reset_gen_tb_behavioral_cfg;

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

Listing B.23: huellkurve_tb.vhd

```
-------------------------------------------------------------------------------
-- Title      : Testbench for design "huellkurve"
-- Project    :
-------------------------------------------------------------------------------
-- File       : huellkurve_tb.vhd<2>
-- Author     :
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-11-01
-- Last update: 2007-01-21
-- Platform   : LFECP20E
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description:
```

```
-------------------------------------------------------------------------------
-- Copyright (c) 2007 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author  Description
-- 2006-11-01  1.0      sidaglas        Created
-------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

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

entity huellkurve_tb is

end huellkurve_tb;

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

architecture huellkurve_tb_behavioral of huellkurve_tb is

  component huellkurve
    generic (
      DATA_WIDTH : positive);
    port (
      CLK_IN      : in  std_logic;
      nRESET      : in  std_logic;
      DVAL_IN     : in  std_logic;
      DVAL_OUT    : out std_logic;
      DATA_IN_A   : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      DATA_IN_B   : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      DATA_OUT_A  : out std_logic_vector(DATA_WIDTH-1 downto 0);
      DATA_OUT_B  : out std_logic_vector(DATA_WIDTH-1 downto 0));
  end component;

  component dds_sinus
    generic (
      OUTPUT_WIDTH : positive;
      TABLE_WIDTH  : positive;
      PHASE_WIDTH  : positive;
      CLK_DIV      : positive);
    port (
      CLK_IN     : in  std_logic;
      nRESET     : in  std_logic;
```

```
      PHASE_INC : in  std_logic_vector(PHASE_WIDTH-1 downto 0);
      OUTPUT    : out std_logic_vector(OUTPUT_WIDTH-1 downto 0));
  end component;

  -- component generics
  constant DATA_WIDTH : positive := 8;

  -- component ports
  signal CLK_IN      : std_logic;
  signal nRESET      : std_logic;
  signal DVAL_IN     : std_logic;
  signal DVAL_OUT    : std_logic;
  signal DATA_IN_A   : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal DATA_IN_B   : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal DATA_OUT_A  : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal DATA_OUT_B  : std_logic_vector(DATA_WIDTH-1 downto 0);

  signal sv_sine_10_kHz : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal sv_sine_14_kHz : std_logic_vector(DATA_WIDTH-1 downto 0);

  -- clock
  signal Clk : std_logic := '1';

begin  -- huellkurve_tb_behavioral

  -- component instantiation
  DUT: huellkurve
    generic map (
      DATA_WIDTH => DATA_WIDTH)
    port map (
      CLK_IN      => CLK_IN,
      nRESET      => nRESET,
      DVAL_IN     => DVAL_IN,
      DVAL_OUT    => DVAL_OUT,
      DATA_IN_A   => DATA_IN_A,
      DATA_IN_B   => DATA_IN_B,
      DATA_OUT_A  => DATA_OUT_A,
      DATA_OUT_B  => DATA_OUT_B);

  dds_sinus_1 : dds_sinus
    generic map (
      OUTPUT_WIDTH => DATA_WIDTH,
      TABLE_WIDTH  => 4,
      PHASE_WIDTH  => DATA_WIDTH,
```

```
    CLK_DIV       => 250)
  port map (
    CLK_IN     => CLK_IN,
    nRESET     => nRESET,
    PHASE_INC => "00110010",
    OUTPUT     => sv_sine_10_kHz);

  dds_sinus_2 : dds_sinus
  generic map (
    OUTPUT_WIDTH => DATA_WIDTH,
    TABLE_WIDTH  => 4,
    PHASE_WIDTH  => DATA_WIDTH,
    CLK_DIV       => 250)
  port map (
    CLK_IN     => CLK_IN,
    nRESET     => nRESET,
    PHASE_INC => "01000110",
    OUTPUT     => sv_sine_14_kHz);

-- clock generation
Clk <= not Clk after 10 ns;
CLK_IN <= Clk;

DATA_IN_A <= sv_sine_10_kHz;
DATA_IN_B <= sv_sine_14_kHz;

-- waveform generation
WaveGen_Proc: process
begin
  -- insert signal assignments here
  nRESET <= '0';
  wait for 1 ns;
  nRESET <= '1';
  wait;
end process WaveGen_Proc;

gen_dval: process
begin  -- process gen_dval
  DVAL_IN <= '0';
  wait for 56 ns;
  DVAL_IN <= '1';
  wait for 20 ns;
  DVAL_IN <= '0';
  wait for 10 us;
```

```
    wait for 340 ns;
  end process gen_dval;

end huellkurve_tb_behavioral;
```

```
configuration huellkurve_tb_huellkurve_tb_behavioral_cfg of huellkurve_tb is
  for huellkurve_tb_behavioral
  end for;
end huellkurve_tb_huellkurve_tb_behavioral_cfg;
```

Listing B.24: cordic_siggen_tb.vhd

```
-- Title      : Testbench for design "cordic_siggen"
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : cordic_siggen_tb.vhd
-- Author     : Daniel Glaser
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-11-30
-- Last update: 2006-11-30
-- Platform   : LFECP20E
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description:
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author  Description
-- 2006-11-30  1.0      sidaglas        Created
-------------------------------------------------------------------------------

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

use ieee.math_real.all;
```

```
entity cordic_siggen_tb is

end cordic_siggen_tb;

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

architecture cordig_siggen_tb_behavioral of cordic_siggen_tb is

  component cordic_siggen
    generic (
      DATA_WIDTH : positive;
      ITERATIONS : positive;
      ITER_WIDTH : positive);
    port (
      CLK_IN     : in  std_logic;
      nRESET     : in  std_logic;
      PHASE_INC : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      DATA_OUT   : out std_logic_vector(DATA_WIDTH-1 downto 0));
  end component;

  -- component generics
  constant DATA_WIDTH : positive := 16;
  constant ITERATIONS : positive := 12;
  constant ITER_WIDTH : positive := 4;

  constant cn_angle_scale_factor : natural := 2**(DATA_WIDTH-3);

  constant cn_pi       : natural := integer(MATH_PI*real(cn_angle_scale_factor));
  constant cn_two_pi   : natural := integer(2*MATH_PI*real(cn_angle_scale_factor));
  constant cn_quart_pi : natural := integer(MATH_PI*real(cn_angle_scale_factor)/real(4))

  -- component ports
  signal CLK_IN     : std_logic;
  signal nRESET     : std_logic;
  signal PHASE_INC : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal DATA_OUT   : std_logic_vector(DATA_WIDTH-1 downto 0);

  -- clock
  signal Clk : std_logic := '1';

begin  -- cordig_siggen_tb_behavioral

  -- component instantiation
```

```
  DUT : cordic_siggen
    generic map (
      DATA_WIDTH => DATA_WIDTH,
      ITERATIONS => ITERATIONS,
      ITER_WIDTH => ITER_WIDTH)
    port map (
      CLK_IN     => CLK_IN,
      nRESET     => nRESET,
      PHASE_INC => PHASE_INC,
      DATA_OUT   => DATA_OUT);

  -- clock generation
  Clk     <= not Clk after 10 ns;
  CLK_IN <= Clk;

  -- waveform generation
  WaveGen_Proc : process
  begin
    -- insert signal assignments here
    nRESET     <= '0';
    wait for 100 ns;
    nRESET     <= '1';
    wait for 1 us;
    PHASE_INC <=
      wait until Clk = '1';
  end process WaveGen_Proc;



end cordig_siggen_tb_behavioral;

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

configuration cordic_siggen_tb_cordig_siggen_tb_behavioral_cfg of cordic_siggen_tb is
  for cordig_siggen_tb_behavioral
  end for;
end cordic_siggen_tb_cordig_siggen_tb_behavioral_cfg;

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

Listing B.25: barrel_shifter_tb.vhd

```
-------------------------------------------------------------------------------
-- Title      : Testbench for design "barrel_shifter"
-- Project    :
```

```
-------------------------------------------------------------------------------
-- File       : barrel_shifter_tb.vhd
-- Author     :
-- Company    :
-- Created    : 2006-11-01
-- Last update: 2007-01-22
-- Platform   :
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description:
-------------------------------------------------------------------------------
-- Copyright (c) 2007
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-11-01  1.0      sidaglas        Created
-------------------------------------------------------------------------------

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

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

entity barrel_shifter_tb is

end barrel_shifter_tb;

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

architecture barrel_shifter_tb_behavioral of barrel_shifter_tb is

  component barrel_shifter
    generic (
      USE_MULTIPLIER : boolean;
      AMOUNT_WIDTH   : positive;
      DATA_WIDTH     : positive);
    port (
      SIGNED_SHIFT : in  std_logic;
      SHIFT_AMOUNT : in  std_logic_vector(AMOUNT_WIDTH-1 downto 0);
      DATA_IN      : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      DATA_OUT     : out std_logic_vector(DATA_WIDTH-1 downto 0));
```

```
  end component;

  -- component generics
  constant USE_MULTIPLIER : boolean  := true;
  constant AMOUNT_WIDTH   : positive := 4;
  constant DATA_WIDTH     : positive := 8;

  -- component ports
  signal SIGNED_SHIFT : std_logic := '0';
  signal SHIFT_AMOUNT : std_logic_vector(AMOUNT_WIDTH-1 downto 0) := "0000";
  signal DATA_IN      : std_logic_vector(DATA_WIDTH-1 downto 0) := "10011001";
  signal DATA_OUT     : std_logic_vector(DATA_WIDTH-1 downto 0);

  -- clock
  signal Clk : std_logic := '1';

begin  -- barrel_shifter_tb_behavioral

  -- component instantiation
  DUT: barrel_shifter
    generic map (
      USE_MULTIPLIER => USE_MULTIPLIER,
      AMOUNT_WIDTH   => AMOUNT_WIDTH,
      DATA_WIDTH     => DATA_WIDTH)
    port map (
      SIGNED_SHIFT => SIGNED_SHIFT,
      SHIFT_AMOUNT => SHIFT_AMOUNT,
      DATA_IN      => DATA_IN,
      DATA_OUT     => DATA_OUT);

  -- clock generation
  Clk <= not Clk after 10 ns;
  SIGNED_SHIFT <= not SIGNED_SHIFT after 1 us;
  DATA_IN <= "00011001" after 1700 ns;
  -- waveform generation
  WaveGen_Proc: process
  begin
    -- insert signal assignments here
    wait for 60 ns;
    SHIFT_AMOUNT <= SHIFT_AMOUNT + "0001";

  end process WaveGen_Proc;
```

```
end barrel_shifter_tb_behavioral;

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

configuration barrel_shifter_tb_barrel_shifter_tb_behavioral_cfg of barrel_shifter_tb is
  for barrel_shifter_tb_behavioral
  end for;
end barrel_shifter_tb_barrel_shifter_tb_behavioral_cfg;

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

Listing B.26: gain_control_tb.vhd

```
-------------------------------------------------------------------------------
-- Title      : Testbench for design "gain_control"
-- Project    :
-------------------------------------------------------------------------------
-- File       : gain_control_tb.vhd
-- Author     :
-- Company    :
-- Created    : 2006-11-01
-- Last update: 2007-01-22
-- Platform   :
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description:
-------------------------------------------------------------------------------
-- Copyright (c) 2007
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-11-01  1.0      sidaglas        Created
-------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

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

entity gain_control_tb is

end gain_control_tb;

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

```
architecture gain_control_tb_behavioral of gain_control_tb is

  component gain_control
    generic (
      CLK_FREQ : positive;
      CLK_MAX  : positive);
    port (
      CLK_IN       : in  std_logic;
      nRESET       : in  std_logic;
      GAIN         : in  std_logic_vector(5 downto 0);
      GPO          : in  std_logic_vector(3 downto 0);
      DC_SERVO_EN  : in  std_logic;
      CM_SERVO_EN  : in  std_logic;
      OVERLOAD     : in  std_logic;
      SCKL         : out std_logic;
      nCS          : out std_logic;
      DOUT         : out std_logic;
      DIN          : in  std_logic);
  end component;

  -- component generics
  constant CLK_FREQ : positive := 49152000;
  constant CLK_MAX  : positive := 6250000;

  -- component ports
  signal CLK_IN       : std_logic;
  signal nRESET       : std_logic := '0';
  signal GAIN         : std_logic_vector(5 downto 0) := (others => '0');
  signal GPO          : std_logic_vector(3 downto 0) := (others => '0');
  signal DC_SERVO_EN  : std_logic := '0';
  signal CM_SERVO_EN  : std_logic := '0';
  signal OVERLOAD     : std_logic := '0';
  signal SCKL         : std_logic;
  signal nCS          : std_logic;
  signal DOUT         : std_logic;
  signal DIN          : std_logic;

  -- clock
  signal Clk : std_logic := '1';

begin  -- gain_control_tb_behavioral

  -- component instantiation
```

```
  DUT: gain_control
    generic map (
      CLK_FREQ => CLK_FREQ,
      CLK_MAX  => CLK_MAX)
    port map (
      CLK_IN       => CLK_IN,
      nRESET       => nRESET,
      GAIN         => GAIN,
      GPO          => GPO,
      DC_SERVO_EN  => DC_SERVO_EN,
      CM_SERVO_EN  => CM_SERVO_EN,
      OVERLOAD     => OVERLOAD,
      SCKL         => SCKL,
      nCS          => nCS,
      DOUT         => DOUT,
      DIN          => DIN);

  -- clock generation
  Clk <= not Clk after 10 ns;
  CLK_IN <= Clk;

  -- waveform generation
  WaveGen_Proc: process
  begin
    -- insert signal assignments here
    wait for 20 ns;
    nRESET <= '1';
    wait for 20 ns;
    GAIN <= "010011";
    wait for 5 us;
    GPO <= "1011";
    wait;
  end process WaveGen_Proc;



end gain_control_tb_behavioral;
```

```
configuration gain_control_tb_gain_control_tb_behavioral_cfg of gain_control_tb is
  for gain_control_tb_behavioral
  end for;
end gain_control_tb_gain_control_tb_behavioral_cfg;
```

Listing B.27: i2s_receiver_tb.vhd

```
-------------------------------------------------------------------------------
-- Title      : Testbench for design "i2s_receiver"
-- Project    :
-------------------------------------------------------------------------------
-- File       : i2s_receiver_tb.vhd
-- Author     :
-- Company    :
-- Created    : 2006-11-01
-- Last update: 2007-01-22
-- Platform   :
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description:
-------------------------------------------------------------------------------
-- Copyright (c) 2007
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author  Description
-- 2006-11-01  1.0      sidaglas        Created
-------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

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

entity i2s_receiver_tb is

end i2s_receiver_tb;

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

architecture i2s_receiver_tb_behavioral of i2s_receiver_tb is

  component i2s_receiver
    generic (
      DATA_WIDTH : positive);
    port (
      CLK_IN : in  std_logic;
      nRESET : in  std_logic;
```

```
      LRCK    : in  std_logic;
      BCK     : in  std_logic;
      DIN     : in  std_logic;
      DOUTL   : out std_logic_vector(DATA_WIDTH-1 downto 0);
      DOUTR   : out std_logic_vector(DATA_WIDTH-1 downto 0);
      DVAL    : out std_logic);
  end component;

  -- component generics
  constant DATA_WIDTH : positive := 24;

  -- component ports
  signal CLK_IN : std_logic;
  signal nRESET : std_logic;
  signal LRCK   : std_logic;
  signal BCK    : std_logic;
  signal DIN    : std_logic;
  signal DOUTL  : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal DOUTR  : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal DVAL   : std_logic;

  -- clock
  signal Clk : std_logic := '1';

  constant clk_period      : time := 10172 ps;
  constant clk_full_period : time := 2 * clk_period;

  signal sl_bck, sl_lrck : std_logic := '0';

  constant cn_clk_per_bck  : natural := 8;
  constant cn_bck_per_lrck : natural := 64;

  signal sv_left_channel, sv_right_channel : std_logic_vector(DATA_WIDTH-1 downto 0) :=

  -- pragma synthesis_off
  signal sv_active : std_logic_vector(DATA_WIDTH downto 0);
  -- pragma synthesis_on

begin  -- i2s_receiver_tb_behavioral

  -- component instantiation
  DUT : i2s_receiver
    generic map (
      DATA_WIDTH => DATA_WIDTH)
```

```
    port map (
      CLK_IN => CLK_IN,
      nRESET => nRESET,
      LRCK   => LRCK,
      BCK    => BCK,
      DIN    => DIN,
      DOUTL  => DOUTL,
      DOUTR  => DOUTR,
      DVAL   => DVAL);

  -- clock generation
  Clk    <= not Clk after clk_period;
  CLK_IN <= Clk;

  bck_gen : process (Clk, nRESET)
    variable vn_clk_count : natural := 0;
  begin  -- process bck_gen
    if nRESET = '0' then                -- asynchronous reset (active low)
      sl_bck       <= '0';
      vn_clk_count := 0;
    elsif rising_edge(CLK) then         -- rising clock edge

      if vn_clk_count = 0 then
        sl_bck <= '1';
      elsif vn_clk_count = cn_clk_per_bck/2 then
        sl_bck <= '0';
      end if;

      if vn_clk_count = 0 then
        vn_clk_count := cn_clk_per_bck - 1;
      else
        vn_clk_count := vn_clk_count - 1;
      end if;

    end if;
  end process bck_gen;

  BCK <= sl_bck;

  lrck_gen : process (CLK, nRESET)
    variable vl_last_bck  : std_logic := '0';
    variable vn_bck_count : natural   := 0;
  begin  -- process lrck_gen
    if nRESET = '0' then                -- asynchronous reset (active low)
```

```
      sl_lrck <= '1';
    elsif rising_edge(CLK) then          -- rising clock edge

      if vl_last_bck = '1' and sl_bck = '0' then

        if vn_bck_count = 0 then
          sl_lrck <= '0';                -- Left Channel active
        elsif vn_bck_count = cn_bck_per_lrck/2 then
          sl_lrck <= '1';                -- Right Channel active
        end if;

        if vn_bck_count = 0 then
          vn_bck_count := cn_bck_per_lrck - 1;
        else
          vn_bck_count := vn_bck_count - 1;
        end if;

      end if;

      vl_last_bck := sl_bck;
    end if;
  end process lrck_gen;

  LRCK <= sl_lrck;

  gen_data : process (CLK, nRESET)
    variable vv_active_word            : std_logic_vector(DATA_WIDTH downto 0) := (other
    variable vl_last_bck, vl_last_lrck : std_logic
:= '0';
  begin  -- process gen_data
    if nRESET = '0' then                 -- asynchronous reset (active low)
      DIN <= '0';
    elsif rising_edge(CLK) then          -- rising clock edge

      if vl_last_lrck = '0' and sl_lrck = '1' then
        vv_active_word := '0' & sv_right_channel;
      elsif vl_last_lrck = '1' and sl_lrck = '0' then
        vv_active_word := '0' & sv_left_channel;
      elsif vl_last_bck = '0' and sl_bck = '1' then
        DIN            <= vv_active_word(DATA_WIDTH);
        vv_active_word := vv_active_word(DATA_WIDTH-1 downto 0) & '0';
      end if;

      -- pragma synthesis_off
```

```
      sv_active <= vv_active_word;
      -- pragma synthesis_on

      vl_last_lrck := sl_lrck;
      vl_last_bck  := sl_bck;

    end if;
  end process gen_data;

  -- waveform generation
  WaveGen_Proc : process
  begin
    -- insert signal assignments here
    nRESET           <= '0';
    wait for 20 ns;
    nRESET           <= '1';
    -- ------------------  "01234567890123456789 0123"
    sv_left_channel  <= "101100111000111100001111";
    sv_right_channel <= "111010101010101010101111";
    wait for 10 us;
    sv_left_channel  <= "000000000000111111111111";
    sv_right_channel <= "111000111000111000111000";
    wait;

  end process WaveGen_Proc;



end i2s_receiver_tb_behavioral;
```

```
configuration i2s_receiver_tb_i2s_receiver_tb_behavioral_cfg of i2s_receiver_tb is
  for i2s_receiver_tb_behavioral
  end for;
end i2s_receiver_tb_i2s_receiver_tb_behavioral_cfg;
```

Listing B.28: i2s_transmitter_tb.vhd

```
-- Title      : Testbench for design "i2s_transmitter"
-- Project    :
```

```
-- File        : i2s_transmitter_tb.vhd
-- Author      :
-- Company     :
-- Created     : 2006-11-01
-- Last update: 2007-01-22
-- Platform    :
-- Standard    : VHDL'87
-------------------------------------------------------------------------------
-- Description:
-------------------------------------------------------------------------------
-- Copyright (c) 2007
-------------------------------------------------------------------------------
-- Revisions   :
-- Date        Version  Author          Description
-- 2006-11-01  1.0      sidaglas        Created
-------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

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

entity i2s_transmitter_tb is

end i2s_transmitter_tb;

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

architecture i2s_transmitter_tb_behavioral of i2s_transmitter_tb is

  component i2s_transmitter
    generic (
      DATA_WIDTH      : positive;
      CLK_IN_PER_BCK  : positive;
      BCK_PER_LRCK    : positive);
    port (
      CLK_IN : in  std_logic;
      nRESET : in  std_logic;
      LRCK   : out std_logic;
      BCK    : out std_logic;
      DOUT   : out std_logic;
      DINL   : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      DINR   : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      DVAL   : in  std_logic);
```

```
  end component;

  -- component generics
  constant DATA_WIDTH     : positive := 24;
  constant CLK_IN_PER_BCK : positive := 8;
  constant BCK_PER_LRCK   : positive := 64;

  -- component ports
  signal CLK_IN : std_logic := '0';
  signal nRESET : std_logic := '0';
  signal LRCK   : std_logic;
  signal BCK    : std_logic;
  signal DOUT   : std_logic;
  signal DINL   : std_logic_vector(DATA_WIDTH-1 downto 0) := (others => '0');
  signal DINR   : std_logic_vector(DATA_WIDTH-1 downto 0) := (others => '0');
  signal DVAL   : std_logic := '0';

  -- clock
  signal Clk : std_logic := '1';

  constant clk_period      : time := 10172 ps;
  constant clk_full_period : time := 2 * clk_period;

begin  -- i2s_transmitter_tb_behavioral

  -- component instantiation
  DUT: i2s_transmitter
    generic map (
      DATA_WIDTH     => DATA_WIDTH,
      CLK_IN_PER_BCK => CLK_IN_PER_BCK,
      BCK_PER_LRCK   => BCK_PER_LRCK)
    port map (
      CLK_IN => CLK_IN,
      nRESET => nRESET,
      LRCK   => LRCK,
      BCK    => BCK,
      DOUT   => DOUT,
      DINL   => DINL,
      DINR   => DINR,
      DVAL   => DVAL);

  -- clock generation
  Clk <= not Clk after clk_period;
  CLK_IN <= Clk;
```

```
  -- waveform generation
  WaveGen_Proc: process
  begin
    -- insert signal assignments here
    nRESET <= '0';
    wait for 20 ns;
    nRESET <= '1';
    -- ____ "01234567890123456789 0123"
    wait for 100 ns;
    DVAL <= '1';
    wait for 5 us;
    DINL <= "10101010101010101010101010";
    DINR <= "00111100001111000011110 0";
    DVAL <= '0';

    wait;
  end process WaveGen_Proc;



end i2s_transmitter_tb_behavioral;
```

```
configuration i2s_transmitter_tb_i2s_transmitter_tb_behavioral_cfg of i2s_transmitter_tb
  for i2s_transmitter_tb_behavioral
  end for;
end i2s_transmitter_tb_i2s_transmitter_tb_behavioral_cfg;
```

Listing B.29: my_multiplexer_tb.vhd

```
-- Title      : Testbench for design "my_multiplexer"
-- Project    :
---------------------------------------------------------------------
-- File       : my_multiplexer_tb.vhd
-- Author     :
-- Company    :
-- Created    : 2006-11-01
-- Last update: 2007-01-22
-- Platform   :
-- Standard   : VHDL'87
```

```
-------------------------------------------------------------------------------
-- Description:
-------------------------------------------------------------------------------
-- Copyright (c) 2007
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-11-01  1.0      sidaglas        Created
-------------------------------------------------------------------------------

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

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

entity my_multiplexer_tb is

end my_multiplexer_tb;

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

architecture my_multiplexer_tb_behavioral of my_multiplexer_tb is

  component my_multiplexer
    port (
      sine_5kHz : in  std_logic_vector(7 downto 0);
      sine_7kHz : in  std_logic_vector(7 downto 0);
      adc_data  : in  std_logic_vector(7 downto 0);
      dac_data  : out std_logic_vector(7 downto 0);
      sel_s1    : in  std_logic;
      sel_s2    : in  std_logic);
  end component;

  -- component ports
  signal sine_5kHz : std_logic_vector(7 downto 0) := (others => '0');
  signal sine_7kHz : std_logic_vector(7 downto 0) := (others => '0');
  signal adc_data  : std_logic_vector(7 downto 0) := (others => '0');
  signal dac_data  : std_logic_vector(7 downto 0) := (others => '0');
  signal sel_s1    : std_logic;
  signal sel_s2    : std_logic;
```

```
  -- clock
  signal Clk : std_logic := '1';

begin  -- my_multiplexer_tb_behavioral

  -- component instantiation
  DUT: my_multiplexer
    port map (
      sine_5kHz => sine_5kHz,
      sine_7kHz => sine_7kHz,
      adc_data  => adc_data,
      dac_data  => dac_data,
      sel_s1    => sel_s1,
      sel_s2    => sel_s2);

  -- clock generation
  Clk <= not Clk after 10 ns;

  Signal_gen: process
  begin  -- process Signal_gen
    sine_7kHz <= sine_7kHz + "00000110";
    sine_5kHz <= sine_5kHz + "00000100";
    adc_data <= adc_data + "11011011";
    wait for 1 ns;
  end process Signal_gen;

  -- waveform generation
  WaveGen_Proc: process
  begin
    -- insert signal assignments here
    sel_s1 <= '0';
    sel_s2 <= '0';
    wait for 100 ns;
    sel_s1 <= '1';
    wait for 100 ns;
    sel_s1 <= '0';
    sel_s2 <= '1';
    wait for 100 ns;
    sel_s1 <= '1';
    wait for 100 ns;
  end process WaveGen_Proc;

end my_multiplexer_tb_behavioral;
```

```
configuration my_multiplexer_tb_my_multiplexer_tb_behavioral_cfg of my_multiplexer_tb is
  for my_multiplexer_tb_behavioral
  end for;
end my_multiplexer_tb_my_multiplexer_tb_behavioral_cfg;
```

Listing B.30: levelmeter_tb.vhd

```
-- Title      : Testbench for design "levelmeter"
-- Project    :
-------------------------------------------------------------------------------
-- File       : levelmeter_tb.vhd
-- Author     :
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-11-01
-- Last update: 2007-01-22
-- Platform   : LFECP20E
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description:
-------------------------------------------------------------------------------
-- Copyright (c) 2007 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author  Description
-- 2006-11-01  1.0      sidaglas        Created
-------------------------------------------------------------------------------

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

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

entity levelmeter_tb is

end levelmeter_tb;

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

```
architecture levelmeter_tb_behavioral of levelmeter_tb is

  component levelmeter
    generic (
      DATA_WIDTH        : positive;
      INTEGRATION_WIDTH : positive);
    port (
      CLK_IN    : in  std_logic;
      nRESET    : in  std_logic;
      DIN_VAL   : in  std_logic;
      DOUT_VAL  : out std_logic;
      DATA_IN   : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      DATA_OUT  : out std_logic_vector(DATA_WIDTH-1 downto 0));
  end component;

  -- component generics
  constant DATA_WIDTH        : positive := 8;
  constant INTEGRATION_WIDTH : positive := 8;

  -- component ports
  signal CLK_IN    : std_logic;
  signal nRESET    : std_logic;
  signal DIN_VAL   : std_logic;
  signal DOUT_VAL  : std_logic;
  signal DATA_IN   : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal DATA_OUT  : std_logic_vector(DATA_WIDTH-1 downto 0);

  -- clock
  signal Clk : std_logic := '1';

  signal sv_random : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal sv_other  : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal sl_random : std_logic := '1';

begin  -- levelmeter_tb_behavioral

  -- component instantiation
  DUT : levelmeter
    generic map (
      DATA_WIDTH        => DATA_WIDTH,
      INTEGRATION_WIDTH => INTEGRATION_WIDTH)
    port map (
      CLK_IN    => CLK_IN,
```

```
      nRESET    => nRESET,
      DIN_VAL   => DIN_VAL,
      DOUT_VAL  => DOUT_VAL,
      DATA_IN   => DATA_IN,
      DATA_OUT  => DATA_OUT);

  -- clock generation
  Clk     <= not Clk after 10 ns;
  CLK_IN  <= Clk;

  DATA_IN <= sv_random when sl_random = '1' else sv_other;

  -- waveform generation
  WaveGen_Proc : process
  begin
    -- insert signal assignments here
    nRESET <= '0';
    wait for 1 ns;
    nRESET <= '1';
    wait for 500 us;
    sl_random <= '0';
    sv_other <= conv_std_logic_vector(127, DATA_WIDTH);
    wait for 100 us;                    -- 600 us
    sv_other <= conv_std_logic_vector(63, DATA_WIDTH);
    wait for 100 us;                    -- 700 us
    sv_other <= conv_std_logic_vector(0, DATA_WIDTH);
    wait for 100 us;                    -- 800 us
    sv_other <=  conv_std_logic_vector(-127,DATA_WIDTH);
    wait for 100 us;                    -- 900 us
    sv_other <= conv_std_logic_vector(-128, DATA_WIDTH);
    wait;
  end process WaveGen_Proc;

  gen_random : process (CLK_IN, nRESET)
    variable vv_random1, vv_random2 : std_logic_vector(sv_random'range);
  begin  -- process gen_random
    if nRESET = '0' then                -- asynchronous reset (active low)
      sv_random  <= (others => '0');
      vv_random1 := (others => '1');
      vv_random2 := (others => '1');
    elsif rising_edge(CLK_IN) then      -- rising clock edge
      vv_random1 := (vv_random1(2) xor vv_random1(1) xor vv_random1(0) xor sv_random(sv_
                     vv_random1(vv_random1'left downto 1);
      vv_random2 := vv_random2(vv_random2'left-1 downto 0) &
```

```
                    (vv_random2(vv_random2'left) xor vv_random2(vv_random2'left-2) xor s
      sv_random <= vv_random1 xor not vv_random2;
    end if;
  end process gen_random;

  Gen_dvals : process
  begin -- process Gen_dvals

    for i in 0 to 10 loop
      DIN_VAL <= '1';
      wait for 20 ns;
      DIN_VAL <= '0';
      wait for 80 ns;
    end loop; -- i

  end process Gen_dvals;


end levelmeter_tb_behavioral;
```

```
configuration levelmeter_tb_levelmeter_tb_behavioral_cfg of levelmeter_tb is
  for levelmeter_tb_behavioral
  end for;
end levelmeter_tb_levelmeter_tb_behavioral_cfg;
```

Listing B.31: prn_shiftreg_tb.vhd

```
-------------------------------------------------------------------------------
-- Title      : Testbench for design "prn_shiftreg"
-- Project    :
-------------------------------------------------------------------------------
-- File       : prn_shiftreg_tb.vhd
-- Author     :
-- Company    :
-- Created    : 2006-11-01
-- Last update: 2007-01-22
-- Platform   :
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description:
-------------------------------------------------------------------------------
```

```
-- Copyright (c) 2007
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author  Description
-- 2006-11-01  1.0      sidaglas        Created
-------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

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

entity prn_shiftreg_tb is

end prn_shiftreg_tb;

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

architecture prn_shiftreg_tb_behavioral of prn_shiftreg_tb is

  component prn_shiftreg
    generic (
      REG_LENGTH : positive);
    port (
      CLK_IN           : in  std_logic;
      nRESET           : in  std_logic;
      CE               : in  std_logic;
      SHIFTREG_CONTENT : out std_logic_vector(REG_LENGTH-1 downto 0);
      PRN_OUT          : out std_logic);
  end component;

  -- component generics
  constant REG_LENGTH : positive := 16;

  -- component ports
  signal CLK_IN           : std_logic;
  signal nRESET           : std_logic;
  signal CE               : std_logic;
  signal SHIFTREG_CONTENT : std_logic_vector(REG_LENGTH-1 downto 0);
  signal PRN_OUT          : std_logic;

  -- clock
  signal Clk : std_logic := '1';
```

```
begin  -- prn_shiftreg_tb_behavioral

  -- component instantiation
  DUT: prn_shiftreg
    generic map (
      REG_LENGTH => REG_LENGTH)
    port map (
      CLK_IN            => CLK_IN,
      nRESET            => nRESET,
      CE                => CE,
      SHIFTREG_CONTENT  => SHIFTREG_CONTENT,
      PRN_OUT           => PRN_OUT);

  -- clock generation
  Clk <= not Clk after 10 ns;
  CLK_IN <= Clk;

  -- waveform generation
  WaveGen_Proc: process
  begin
    -- insert signal assignments here
    nRESET <= '0';
    wait for 1 ns;
    nRESET <= '1';
    CE <= '1';
    wait for 9300 ns;
    CE <= '0';
    wait;
  end process WaveGen_Proc;

end prn_shiftreg_tb_behavioral;
```

```
configuration prn_shiftreg_tb_prn_shiftreg_tb_behavioral_cfg of prn_shiftreg_tb is
  for prn_shiftreg_tb_behavioral
  end for;
end prn_shiftreg_tb_prn_shiftreg_tb_behavioral_cfg;
```

Listing B.32: siggen_tb.vhd

```
-- Title      : Testbench for design "siggen"
```

```
-- Project     :
-------------------------------------------------------------------------------
-- File        : siggen_tb.vhd
-- Author      : Daniel Glaser
-- Company     : LTE, FAU Erlangen-Nuremberg, Germany
-- Created     : 2006-11-01
-- Last update: 2007-01-22
-- Platform    : LFECP20E
-- Standard    : VHDL'87
-------------------------------------------------------------------------------
-- Description:
-------------------------------------------------------------------------------
-- Copyright (c) 2007 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions   :
-- Date        Version  Author          Description
-- 2006-11-01  1.0      sidaglas        Created
-------------------------------------------------------------------------------

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

use ieee.math_real.all;

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

entity siggen_tb is

end siggen_tb;

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

architecture siggen_tb_behavioral of siggen_tb is

  component siggen
    generic (
      DATA_WIDTH : positive;
      ITERATIONS : positive;
      ITER_WIDTH : positive;
      CLK_DIV    : positive);
    port (
```

```
      CLK_IN           : in  std_logic;
      nRESET           : in  std_logic;
      CE               : in  std_logic;
      PHASE_INCREMENT  : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      SIGNAL_OUT       : out std_logic_vector(DATA_WIDTH-1 downto 0);
      DVAL             : out std_logic);
  end component;

  -- component generics
  constant DATA_WIDTH : positive := 16;
  constant ITERATIONS : positive := 10;
  constant ITER_WIDTH : positive := 4;
  constant CLK_DIV    : positive := 49;

  -- component ports
  signal CLK_IN           : std_logic;
  signal nRESET           : std_logic                                  := '0';
  signal CE               : std_logic                                  := '0';
  signal PHASE_INCREMENT  : std_logic_vector(DATA_WIDTH-1 downto 0) := (others => '0');
  signal SIGNAL_OUT       : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal DVAL             : std_logic;

  -- clock
  signal Clk : std_logic := '1';

begin  -- siggen_tb_behavioral

  -- component instantiation
  DUT : siggen
    generic map (
      DATA_WIDTH => DATA_WIDTH,
      ITERATIONS => ITERATIONS,
      ITER_WIDTH => ITER_WIDTH,
      CLK_DIV    => CLK_DIV)
    port map (
      CLK_IN           => CLK_IN,
      nRESET           => nRESET,
      CE               => CE,
      PHASE_INCREMENT  => PHASE_INCREMENT,
      SIGNAL_OUT       => SIGNAL_OUT,
      DVAL             => DVAL);

  -- clock generation
  Clk    <= not Clk after 10 ns;
```

```
  CLK_IN <= Clk;

  -- waveform generation
  WaveGen_Proc : process
  begin
    -- insert signal assignments here
    wait for 1 ns;
    nRESET           <= '1';
    wait for 20 ns;
    CE               <= '1';
    wait for 200 ns;
    PHASE_INCREMENT <= conv_std_logic_vector(515, DATA_WIDTH);  -- ~3.6° = ~10kHz
    wait;

    wait until Clk = '1';
  end process WaveGen_Proc;



end siggen_tb_behavioral;

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

configuration siggen_tb_siggen_tb_behavioral_cfg of siggen_tb is
  for siggen_tb_behavioral
  end for;
end siggen_tb_siggen_tb_behavioral_cfg;

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

Listing B.33: sigregen_tb.vhd

```
-------------------------------------------------------------------------------
-- Title      : Testbench for design "sigregen"
-- Project    :
-------------------------------------------------------------------------------
-- File       : sigregen_tb.vhd
-- Author     :
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-11-01
-- Last update: 2007-01-21
-- Platform   : LFECP20E
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description:
```

```
-------------------------------------------------------------------------------
-- Copyright (c) 2007 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-11-01  1.0      sidaglas        Created
-------------------------------------------------------------------------------

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

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

entity sigregen_tb is

end sigregen_tb;

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

architecture sigregen_tb_behavioral of sigregen_tb is

  component sigregen
    generic (
      DATA_WIDTH : positive;
      HYSTERESIS : natural);
    port (
      CLK_IN     : in  std_logic;
      nRESET     : in  std_logic;
      DVAL_IN    : in  std_logic;
      DVAL_OUT   : out std_logic;
      DATA_IN_A  : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      DATA_IN_B  : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      DATA_OUT   : out std_logic);
  end component;

  component dds_sinus
    generic (
      OUTPUT_WIDTH : positive;
      TABLE_WIDTH  : positive;
      PHASE_WIDTH  : positive;
      CLK_DIV      : positive);
```

```
    port (
      CLK_IN     : in  std_logic;
      nRESET     : in  std_logic;
      PHASE_INC  : in  std_logic_vector(PHASE_WIDTH-1 downto 0);
      OUTPUT     : out std_logic_vector(OUTPUT_WIDTH-1 downto 0));
  end component;

  -- component generics
  constant DATA_WIDTH : positive := 8;
  constant HYSTERESIS : natural  := 2;

  -- component ports
  signal CLK_IN     : std_logic;
  signal nRESET     : std_logic;
  signal DVAL_IN    : std_logic;
  signal DVAL_OUT   : std_logic;
  signal DATA_IN_A  : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal DATA_IN_B  : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal DATA_OUT   : std_logic;

  -- clock
  signal Clk : std_logic := '1';

  signal sv_sine_10_kHz : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal sv_sine_14_kHz : std_logic_vector(DATA_WIDTH-1 downto 0);

  signal sr_scale_a, sr_scale_b : real;

  signal sl_data_in : std_logic;

begin  -- sigregen_tb_behavioral

  -- component instantiation
  DUT : sigregen
    generic map (
      DATA_WIDTH => DATA_WIDTH,
      HYSTERESIS => HYSTERESIS)
    port map (
      CLK_IN     => CLK_IN,
      nRESET     => nRESET,
      DVAL_IN    => DVAL_IN,
      DVAL_OUT   => DVAL_OUT,
      DATA_IN_A  => DATA_IN_A,
      DATA_IN_B  => DATA_IN_B,
```

```
    DATA_OUT   => DATA_OUT);

dds_sinus_1 : dds_sinus
  generic map (
    OUTPUT_WIDTH => DATA_WIDTH,
    TABLE_WIDTH  => 4,
    PHASE_WIDTH  => DATA_WIDTH,
    CLK_DIV      => 250)
  port map (
    CLK_IN     => CLK_IN,
    nRESET     => nRESET,
    PHASE_INC => "00110010",
    OUTPUT     => sv_sine_10_kHz);

  dds_sinus_2 : dds_sinus
  generic map (
    OUTPUT_WIDTH => DATA_WIDTH,
    TABLE_WIDTH  => 4,
    PHASE_WIDTH  => DATA_WIDTH,
    CLK_DIV      => 250)
  port map (
    CLK_IN     => CLK_IN,
    nRESET     => nRESET,
    PHASE_INC => "01000110",
    OUTPUT     => sv_sine_14_kHz);

-- clock generation
Clk    <= not Clk after 10 ns;
CLK_IN <= Clk;

sl_data_in <= '1' when sr_scale_a > sr_scale_b else '0';

DATA_IN_A <= conv_std_logic_vector(
  integer(real(conv_integer(sv_sine_10_kHz)) * sr_scale_a), DATA_WIDTH);
DATA_IN_B <= conv_std_logic_vector(
  integer(real(conv_integer(sv_sine_14_kHz)) * sr_scale_b), DATA_WIDTH);

-- waveform generation
WaveGen_Proc : process
begin
  -- insert signal assignments here
  sr_scale_a <= 0.0;
  sr_scale_b <= 0.0;
  nRESET     <= '0';
```

```
    wait for 1 ns;
    nRESET     <= '1';
    wait for 2 ms;
    sr_scale_a <= 0.05;
    sr_scale_b <= 0.0;
    wait for 1 ms;
    sr_scale_a <= 0.1;
    sr_scale_b <= 0.0;
    wait for 1 ms;
    sr_scale_a <= 0.2;
    sr_scale_b <= 0.0;
    wait for 1 ms;
    sr_scale_a <= 0.4;
    sr_scale_b <= 0.0;
    wait for 1 ms;
    sr_scale_a <= 1.0;
    sr_scale_b <= 0.0;
    wait for 2 ms;
    sr_scale_a <= 0.0;
    sr_scale_b <= 1.0;
    wait for 2 ms;
    sr_scale_a <= 1.0;
    sr_scale_b <= 0.9;
    wait for 2 ms;
    sr_scale_a <= 0.9;
    sr_scale_b <= 1.0;
    wait until Clk = '1';
  end process WaveGen_Proc;

  gen_dval : process
  begin  -- process gen_dval
    DVAL_IN <= '0';
    wait for 56 ns;
    DVAL_IN <= '1';
    wait for 20 ns;
    DVAL_IN <= '0';
    wait for 10 us;
    wait for 340 ns;
  end process gen_dval;

end sigregen_tb_behavioral;
```

```
configuration sigregen_tb_sigregen_tb_behavioral_cfg of sigregen_tb is
  for sigregen_tb_behavioral
  end for;
end sigregen_tb_sigregen_tb_behavioral_cfg;
```

B.3 Versuchsspezifische Module

B.3.1 Multiplexer

Listing B.34: my_multiplexer.vhd

```
-------------------------------------------------------------------------------
-- Title      : Multiplexer
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : multiplexer.vhd
-- Author     : Daniel Glaser
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-06-27
-- Last update: 2006-11-23
-- Platform   : LFECP20E
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description: This module defines an 8-bit multiplexer for digital audio data
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-06-27  1.0      sidaglas        Created
-- 2006-11-10  1.1      sidaglas        Fixed entity name to something not
--                                      conflicting with Project name. This
--                                      seemed to raise some error
-------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

entity my_multiplexer is

  port (
    sine_5kHz : in std_logic_vector(7 downto 0);
```

```
    sine_7kHz : in std_logic_vector(7 downto 0);
    adc_data  : in std_logic_vector(7 downto 0);
    dac_data : out std_logic_vector(7 downto 0);

    sel_s1 : in std_logic;
    sel_s2 : in std_logic);

end my_multiplexer;

architecture multiplexer_behavioral of my_multiplexer is
begin

-- STARTL

  dac_data <= sine_5kHz when sel_s1 = '1' and sel_s2 = '0' else
              sine_7kHz when sel_s1 = '1' and sel_s2 = '1' else
              adc_data  when sel_s1 = '0' and sel_s2 = '1' else (others => '0');

-- STOPL

end multiplexer_behavioral;
```

Listing B.35: stud_toplevel

```
-------------------------------------------------------------------------------
-- Title      : Students toplevel
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : stud_toplevel.vhd
-- Author     : Daniel Glaser
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-09-04
-- Last update: 2006-11-26
-- Platform   : LFECP20E
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description: This Toplevel is for student use. They should start here, as
--              here are already usable signals like parallel data from adc. It
--              simplyfies the work to be done.
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-09-04  1.0      sidaglas        Created
```

```
-- 2006-11-22  1.1      sidaglas        Modified
------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;

entity stud_toplevel is

  generic (
    DATA_WIDTH : positive := 8);

  port (
    ----------------------------------------------------------------------------
    -- Generic signals
    ----------------------------------------------------------------------------
    CLK_FAST : in std_logic;            -- 49,152 MHz
    CLK_SLOW : in std_logic;            -- 0,768 MHz = 768 kHz

    nRESET : in std_logic;

    ----------------------------------------------------------------------------
    -- Human device interface (HDI)
    ----------------------------------------------------------------------------

    -- Inputs
    SWITCH_1 : in std_logic;
    SWITCH_2 : in std_logic;
    SWITCH_3 : in std_logic;
    SWITCH_4 : in std_logic;
    SWITCH_5 : in std_logic;
    SWITCH_6 : in std_logic;
    SWITCH_7 : in std_logic;
    SWITCH_8 : in std_logic;

    BUTTON_1 : in std_logic;
    BUTTON_2 : in std_logic;
    BUTTON_3 : in std_logic;
    BUTTON_4 : in std_logic;
    BUTTON_5 : in std_logic;
    BUTTON_6 : in std_logic;
    BUTTON_7 : in std_logic;
    BUTTON_8 : in std_logic;
```

```
-- Outputs
STATUS_L_RED : out std_logic;
STATUS_L_YEL : out std_logic;
STATUS_L_GRE : out std_logic;
STATUS_M_RED : out std_logic;
STATUS_M_YEL : out std_logic;
STATUS_M_GRE : out std_logic;
STATUS_R_RED : out std_logic;
STATUS_R_YEL : out std_logic;
STATUS_R_GRE : out std_logic;

-- The segment leds take hex numbers and show them
SEGMENT_LED1 : out std_logic_vector(3 downto 0);
SEGMENT_LED2 : out std_logic_vector(3 downto 0);
SEGMENT_LED3 : out std_logic_vector(3 downto 0);
SEGMENT_LED4 : out std_logic_vector(3 downto 0);

-- The Bargraph switches on the count of lights you say
BARGRAPH_LEFT   : out std_logic_vector(7 downto 0);
BARGRAPH_RIGHT  : out std_logic_vector(7 downto 0);
BARGRAPH_DEC_EN : out std_logic;

-----------------------------------------------------------------------
-- PC communications
-----------------------------------------------------------------------
PC_SDIN  : in  std_logic;
PC_SDOUT : out std_logic;

-----------------------------------------------------------------------
-- Peripherial connections
-----------------------------------------------------------------------
-- Digitally controlled microphone preamplifier
MA_GAIN : out std_logic_vector(5 downto 0);

-- Analog to digital converter
AD_PDIN_L : in std_logic_vector(23 downto 0);
AD_PDIN_R : in std_logic_vector(23 downto 0);
AD_DVAL   : in std_logic;

-- Digital to analog converter
DA_PDOUT_L : out std_logic_vector(23 downto 0);
DA_PDOUT_R : out std_logic_vector(23 downto 0);
DA_DVAL    : out std_logic);
```

```
end stud_toplevel;

architecture behavioral of stud_toplevel is

  component dds_sinus
    generic (
      OUTPUT_WIDTH : positive;
      TABLE_WIDTH  : positive;
      PHASE_WIDTH  : positive;
      CLK_DIV      : positive);
    port (
      CLK_IN    : in  std_logic;
      nRESET    : in  std_logic;
      PHASE_INC : in  std_logic_vector(PHASE_WIDTH-1 downto 0);
      OUTPUT    : out std_logic_vector(OUTPUT_WIDTH-1 downto 0));
  end component;

  component my_multiplexer
    port (
      sine_5kHz : in  std_logic_vector(7 downto 0);
      sine_7kHz : in  std_logic_vector(7 downto 0);
      adc_data  : in  std_logic_vector(7 downto 0);
      dac_data  : out std_logic_vector(7 downto 0);
      sel_s1    : in  std_logic;
      sel_s2    : in  std_logic);
  end component;

  signal sv_sine_5kHz : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal sv_sine_7kHz : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal sv_dac_data  : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal sv_adc_data  : std_logic_vector(DATA_WIDTH-1 downto 0);

  signal sl_sel_s1, sl_sel_s2 : std_logic;

begin  -- behavioral

  dds_sinus_5kHz : dds_sinus
    generic map (
      OUTPUT_WIDTH => 8,
      TABLE_WIDTH  => 3,
      PHASE_WIDTH  => 8,
      CLK_DIV      => 8)
    port map (
```

```
      CLK_IN     => CLK_SLOW,
      nRESET     => nRESET,
      PHASE_INC => "00001101",
      OUTPUT     => sv_sine_5kHz);

  dds_sinus_7kHz : dds_sinus
    generic map (
      OUTPUT_WIDTH => 8,
      TABLE_WIDTH  => 3,
      PHASE_WIDTH  => 8,
      CLK_DIV      => 8)
    port map (
      CLK_IN     => CLK_SLOW,
      nRESET     => nRESET,
      PHASE_INC => "00010010",
      OUTPUT     => sv_sine_5kHz);

  my_multiplexer_1 : my_multiplexer
    port map (
      sine_5kHz => sv_sine_5kHz,
      sine_7kHz => sv_sine_7kHz,
      adc_data  => sv_adc_data,
      dac_data  => sv_dac_data,
      sel_s1    => sl_sel_s1,
      sel_s2    => sl_sel_s2);

  DA_DVAL <= AD_DVAL;

  DA_PDOUT_L  <= sv_dac_data;
  sv_adc_data <= AD_PDIN_L;

  sl_sel_s1 <= SWITCH_1 xor BUTTON_1;
  sl_sel_s2 <= SWITCH_2 xor BUTTON_2;

end behavioral;
```

B.3.2 Zufallsfolgengenerator

Listing B.36: prn_shiftreg.vhd

```
-------------------------------------------------------------------------------
-- Title      : PRN-Schieberegister
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : prn_shiftreg.vhd
```

```
-- Author      : Daniel Glaser
-- Company     : LTE, FAU Erlangen-Nuremberg, Germany
-- Created     : 2006-11-01
-- Last update: 2007-01-23
-- Platform    : LFECP20E
-- Standard    : VHDL'87
-------------------------------------------------------------------------------
-- Description: This module implements a PRN shift register.
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-11-01  1.0      sidaglas        Created
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;

entity prn_shiftreg is

  generic (
    REG_LENGTH : positive := 16);

  port (
    CLK_IN           : in  std_logic;
    nRESET           : in  std_logic;
    CE               : in  std_logic;
    SHIFTREG_CONTENT : out std_logic_vector(REG_LENGTH-1 downto 0);
    PRN_OUT          : out std_logic);

end prn_shiftreg;

architecture behavioral of prn_shiftreg is

--STARTL
  constant cv_init : std_logic_vector(REG_LENGTH-1 downto 0) := "1010101010101010";

--STOPL
begin  -- behavioral

--STARTL
```

```
  proc_shiftreg : process (CLK_IN, nRESET)
    variable vv_shiftreg : std_logic_vector(REG_LENGTH-1 downto 0);
  begin  -- process proc_shiftreg
    if nRESET = '0' then                -- asynchronous reset (active low)
      vv_shiftreg := cv_init;
    elsif rising_edge(CLK_IN) then      -- rising clock edge
      if CE = '1' then
        PRN_OUT          <= vv_shiftreg(0);
        SHIFTREG_CONTENT <= vv_shiftreg;

        vv_shiftreg(REG_LENGTH-1 downto 0) :=
          (vv_shiftreg(5) xor vv_shiftreg(1)) &
          vv_shiftreg(REG_LENGTH-1 downto 1);
      end if;
    end if;
  end process proc_shiftreg;

--STOPL
end behavioral;
```

Listing B.37: stud_toplevel.vhd

```
-------------------------------------------------------------------------------
-- Title      : Students toplevel
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : stud_toplevel.vhd
-- Author     : Daniel Glaser
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-09-04
-- Last update: 2006-12-02
-- Platform   : LFECP20E
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description: This Toplevel is for student use. They should start here, as
--              here are already usable signals like parallel data from adc. It
--              simplyfies the work to be done.
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-09-04  1.0      sidaglas        Created
-- 2006-11-22  1.1      sidaglas        Modified
-------------------------------------------------------------------------------
```

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

entity stud_toplevel is

  generic (
    DATA_WIDTH : positive := 8);

  port (
    ------------------------------------------------------------------------
    -- Generic signals
    ------------------------------------------------------------------------
    CLK_FAST : in std_logic;            -- 49,152 MHz
    CLK_SLOW : in std_logic;            -- 0,768 MHz = 768 kHz

    nRESET : in std_logic;

    ------------------------------------------------------------------------
    -- Human device interface (HDI)
    ------------------------------------------------------------------------

    -- Inputs
    SWITCH_1 : in std_logic;
    SWITCH_2 : in std_logic;
    SWITCH_3 : in std_logic;
    SWITCH_4 : in std_logic;
    SWITCH_5 : in std_logic;
    SWITCH_6 : in std_logic;
    SWITCH_7 : in std_logic;
    SWITCH_8 : in std_logic;

    BUTTON_1 : in std_logic;
    BUTTON_2 : in std_logic;
    BUTTON_3 : in std_logic;
    BUTTON_4 : in std_logic;
    BUTTON_5 : in std_logic;
    BUTTON_6 : in std_logic;
    BUTTON_7 : in std_logic;
    BUTTON_8 : in std_logic;

    -- Outputs
```

```
    STATUS_L_RED : out std_logic;
    STATUS_L_YEL : out std_logic;
    STATUS_L_GRE : out std_logic;
    STATUS_M_RED : out std_logic;
    STATUS_M_YEL : out std_logic;
    STATUS_M_GRE : out std_logic;
    STATUS_R_RED : out std_logic;
    STATUS_R_YEL : out std_logic;
    STATUS_R_GRE : out std_logic;

    -- The segment leds take hex numbers and show them
    SEGMENT_LED1 : out std_logic_vector(3 downto 0);
    SEGMENT_LED2 : out std_logic_vector(3 downto 0);
    SEGMENT_LED3 : out std_logic_vector(3 downto 0);
    SEGMENT_LED4 : out std_logic_vector(3 downto 0);

    -- The Bargraph switches on the count of lights you say
    BARGRAPH_LEFT    : out std_logic_vector(7 downto 0);
    BARGRAPH_RIGHT   : out std_logic_vector(7 downto 0);
    BARGRAPH_DEC_EN : out std_logic;

    ------------------------------------------------------------------------
    -- PC communications
    ------------------------------------------------------------------------
    PC_SDIN  : in  std_logic;
    PC_SDOUT : out std_logic;

    ------------------------------------------------------------------------
    -- Peripherial connections
    ------------------------------------------------------------------------
    -- Digitally controlled microphone preamplifier
    MA_GAIN : out std_logic_vector(5 downto 0);

    -- Analog to digital converter
    AD_PDIN_L : in std_logic_vector(23 downto 0);
    AD_PDIN_R : in std_logic_vector(23 downto 0);
    AD_DVAL   : in std_logic;

    -- Digital to analog converter
    DA_PDOUT_L : out std_logic_vector(23 downto 0);
    DA_PDOUT_R : out std_logic_vector(23 downto 0);
    DA_DVAL    : out std_logic);

end stud_toplevel;
```

```
architecture behavioral of stud_toplevel is

  constant cp_clk_divider : positive := 10000;
  constant cp_reg_length : positive := 16;

  component prn_shiftreg
    generic (
      REG_LENGTH : positive);
    port (
      CLK_IN           : in  std_logic;
      nRESET           : in  std_logic;
      CE               : in  std_logic;
      SHIFTREG_CONTENT : out std_logic_vector(REG_LENGTH-1 downto 0);
      PRN_OUT          : out std_logic);
  end component;

  signal sv_prnreg : std_logic_vector(REG_LENGTH-1 downto 0);
  signal sl_prn_ce : std_logic;

begin  -- behavioral

  prn_shiftreg_1 : prn_shiftreg
    generic map (
      REG_LENGTH => cp_reg_length)
    port map (
      CLK_IN           => CLK_SLOW,
      nRESET           => nRESET,
      CE               => sl_prn_ce,
      SHIFTREG_CONTENT => sv_prnreg,
      PRN_OUT          => STATUS_R_GRE);

  BARGRAPH_LEFT  <= sv_prnreg(cp_reg_length-1 downto cp_reg_length-8);
  BARGRAPH_RIGHT <= sv_prnreg(7 downto 0);

  BARGRAPH_DEC_EN <= '0';

  -- This process is for debugging the prn
  gen_ce : process (CLK_SLOW, nRESET)
    variable vl_last_button_state : std_logic;
    variable vn_count : natural range 0 to cp_clk_divider-1;
  begin  -- process gen_ce
    if nRESET = '0' then                -- asynchronous reset (active low)
      sl_prn_ce <= '0';
```

```
      vn_count := cp_clk_divider-1;
    elsif rising_edge(CLK_SLOW) then    -- rising clock edge
      if SWITCH_1 = '0' then
        -- SWITCH active = '0'
        if vn_count = 0 then
          sl_prn_ce <= '1';
          vn_count := cp_clk_divider-1;
        else
          sl_prn_ce <= '1';
          vn_count := vn_count + 1;
        end if;
      elsif vl_last_button_state = '1' and BUTTON_1 = '0' then
        -- Button signal edge detect (BUTTON active = '0')
        sl_prn_ce <= '1';
      else
        sl_prn_ce <= '0';
      end if;
    end if;
  end process gen_ce;

end behavioral;
```

B.3.3 Signalgenerator

Listing B.38: cordic_lut.vhd

```
-------------------------------------------------------------------------------
-- Title      : Cordic look-up table
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : cordic_lut.vhd
-- Author     : Daniel Glaser
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-11-16
-- Last update: 2007-01-23
-- Platform   : LFECP20E
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description: This modlue implements the look-up table needed for cordic
--              algorithm. It uses math_real to claculate the values needed.
--              Math_real is not needed after synthesis time, because the
--              values are converted to std_logic_vector representing a scaled
--              integer value. The values are stored within distributed RAM
--              (ROM-mode).
-------------------------------------------------------------------------------
```

```
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
--------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author         Description
-- 2006-11-16  1.0      sidaglas       Created
-- 2006-11-24  1.1      sidaglas       Improved
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;
use ieee.math_real.all;

-- NOCOPY -- Don't copy this script to students work folder

entity cordic_lut is

  generic (
    DATA_WIDTH : positive := 16;
    ITER_WIDTH : positive := 4);

  port (
    STEP        : in  std_logic_vector(ITER_WIDTH-1 downto 0);
    ALPHA_I     : out std_logic_vector(DATA_WIDTH-1 downto 0);
    TAN_ALPHA_I : out std_logic_vector(DATA_WIDTH-1 downto 0);
    K_I         : out std_logic_vector(DATA_WIDTH-1 downto 0);
    K_G         : out std_logic_vector(DATA_WIDTH-1 downto 0));

end cordic_lut;

architecture behavioral of cordic_lut is

  constant cn_iterations         : positive := 2**ITER_WIDTH;
  constant cn_max_value          : positive := (2**DATA_WIDTH)-1;
  constant cn_scale_factor       : natural  := 2**(DATA_WIDTH-2);
  constant cn_angle_scale_factor : natural  := 2**(DATA_WIDTH-3);

  type tur_lut is record
    ALPHA_I     : natural range 0 to cn_max_value;
    TAN_ALPHA_I : natural range 0 to cn_max_value;
    K_I         : natural range 0 to cn_max_value;
    K_G         : natural range 0 to cn_max_value;
  end record tur_lut;
```

```
  type tuar_lut is array (0 to cn_iterations-1) of tur_lut;
  signal suar_cordic_lut : tuar_lut;

--  signal sr_tan_alpha_i : real := real(0);
--  signal sr_alphi_i     : real := real(0);
  signal si_k_i_helper : integer := 0;
  signal si_k_helper   : integer := 0;
  signal sn_step       : natural range 0 to (2**ITER_WIDTH)-1;

begin  -- behavioral

  gen_lut : for i in 0 to cn_iterations-1 generate
    suar_cordic_lut(i).TAN_ALPHA_I <=
      integer(ROUND(real(cn_scale_factor)/(real(2**i))));

    suar_cordic_lut(i).ALPHA_I <=
      integer(ROUND(arctan(real(1)/(real(2**i)))*real(cn_angle_scale_factor)));

    suar_cordic_lut(i).K_I <=
      integer(real(cn_scale_factor)/sqrt(real(1)+(real(2)**(-2*i))));

    gen_zeroi : if i = 0 generate
      suar_cordic_lut(i).K_G <=
        integer(real(cn_scale_factor)/
                sqrt(real(1)+(real(2)**(-2*i))));
    end generate gen_zeroi;
    gen_nonzeroi : if i /= 0 generate
      suar_cordic_lut(i).K_G <=
        integer(real(suar_cordic_lut(i-1).K_G)/
                sqrt(real(1)+(real(2)**(-2*i))));
    end generate gen_nonzeroi;

  end generate gen_lut;

  sn_step <= conv_integer("0" & STEP);

  ALPHA_I <= conv_std_logic_vector(
    conv_signed(suar_cordic_lut(sn_step).ALPHA_I, DATA_WIDTH), DATA_WIDTH);
  TAN_ALPHA_I <= conv_std_logic_vector(
    conv_signed(suar_cordic_lut(sn_step).TAN_ALPHA_I, DATA_WIDTH), DATA_WIDTH);
  K_I <= conv_std_logic_vector(
    conv_signed(suar_cordic_lut(sn_step).K_I, DATA_WIDTH), DATA_WIDTH);
  K_G <= conv_std_logic_vector(
```

```
    conv_signed(suar_cordic_lut(sn_step).K_G, DATA_WIDTH), DATA_WIDTH);

end behavioral;
```

Listing B.39: cordic.vhd

```
-------------------------------------------------------------------------------
-- Title      : Cordic Base
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : cordic.vhd
-- Author     : Daniel Glaser
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-11-01
-- Last update: 2007-01-23
-- Platform   : LFECP20E
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description: This is the beatiful cordic algorithm. It is implemented as a
--              parametrizable module. The iteration depth is variable as the
--              width of the data is.
--                It is completely written in VHDL and doesn't need any tools
--              from your fpga vendor, but it needs the math_real package at
--              synthesis time. If you can't cope with this, implement your
--              own, hand calculated lut or try to solve it in integer
--              arithmetic and you should be happy again.
--                The algorithm takes ITERATIONS+1 cycles to complete.
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-11-01  1.0      sidaglas        Created
-- 2006-11-24  1.0      sidaglas        Finished
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
--use ieee.numeric_std.all;

entity cordic_base is

  generic (
    ITER_WIDTH    : positive := 4;
```

```
    ITERATIONS    : positive := 10;
    DATA_WIDTH    : positive := 16;
    SCALE_OUTPUT : boolean  := true);

  port (
    CLK_IN    : in  std_logic;
    nRESET    : in  std_logic;
    DVAL_IN   : in  std_logic;
    DVAL_OUT  : out std_logic;
    MODE      : in  std_logic;                      -- unused
    PARAM_M   : in  std_logic_vector(1 downto 0);   -- unused
    X_IN      : in  std_logic_vector(DATA_WIDTH-1 downto 0);
    Y_IN      : in  std_logic_vector(DATA_WIDTH-1 downto 0);
    Z_IN      : in  std_logic_vector(DATA_WIDTH-1 downto 0);
    X_OUT     : out std_logic_vector(DATA_WIDTH-1 downto 0);
    Y_OUT     : out std_logic_vector(DATA_WIDTH-1 downto 0);
    Z_OUT     : out std_logic_vector(DATA_WIDTH-1 downto 0));

end cordic_base;

architecture behavioral of cordic_base is

  --STARTL
  constant cn_scale_factor : natural := 2**(DATA_WIDTH-2);

  ------------------------------------------------------------------------
  -- functions
  ------------------------------------------------------------------------

  function rshift(ARG : signed; COUNT : natural) return signed is
    variable vs_tmp : signed(ARG'range);
    constant ci_lo  : integer := ARG'high - COUNT + 1;
  begin
    for n in ARG'high downto ci_lo loop
      vs_tmp(n) := ARG(ARG'high);
    end loop;
    for n in ARG'high - COUNT downto 0 loop
      vs_tmp(n) := ARG(n + COUNT);
    end loop;
    return vs_tmp;
  end function rshift;

  component cordic_lut
    generic (
```

```
      DATA_WIDTH : positive;
      ITER_WIDTH : positive);
    port (
      STEP        : in  std_logic_vector(ITER_WIDTH-1 downto 0);
      ALPHA_I     : out std_logic_vector(DATA_WIDTH-1 downto 0);
      TAN_ALPHA_I : out std_logic_vector(DATA_WIDTH-1 downto 0);
      K_I         : out std_logic_vector(DATA_WIDTH-1 downto 0);
      K_G         : out std_logic_vector(DATA_WIDTH-1 downto 0));
  end component;

  signal sv_cordic_step : std_logic_vector(ITER_WIDTH-1 downto 0);
  signal sv_alpha_i     : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal sv_tan_aplha_i : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal sv_k_i         : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal sv_k_g         : std_logic_vector(DATA_WIDTH-1 downto 0);
--  signal sv_pings       : std_logic_vector(8 downto 0);

  signal si_x, si_y, si_z : integer
    range -(2**DATA_WIDTH) to 2**DATA_WIDTH-1 := 0;
  signal si_x_s, si_y_s, si_z_s : integer
    range -(2**DATA_WIDTH) to 2**DATA_WIDTH-1 := 0;
  signal si_alpha_i : integer
    range -(2**DATA_WIDTH) to 2**DATA_WIDTH-1 := 0;

  signal si_scale : integer range -(2**DATA_WIDTH) to 2**DATA_WIDTH-1;

  signal sn_iter_count : natural range 0 to ITERATIONS;
  signal sn_debug      : integer;

  signal sl_active : std_logic;

  signal sb_sigma_pos : boolean;
  --STOPL

begin  -- behavioral

  --STARTL
  cordic_lut_1 : cordic_lut
    generic map (
      DATA_WIDTH => DATA_WIDTH,
      ITER_WIDTH => ITER_WIDTH)
    port map (
      STEP        => sv_cordic_step,
      ALPHA_I     => sv_alpha_i,
```

```
    TAN_ALPHA_I => sv_tan_aplha_i,
    K_I         => sv_k_i,
    K_G         => sv_k_g);

cordic_control : process (CLK_IN, nRESET)
  variable vl_dval : std_logic;
begin  -- process cordic_control
  if nRESET = '0' then                  -- asynchronous reset (active low)
    vl_dval   := '0';
    sl_active <= '0';
    DVAL_OUT  <= '0';
    X_OUT     <= (others => '0');
    Y_OUT     <= (others => '0');
    Z_OUT     <= (others => '0');
  elsif rising_edge(CLK_IN) then        -- rising clock edge

    if DVAL_IN = '1' and vl_dval = '0' and sl_active = '0' then
      sl_active     <= '1';
      si_x          <= conv_integer(X_IN);
      si_y          <= conv_integer(Y_IN);
      si_z          <= conv_integer(Z_IN);
      sn_iter_count <= 0;
      DVAL_OUT      <= '0';
    elsif (sn_iter_count = ITERATIONS and SCALE_OUTPUT)
      or (sn_iter_count = ITERATIONS-1 and not SCALE_OUTPUT) then
      sl_active     <= '0';
      sn_iter_count <= 0;
    elsif sl_active = '1' then
      sn_iter_count <= sn_iter_count + 1;
    elsif sl_active = '0' then
      DVAL_OUT <= '1';
      X_OUT    <= conv_std_logic_vector(si_x, DATA_WIDTH);
      Y_OUT    <= conv_std_logic_vector(si_y, DATA_WIDTH);
      Z_OUT    <= conv_std_logic_vector(si_z, DATA_WIDTH);
    end if;

    if sl_active = '1' then
      if sn_iter_count = ITERATIONS then
        -- If no SCALING, we never get to sn_iter_count = ITERATIONS
        si_x <= si_x * si_scale / cn_scale_factor;
        si_y <= si_y * si_scale / cn_scale_factor;
        si_z <= si_z;
      else
        si_x <= si_x - conv_integer(rshift(signed(
```

```
            conv_std_logic_vector(si_y_s, DATA_WIDTH)), sn_iter_count));
          si_y <= si_y + conv_integer(rshift(signed(
            conv_std_logic_vector(si_x_s, DATA_WIDTH)), sn_iter_count));
          si_z <= si_z - si_z_s;
        end if;
      end if;

      vl_dval := DVAL_IN;
    end if;
  end process cordic_control;

  sn_debug <= cn_scale_factor;

  si_alpha_i <= conv_integer(sv_alpha_i);
  si_scale   <= conv_integer(sv_k_g);

  sb_sigma_pos <= true when si_z >= 0 else false;

  si_x_s <= si_x       when sb_sigma_pos else -si_x;
  si_y_s <= si_y       when sb_sigma_pos else -si_y;
  si_z_s <= si_alpha_i when sb_sigma_pos else -si_alpha_i;

  sv_cordic_step <= conv_std_logic_vector(sn_iter_count, ITER_WIDTH);
  --STOPL

end behavioral;
```

Listing B.40: cordic_full.vhd

```
-------------------------------------------------------------------------------
-- Title      : Cordic Full
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : cordic_full.vhd
-- Author     : Daniel Glaser
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-11-01
-- Last update: 2007-01-21
-- Platform   : LFECP20E
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description: This module improves the cordic with full scale 2*PI rotation
--              capability.
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
```

```
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author  Description
-- 2006-11-01  1.0      sidaglas        Created
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
use ieee.numeric_std.all;

use ieee.math_real.all;

entity cordic_full is

  generic (
    ITER_WIDTH    : positive := 4;
    ITERATIONS    : positive := 10;
    DATA_WIDTH    : positive := 16;
    SCALE_OUTPUT : boolean  := true);

  port (
    CLK_IN    : in  std_logic;
    nRESET    : in  std_logic;
    DVAL_IN   : in  std_logic;
    DVAL_OUT  : out std_logic;
    MODE      : in  std_logic;
    PARAM_M   : in  std_logic_vector(1 downto 0);
    X_IN      : in  std_logic_vector(DATA_WIDTH-1 downto 0);
    Y_IN      : in  std_logic_vector(DATA_WIDTH-1 downto 0);
    Z_IN      : in  std_logic_vector(DATA_WIDTH-1 downto 0);
    X_OUT     : out std_logic_vector(DATA_WIDTH-1 downto 0);
    Y_OUT     : out std_logic_vector(DATA_WIDTH-1 downto 0);
    Z_OUT     : out std_logic_vector(DATA_WIDTH-1 downto 0));

end cordic_full;

architecture behavioral of cordic_full is

  --STARTL
  component cordic_base
    generic (
      ITER_WIDTH    : positive;
      ITERATIONS    : positive;
```

```
      DATA_WIDTH    : positive;
      SCALE_OUTPUT : boolean);
    port (
      CLK_IN    : in  std_logic;
      nRESET    : in  std_logic;
      DVAL_IN   : in  std_logic;
      DVAL_OUT  : out std_logic;
      MODE      : in  std_logic;                        -- unused
      PARAM_M   : in  std_logic_vector(1 downto 0);  -- unused
      X_IN      : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      Y_IN      : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      Z_IN      : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      X_OUT     : out std_logic_vector(DATA_WIDTH-1 downto 0);
      Y_OUT     : out std_logic_vector(DATA_WIDTH-1 downto 0);
      Z_OUT     : out std_logic_vector(DATA_WIDTH-1 downto 0));
  end component;

  constant cn_quart_pi : natural := integer((MATH_PI*real(2**(DATA_WIDTH-3)))/real(4));
  constant cn_half_pi  : natural := integer((MATH_PI*real(2**(DATA_WIDTH-3)))/real(2));
  constant cn_pi       : natural := integer(MATH_PI*real(2**(DATA_WIDTH-3)));

  type tab_saver is array (1 to 4) of boolean;

  signal sb_rot_sec_1, sb_rot_sec_2, sb_rot_sec_3, sb_rot_sec_4 : boolean;
  signal sb_active                                              : boolean;

  signal sv_in_x, sv_in_y, sv_in_z    : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal sv_out_x, sv_out_y, sv_out_z : std_logic_vector(DATA_WIDTH-1 downto 0);

  signal sl_dval_in, sl_dval_out : std_logic;

  signal si_x, si_y, si_z : integer range -(2**DATA_WIDTH) to 2**DATA_WIDTH-1;
  signal si_z_rot         : integer range -(2**DATA_WIDTH) to 2**DATA_WIDTH-1;

  --STOPL

begin  -- behavioral

  --STARTL
  cordic_1 : cordic_base
    generic map (
      ITER_WIDTH    => ITER_WIDTH,
      ITERATIONS    => ITERATIONS-2,
      DATA_WIDTH    => DATA_WIDTH,
```

```
      SCALE_OUTPUT => SCALE_OUTPUT)
    port map (
      CLK_IN    => CLK_IN,
      nRESET    => nRESET,
      DVAL_IN   => sl_dval_in,
      DVAL_OUT  => sl_dval_out,
      MODE      => MODE,
      PARAM_M   => PARAM_M,
      X_IN      => sv_in_x,
      Y_IN      => sv_in_y,
      Z_IN      => sv_in_z,
      X_OUT     => sv_out_x,
      Y_OUT     => sv_out_y,
      Z_OUT     => sv_out_z);

  --       +-----+-----+
  --       |  2  |  1  |
  --       +-----0-----+
  --       |  3  |  4  |
  --       +-----+-----+

  si_x <= conv_integer(X_IN);
  si_y <= conv_integer(Y_IN);
  si_z <= conv_integer(Z_IN);

  -- Make the angle between -90 and 0 degree, the vector will be in
  -- the first Quadrant. This ensures, that the cordic will converge
--  si_z_rot <= si_z - integer(2*cn_half_pi) when sb_rot_sec_2 else
--              si_z + integer(cn_half_pi) when sb_rot_sec_3 else
--              si_z - integer(cn_half_pi) when sb_rot_sec_1 else
--              si_z;

  proc_rot : process (CLK_IN, nRESET)
    variable vl_dval_in, vl_dval_out      : std_logic;
    variable vab_rot_saver, vab_vec_saver : tab_saver;
    variable vb_correct_angle             : boolean;
    variable vv_half_pi_rots              : std_logic_vector(1 downto 0);
    variable vi_x, vi_y, vi_z             : integer
      range -(2**DATA_WIDTH) to 2**DATA_WIDTH-1;
  begin  -- process proc_rot_before
    if nRESET = '0' then                  -- asynchronous reset (active low)
      vl_dval_out      := '0';
      vl_dval_in       := '0';
      vb_correct_angle := false;
```

```
    vv_half_pi_rots  := (others => '0');
  elsif rising_edge(CLK_IN) then      -- rising clock edge

    if DVAL_IN = '1' and vl_dval_in = '0' and not sb_active then

      vi_x := si_x;
      vi_y := si_y;
      vi_z := si_z;

      if vi_z >= -integer(cn_half_pi) and vi_z <= integer(cn_half_pi) then
        sl_dval_in <= '1';
      else
        vb_correct_angle := true;
      end if;

    elsif vb_correct_angle then

      if vi_z < -integer(cn_half_pi) then
        vi_x := -vi_x;
        vi_y := -vi_y;
        vi_z := vi_z + integer(cn_pi);

      elsif vi_z > integer(cn_half_pi) then
        vi_x := -vi_x;
        vi_y := -vi_y;
        vi_z := vi_z - integer(cn_pi);

      else
        sl_dval_in       <= '1';
        vb_correct_angle := false;
      end if;

    else
      sl_dval_in <= '0';
    end if;

    sv_in_x <= conv_std_logic_vector(vi_x, DATA_WIDTH);
    sv_in_y <= conv_std_logic_vector(vi_y, DATA_WIDTH);
    sv_in_z <= conv_std_logic_vector(vi_z, DATA_WIDTH);

    vl_dval_out := sl_dval_out;
    vl_dval_in  := DVAL_IN;
  end if;
end process proc_rot;
```

```
  DVAL_OUT <= sl_dval_out;
  X_OUT    <= sv_out_x;
  Y_OUT    <= sv_out_y;
  Z_OUT    <= sv_out_z;

  --STOPL

end behavioral;
```

Listing B.41: cordic_siggen.vhd

```
-------------------------------------------------------------------------------
-- Title      : Cordig Siggen
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : cordic_siggen.vhd
-- Author     : Daniel Glaser
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-11-01
-- Last update: 2007-01-21
-- Platform   : LFECP20E
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description: This module uses the cordic algorithm to implement a sinusodial
--              signal generator.
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-11-01  1.0      sidaglas        Created
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
use ieee.numeric_std.all;

use ieee.math_real.all;

entity cordic_siggen is

  generic (
    DATA_WIDTH : positive := 8;
```

```
    ITERATIONS : positive := 10;
    ITER_WIDTH : positive := 4);

  port (
    CLK_IN     : in  std_logic;
    nRESET     : in  std_logic;
    PHASE_INC  : in  std_logic_vector(DATA_WIDTH-1 downto 0);
    DATA_OUT   : out std_logic_vector(DATA_WIDTH-1 downto 0));

end cordic_siggen;

architecture behavioral of cordic_siggen is

  constant cn_angle_scale_factor : natural := 2**(DATA_WIDTH-3);

  constant ci_pi     : natural := integer(MATH_PI*real(cn_angle_scale_factor));
  constant ci_two_pi : natural := 2*ci_pi;

  -- This lut is only used for getting the scale_factor
  component cordic_lut
    generic (
      DATA_WIDTH : positive;
      ITER_WIDTH : positive);
    port (
      STEP        : in  std_logic_vector(ITER_WIDTH-1 downto 0);
      ALPHA_I     : out std_logic_vector(DATA_WIDTH-1 downto 0);
      TAN_ALPHA_I : out std_logic_vector(DATA_WIDTH-1 downto 0);
      K_I         : out std_logic_vector(DATA_WIDTH-1 downto 0);
      K_G         : out std_logic_vector(DATA_WIDTH-1 downto 0));
  end component;

  signal sv_vector_full_k : std_logic_vector(DATA_WIDTH-1 downto 0);

  component cordic_full
    generic (
      ITER_WIDTH : positive;
      ITERATIONS : positive;
      DATA_WIDTH : positive;
      SCALE_OUTPUT : boolean);
    port (
      CLK_IN    : in  std_logic;
      nRESET    : in  std_logic;
      DVAL_IN   : in  std_logic;
      DVAL_OUT  : out std_logic;
```

```
      MODE      : in  std_logic;
      PARAM_M   : in  std_logic_vector(1 downto 0);
      X_IN      : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      Y_IN      : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      Z_IN      : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      X_OUT     : out std_logic_vector(DATA_WIDTH-1 downto 0);
      Y_OUT     : out std_logic_vector(DATA_WIDTH-1 downto 0);
      Z_OUT     : out std_logic_vector(DATA_WIDTH-1 downto 0));
  end component;

  signal si_phase_inc : integer range -(2**(DATA_WIDTH-1)) to (2**(DATA_WIDTH-1))-1;
  signal sv_phase     : std_logic_vector(DATA_WIDTH-1 downto 0);

begin -- behavioral

  -- This lut is only used for getting the scale_factor. It will hopefully
  -- optimized down to a simple constant value
  cordic_lut_1: cordic_lut
    generic map (
      DATA_WIDTH => DATA_WIDTH,
      ITER_WIDTH => ITER_WIDTH)
    port map (
      STEP        => ITERATIONS-1,
      ALPHA_I     => open,
      TAN_ALPHA_I => open,
      K_I         => open,
      K_G         => sv_vector_full_k);

  cordic_full_1 : cordic_full
    generic map (
      ITER_WIDTH => ITER_WIDTH,
      ITERATIONS => ITERATIONS,
      DATA_WIDTH => DATA_WIDTH,
      SCALE_OUTPUT => false)
    port map (
      CLK_IN    => CLK_IN,
      nRESET    => nRESET,
      DVAL_IN   => DVAL_IN,
      DVAL_OUT  => DVAL_OUT,
      MODE      => '0',
      PARAM_M   => "01",
      X_IN      => sv_vector_full_k,
      Y_IN      => sv_vector_null,
      Z_IN      => sv_phase,
```

```
      X_OUT     => X_OUT,
      Y_OUT     => Y_OUT,
      Z_OUT     => Z_OUT);

  -- This process accumulates the phase increment to the full angle of the
  -- rotation
  proc_phase_inc : process (CLK_IN, nRESET)
    variable vi_phase : integer range -(2**(DATA_WIDTH-1)) to (2**(DATA_WIDTH-1))-1;
    variable vn_count : natural range 0 to ITERATIONS-1;
  begin  -- process proc_phase_inv
    if nRESET = '0' then                -- asynchronous reset (active low)
      vi_phase := 0;
    elsif rising_edge(CLK_IN) then      -- rising clock edge

      if vn_count = 0 then
        if vi_phase + si_phase_inc > ci_pi then
          vi_phase := ci_two_pi - vi_phase;
        else
          vi_phase := vi_phase + si_phase_inc;
        end if;
        sv_phase <= conv_std_logic_vector(vi_phase);
      end if;

      if vn_count = 0 then
        vn_count := ITERATIONS-1;
        sl_cordic_dval_in <= '1';
      else
        vn_count := vn_count - 1;
        sl_cordic_dval_in <= '0';
      end if;

    end if;

  end process proc_phase_inc;

  sv_vector_null <= (others => '0');
  si_phase_inc <= conv_integer(PHASE_INC);

end behavioral;
```

Listing B.42: stud_toplevel.vhd

```
-------------------------------------------------------------------------------
-- Title      : Students toplevel
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
```

```
-------------------------------------------------------------------------------
-- File       : stud_toplevel.vhd
-- Author     : Daniel Glaser
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-09-04
-- Last update: 2007-01-21
-- Platform   : LFECP20E
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description: This Toplevel is for student use. They should start here, as
--              here are already usable signals like parallel data from adc. It
--              simplyfies the work to be done.
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-09-04  1.0      sidaglas        Created
-- 2006-11-22  1.1      sidaglas        Modified
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;

entity stud_toplevel is

  generic (
    DATA_WIDTH : positive := 16);

  port (
    ---------------------------------------------------------------------------
    -- Generic signals
    ---------------------------------------------------------------------------
    CLK_FAST : in std_logic;            -- 49,152 MHz
    CLK_SLOW : in std_logic;            -- 0,768 MHz = 768 kHz

    nRESET : in std_logic;

    ---------------------------------------------------------------------------
    -- Human device interface (HDI)
    ---------------------------------------------------------------------------
```

```
-- Inputs
SWITCH_1 : in std_logic;
SWITCH_2 : in std_logic;
SWITCH_3 : in std_logic;
SWITCH_4 : in std_logic;
SWITCH_5 : in std_logic;
SWITCH_6 : in std_logic;
SWITCH_7 : in std_logic;
SWITCH_8 : in std_logic;

BUTTON_1 : in std_logic;
BUTTON_2 : in std_logic;
BUTTON_3 : in std_logic;
BUTTON_4 : in std_logic;
BUTTON_5 : in std_logic;
BUTTON_6 : in std_logic;
BUTTON_7 : in std_logic;
BUTTON_8 : in std_logic;

-- Outputs
STATUS_L_RED : out std_logic;
STATUS_L_YEL : out std_logic;
STATUS_L_GRE : out std_logic;
STATUS_M_RED : out std_logic;
STATUS_M_YEL : out std_logic;
STATUS_M_GRE : out std_logic;
STATUS_R_RED : out std_logic;
STATUS_R_YEL : out std_logic;
STATUS_R_GRE : out std_logic;

-- The segment leds take hex numbers and show them
SEGMENT_LED1 : out std_logic_vector(3 downto 0);
SEGMENT_LED2 : out std_logic_vector(3 downto 0);
SEGMENT_LED3 : out std_logic_vector(3 downto 0);
SEGMENT_LED4 : out std_logic_vector(3 downto 0);

-- The Bargraph switches on the count of lights you say
BARGRAPH_LEFT    : out std_logic_vector(7 downto 0);
BARGRAPH_RIGHT   : out std_logic_vector(7 downto 0);
BARGRAPH_DEC_EN : out std_logic;

-----------------------------------------------------------------------------
-- PC communications
-----------------------------------------------------------------------------
```

```
    PC_SDIN  : in  std_logic;
    PC_SDOUT : out std_logic;

    ------------------------------------------------------------------------
    -- Peripherial connections
    ------------------------------------------------------------------------
    -- Digitally controlled microphone preamplifier
    MA_GAIN : out std_logic_vector(5 downto 0);

    -- Analog to digital converter
    AD_PDIN_L : in std_logic_vector(23 downto 0);
    AD_PDIN_R : in std_logic_vector(23 downto 0);
    AD_DVAL   : in std_logic;

    -- Digital to analog converter
    DA_PDOUT_L : out std_logic_vector(23 downto 0);
    DA_PDOUT_R : out std_logic_vector(23 downto 0);
    DA_DVAL    : out std_logic);

end stud_toplevel;

architecture behavioral of stud_toplevel is

  --STARL
  component cordic_siggen
    generic (
      DATA_WIDTH : positive;
      ITERATIONS : positive;
      ITER_WIDTH : positive);
    port (
      CLK_IN     : in  std_logic;
      nRESET     : in  std_logic;
      PHASE_INC : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      DATA_OUT   : out std_logic_vector(DATA_WIDTH-1 downto 0));
  end component;

  signal sl_cordic_dval_in  : std_logic;
  signal sl_cordic_dval_out : std_logic;

  signal sv_siggen_dout : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal sv_phase_inc : std_logic_vector(DATA_WIDTH-1 downto 0);
  --STOPL

begin  -- behavioral
```

```
  --STARL
  cordic_siggen_1: cordic_siggen
    generic map (
      DATA_WIDTH => DATA_WIDTH,
      ITERATIONS => 10,
      ITER_WIDTH => 4)
    port map (
      CLK_IN     => CLK_SLOW,
      nRESET     => nRESET,
      PHASE_INC => sv_phase_inc,
      DATA_OUT   => sv_siggen_dout);

  process (CLK_SLOW, nRESET)
    variable vl_edge_detect : std_logic;
  begin  -- process
    if nRESET = '0' then                  -- asynchronous reset (active low)
      sv_phase_inc <= (others => '0');
    elsif rising_edge(CLK_SLOW) then      -- rising clock edge
      if vl_edge_detect = '0' and BUTTON_1 = '1' then
        sv_phase_inc <= conv_std_logic_vector(cn_angle_scale_factor/20, DATA_WIDTH);
      else
        sv_phase_inc <= (others => '0');
      end if;
      vl_edge_detect := BUTTON_1
    end if;
  end process;

  BARGRAPH_DEC_EN <= '0';
  BARGRAPH_LEFT   <= sv_siggen_dout;
  --STOPL

end behavioral;
```

B.3.4 Levelanzeige

Listing B.43: levelmeter.vhd

```
-------------------------------------------------------------------------------
-- Title      : Levelmeter
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : levelmeter.vhd
-- Author     : Daniel Glaser
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
```

```
-- Created    : 2006-11-01
-- Last update: 2007-01-21
-- Platform   : LFECP20E
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description: This module calculates the mean square of an audio signal. Be
--              careful with the output signal. It is an unsigned vector.
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-11-01  1.0      sidaglas        Created
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
use ieee.numeric_std.all;

entity levelmeter is

  generic (
    DATA_WIDTH        : positive := 8;
    INTEGRATION_WIDTH : positive := 10);

  port (
    CLK_IN    : in  std_logic;
    nRESET    : in  std_logic;
    DIN_VAL   : in  std_logic;
    DOUT_VAL  : out std_logic;
    DATA_IN   : in  std_logic_vector(DATA_WIDTH-1 downto 0);
    DATA_OUT  : out std_logic_vector(DATA_WIDTH-1 downto 0));

end levelmeter;

architecture behavioral of levelmeter is

  --STARTL
  constant cn_integration_time : natural := (2**INTEGRATION_WIDTH)-1;

  constant ci_data_min : integer := -(2**(DATA_WIDTH-1));
  constant ci_data_max : integer := (2**(DATA_WIDTH-1))-1;
```

```
  constant ci_square_min : integer := ci_data_min;
  constant ci_square_max : integer := ci_data_max;

--  constant ci_square_min : integer := -(2**(2*(DATA_WIDTH-1)-1));
--  constant ci_square_max : integer := (2**(2*(DATA_WIDTH-1)-1))-1;

  constant ci_acc_min : integer := -(2**(DATA_WIDTH+INTEGRATION_WIDTH-1));
  constant ci_acc_max : integer := (2**(DATA_WIDTH+INTEGRATION_WIDTH-1))-1;

--  constant ci_acc_min : integer := -(2**(2*(DATA_WIDTH+INTEGRATION_WIDTH-1)-1));
--  constant ci_acc_max : integer := (2**(2*(DATA_WIDTH+INTEGRATION_WIDTH-1)-1))-1;

  constant cn_acc_width : natural := DATA_WIDTH+INTEGRATION_WIDTH-1;

  signal sl_clear_intreg : std_logic;
  signal sl_reg_input    : std_logic;

  signal si_input_data : integer range ci_data_min to ci_data_max;

  -- pragma synthesis_off
  signal si_square     : integer range ci_square_min to ci_square_max;
  signal si_square_acc : integer range ci_acc_min to ci_acc_max;
  -- pragma synthesis_on

  --STOPL

begin  -- behavioral

  assert DATA_WIDTH < 17
    report "DATA_WIDTH must be equal or less than 16"
    severity error;
  assert INTEGRATION_WIDTH + DATA_WIDTH < 33
    report "DATA_WIDTH or INTEGRATION_WIDTH to large. DATA_WIDTH + INTEGRATION_WIDTH mus
    severity error;
  --STARTL
  proc_count_indata : process (CLK_IN, nRESET)
    variable vn_count        : natural range 0 to cn_integration_time;
    variable vl_dval_in_edge : std_logic := '0';
  begin  -- process proc_count_indata
    if nRESET = '0' then                -- asynchronous reset (active low)
      vn_count      := 1;
      si_input_data <= 0;
    elsif rising_edge(CLK_IN) then      -- rising clock edge
      if vl_dval_in_edge = '0' and DIN_VAL = '1' then
```

```
      if vn_count = 0 then
        vn_count        := cn_integration_time;
        sl_clear_intreg <= '1';
      else
        sl_clear_intreg <= '0';
        vn_count        := vn_count - 1;
      end if;
      sl_reg_input  <= '1';
      si_input_data <= conv_integer(DATA_IN);
    else
      sl_reg_input <= '0';
    end if;
  end if;
end process proc_count_indata;

proc_square_and_acc_data : process (CLK_IN, nRESET)
  variable vi_square     : integer range ci_square_min to ci_square_max;
  variable vi_square_acc : integer range ci_acc_min to ci_acc_max;
  variable vv_output     : std_logic_vector(cn_acc_width-1 downto 0);
begin  -- process proc_square_and_acc_data
  if nRESET = '0' then                  -- asynchronous reset (active low)
    vi_square     := 0;
    vi_square_acc := 0;
    vv_output     := (others => '0');
  elsif rising_edge(CLK_IN) then        -- rising clock edge

    if sl_reg_input = '1' then
      if sl_clear_intreg = '1' then
        -- The following order of vv_output is intended
        vv_output     := conv_std_logic_vector(vi_square_acc, cn_acc_width);
        -- Throwing away the first bit, because it's only the sign and so
        -- never set
        DATA_OUT      <= vv_output(vv_output'left downto vv_output'left-DATA_WIDTH+1);
        vi_square_acc := vi_square;
        DOUT_VAL      <= '1';
      else
        DOUT_VAL      <= '0';
        vi_square_acc := vi_square_acc + vi_square;
      end if;
      vi_square     := ((si_input_data * si_input_data)-1)/(2**(DATA_WIDTH-1));
      -- pragma synthesis_off
      si_square     <= vi_square;
      si_square_acc <= vi_square_acc;
      -- pragma synthesis_on
```

```
      end if;

    end if;
  end process proc_square_and_acc_data;
  --STOPL

end behavioral;
```

Listing B.44: stud_toplevel.vhd

```
-------------------------------------------------------------------------------
-- Title      : Students toplevel
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : stud_toplevel.vhd
-- Author     : Daniel Glaser
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-09-04
-- Last update: 2006-11-26
-- Platform   : LFECP20E
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description: This Toplevel is for student use. They should start here, as
--              here are already usable signals like parallel data from adc. It
--              simplyfies the work to be done.
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-09-04  1.0      sidaglas        Created
-- 2006-11-22  1.1      sidaglas        Modified
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;

entity stud_toplevel is

  generic (
    DATA_WIDTH : positive := 8);

  port (
```

```
-------------------------------------------------------------------------------
-- Generic signals
-------------------------------------------------------------------------------
CLK_FAST : in std_logic;                -- 49,152 MHz
CLK_SLOW : in std_logic;                -- 0,768 MHz = 768 kHz

nRESET : in std_logic;

-------------------------------------------------------------------------------
-- Human device interface (HDI)
-------------------------------------------------------------------------------

-- Inputs
SWITCH_1 : in std_logic;
SWITCH_2 : in std_logic;
SWITCH_3 : in std_logic;
SWITCH_4 : in std_logic;
SWITCH_5 : in std_logic;
SWITCH_6 : in std_logic;
SWITCH_7 : in std_logic;
SWITCH_8 : in std_logic;

BUTTON_1 : in std_logic;
BUTTON_2 : in std_logic;
BUTTON_3 : in std_logic;
BUTTON_4 : in std_logic;
BUTTON_5 : in std_logic;
BUTTON_6 : in std_logic;
BUTTON_7 : in std_logic;
BUTTON_8 : in std_logic;

-- Outputs
STATUS_L_RED : out std_logic;
STATUS_L_YEL : out std_logic;
STATUS_L_GRE : out std_logic;
STATUS_M_RED : out std_logic;
STATUS_M_YEL : out std_logic;
STATUS_M_GRE : out std_logic;
STATUS_R_RED : out std_logic;
STATUS_R_YEL : out std_logic;
STATUS_R_GRE : out std_logic;

-- The segment leds take hex numbers and show them
SEGMENT_LED1 : out std_logic_vector(3 downto 0);
```

```
    SEGMENT_LED2 : out std_logic_vector(3 downto 0);
    SEGMENT_LED3 : out std_logic_vector(3 downto 0);
    SEGMENT_LED4 : out std_logic_vector(3 downto 0);

    -- The Bargraph switches on the count of lights you say
    BARGRAPH_LEFT    : out std_logic_vector(7 downto 0);
    BARGRAPH_RIGHT   : out std_logic_vector(7 downto 0);
    BARGRAPH_DEC_EN : out std_logic;

    ---------------------------------------------------------------------------
    -- PC communications
    ---------------------------------------------------------------------------
    PC_SDIN  : in  std_logic;
    PC_SDOUT : out std_logic;

    ---------------------------------------------------------------------------
    -- Peripherial connections
    ---------------------------------------------------------------------------
    -- Digitally controlled microphone preamplifier
    MA_GAIN : out std_logic_vector(5 downto 0);

    -- Analog to digital converter
    AD_PDIN_L : in std_logic_vector(23 downto 0);
    AD_PDIN_R : in std_logic_vector(23 downto 0);
    AD_DVAL   : in std_logic;

    -- Digital to analog converter
    DA_PDOUT_L : out std_logic_vector(23 downto 0);
    DA_PDOUT_R : out std_logic_vector(23 downto 0);
    DA_DVAL    : out std_logic);

end stud_toplevel;

architecture behavioral of stud_toplevel is

  --STARTL
  component levelmeter
    generic (
      DATA_WIDTH         : positive;
      INTEGRATION_WIDTH : positive);
    port (
      CLK_IN   : in  std_logic;
      nRESET   : in  std_logic;
      DIN_VAL  : in  std_logic;
```

```
      DOUT_VAL : out std_logic;
      DATA_IN  : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      DATA_OUT : out std_logic_vector(DATA_WIDTH-1 downto 0));
  end component;

  signal sv_lm_din      : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal sl_lm_dout_val : std_logic;
  --STOPL

begin  -- behavioral

  --STARTL
  levelmeter_1 : levelmeter
    generic map (
      DATA_WIDTH         => 8,
      INTEGRATION_WIDTH => 10)
    port map (
      CLK_IN    => CLK_SLOW,
      nRESET    => nRESET,
      DIN_VAL   => AD_DVAL,
      DOUT_VAL => sl_lm_dout_val,
      DATA_IN   => sv_lm_din,
      DATA_OUT => BARGRAPH_LEFT);

  sv_lm_din <= AD_DOUT(AD_DOUT'left downto AD_DOUT'left-DATA_WIDTH+1);
  BARGRAPH_DEC_EN <= '1';
  --STOPL

end behavioral;
```

B.3.5 Modulator

Listing B.45: stud_toplevel.vhd

```
-------------------------------------------------------------------------------
-- Title      : Students toplevel
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : stud_toplevel.vhd
-- Author     : Daniel Glaser
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-09-04
-- Last update: 2007-01-21
-- Platform   : LFECP20E
-- Standard   : VHDL'87
```

```
-------------------------------------------------------------------------------
-- Description: This Toplevel is for student use. They should start here, as
--              here are already usable signals like parallel data from adc. It
--              simplyfies the work to be done.
--STARTL
--              Tutors should know, that BUTTON1 is the test source for this
--              solution.
--STOPL
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-09-04  1.0      sidaglas        Created
-- 2006-11-22  1.1      sidaglas        Modified
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;

entity stud_toplevel is

  generic (
    DATA_WIDTH : positive := 8);

  port (
    ---------------------------------------------------------------------------
    -- Generic signals
    ---------------------------------------------------------------------------
    CLK_FAST : in std_logic;            -- 49,152 MHz
    CLK_SLOW : in std_logic;            -- 0,768 MHz = 768 kHz

    nRESET : in std_logic;

    ---------------------------------------------------------------------------
    -- Human device interface (HDI)
    ---------------------------------------------------------------------------

    -- Inputs
    SWITCH_1 : in std_logic;
    SWITCH_2 : in std_logic;
    SWITCH_3 : in std_logic;
```

```
SWITCH_4 : in std_logic;
SWITCH_5 : in std_logic;
SWITCH_6 : in std_logic;
SWITCH_7 : in std_logic;
SWITCH_8 : in std_logic;

BUTTON_1 : in std_logic;
BUTTON_2 : in std_logic;
BUTTON_3 : in std_logic;
BUTTON_4 : in std_logic;
BUTTON_5 : in std_logic;
BUTTON_6 : in std_logic;
BUTTON_7 : in std_logic;
BUTTON_8 : in std_logic;

-- Outputs
STATUS_L_RED : out std_logic;
STATUS_L_YEL : out std_logic;
STATUS_L_GRE : out std_logic;
STATUS_M_RED : out std_logic;
STATUS_M_YEL : out std_logic;
STATUS_M_GRE : out std_logic;
STATUS_R_RED : out std_logic;
STATUS_R_YEL : out std_logic;
STATUS_R_GRE : out std_logic;

-- The segment leds take hex numbers and show them
SEGMENT_LED1 : out std_logic_vector(3 downto 0);
SEGMENT_LED2 : out std_logic_vector(3 downto 0);
SEGMENT_LED3 : out std_logic_vector(3 downto 0);
SEGMENT_LED4 : out std_logic_vector(3 downto 0);

-- The Bargraph switches on the count of lights you say
BARGRAPH_LEFT    : out std_logic_vector(7 downto 0);
BARGRAPH_RIGHT   : out std_logic_vector(7 downto 0);
BARGRAPH_DEC_EN : out std_logic;

-----------------------------------------------------------------------
-- PC communications
-----------------------------------------------------------------------

PC_SDIN  : in  std_logic;
PC_SDOUT : out std_logic;

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

```
    -- Peripherial connections
    ------------------------------------------------------------------------
    -- Digitally controlled microphone preamplifier
    MA_GAIN : out std_logic_vector(5 downto 0);

    -- Analog to digital converter
    AD_PDIN_L : in std_logic_vector(23 downto 0);
    AD_PDIN_R : in std_logic_vector(23 downto 0);
    AD_DVAL   : in std_logic;

    -- Digital to analog converter
    DA_PDOUT_L : out std_logic_vector(23 downto 0);
    DA_PDOUT_R : out std_logic_vector(23 downto 0);
    DA_DVAL    : out std_logic);

end stud_toplevel;

architecture behavioral of stud_toplevel is

  --STARL
  component cordic_siggen
    generic (
      DATA_WIDTH : positive;
      ITERATIONS : positive;
      ITER_WIDTH : positive);
    port (
      CLK_IN    : in  std_logic;
      nRESET    : in  std_logic;
      PHASE_INC : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      DATA_OUT  : out std_logic_vector(DATA_WIDTH-1 downto 0));
  end component;

  signal sl_cordic_dval_in  : std_logic;
  signal sl_cordic_dval_out : std_logic;

  signal sv_siggen_dout : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal sv_phase_inc : std_logic_vector(DATA_WIDTH-1 downto 0);
  --STOPL

begin  -- behavioral

  --STARL
  cordic_siggen_1: cordic_siggen
    generic map (
```

```
      DATA_WIDTH => DATA_WIDTH,
      ITERATIONS => 10,
      ITER_WIDTH => 4)
    port map (
      CLK_IN     => CLK_SLOW,
      nRESET     => nRESET,
      PHASE_INC => sv_phase_inc,
      DATA_OUT   => sv_siggen_dout);

  process (CLK_SLOW, nRESET)
    variable vl_edge_detect : std_logic;
  begin  -- process
    if nRESET = '0' then                  -- asynchronous reset (active low)
      sv_phase_inc <= (others => '0');
    elsif rising_edge(CLK_SLOW) then    -- rising clock edge
      if vl_edge_detect = '0' and BUTTON_1 = '1' then
        sv_phase_inc <= conv_std_logic_vector(cn_angle_scale_factor/20, DATA_WIDTH);
      else
        sv_phase_inc <= conv_std_logic_vector(cn_angle_scale_factor/10, DATA_WIDTH);
      end if;
      vl_edge_detect := BUTTON_1
    end if;
  end process;

  DA_PDOUT_L   <= sv_siggen_dout;
  --STOPL

end behavioral;
```

B.3.6 Bandpass

Listing B.46: bandpass.vhd

```
-------------------------------------------------------------------------------
-- Title      : Bandpass
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : bandpass.vhd
-- Author     : Daniel Glaser
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-11-01
-- Last update: 2007-01-21
-- Platform   : LFECP20E
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
```

```
-- Description: This is the Bandpass in linear phase structure.
-----------------------------------------------------------------------------
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
-----------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-11-01  1.0      sidaglas        Created
-----------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
--use ieee.numeric_std.all;

use work.my_funcs.all;

use ieee.math_real.all;

--NOCOPY

entity bandpass is

  generic (
    DATA_WIDTH  : positive := 8;
    ORDER_WIDTH : positive := 4);
--    ORDER       : positive := 16);

  port (
    CLK_IN    : in  std_logic;
    nRESET    : in  std_logic;
    CE        : in  std_logic;
    DATA_IN   : in  std_logic_vector(DATA_WIDTH-1 downto 0);
    DATA_OUT  : out std_logic_vector(DATA_WIDTH-1 downto 0));

end bandpass;

architecture behavioral of bandpass is

  constant cn_order : natural := 2**ORDER_WIDTH;

  constant ci_pipe_min : integer := integer_min(DATA_WIDTH);  -- 8 Bit
  constant ci_pipe_max : integer := integer_max(DATA_WIDTH);

  constant ci_pipe_sum_min : integer := integer_min(DATA_WIDTH+1);  -- 9 Bit
```

```
  constant ci_pipe_sum_max : integer := integer_max(DATA_WIDTH+1);

  constant ci_pipe_sum_scale : integer := 2**DATA_WIDTH;

  constant ci_prod_min : integer := integer_min((2*(DATA_WIDTH+1))-1);  -- 17 Bit
  constant ci_prod_max : integer := integer_max((2*(DATA_WIDTH+1))-1);

  constant ci_prod_sum_min : integer := integer_min((2*(DATA_WIDTH+1)));  -- 18 Bit
  constant ci_prod_sum_max : integer := integer_max((2*(DATA_WIDTH+1)));

  constant ci_prod_sum_sum_min : integer := integer_min((2*(DATA_WIDTH+1))+1);
-- 19 Bit
  constant ci_prod_sum_sum_max : integer := integer_max((2*(DATA_WIDTH+1))+1);

  -- (DATA_WIDTH+1)              for the adder before the multiplier
  -- 2*(DATA_WIDTH+1)-1          for the multiplier itself
  -- ORDER_WIDTH                 for the sum of adders
  -- ORDER_WIDTH-1               adder before the multiplier subtracted
  -- ============================================================
  -- Gives: 2*(DATA_WIDTH+1) - 1 + ORDER_WIDTH - 1
  constant ci_data_out_min : integer := integer_min((2*(DATA_WIDTH+1))+ORDER_WIDTH-2);
-- 20 Bit
  constant ci_data_out_max : integer := integer_max((2*(DATA_WIDTH+1))+ORDER_WIDTH-2);

  -- This should be fixed for orders not powers of two
--  constant cn_data_out_scale : natural := 2**(DATA_WIDTH+1);
  constant cn_data_out_scale : natural := 2**DATA_WIDTH;

  subtype tui_pipe is integer range ci_pipe_min to ci_pipe_max;
  subtype tui_pipe_sum is integer range ci_pipe_sum_min to ci_pipe_sum_max;
  subtype tui_prod is integer range ci_prod_min to ci_prod_max;
  subtype tui_prod_sum is integer range ci_prod_sum_min to ci_prod_sum_max;
  subtype tui_prod_sum_sum is integer range ci_prod_sum_sum_min to ci_prod_sum_sum_max;

  subtype tui_whole_sum is integer range ci_data_out_min to ci_data_out_max;
  subtype tui_whole_scaled is integer range ci_pipe_min to ci_pipe_max;

  type tuai_pipe is array (0 to cn_order-1) of tui_pipe;
  type tuai_pipe_sum is array (0 to (cn_order/2)-1) of tui_pipe_sum;
  type tuai_prod is array (0 to (cn_order/2)-1) of tui_prod;
  type tuai_prod_sum is array (0 to (cn_order/4)-1) of tui_prod_sum;
  type tuai_prod_sum_sum is array (0 to (cn_order/4)-1) of tui_prod_sum_sum;

  --Synopsys translate_off
```

```
  signal sai_pipe          : tuai_pipe;
  --Synopsys translate_on
  signal sai_pipe_sum      : tuai_pipe_sum;
  signal sai_prod          : tuai_prod;
  signal sai_prod_sum      : tuai_prod_sum;
  signal sai_prod_sum_sum  : tuai_prod_sum_sum;

  signal si_whole_sum     : tui_whole_sum;
  signal si_whole_scaled  : tui_whole_scaled;

  constant cai_coefficients : tuai_pipe_sum :=
    (integer(real(-0.00101)*real(ci_pipe_sum_max)),     -- c0 = c15
     (integer(real(-0.00521)*real(ci_pipe_sum_max))),   -- c1 = c14
     (integer(real(-0.01269)*real(ci_pipe_sum_max))),   -- c2 = c13
     (integer(real(-0.01214)*real(ci_pipe_sum_max))),   -- c3 = c12
     (integer(real(+0.01830)*real(ci_pipe_sum_max))),   -- c4 = c11
     (integer(real(+0.08914)*real(ci_pipe_sum_max))),   -- c5 = c10
     (integer(real(+0.17962)*real(ci_pipe_sum_max))),   -- c6 = c9
     (integer(real(+0.24399)*real(ci_pipe_sum_max))));  -- c7 = c8

begin  -- behavioral

  proc_filter : process (CLK_IN, nRESET)
    variable vi_whole_sum : tui_whole_sum;
    variable vai_pipe     : tuai_pipe;
  begin  -- process proc_filter
    if nRESET = '0' then                -- asynchronous reset (active low)
      vai_pipe         := (others => 0);
      sai_pipe         <= (others => 0);
      sai_pipe_sum     <= (others => 0);
      sai_prod         <= (others => 0);
      sai_prod_sum     <= (others => 0);
      sai_prod_sum_sum <= (others => 0);
      si_whole_sum     <= 0;
      si_whole_scaled  <= 0;
      DATA_OUT         <= (others => '0');
    elsif rising_edge(CLK_IN) then      -- rising clock edge

      if CE = '1' then

        vai_pipe(1 to cn_order-1) := vai_pipe(0 to cn_order-2);

        --Synopsys translate_off
        sai_pipe <= vai_pipe;
```

```
        --Synopsys translate_on

        -- We put new data into the pipeline at position 0
        vai_pipe(0) := conv_integer(DATA_IN);

      end if;

      for j in 0 to (cn_order/2)-1 loop
        sai_prod(j)     <= sai_pipe_sum(j) * cai_coefficients(j);
        sai_pipe_sum(j) <= vai_pipe(j) + vai_pipe(cn_order-j-1);
      end loop;  -- j

      for k in 0 to (cn_order/4)-1 loop
        sai_prod_sum(k) <= sai_prod(2*k) + sai_prod((2*k)+1);
      end loop;  -- k

      vi_whole_sum := 0;
      for l in 0 to (cn_order/8)-1 loop
        vi_whole_sum         := vi_whole_sum + sai_prod_sum_sum(l);
        sai_prod_sum_sum(l) <= sai_prod_sum(2*l) + sai_prod_sum((2*l)+1);
      end loop;  -- l

      si_whole_sum    <= vi_whole_sum;
      si_whole_scaled <= si_whole_sum/cn_data_out_scale;
      DATA_OUT        <= conv_std_logic_vector(si_whole_scaled, DATA_WIDTH);


    end if;
  end process proc_filter;

end behavioral;
```

Listing B.47: stud_toplevel.vhd

```
-------------------------------------------------------------------------------
-- Title      : Students toplevel
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : stud_toplevel.vhd
-- Author     : Daniel Glaser
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-09-04
-- Last update: 2006-11-27
-- Platform   : LFECP20E
-- Standard   : VHDL'87
```

```
-------------------------------------------------------------------------------
-- Description: This Toplevel is for student use. They should start here, as
--              here are already usable signals like parallel data from adc. It
--              simplyfies the work to be done.
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-09-04  1.0      sidaglas        Created
-- 2006-11-22  1.1      sidaglas        Modified
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;

entity stud_toplevel is

  generic (
    DATA_WIDTH : positive := 8);

  port (
    -------------------------------------------------------------------------
    -- Generic signals
    -------------------------------------------------------------------------
    CLK_FAST : in std_logic;            -- 49,152 MHz
    CLK_SLOW : in std_logic;            -- 0,768 MHz = 768 kHz

    nRESET : in std_logic;

    -------------------------------------------------------------------------
    -- Human device interface (HDI)
    -------------------------------------------------------------------------

    -- Inputs
    SWITCH_1 : in std_logic;
    SWITCH_2 : in std_logic;
    SWITCH_3 : in std_logic;
    SWITCH_4 : in std_logic;
    SWITCH_5 : in std_logic;
    SWITCH_6 : in std_logic;
    SWITCH_7 : in std_logic;
```

```
SWITCH_8 : in std_logic;

BUTTON_1 : in std_logic;
BUTTON_2 : in std_logic;
BUTTON_3 : in std_logic;
BUTTON_4 : in std_logic;
BUTTON_5 : in std_logic;
BUTTON_6 : in std_logic;
BUTTON_7 : in std_logic;
BUTTON_8 : in std_logic;

-- Outputs
STATUS_L_RED : out std_logic;
STATUS_L_YEL : out std_logic;
STATUS_L_GRE : out std_logic;
STATUS_M_RED : out std_logic;
STATUS_M_YEL : out std_logic;
STATUS_M_GRE : out std_logic;
STATUS_R_RED : out std_logic;
STATUS_R_YEL : out std_logic;
STATUS_R_GRE : out std_logic;

-- The segment leds take hex numbers and show them
SEGMENT_LED1 : out std_logic_vector(3 downto 0);
SEGMENT_LED2 : out std_logic_vector(3 downto 0);
SEGMENT_LED3 : out std_logic_vector(3 downto 0);
SEGMENT_LED4 : out std_logic_vector(3 downto 0);

-- The Bargraph switches on the count of lights you say
BARGRAPH_LEFT    : out std_logic_vector(7 downto 0);
BARGRAPH_RIGHT   : out std_logic_vector(7 downto 0);
BARGRAPH_DEC_EN : out std_logic;

-----------------------------------------------------------------------
-- PC communications
-----------------------------------------------------------------------

PC_SDIN  : in  std_logic;
PC_SDOUT : out std_logic;

-----------------------------------------------------------------------
-- Peripherial connections
-----------------------------------------------------------------------

-- Digitally controlled microphone preamplifier
MA_GAIN : out std_logic_vector(5 downto 0);
```

```
    -- Analog to digital converter
    AD_PDIN_L : in std_logic_vector(23 downto 0);
    AD_PDIN_R : in std_logic_vector(23 downto 0);
    AD_DVAL   : in std_logic;

    -- Digital to analog converter
    DA_PDOUT_L : out std_logic_vector(23 downto 0);
    DA_PDOUT_R : out std_logic_vector(23 downto 0);
    DA_DVAL    : out std_logic);

end stud_toplevel;

architecture behavioral of stud_toplevel is

  --STARTL
  component bandpass
    generic (
      DATA_WIDTH  : positive;
      ORDER_WIDTH : positive);
    port (
      CLK_IN   : in  std_logic;
      nRESET   : in  std_logic;
      CE       : in  std_logic;
      DATA_IN  : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      DATA_OUT : out std_logic_vector(DATA_WIDTH-1 downto 0));
  end component;

  signal sl_bp_ce                      : std_logic;
  signal sv_bp_data_in, sv_bp_data_out : std_logic_vector(DATA_WIDTH-1 downto 0);
  --STOPL

begin  -- behavioral

  -- Hint: Order should be multiples of 8 for easier scaling with DSP-Blocks
  --STARTL
  bandpass_1 : bandpass
    generic map (
      DATA_WIDTH  => DATA_WIDTH,
      ORDER_WIDTH => 4)
    port map (
      CLK_IN   => CLK_SLOW,
      nRESET   => nRESET,
      CE       => sl_bp_ce,
```

```
      DATA_IN  => sv_bp_data_in,
      DATA_OUT => sv_bp_data_out);

  proc_bp_test : process (CLK_SLOW, nRESET)
    variable vl_edge_detect : std_logic;
  begin -- process proc_bp_test
    if nRESET = '0' then                -- asynchronous reset (active low)
      sl_bp_ce       <= '0';
      vl_edge_detect := '0';
      sv_bp_data_out <= (others => '0');
      DA_PDOUT_L     <= (others => '0');
    elsif rising_edge(CLK_SLOW) then    -- rising clock edge
      if vl_edge_detect = '0' and AD_DVAL = '1' then
        sl_bp_ce <= '1';
      else
        sl_bp_ce <= '0';
      end if;
      vl_edge_detect := AD_DVAL;

      sv_bp_data_in <= AD_PDIN_L(AD_PDIN_L'left downto AD_PDIN_L'left-DATA_WIDTH+1);

      DA_PDOUT_L(DA_PDOUT_L'left downto DA_PDOUT_L'left-DATA_WIDTH+1) <= sv_bp_data_in;
    end if;
  end process proc_bp_test;

  --STOPL

end behavioral;
```

B.3.7 Filteroptimierung

Listing B.48: filteropt.vhd

```
-------------------------------------------------------------------------------
-- Title      : Filter Optimization
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : filteropt.vhd
-- Author     : Daniel Glaser
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-11-01
-- Last update: 2007-01-23
-- Platform   : LFECP20E
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
```

```
-- Description: This module implements a DSP-Block-Optimized version of the
--              previosly discussed filter.
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author  Description
-- 2006-11-01  1.0      sidaglas        Created
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

use ieee.math_real.all;

--NOCOPY

entity filteropt is

  generic (
    FILTER_ORDER : positive := 37;
    DATA_WIDTH   : positive := 8);

  port (
    CLK_IN    : in  std_logic;
    nRESET    : in  std_logic;
    DVAL_IN   : in  std_logic;
    DVAL_OUT  : out std_logic;
    DATA_IN   : in  std_logic_vector(DATA_WIDTH-1 downto 0);
    DATA_OUT  : out std_logic_vector(DATA_WIDTH-1 downto 0));

end filteropt;

architecture behavioral of filteropt is

  constant cn_mult_per_block : natural := 8;
  constant cn_pipe_sum_count : natural := (FILTER_ORDER+1)/2;
  constant cn_block_count    : natural := (FILTER_ORDER-1)/(2*cn_mult_per_block)+1;
  constant cn_scale_width    : natural := DATA_WIDTH+1; -- Empirisch emitteln!

  subtype ti_pipe is integer range -2**(DATA_WIDTH-1) to 2**(DATA_WIDTH-1)-1;
-- 8 Bit
```

```
  subtype ti_pipe_sum is integer range -2**(DATA_WIDTH) to 2**(DATA_WIDTH)-1;
-- 9 Bit
  subtype ti_multed is integer range -2**(2*(DATA_WIDTH+1)-1) to 2**(2*(DATA_WIDTH+1)-1)
-- 18 Bit
  subtype ti_mult_sumed is integer range -2**(2*(DATA_WIDTH+1)) to 2**(2*(DATA_WIDTH+1))
-- 19 Bit
  subtype ti_maced is integer range -2**(2*(DATA_WIDTH+2)-1) to 2**(2*(DATA_WIDTH+2)-1)-
-- 20 Bit
  subtype ti_endsum is integer range -2**(2*(DATA_WIDTH+2)) to 2**(2*(DATA_WIDTH+2))-1;
-- 21 Bit

  type tai_pipe is array (0 to FILTER_ORDER-1) of ti_pipe;
  type tai_pipe_sum is array (0 to cn_mult_per_block*cn_block_count-1) of ti_pipe_sum;
  type tai_mult_in is array (0 to cn_mult_per_block-1) of ti_pipe_sum;
  type tai_mult_out is array (0 to cn_mult_per_block-1) of ti_multed;
  type tai_mult_sumed is array (0 to cn_mult_per_block/2-1) of ti_mult_sumed;
  type tai_maced is array (0 to 1) of ti_maced;

  constant ci_pipe_sum_max : integer := ti_pipe_sum'high;

  constant cai_coefficients : tai_pipe_sum :=
    (integer(real(-0.00128)*real(ci_pipe_sum_max)),     -- c0, c36
     (integer(real(-0.04028)*real(ci_pipe_sum_max))),   -- c1, c35
     (integer(real(-0.02579)*real(ci_pipe_sum_max))),   -- c2, c34
     (integer(real(-0.02100)*real(ci_pipe_sum_max))),   -- c3, c33
     (integer(real(-0.00510)*real(ci_pipe_sum_max))),   -- c4, c32
     (integer(real(+0.01782)*real(ci_pipe_sum_max))),   -- c5, c31
     (integer(real(+0.04010)*real(ci_pipe_sum_max))),   -- c6, c30
     (integer(real(+0.05267)*real(ci_pipe_sum_max))),   -- c7, c29
     (integer(real(+0.04843)*real(ci_pipe_sum_max))),   -- c8, c28
     (integer(real(+0.02582)*real(ci_pipe_sum_max))),   -- c9, c27
     (integer(real(-0.00974)*real(ci_pipe_sum_max))),   -- c10, c26
     (integer(real(-0.04694)*real(ci_pipe_sum_max))),   -- c11, c25
     (integer(real(-0.07233)*real(ci_pipe_sum_max))),   -- c12, c24
     (integer(real(-0.07553)*real(ci_pipe_sum_max))),   -- c13, c23
     (integer(real(-0.05338)*real(ci_pipe_sum_max))),   -- c14, c22
     (integer(real(-0.01187)*real(ci_pipe_sum_max))),   -- c15, c21
     (integer(real(+0.03558)*real(ci_pipe_sum_max))),   -- c16, c20
     (integer(real(+0.07288)*real(ci_pipe_sum_max))),   -- c17, c19
     (integer(real(+0.08694)*real(ci_pipe_sum_max))),   -- c18

     (integer(real(+0.00000)*real(ci_pipe_sum_max))),
     (integer(real(+0.00000)*real(ci_pipe_sum_max))),
     (integer(real(+0.00000)*real(ci_pipe_sum_max))),
```

```
    (integer(real(+0.00000)*real(ci_pipe_sum_max))),
    (integer(real(+0.00000)*real(ci_pipe_sum_max))));

  signal sai_pipe     : tai_pipe;
  signal sai_pipe_sum : tai_pipe_sum := (others => 0);

  signal sai_multiplier_in_a, sai_multiplier_in_b : tai_mult_in;
  signal sai_multiplier_out                       : tai_mult_out;
  signal sai_mult_sum                             : tai_mult_sumed;

  signal sai_mac    : tai_maced;
  signal si_endsum : ti_endsum;

  signal si_filter_outreg, si_filter_savereg : integer range -2**(2*DATA_WIDTH-1) to 2**

  signal sl_mult_active : std_logic;

  signal sn_current_block : natural range 0 to cn_block_count-1;

begin  -- behavioral


  ------------------------------------------------------------------------------
  -- Let the pipe flow/smoke
  ------------------------------------------------------------------------------
  shift_pipe_regs : process (CLK_IN, nRESET)
    variable vl_dval_in : std_logic := '0';
  begin  -- process shift_pipe_regs
    if nRESET = '0' then                -- asynchronous reset (active low)
      sai_pipe <= (others => 0);
    elsif rising_edge(CLK_IN) then      -- rising clock edge
      if DVAL_IN = '1' and vl_dval_in = '0' then
        -- Shift the values in the pipeline by one
        sai_pipe <= conv_integer(DATA_IN) & sai_pipe(0 to FILTER_ORDER-2);
      end if;
      vl_dval_in := DVAL_IN;
    end if;
  end process shift_pipe_regs;
  ------------------------------------------------------------------------------
  -- Let the pipe flow/smoke
  ------------------------------------------------------------------------------


  ------------------------------------------------------------------------------
  -- Odd or even symmetry ??? Here we handle both ;-)
```

```
-------------------------------------------------------------------------------
gen_pipe_sums : for li_pipe_sum in 0 to cn_pipe_sum_count-1 generate
  process (CLK_IN, nRESET)
    variable vi_sum_helper : ti_pipe_sum;
  begin  -- process
    if nRESET = '0' then                -- asynchronous reset (active low)
      sai_pipe_sum(li_pipe_sum) <= 0;
    elsif rising_edge(CLK_IN) then      -- rising clock edge

      vi_sum_helper := sai_pipe(li_pipe_sum);
      if li_pipe_sum /= (FILTER_ORDER - li_pipe_sum - 1) then
        vi_sum_helper := vi_sum_helper + sai_pipe(FILTER_ORDER - li_pipe_sum - 1);
      end if;
      sai_pipe_sum(li_pipe_sum) <= vi_sum_helper;

    end if;
  end process;
end generate gen_pipe_sums;
-------------------------------------------------------------------------------
-- Odd or even symmetry ??? Here we handle both ;-)
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
-- Walk through the pipes
-------------------------------------------------------------------------------
proc_pipe_walk : process (CLK_IN, nRESET)
  variable vl_dval_in : std_logic := '0';
begin  -- process proc_pipe_walk
  if nRESET = '0' then                  -- asynchronous reset (active low)
    sn_current_block <= 0;
  elsif rising_edge(CLK_IN) then        -- rising clock edge
    if DVAL_IN = '1' and vl_dval_in = '0' then
      sl_mult_active   <= '1';
      sn_current_block <= cn_block_count - 1;
    elsif sn_current_block = 0 then
      sl_mult_active <= '0';
    else
      sn_current_block <= sn_current_block - 1;
    end if;

    vl_dval_in := DVAL_IN;
  end if;
end process proc_pipe_walk;
-------------------------------------------------------------------------------
```

```
-- Walk through the pipes
------------------------------------------------------------------------------


------------------------------------------------------------------------------
-- Multiplexer before Multiplier and Multiplier itself
------------------------------------------------------------------------------
gen_mult_mux : for li_mult_in_block in 0 to cn_mult_per_block-1 generate

  sai_multiplier_in_a(li_mult_in_block) <=
    sai_pipe_sum(8*sn_current_block+li_mult_in_block)
    when sl_mult_active = '1' else 0;
  sai_multiplier_in_b(li_mult_in_block) <=
    cai_coefficients(8*sn_current_block+li_mult_in_block)
    when sl_mult_active = '1' else 0;

  -- If the synthesis is smart enough, it will see the pipeline register,
  -- available in the sysDSP blocks, otherwise, coment out the process
  proc_pipeline_reg : process (CLK_IN)
  begin  -- process gen_pipeline_reg
    if rising_edge(CLK_IN) then       -- rising clock edge
      sai_multiplier_out(li_mult_in_block) <=
        sai_multiplier_in_a(li_mult_in_block) *
        sai_multiplier_in_b(li_mult_in_block);
    end if;
  end process proc_pipeline_reg;

end generate gen_mult_mux;
------------------------------------------------------------------------------
-- Multiplexer before Multiplier and Multiplier itself
------------------------------------------------------------------------------


------------------------------------------------------------------------------
-- Adder after Multiplier
------------------------------------------------------------------------------
gen_mult_sum : for li_sum_after_mult in 0 to cn_mult_per_block/2-1 generate
  sai_mult_sum(li_sum_after_mult) <=
    sai_multiplier_out(2*li_sum_after_mult) +
    sai_multiplier_out(2*li_sum_after_mult + 1);
end generate gen_mult_sum;
------------------------------------------------------------------------------
-- Adder after Multiplier
------------------------------------------------------------------------------


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

```
-- MAC Unit with adder in front
------------------------------------------------------------------------
gen_macs : for li_macs in 0 to 1 generate
  gen_accumulate : process (CLK_IN, nRESET)
    variable vl_dval_in : std_logic := '0';
  begin  -- process gen_accumulate
    if nRESET = '0' then                -- asynchronous reset (active low)
      sai_mac(li_macs) <= 0;
    elsif rising_edge(CLK_IN) then      -- rising clock edge
      if DVAL_IN = '1' and vl_dval_in = '0' then
        -- First we save the accumulated values
        -- Then we do some other things
        sai_mac(li_macs) <= 0;
      else
        sai_mac(li_macs) <=
          sai_mac(li_macs) +
          sai_mult_sum(2*li_macs) + sai_mult_sum(2*li_macs+1);
      end if;
    end if;
  end process gen_accumulate;
end generate gen_macs;
------------------------------------------------------------------------
-- MAC Unit with adder in front
------------------------------------------------------------------------


------------------------------------------------------------------------
-- Do the output
------------------------------------------------------------------------
proc_out : process (CLK_IN, nRESET)
  variable vl_dval_in : std_logic := '0';
  variable vi_endsum  : ti_endsum;
begin  -- process proc_out
  if nRESET = '0' then                  -- asynchronous reset (active low)
    DATA_OUT <= (others => '0');
    DVAL_OUT <= '0';
  elsif rising_edge(CLK_IN) then        -- rising clock edge

    if DVAL_IN = '1' and vl_dval_in = '0' then
      vi_endsum := sai_mac(0) + sai_mac(1);
      si_endsum <= vi_endsum;
      DATA_OUT  <= conv_std_logic_vector(vi_endsum/2**cn_scale_width, DATA_WIDTH);
      DVAL_OUT  <= '1';
    else
      DVAL_OUT <= '0';
```

```
      end if;

    end if;
  end process proc_out;
  -------------------------------------------------------------------------------
  -- Do the output
  -------------------------------------------------------------------------------

end behavioral;
```

Listing B.49: stud_toplevel.vhd

```
-------------------------------------------------------------------------------
-- Title      : Students toplevel
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : stud_toplevel.vhd
-- Author     : Daniel Glaser
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-09-04
-- Last update: 2007-01-21
-- Platform   : LFECP20E
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description: This Toplevel is for student use. They should start here, as
--              here are already usable signals like parallel data from adc. It
--              simplyfies the work to be done.
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-09-04  1.0      sidaglas        Created
-- 2006-11-22  1.1      sidaglas        Modified
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;

entity stud_toplevel is

  generic (
    DATA_WIDTH : positive := 8);
```

```
port (
    ------------------------------------------------------------------------
    -- Generic signals
    ------------------------------------------------------------------------
    CLK_FAST : in std_logic;                -- 49,152 MHz
    CLK_SLOW : in std_logic;                -- 0,768 MHz = 768 kHz

    nRESET : in std_logic;

    ------------------------------------------------------------------------
    -- Human device interface (HDI)
    ------------------------------------------------------------------------

    -- Inputs
    SWITCH_1 : in std_logic;
    SWITCH_2 : in std_logic;
    SWITCH_3 : in std_logic;
    SWITCH_4 : in std_logic;
    SWITCH_5 : in std_logic;
    SWITCH_6 : in std_logic;
    SWITCH_7 : in std_logic;
    SWITCH_8 : in std_logic;

    BUTTON_1 : in std_logic;
    BUTTON_2 : in std_logic;
    BUTTON_3 : in std_logic;
    BUTTON_4 : in std_logic;
    BUTTON_5 : in std_logic;
    BUTTON_6 : in std_logic;
    BUTTON_7 : in std_logic;
    BUTTON_8 : in std_logic;

    -- Outputs
    STATUS_L_RED : out std_logic;
    STATUS_L_YEL : out std_logic;
    STATUS_L_GRE : out std_logic;
    STATUS_M_RED : out std_logic;
    STATUS_M_YEL : out std_logic;
    STATUS_M_GRE : out std_logic;
    STATUS_R_RED : out std_logic;
    STATUS_R_YEL : out std_logic;
    STATUS_R_GRE : out std_logic;
```

```
    -- The segment leds take hex numbers and show them
    SEGMENT_LED1 : out std_logic_vector(3 downto 0);
    SEGMENT_LED2 : out std_logic_vector(3 downto 0);
    SEGMENT_LED3 : out std_logic_vector(3 downto 0);
    SEGMENT_LED4 : out std_logic_vector(3 downto 0);

    -- The Bargraph switches on the count of lights you say
    BARGRAPH_LEFT   : out std_logic_vector(7 downto 0);
    BARGRAPH_RIGHT  : out std_logic_vector(7 downto 0);
    BARGRAPH_DEC_EN : out std_logic;

    ---------------------------------------------------------------------------
    -- PC communications
    ---------------------------------------------------------------------------
    PC_SDIN  : in  std_logic;
    PC_SDOUT : out std_logic;

    ---------------------------------------------------------------------------
    -- Peripherial connections
    ---------------------------------------------------------------------------
    -- Digitally controlled microphone preamplifier
    MA_GAIN : out std_logic_vector(5 downto 0);

    -- Analog to digital converter
    AD_PDIN_L : in std_logic_vector(23 downto 0);
    AD_PDIN_R : in std_logic_vector(23 downto 0);
    AD_DVAL   : in std_logic;

    -- Digital to analog converter
    DA_PDOUT_L : out std_logic_vector(23 downto 0);
    DA_PDOUT_R : out std_logic_vector(23 downto 0);
    DA_DVAL    : out std_logic);

end stud_toplevel;

architecture behavioral of stud_toplevel is

-- Here's the place for declarations

--STARTL
  component filteropt
    generic (
      FILTER_ORDER : positive;
      DATA_WIDTH   : positive);
```

```
    port (
      CLK_IN    : in  std_logic;
      nRESET    : in  std_logic;
      DVAL_IN   : in  std_logic;
      DVAL_OUT  : out std_logic;
      DATA_IN   : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      DATA_OUT  : out std_logic_vector(DATA_WIDTH-1 downto 0));
  end component;

  component dds_sinus
    generic (
      OUTPUT_WIDTH : positive;
      TABLE_WIDTH  : positive;
      PHASE_WIDTH  : positive;
      CLK_DIV      : positive);
    port (
      CLK_IN     : in  std_logic;
      nRESET     : in  std_logic;
      PHASE_INC  : in  std_logic_vector(PHASE_WIDTH-1 downto 0);
      OUTPUT     : out std_logic_vector(OUTPUT_WIDTH-1 downto 0));
  end component;

  signal sl_filter_dval_in, sl_filter_dval_out : std_logic;
  signal sv_filter_data_in, sv_filter_data_out : std_logic_vector(DATA_WIDTH-1 downto 0)

  signal sv_phase_inc : std_logic_vector(DATA_WIDTH-1 downto 0);
--STOPL

begin  -- behavioral

-- Here's the place for instantiations and code
--STARTL
  filteropt_1 : filteropt
    generic map (
      FILTER_ORDER => 37,
      DATA_WIDTH   => DATA_WIDTH)
    port map (
      CLK_IN    => CLK_FAST,
      nRESET    => nRESET,
      DVAL_IN   => sl_filter_dval_in,
      DVAL_OUT  => sl_filter_dval_out,
      DATA_IN   => sv_filter_data_in,
      DATA_OUT  => sv_filter_data_out);
```

```
  dds_sinus_1 : dds_sinus
    generic map (
      OUTPUT_WIDTH => 8,
      TABLE_WIDTH  => 4,
      PHASE_WIDTH  => 8,
      CLK_DIV      => 250)
    port map (
      CLK_IN     => CLK_FAST,
      nRESET     => nRESET,
      PHASE_INC => sv_phase_inc,
      OUTPUT     => sv_filter_data_in);

  -- We can use this as a 96 kHz-Source ;-)
  sl_filter_dval_in <= AD_DVAL;

  DA_PDOUT_L <= sv_filter_data_out;
  DA_DVAL    <= sl_filter_dval_out;

  proc_phase_inc : process (CLK_FAST, nRESET)
    variable vi_phase_inc : natural range 0 to 120;
    variable vl_dval : std_logic;
  begin  -- process proc_phase_inc
    if nRESET = '0' then                -- asynchronous reset (active low)
      sv_phase_inc <= (others => '0');
    elsif rising_edge(CLK_FAST) then      -- rising clock edge
      if AD_DVAL = '1' and vl_dval = '0' then

        if vi_phase_inc = 100 then
          vi_phase_inc := 0;
        else
          vi_phase_inc := vi_phase_inc + 1;
        end if;

      end if;

      vl_dval := AD_DVAL;

    end if;
  end process proc_phase_inc;
--STOPL
end behavioral;
```

B.3.8 Demodulator

Listing B.50: .vhd

```
-------------------------------------------------------------------------------
-- Title      : Hüllkurvendemodulator
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : huellkurve.vhd
-- Author     : Daniel Glaser
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-11-01
-- Last update: 2007-01-22
-- Platform   : LFECP20E
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description: Einfache Quadrierung des Signals
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-11-01  1.0      sidaglas        Created
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;


--NOCOPY

entity huellkurve is

  generic (
    DATA_WIDTH : positive := 8);

  port (
    CLK_IN      : in  std_logic;
    nRESET      : in  std_logic;
    DVAL_IN     : in  std_logic;
    DVAL_OUT    : out std_logic;
    DATA_IN_A   : in  std_logic_vector(DATA_WIDTH-1 downto 0);
    DATA_IN_B   : in  std_logic_vector(DATA_WIDTH-1 downto 0);
    DATA_OUT_A  : out std_logic_vector(DATA_WIDTH-1 downto 0);
    DATA_OUT_B  : out std_logic_vector(DATA_WIDTH-1 downto 0));
```

```
end huellkurve;

architecture behavioral of huellkurve is

begin  -- behavioral

  process (CLK_IN, nRESET)
    variable vl_dval : std_logic := '0';
    variable vn_count : natural range 0 to 2 := 0;
  begin  -- process
    if nRESET = '0' then                -- asynchronous reset (active low)
      DATA_OUT_A <= (others => '0');
      DATA_OUT_B <= (others => '0');
      DVAL_OUT <= '0';
    elsif rising_edge(CLK_IN) then      -- rising clock edge
      if DVAL_IN = '1' and vl_dval = '0' then
        vn_count := 2;
      elsif vn_count /= 0 then
        vn_count := vn_count - 1;
      end if;

      if vn_count = 2 then
        DATA_OUT_B <= conv_std_logic_vector(
          (conv_integer(DATA_IN_B) * conv_integer(DATA_IN_B))/
          2**(DATA_WIDTH-2), DATA_WIDTH);
      elsif vn_count = 1 then
        DATA_OUT_A <= conv_std_logic_vector(
          (conv_integer(DATA_IN_A) * conv_integer(DATA_IN_A))/
          2**(DATA_WIDTH-2), DATA_WIDTH);
        DVAL_OUT <= '1';
      else
        DVAL_OUT <= '0';
      end if;
    end if;
  end process;

end behavioral;
```

B.3.9 Tiefpass

Listing B.51: filteropt.vhd

```
-------------------------------------------------------------------------------
-- Title      : Filter Optimization
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
```

```
-------------------------------------------------------------------------------
-- File        : filteropt.vhd
-- Author      : Daniel Glaser
-- Company     : LTE, FAU Erlangen-Nuremberg, Germany
-- Created     : 2006-11-01
-- Last update: 2007-01-23
-- Platform    : LFECP20E
-- Standard    : VHDL'87
-------------------------------------------------------------------------------
-- Description: This module implements a DSP-Block-Optimized version of the
--              previosly discussed filter.
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-11-01  1.0      sidaglas        Created
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

use ieee.math_real.all;

--NOCOPY

entity filteropt is

  generic (
    FILTER_ORDER : positive := 32;
    DATA_WIDTH   : positive := 8);

  port (
    CLK_IN    : in  std_logic;
    nRESET    : in  std_logic;
    DVAL_IN   : in  std_logic;
    DVAL_OUT  : out std_logic;
    DATA_IN   : in  std_logic_vector(DATA_WIDTH-1 downto 0);
    DATA_OUT  : out std_logic_vector(DATA_WIDTH-1 downto 0));

end filteropt;

architecture behavioral of filteropt is
```

```
  constant cn_mult_per_block : natural := 8;
  constant cn_pipe_sum_count : natural := (FILTER_ORDER+1)/2;
  constant cn_block_count    : natural := (FILTER_ORDER-1)/(2*cn_mult_per_block)+1;
  constant cn_scale_width    : natural := DATA_WIDTH+1;  -- Empirisch emitteln!

  subtype ti_pipe is integer range -2**(DATA_WIDTH-1) to 2**(DATA_WIDTH-1)-1;
-- 8 Bit
  subtype ti_pipe_sum is integer range -2**(DATA_WIDTH) to 2**(DATA_WIDTH)-1;
-- 9 Bit
  subtype ti_multed is integer range -2**(2*(DATA_WIDTH+1)-1) to 2**(2*(DATA_WIDTH+1)-1)
-- 18 Bit
  subtype ti_mult_sumed is integer range -2**(2*(DATA_WIDTH+1)) to 2**(2*(DATA_WIDTH+1))
-- 19 Bit
  subtype ti_maced is integer range -2**(2*(DATA_WIDTH+2)-1) to 2**(2*(DATA_WIDTH+2)-1)-
-- 20 Bit
  subtype ti_endsum is integer range -2**(2*(DATA_WIDTH+2)) to 2**(2*(DATA_WIDTH+2))-1;
-- 21 Bit

  type tai_pipe is array (0 to FILTER_ORDER-1) of ti_pipe;
  type tai_pipe_sum is array (0 to cn_mult_per_block*cn_block_count-1) of ti_pipe_sum;
  type tai_mult_in is array (0 to cn_mult_per_block-1) of ti_pipe_sum;
  type tai_mult_out is array (0 to cn_mult_per_block-1) of ti_multed;
  type tai_mult_sumed is array (0 to cn_mult_per_block/2-1) of ti_mult_sumed;
  type tai_maced is array (0 to 1) of ti_maced;

  constant ci_pipe_sum_max : integer := ti_pipe_sum'high;

  constant cai_coefficients : tai_pipe_sum :=
    (integer(real(+0.00077)*real(ci_pipe_sum_max)),     -- c0, c31
     (integer(real(-0.00132)*real(ci_pipe_sum_max))),   -- c1, c30
     (integer(real(+0.00236)*real(ci_pipe_sum_max))),   -- c2, c29
     (integer(real(+0.00417)*real(ci_pipe_sum_max))),   -- c3, c28
     (integer(real(+0.00698)*real(ci_pipe_sum_max))),   -- c4, c27
     (integer(real(+0.01095)*real(ci_pipe_sum_max))),   -- c5, c26
     (integer(real(+0.01613)*real(ci_pipe_sum_max))),   -- c6, c25
     (integer(real(+0.02244)*real(ci_pipe_sum_max))),   -- c7, c24
     (integer(real(+0.02968)*real(ci_pipe_sum_max))),   -- c8, c23
     (integer(real(+0.03754)*real(ci_pipe_sum_max))),   -- c9, c22
     (integer(real(+0.04559)*real(ci_pipe_sum_max))),   -- c10, c21
     (integer(real(+0.05335)*real(ci_pipe_sum_max))),   -- c11, c20
     (integer(real(+0.06033)*real(ci_pipe_sum_max))),   -- c12, c19
     (integer(real(+0.06606)*real(ci_pipe_sum_max))),   -- c13, c18
     (integer(real(+0.07012)*real(ci_pipe_sum_max))),   -- c14, c17
```

```
    (integer(real(+0.07222)*real(ci_pipe_sum_max))));   -- c15, c16
--    (integer(real(+0.00000)*real(ci_pipe_sum_max))),
--    (integer(real(+0.00000)*real(ci_pipe_sum_max))),
--    (integer(real(+0.00000)*real(ci_pipe_sum_max))),

--    (integer(real(+0.00000)*real(ci_pipe_sum_max))),
--    (integer(real(+0.00000)*real(ci_pipe_sum_max))),
--    (integer(real(+0.00000)*real(ci_pipe_sum_max))),
--    (integer(real(+0.00000)*real(ci_pipe_sum_max))),
--    (integer(real(+0.00000)*real(ci_pipe_sum_max))));

  signal sai_pipe     : tai_pipe;
  signal sai_pipe_sum : tai_pipe_sum := (others => 0);

  signal sai_multiplier_in_a, sai_multiplier_in_b : tai_mult_in;
  signal sai_multiplier_out                       : tai_mult_out;
  signal sai_mult_sum                             : tai_mult_sumed;

  signal sai_mac    : tai_maced;
  signal si_endsum : ti_endsum;

  signal si_filter_outreg, si_filter_savereg : integer range -2**(2*DATA_WIDTH-1) to 2**

  signal sl_mult_active : std_logic;

  signal sn_current_block : natural range 0 to cn_block_count-1;

begin  -- behavioral


  ------------------------------------------------------------------------------
  -- Let the pipe flow/smoke
  ------------------------------------------------------------------------------
  shift_pipe_regs : process (CLK_IN, nRESET)
    variable vl_dval_in : std_logic := '0';
  begin  -- process shift_pipe_regs
    if nRESET = '0' then                -- asynchronous reset (active low)
      sai_pipe <= (others => 0);
    elsif rising_edge(CLK_IN) then      -- rising clock edge
      if DVAL_IN = '1' and vl_dval_in = '0' then
        -- Shift the values in the pipeline by one
        sai_pipe <= conv_integer(DATA_IN) & sai_pipe(0 to FILTER_ORDER-2);
      end if;
      vl_dval_in := DVAL_IN;
```

```
  end if;
end process shift_pipe_regs;
------------------------------------------------------------------------------
-- Let the pipe flow/smoke
------------------------------------------------------------------------------


------------------------------------------------------------------------------
-- Odd or even symmetry ??? Here we handle both ;-)
------------------------------------------------------------------------------
gen_pipe_sums : for li_pipe_sum in 0 to cn_pipe_sum_count-1 generate
  process (CLK_IN, nRESET)
    variable vi_sum_helper : ti_pipe_sum;
  begin  -- process
    if nRESET = '0' then                -- asynchronous reset (active low)
      sai_pipe_sum(li_pipe_sum) <= 0;
    elsif rising_edge(CLK_IN) then      -- rising clock edge

      vi_sum_helper := sai_pipe(li_pipe_sum);
      if li_pipe_sum /= (FILTER_ORDER - li_pipe_sum - 1) then
        vi_sum_helper := vi_sum_helper + sai_pipe(FILTER_ORDER - li_pipe_sum - 1);
      end if;
      sai_pipe_sum(li_pipe_sum) <= vi_sum_helper;

    end if;
  end process;
end generate gen_pipe_sums;
------------------------------------------------------------------------------
-- Odd or even symmetry ??? Here we handle both ;-)
------------------------------------------------------------------------------


------------------------------------------------------------------------------
-- Walk through the pipes
------------------------------------------------------------------------------
proc_pipe_walk : process (CLK_IN, nRESET)
  variable vl_dval_in : std_logic := '0';
begin  -- process proc_pipe_walk
  if nRESET = '0' then                  -- asynchronous reset (active low)
    sn_current_block <= 0;
  elsif rising_edge(CLK_IN) then        -- rising clock edge
    if DVAL_IN = '1' and vl_dval_in = '0' then
      sl_mult_active   <= '1';
      sn_current_block <= cn_block_count - 1;
    elsif sn_current_block = 0 then
      sl_mult_active <= '0';
```

```
      else
        sn_current_block <= sn_current_block - 1;
      end if;

      vl_dval_in := DVAL_IN;
    end if;
  end process proc_pipe_walk;
-------------------------------------------------------------------------------
-- Walk through the pipes
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
-- Multiplexer before Multiplier and Multiplier itself
-------------------------------------------------------------------------------
  gen_mult_mux : for li_mult_in_block in 0 to cn_mult_per_block-1 generate

    sai_multiplier_in_a(li_mult_in_block) <=
      sai_pipe_sum(8*sn_current_block+li_mult_in_block)
      when sl_mult_active = '1' else 0;
    sai_multiplier_in_b(li_mult_in_block) <=
      cai_coefficients(8*sn_current_block+li_mult_in_block)
      when sl_mult_active = '1' else 0;

    -- If the synthesis is smart enough, it will see the pipeline register,
    -- available in the sysDSP blocks, otherwise, coment out the process
    proc_pipeline_reg : process (CLK_IN)
    begin  -- process gen_pipeline_reg
      if rising_edge(CLK_IN) then       -- rising clock edge
        sai_multiplier_out(li_mult_in_block) <=
          sai_multiplier_in_a(li_mult_in_block) *
          sai_multiplier_in_b(li_mult_in_block);
      end if;
    end process proc_pipeline_reg;

  end generate gen_mult_mux;
-------------------------------------------------------------------------------
-- Multiplexer before Multiplier and Multiplier itself
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
-- Adder after Multiplier
-------------------------------------------------------------------------------
  gen_mult_sum : for li_sum_after_mult in 0 to cn_mult_per_block/2-1 generate
    sai_mult_sum(li_sum_after_mult) <=
```

```
    sai_multiplier_out(2*li_sum_after_mult) +
    sai_multiplier_out(2*li_sum_after_mult + 1);
end generate gen_mult_sum;
-------------------------------------------------------------------------------
-- Adder after Multiplier
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
-- MAC Unit with adder in front
-------------------------------------------------------------------------------
gen_macs : for li_macs in 0 to 1 generate
  gen_accumulate : process (CLK_IN, nRESET)
    variable vl_dval_in : std_logic := '0';
  begin  -- process gen_accumulate
    if nRESET = '0' then                -- asynchronous reset (active low)
      sai_mac(li_macs) <= 0;
    elsif rising_edge(CLK_IN) then      -- rising clock edge
      if DVAL_IN = '1' and vl_dval_in = '0' then
        -- First we save the accumulated values
        -- Then we do some other things
        sai_mac(li_macs) <= 0;
      else
        sai_mac(li_macs) <=
          sai_mac(li_macs) +
          sai_mult_sum(2*li_macs) + sai_mult_sum(2*li_macs+1);
      end if;
    end if;
  end process gen_accumulate;
end generate gen_macs;
-------------------------------------------------------------------------------
-- MAC Unit with adder in front
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
-- Do the output
-------------------------------------------------------------------------------
proc_out : process (CLK_IN, nRESET)
  variable vl_dval_in : std_logic := '0';
  variable vi_endsum  : ti_endsum;
begin  -- process proc_out
  if nRESET = '0' then                  -- asynchronous reset (active low)
    DATA_OUT <= (others => '0');
    DVAL_OUT <= '0';
  elsif rising_edge(CLK_IN) then        -- rising clock edge
```

```
      if DVAL_IN = '1' and vl_dval_in = '0' then
        vi_endsum := sai_mac(0) + sai_mac(1);
        si_endsum <= vi_endsum;
        DATA_OUT  <= conv_std_logic_vector(vi_endsum/2**cn_scale_width, DATA_WIDTH);
        DVAL_OUT  <= '1';
      else
        DVAL_OUT <= '0';
      end if;

    end if;
  end process proc_out;
  ------------------------------------------------------------------------------
  -- Do the output
  ------------------------------------------------------------------------------

end behavioral;
```

Listing B.52: stud_toplevel.vhd

```
------------------------------------------------------------------------------
-- Title      : Students toplevel
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
------------------------------------------------------------------------------
-- File       : stud_toplevel.vhd
-- Author     : Daniel Glaser
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-09-04
-- Last update: 2006-11-23
-- Platform   : LFECP20E
-- Standard   : VHDL'87
------------------------------------------------------------------------------
-- Description: This Toplevel is for student use. They should start here, as
--              here are already usable signals like parallel data from adc. It
--              simplyfies the work to be done.
------------------------------------------------------------------------------
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-09-04  1.0      sidaglas        Created
-- 2006-11-22  1.1      sidaglas        Modified
------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
```

```
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;

entity stud_toplevel is

  generic (
    DATA_WIDTH : positive := 8);

  port (
    ---------------------------------------------------------------------------
    -- Generic signals
    ---------------------------------------------------------------------------
    CLK_FAST : in std_logic;            -- 49,152 MHz
    CLK_SLOW : in std_logic;            -- 0,768 MHz = 768 kHz

    nRESET : in std_logic;

    ---------------------------------------------------------------------------
    -- Human device interface (HDI)
    ---------------------------------------------------------------------------

    -- Inputs
    SWITCH_1 : in std_logic;
    SWITCH_2 : in std_logic;
    SWITCH_3 : in std_logic;
    SWITCH_4 : in std_logic;
    SWITCH_5 : in std_logic;
    SWITCH_6 : in std_logic;
    SWITCH_7 : in std_logic;
    SWITCH_8 : in std_logic;

    BUTTON_1 : in std_logic;
    BUTTON_2 : in std_logic;
    BUTTON_3 : in std_logic;
    BUTTON_4 : in std_logic;
    BUTTON_5 : in std_logic;
    BUTTON_6 : in std_logic;
    BUTTON_7 : in std_logic;
    BUTTON_8 : in std_logic;

    -- Outputs
    STATUS_L_RED : out std_logic;
    STATUS_L_YEL : out std_logic;
```

```
    STATUS_L_GRE : out std_logic;
    STATUS_M_RED : out std_logic;
    STATUS_M_YEL : out std_logic;
    STATUS_M_GRE : out std_logic;
    STATUS_R_RED : out std_logic;
    STATUS_R_YEL : out std_logic;
    STATUS_R_GRE : out std_logic;

    -- The segment leds take hex numbers and show them
    SEGMENT_LED1 : out std_logic_vector(3 downto 0);
    SEGMENT_LED2 : out std_logic_vector(3 downto 0);
    SEGMENT_LED3 : out std_logic_vector(3 downto 0);
    SEGMENT_LED4 : out std_logic_vector(3 downto 0);

    -- The Bargraph switches on the count of lights you say
    BARGRAPH_LEFT    : out std_logic_vector(7 downto 0);
    BARGRAPH_RIGHT   : out std_logic_vector(7 downto 0);
    BARGRAPH_DEC_EN : out std_logic;

    ------------------------------------------------------------------------
    -- PC communications
    ------------------------------------------------------------------------
    PC_SDIN  : in  std_logic;
    PC_SDOUT : out std_logic;

    ------------------------------------------------------------------------
    -- Peripherial connections
    ------------------------------------------------------------------------
    -- Digitally controlled microphone preamplifier
    MA_GAIN : out std_logic_vector(5 downto 0);

    -- Analog to digital converter
    AD_PDIN_L : in std_logic_vector(23 downto 0);
    AD_PDIN_R : in std_logic_vector(23 downto 0);

    -- Digital to analog converter
    DA_PDOUT_L : out std_logic_vector(23 downto 0);
    DA_PDOUT_R : out std_logic_vector(23 downto 0));

end stud_toplevel;

architecture behavioral of stud_toplevel is

-- Here's the place for declarations
```

```
begin  -- behavioral

-- Here's the place for instantiations and code

end behavioral;
```

B.3.10 Signalrückgewinnung

Listing B.53: sigregen.vhd

```
-------------------------------------------------------------------------------
-- Title      : Signal Regeneration
-- Project    :
-------------------------------------------------------------------------------
-- File       : sigregen.vhd
-- Author     :
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-11-01
-- Last update: 2007-01-22
-- Platform   : LFECP20E
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description: Ths module regenerates the serial input signal, which is FSK
--              modulated with two different sinusodial signals. Two bandpasses
--              filter out the correct frequencies, a demodulator gives a dc
--              offset and a low pass filter makes is smooth again. This two dc
--              signals are compared and the bigger one gives the valid output.
--              A hysteresis is also applied to kill some noise.
-------------------------------------------------------------------------------
-- Copyright (c) 2007 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-11-01  1.0      sidaglas        Created
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;

--NOCOPY

entity sigregen is
```

```
  generic (
    DATA_WIDTH : positive := 8;
    HYSTERESIS : natural  := 2);

  port (
    CLK_IN     : in  std_logic;
    nRESET     : in  std_logic;
    DVAL_IN    : in  std_logic;
    DVAL_OUT   : out std_logic;
    DATA_IN_A  : in  std_logic_vector(DATA_WIDTH-1 downto 0);
    DATA_IN_B  : in  std_logic_vector(DATA_WIDTH-1 downto 0);
    DATA_OUT   : out std_logic);

end sigregen;

architecture behavioral of sigregen is

  constant FILTER_ORDER : positive := 32;

  signal si_comp_out : integer range -2**(DATA_WIDTH) to 2**(DATA_WIDTH)-1;
  signal sl_add_val  : std_logic;

  component filteropt
    generic (
      FILTER_ORDER : positive;
      DATA_WIDTH   : positive);
    port (
      CLK_IN    : in  std_logic;
      nRESET    : in  std_logic;
      DVAL_IN   : in  std_logic;
      DVAL_OUT  : out std_logic;
      DATA_IN   : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      DATA_OUT  : out std_logic_vector(DATA_WIDTH-1 downto 0));
  end component;

  component huellkurve
    generic (
      DATA_WIDTH : positive);
    port (
      CLK_IN     : in  std_logic;
      nRESET     : in  std_logic;
      DVAL_IN    : in  std_logic;
      DVAL_OUT   : out std_logic;
```

```
      DATA_IN_A  : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      DATA_IN_B  : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      DATA_OUT_A : out std_logic_vector(DATA_WIDTH-1 downto 0);
      DATA_OUT_B : out std_logic_vector(DATA_WIDTH-1 downto 0));
  end component;

  signal sl_hk_dval_in, sl_hk_dval_out      : std_logic;
  signal sv_hk_data_in_a, sv_hk_data_in_b   : std_logic_vector(DATA_WIDTH-1 downto 0);
  signal sv_hk_data_out_a, sv_hk_data_out_b : std_logic_vector(DATA_WIDTH-1 downto 0);

  signal sl_filt_val : std_logic;

  signal sl_lp1_dval_in, sl_lp1_dval_out : std_logic;
  signal sv_lp1_data_in, sv_lp1_data_out : std_logic_vector(DATA_WIDTH-1 downto 0);

  signal sl_lp2_dval_in, sl_lp2_dval_out : std_logic;
  signal sv_lp2_data_in, sv_lp2_data_out : std_logic_vector(DATA_WIDTH-1 downto 0);

begin -- behavioral

  huellkurve_1 : huellkurve
    generic map (
      DATA_WIDTH => DATA_WIDTH)
    port map (
      CLK_IN     => CLK_IN,
      nRESET     => nRESET,
      DVAL_IN    => sl_hk_dval_in,
      DVAL_OUT   => sl_hk_dval_out,
      DATA_IN_A  => sv_hk_data_in_a,
      DATA_IN_B  => sv_hk_data_in_b,
      DATA_OUT_A => sv_hk_data_out_a,
      DATA_OUT_B => sv_hk_data_out_b);

  sv_hk_data_in_a <= DATA_IN_A;
  sv_hk_data_in_b <= DATA_IN_B;
  sl_hk_dval_in   <= DVAL_IN;

  sv_lp1_data_in <= '0' & sv_hk_data_out_a(DATA_WIDTH-1 downto 1);
  sv_lp2_data_in <= '0' & sv_hk_data_out_b(DATA_WIDTH-1 downto 1);
  sl_lp1_dval_in <= sl_hk_dval_out;
  sl_lp2_dval_in <= sl_hk_dval_out;

  filteropt_1 : filteropt
    generic map (
```

```
    FILTER_ORDER => FILTER_ORDER,
    DATA_WIDTH   => DATA_WIDTH)
  port map (
    CLK_IN   => CLK_IN,
    nRESET   => nRESET,
    DVAL_IN  => sl_lp1_dval_in,
    DVAL_OUT => sl_lp1_dval_out,
    DATA_IN  => sv_lp1_data_in,
    DATA_OUT => sv_lp1_data_out);

filteropt_2 : filteropt
  generic map (
    FILTER_ORDER => FILTER_ORDER,
    DATA_WIDTH   => DATA_WIDTH)
  port map (
    CLK_IN   => CLK_IN,
    nRESET   => nRESET,
    DVAL_IN  => sl_lp2_dval_in,
    DVAL_OUT => sl_lp2_dval_out,
    DATA_IN  => sv_lp2_data_in,
    DATA_OUT => sv_lp2_data_out);

sync_filters : process (CLK_IN, nRESET)
  variable vl_filter_val1, vl_filter_val2 : std_logic;
  variable vl_lp1_val, vl_lp2_val         : std_logic;
begin  -- process sync_filters
  if nRESET = '0' then                  -- asynchronous reset (active low)
    vl_filter_val1 := '0';
    vl_filter_val2 := '0';
    sl_filt_val <= '0';
  elsif rising_edge(CLK_IN) then        -- rising clock edge

    if vl_filter_val1 = '1' and vl_filter_val2 = '1' then
      sl_filt_val <= '1';
      vl_filter_val1 := '0';
      vl_filter_val2 := '0';
    else
      sl_filt_val <= '0';
      if sl_lp1_dval_out = '1' and vl_lp1_val = '0' then
        vl_filter_val1 := '1';
      end if;
      if sl_lp2_dval_out = '1' and vl_lp2_val = '0' then
        vl_filter_val2 := '1';
      end if;
```

```
    end if;

    vl_lp1_val := sl_lp1_dval_out;
    vl_lp2_val := sl_lp2_dval_out;
  end if;
end process sync_filters;

proc_comp : process (CLK_IN, nRESET)
  variable vl_dval : std_logic := '0';
begin  -- process proc_comp
  if nRESET = '0' then                  -- asynchronous reset (active low)
    si_comp_out <= 0;
    sl_add_val  <= '0';
  elsif rising_edge(CLK_IN) then        -- rising clock edge
    if sl_filt_val = '1' and vl_dval = '0' then
      si_comp_out <= conv_integer('0' & sv_lp1_data_out) - conv_integer('0' & sv_lp2_d
      sl_add_val  <= '1';
    else
      sl_add_val <= '0';
    end if;

    vl_dval := sl_filt_val;
  end if;
end process proc_comp;

proc_hyst : process (CLK_IN, nRESET)
  variable vl_val : std_logic := '0';
begin  -- process proc_hyst
  if nRESET = '0' then                  -- asynchronous reset (active low)
    DATA_OUT <= '0';
  elsif rising_edge(CLK_IN) then        -- rising clock edge
    if sl_add_val = '1' and vl_val = '0' then
      if si_comp_out >= HYSTERESIS then
        DATA_OUT <= '1';
      elsif si_comp_out <= -(HYSTERESIS) then
        DATA_OUT <= '0';
      end if;
      DVAL_OUT <= '1';
    else
      DVAL_OUT <= '0';
    end if;

    vl_val := sl_add_val;
  end if;
```

```
  end process proc_hyst;
end behavioral;
```

Listing B.54: stud_toplevel.vhd

```
-------------------------------------------------------------------------------
-- Title      : Students toplevel
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : stud_toplevel.vhd
-- Author     : Daniel Glaser
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-09-04
-- Last update: 2007-01-22
-- Platform   : LFECP20E
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description: This Toplevel is for student use. They should start here, as
--              here are already usable signals like parallel data from adc. It
--              simplyfies the work to be done.
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-09-04  1.0      sidaglas        Created
-- 2006-11-22  1.1      sidaglas        Modified
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;

entity stud_toplevel is

  generic (
    DATA_WIDTH : positive := 8);

  port (
    ---------------------------------------------------------------------------
    -- Generic signals
    ---------------------------------------------------------------------------
    CLK_FAST : in std_logic;                -- 49,152 MHz
    CLK_SLOW : in std_logic;                -- 0,768 MHz = 768 kHz
```

```
nRESET : in std_logic;

-------------------------------------------------------------------------
-- Human device interface (HDI)
-------------------------------------------------------------------------

-- Inputs
SWITCH_1 : in std_logic;
SWITCH_2 : in std_logic;
SWITCH_3 : in std_logic;
SWITCH_4 : in std_logic;
SWITCH_5 : in std_logic;
SWITCH_6 : in std_logic;
SWITCH_7 : in std_logic;
SWITCH_8 : in std_logic;

BUTTON_1 : in std_logic;
BUTTON_2 : in std_logic;
BUTTON_3 : in std_logic;
BUTTON_4 : in std_logic;
BUTTON_5 : in std_logic;
BUTTON_6 : in std_logic;
BUTTON_7 : in std_logic;
BUTTON_8 : in std_logic;

-- Outputs
STATUS_L_RED : out std_logic;
STATUS_L_YEL : out std_logic;
STATUS_L_GRE : out std_logic;
STATUS_M_RED : out std_logic;
STATUS_M_YEL : out std_logic;
STATUS_M_GRE : out std_logic;
STATUS_R_RED : out std_logic;
STATUS_R_YEL : out std_logic;
STATUS_R_GRE : out std_logic;

-- The segment leds take hex numbers and show them
SEGMENT_LED1 : out std_logic_vector(3 downto 0);
SEGMENT_LED2 : out std_logic_vector(3 downto 0);
SEGMENT_LED3 : out std_logic_vector(3 downto 0);
SEGMENT_LED4 : out std_logic_vector(3 downto 0);

-- The Bargraph switches on the count of lights you say
```

```
    BARGRAPH_LEFT    : out std_logic_vector(7 downto 0);
    BARGRAPH_RIGHT   : out std_logic_vector(7 downto 0);
    BARGRAPH_DEC_EN  : out std_logic;

    ---------------------------------------------------------------------------
    -- PC communications
    ---------------------------------------------------------------------------
    PC_SDIN  : in  std_logic;
    PC_SDOUT : out std_logic;

    ---------------------------------------------------------------------------
    -- Peripherial connections
    ---------------------------------------------------------------------------
    -- Digitally controlled microphone preamplifier
    MA_GAIN : out std_logic_vector(5 downto 0);

    -- Analog to digital converter
    AD_PDIN_L : in std_logic_vector(23 downto 0);
    AD_PDIN_R : in std_logic_vector(23 downto 0);
    AD_DVAL   : in std_logic;

    -- Digital to analog converter
    DA_PDOUT_L : out std_logic_vector(23 downto 0);
    DA_PDOUT_R : out std_logic_vector(23 downto 0);
    DA_DVAL    : out std_logic);

end stud_toplevel;

architecture behavioral of stud_toplevel is

-- Here's the place for declarations

--STARTL
  component sigregen
    generic (
      DATA_WIDTH : positive;
      HYSTERESIS : natural);
    port (
      CLK_IN     : in  std_logic;
      nRESET     : in  std_logic;
      DVAL_IN    : in  std_logic;
      DVAL_OUT   : out std_logic;
      DATA_IN_A  : in  std_logic_vector(DATA_WIDTH-1 downto 0);
      DATA_IN_B  : in  std_logic_vector(DATA_WIDTH-1 downto 0);
```

```
      DATA_OUT   : out std_logic);
  end component;

  signal sv_data_in_a, sv_data_in_b : std_logic_vector(DATA_WIDTH-1 downto 0);

  component dds_sinus
    generic (
      OUTPUT_WIDTH : positive;
      TABLE_WIDTH  : positive;
      PHASE_WIDTH  : positive;
      CLK_DIV      : positive);
    port (
      CLK_IN     : in  std_logic;
      nRESET     : in  std_logic;
      PHASE_INC  : in  std_logic_vector(PHASE_WIDTH-1 downto 0);
      OUTPUT     : out std_logic_vector(OUTPUT_WIDTH-1 downto 0));
  end component;

  signal sv_sinus_5kHz, sv_sinus_10kHz : std_logic_vector(DATA_WIDTH-1 downto 0);

--STOPL
begin  -- behavioral

-- Here's the place for instantiations and code

--STARTL
  sigregen_1: sigregen
    generic map (
      DATA_WIDTH => DATA_WIDTH,
      HYSTERESIS => 2)
    port map (
      CLK_IN     => CLK_FAST,
      nRESET     => nRESET,
      DVAL_IN    => AD_DVAL,
      DVAL_OUT   => open,
      DATA_IN_A  => sv_data_in_a,
      DATA_IN_B  => sv_data_in_b,
      DATA_OUT   => STATUS_R_GRE);

  dds_sinus_1: dds_sinus
    generic map (
      OUTPUT_WIDTH => DATA_WIDTH,
      TABLE_WIDTH  => 4,
      PHASE_WIDTH  => DATA_WIDTH,
```

```
      CLK_DIV       => 250)
    port map (
      CLK_IN     => CLK_FAST,
      nRESET     => nRESET,
      PHASE_INC => "00011001",
      OUTPUT     => sv_sinus_5kHz);

  sv_data_in_a <= sv_sinus_5kHz when BUTTON_1 = '1' else (others => '0');

  dds_sinus_2: dds_sinus
    generic map (
      OUTPUT_WIDTH => DATA_WIDTH,
      TABLE_WIDTH  => 4,
      PHASE_WIDTH  => DATA_WIDTH,
      CLK_DIV       => 250)
    port map (
      CLK_IN     => CLK_FAST,
      nRESET     => nRESET,
      PHASE_INC => "00110010",
      OUTPUT     => sv_sinus_10kHz);

  sv_data_in_b <= sv_sinus_10kHz when BUTTON_2 = '1' else (others => '0');

--STOPL
end behavioral;
```

B.3.11 Loop-Test

Listing B.55: stud_toplevel.vhd

```
-------------------------------------------------------------------------------
-- Title      : Students toplevel
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : stud_toplevel.vhd
-- Author     : Daniel Glaser
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-09-04
-- Last update: 2006-11-23
-- Platform   : LFECP20E
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description: This Toplevel is for student use. They should start here, as
--              here are already usable signals like parallel data from adc. It
--              simplyfies the work to be done.
```

```
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-09-04  1.0      sidaglas        Created
-- 2006-11-22  1.1      sidaglas        Modified
-------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;

entity stud_toplevel is

  generic (
    DATA_WIDTH : positive := 8);

  port (
    -----------------------------------------------------------------------
    -- Generic signals
    -----------------------------------------------------------------------
    CLK_FAST : in std_logic;            -- 49,152 MHz
    CLK_SLOW : in std_logic;            -- 0,768 MHz = 768 kHz

    nRESET : in std_logic;

    -----------------------------------------------------------------------
    -- Human device interface (HDI)
    -----------------------------------------------------------------------

    -- Inputs
    SWITCH_1 : in std_logic;
    SWITCH_2 : in std_logic;
    SWITCH_3 : in std_logic;
    SWITCH_4 : in std_logic;
    SWITCH_5 : in std_logic;
    SWITCH_6 : in std_logic;
    SWITCH_7 : in std_logic;
    SWITCH_8 : in std_logic;

    BUTTON_1 : in std_logic;
    BUTTON_2 : in std_logic;
```

```
BUTTON_3 : in std_logic;
BUTTON_4 : in std_logic;
BUTTON_5 : in std_logic;
BUTTON_6 : in std_logic;
BUTTON_7 : in std_logic;
BUTTON_8 : in std_logic;

-- Outputs
STATUS_L_RED : out std_logic;
STATUS_L_YEL : out std_logic;
STATUS_L_GRE : out std_logic;
STATUS_M_RED : out std_logic;
STATUS_M_YEL : out std_logic;
STATUS_M_GRE : out std_logic;
STATUS_R_RED : out std_logic;
STATUS_R_YEL : out std_logic;
STATUS_R_GRE : out std_logic;

-- The segment leds take hex numbers and show them
SEGMENT_LED1 : out std_logic_vector(3 downto 0);
SEGMENT_LED2 : out std_logic_vector(3 downto 0);
SEGMENT_LED3 : out std_logic_vector(3 downto 0);
SEGMENT_LED4 : out std_logic_vector(3 downto 0);

-- The Bargraph switches on the count of lights you say
BARGRAPH_LEFT    : out std_logic_vector(7 downto 0);
BARGRAPH_RIGHT   : out std_logic_vector(7 downto 0);
BARGRAPH_DEC_EN : out std_logic;

-----------------------------------------------------------------------
-- PC communications
-----------------------------------------------------------------------
PC_SDIN  : in  std_logic;
PC_SDOUT : out std_logic;

-----------------------------------------------------------------------
-- Peripherial connections
-----------------------------------------------------------------------
-- Digitally controlled microphone preamplifier
MA_GAIN : out std_logic_vector(5 downto 0);

-- Analog to digital converter
AD_PDIN_L : in std_logic_vector(23 downto 0);
AD_PDIN_R : in std_logic_vector(23 downto 0);
```

```
    -- Digital to analog converter
    DA_PDOUT_L : out std_logic_vector(23 downto 0);
    DA_PDOUT_R : out std_logic_vector(23 downto 0));

end stud_toplevel;

architecture behavioral of stud_toplevel is

-- Here's the place for declarations

begin -- behavioral

-- Here's the place for instantiations and code

end behavioral;
```

B.3.12 Chat-Session

Listing B.56: stud_toplevel.vhd

```
-------------------------------------------------------------------------------
-- Title      : Students toplevel
-- Project    : Praktikum zu Architekturen der Digitalen Signalverarbeitung
-------------------------------------------------------------------------------
-- File       : stud_toplevel.vhd
-- Author     : Daniel Glaser
-- Company    : LTE, FAU Erlangen-Nuremberg, Germany
-- Created    : 2006-09-04
-- Last update: 2006-11-23
-- Platform   : LFECP20E
-- Standard   : VHDL'87
-------------------------------------------------------------------------------
-- Description: This Toplevel is for student use. They should start here, as
--              here are already usable signals like parallel data from adc. It
--              simplyfies the work to be done.
-------------------------------------------------------------------------------
-- Copyright (c) 2006 LTE, FAU Erlangen-Nuremberg, Germany
-------------------------------------------------------------------------------
-- Revisions  :
-- Date        Version  Author          Description
-- 2006-09-04  1.0      sidaglas        Created
-- 2006-11-22  1.1      sidaglas        Modified
-------------------------------------------------------------------------------

library ieee;
```

```
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;

entity stud_toplevel is

  generic (
    DATA_WIDTH : positive := 8);

  port (
    ------------------------------------------------------------------------
    -- Generic signals
    ------------------------------------------------------------------------
    CLK_FAST : in std_logic;            -- 49,152 MHz
    CLK_SLOW : in std_logic;            -- 0,768 MHz = 768 kHz

    nRESET : in std_logic;

    ------------------------------------------------------------------------
    -- Human device interface (HDI)
    ------------------------------------------------------------------------

    -- Inputs
    SWITCH_1 : in std_logic;
    SWITCH_2 : in std_logic;
    SWITCH_3 : in std_logic;
    SWITCH_4 : in std_logic;
    SWITCH_5 : in std_logic;
    SWITCH_6 : in std_logic;
    SWITCH_7 : in std_logic;
    SWITCH_8 : in std_logic;

    BUTTON_1 : in std_logic;
    BUTTON_2 : in std_logic;
    BUTTON_3 : in std_logic;
    BUTTON_4 : in std_logic;
    BUTTON_5 : in std_logic;
    BUTTON_6 : in std_logic;
    BUTTON_7 : in std_logic;
    BUTTON_8 : in std_logic;

    -- Outputs
    STATUS_L_RED : out std_logic;
```

```
    STATUS_L_YEL : out std_logic;
    STATUS_L_GRE : out std_logic;
    STATUS_M_RED : out std_logic;
    STATUS_M_YEL : out std_logic;
    STATUS_M_GRE : out std_logic;
    STATUS_R_RED : out std_logic;
    STATUS_R_YEL : out std_logic;
    STATUS_R_GRE : out std_logic;

    -- The segment leds take hex numbers and show them
    SEGMENT_LED1 : out std_logic_vector(3 downto 0);
    SEGMENT_LED2 : out std_logic_vector(3 downto 0);
    SEGMENT_LED3 : out std_logic_vector(3 downto 0);
    SEGMENT_LED4 : out std_logic_vector(3 downto 0);

    -- The Bargraph switches on the count of lights you say
    BARGRAPH_LEFT    : out std_logic_vector(7 downto 0);
    BARGRAPH_RIGHT   : out std_logic_vector(7 downto 0);
    BARGRAPH_DEC_EN : out std_logic;

    ------------------------------------------------------------------------
    -- PC communications
    ------------------------------------------------------------------------
    PC_SDIN  : in  std_logic;
    PC_SDOUT : out std_logic;

    ------------------------------------------------------------------------
    -- Peripherial connections
    ------------------------------------------------------------------------
    -- Digitally controlled microphone preamplifier
    MA_GAIN : out std_logic_vector(5 downto 0);

    -- Analog to digital converter
    AD_PDIN_L : in std_logic_vector(23 downto 0);
    AD_PDIN_R : in std_logic_vector(23 downto 0);

    -- Digital to analog converter
    DA_PDOUT_L : out std_logic_vector(23 downto 0);
    DA_PDOUT_R : out std_logic_vector(23 downto 0));

end stud_toplevel;

architecture behavioral of stud_toplevel is
```

```
-- Here's the place for declarations

begin  -- behavioral

-- Here's the place for instantiations and code

end behavioral;
```

C Praktikumsanleitung

Das Praktikumsskript wurde vom Bearbeiter dieser Studienarbeit und Andreas Schedel, der den MATLAB-Teil übernahm, zu etwa gleichen Teilen erstellt. Um den Zusammenhang des Stoffes zu wahren, wurden die Teile, die nicht im Rahmen dieser Studienarbeit entstanden, im Text belassen. Nicht immer kann genau unterschieden werden, welche Teile wem zuzurechnen sind. Dies trifft insbesondere auf die Einleitung zu.

Dieses Skript findet sich auf der beiliegenden CD.

Literaturverzeichnis

[1] *Omnidirectional Electret Condenser Microphone Unit EM-060A*, Conrad Electronic SE. [Online]. Available: http://www.produktinfo.conrad.com/datenblaetter/300000-324999/302147-da-02-en-Mikrofonkap

[2] M. Huemer, *Skriptum zur Vorlesung "Architekturen der Digitalen Signalverarbeitung"*. Lehrstuhl für Technische Elektronik, Universität Erlangen-Nürnberg, 2006.

[3] *LatticeECP2/M Family Handbook*, 02nd ed., Lattice semiconductor corporation, September 2006. [Online]. Available: http://www.latticesemi.com/dynamic/view_document.cfm?document_id=21733

[4] *LatticeECP/EC Family Data Sheet*, 02nd ed., Lattice semiconductor corporation, March 2006. [Online]. Available: http://www.latticesemi.com/dynamic/view_document.cfm?document_id=8517

[5] *LatticeECP/EC Family Handbook*, 02nd ed., Lattice semiconductor corporation, April 2006. [Online]. Available: http://www.latticesemi.com/dynamic/view_document.cfm?document_id=8518

[6] *TL082 Datasheet - Wide Bandwidth Dual JFET Input Operational Amplifier*, National Semiconductor, August 2000. [Online]. Available: http://www.national.com/ds.cgi/TL/TL082.pdf

[7] *LM320L/LM79LXXAC Series 3-Terminal Negative Regulators*, National Semiconductor, March 2006. [Online]. Available: http://www.national.com/ds.cgi/LM/LM320L.pdf

[8] *SP3222EB/3232EB Datasheet - CorporationTrue +3.0V to +5.5V RS-232 Transceivers*, Sipex Corporation, November 2005. [Online]. Available: http://www.sipex.com/Files/DataSheets/sp3222eb_3232eb.pdf

[9] *STPS340U Datasheet - Power Schottky Rectifier*, STMicroelectronics, July 2003. [Online]. Available: http://www.st.com/stonline/products/literature/ds/3624.pdf

[10] *TPS6755 Datasheet - ADJUSTABLE INVERTING DC/DC CONVERTER*, Texas Instruments Incorporated, December 1996. [Online]. Available: http://www.ti.com/lit/gpn/tps6755

[11] *OPA2134 Datasheet - SoundPLUS High Performance AUDIO OPERATIONAL AMPLIFIERS*, Texas Instruments Incorporated, December 1997. [Online]. Available: http://www.ti.com/lit/gpn/opa2134

[12] *TPS0213 Datasheet - 2-W MONO AUDIO POWER AMPLIFIER WITH HEADPHONE DRIVE*, Texas Instruments Incorporated, January 2000. [Online]. Available: http://www.ti.com/lit/gpn/tpa0213

[13] *PCM1730 Datasheet - 24-BIT, 192-kHz SAMPLING ADVANCED SEGMENT, AUDIO STEREO DIGITAL-TO-ANALOG CONVERTER*, Texas Instruments Incorporated, November 2001. [Online]. Available: http://www.ti.com/lit/gpn/pcm1730

[14] *PGA2500 Datasheet - Digitally Controlled Microphone Preamplifier*, Texas Instruments Incorporated, December 2003. [Online]. Available: http://www.ti.com/lit/gpn/pga2500

[15] *PCM1803 Datasheet - SINGLE-ENDED, ANALOG-INPUT 24-BIT, 96-kHz STEREO A/D CONVERTER*, Texas Instruments Incorporated, November 2004. [Online]. Available: http://www.ti.com/lit/gpn/pcm1803

[16] *PTH12000 Datasheet*, Texas Instruments Incorporated, June 2004. [Online]. Available: http://www.ti.com/lit/gpn/pth12000w

[17] *TPS5430 Datasheet - 3-A, WIDE INPUT RANGE, STEP-DOWN SWIFT(TM) CONVERTER*, Texas Instruments Incorporated, March 2006. [Online]. Available: http://www.ti.com/lit/gpn/tps5430

[18] *TPS732xx Datasheet - Cap-Free, NMOS, 250mA Low Dropout Regulator with Reverse Current Protection*, Texas Instruments Incorporated, May 2006. [Online]. Available: http://www.ti.com/lit/gpn/tps73250

[19] U. Tietze and C. Schenk, *Halbleiter-Schaltungstechnik*, 11st ed. Springer-Verlag Berlin Heidelberg, 1999.

[20] *XC6201 Series positive voltage regulators*, Torex Semiconductor Ltd. [Online]. Available: http://www.torex-usa.com/product/pro02/pdf/XC6201_E.pdf