Studiengang Diplom - Informatik

Diplomarbeit

Redesign eines VLSI Chips zur gehörgerechten Signalvorverarbeitung akustischer Signale

vorgelegt von: Frank Poppen

vorgelegt am: 03.02.1998

Betreuender Gutachter: Matthias Brucke

Zweiter Gutachter: Prof. Dr. Ing. Wolfgang Nebel

Abbildungsverzeichnis

Abbildungsverzeichnis

Abbildungsverzeichnis

1 Konventionen

- arg(x): Argument der imaginären Zahl x.
- AG EIS: ArbeitsGruppe Entwurf Integrierter Schaltungen der Universität Oldenburg.
- AG IMA: ArbeitsGruppe Informatikmethoden für MikroelektronikAnwendungen der Universität Hamburg.
- AG MEDI: ArbeitsGruppe MEDIzinische Physik der Universität Oldenburg.
- BLB: BinauraleLateralitätsBewertung.
- BIST: Build In Self Test. Komponenten mit BIST testen ihre Hardware selbstätig auf physikalische schäden.
- CORDIC: COordinate Rotating DIgital Computer.
- DPR: Dual Ported RAM. Speicher mit zwei getrennten Schnittstellen zum Lesen und/oder Schreiben.
- DSP: Digitaler SignalProzessor.
- DW01: Designware Library 1 von Synopsys. In ihr sind unter anderem diverse Addierer enthalten.
- DW02: Designware Library 2 von Synopsys. In ihr sind unter anderem diverse Multiplizierer enthalten.
- DW03: Designware Library 3 von Synopsys. In ihr sind unter anderem diverse RAMbausteine enthalten.
- FIR: Finite Impulse Response Filter.
- FSM: Finite State Machine, endlicher Automat.
- GATE: Design auf Gatterebene.
- GFB: GammatoneFilterBank.
- ’high’: Entspricht dem elektrischen Äquivalent zu logisch 1 bzw. TRUE.
- HSS: HighSpeedSchnittStelle.
- HWGR: HalbWellenGleichRichter.
- IIF: InputInterFace.
- IIR: Infinite Impulse Response Filter.
- Kanal: Bezeichnet einen der 30 Filterkanäle der Gammatone Filterbank.
- ’low’: Entspricht dem elektrischen Äquivalent zu logisch 0 bzw. FALSE.
- LSS: LowSpeedSchnittstelle.
- LSB: Least Significant Bit.
- LTI-System: Lineares, zeitinvariantes System.
- MPEG: Moving Pictures Experts Group
- MSB: Most Significant Bit.

- NRGL: NachReGeLschleifen. Ein Teil des Perzeptionsmodells, welcher nicht auf dem SiliconEar implementiert wird. Sie sind Bestandteil der Arbeit der AG IMA.
- OIF: OutputInterFace.
- Perzeptionsmodell: Beschreibt das menschliche Ohr aus der Sicht der AG MEDI. Siehe auch Kapitel 3.2.
- PG SiliconEar: Die Projektgruppe „Entwurf eines VLSI-Chips für Hörgeräte“.
- RT: Design auf Registertransferebene.
- SiliconEar: Name des ASIC, der Gegenstand dieser Diplomarbeit ist und in dem Teile des Perzeptionsmodells implementiert sind.
- Stereokanal: Steht für den linken bzw. rechten Kanal des ASIC SiliconEar. Dieser arbeitet binaural.
- Synopsys: Designtool zur Entwicklung von Hardwarekomponenten. Siehe auch [35].
- Sample: Ein Sample besteht aus den beiden 16 Bit breiten Teilsamplen des linken und rechten Stereokanals und wird in das IIF eingelesen.
- Samplingfrequenz: Beträgt 16,276 KHz
- Stereosample: Teilsample eines Samples, also entweder zum linken oder rechten Stereokanal der binauralen Verarbeitung gehörig.
- SINALCO: SiliconCochlea. Eine Projektgruppe, die sich die Implementation des Perzeptionsmodells in Hardware zum Ziel gesetzt hat. Siehe auch Kapitel 2.
- VHDL: Very High speed integrated circuits Description Language.

2 Einleitung

Die Physik versucht, Eigenschaften von realen Systemen mit Hilfe von Modellen zu beschreiben. Wird ein Modell den gewachsenen Ansprüchen der Physik nicht mehr gerecht, so wird nach einem neuen, erweiterten Modell gesucht. Das Verständnis von z. B. atomaren Vorgängen wurde durch die unterschiedlichen Atommodelle im Laufe der Zeit weiter verbessert. Es fällt bei der Erstellung eines Modells naturgemäß leichter, ein physikalisches oder chemisches Phänomen zu beschreiben, als ein biologisch- / psychologisches, wie Wahrnehmungen beim Menschen. Um Eigenschaften des Menschen genügend genau beschreiben zu können, müssen seine subjektiven Empfindungen mit in das Modell einfließen. Die Forschungsdisziplin, die sich mit den subjektiven Empfindungen des Menschen im Zusammenhang mit akustischen Phänomenen beschäftigt, heißt Psychoakustik.

Die AG MEDI an der Universität Oldenburg verfügt über ein Modell zur Beschreibung des menschlichen Hörens, das Perzeptionsmodell [3]. Bei der Entwicklung eines Verständnisses für das Gehör waren Tests mit normalhörenden Versuchspersonen nötig, damit Empfindungen wie „Laut“, „Leise“, „Ton Höher als“, etc. im Modell berücksichtigt werden konnten. Das Perzeptionsmodell wird im nachfolgenden Kapitel 3 weiter erläutert.

Mit Hilfe dieses Modells wird versucht, Verfahren zu entwickeln, die Personen mit Hörschäden über ihre Behinderung im täglichen Leben hinweghelfen sollen. Es sollen Möglichkeiten zur gehörgerechten Signalvorverarbeitung bei der Spracherkennung implementiert werden. Es können an das menschliche Ohr angepaßte Kompressionsalgorithmen im Audiobereich entwickelt werden. Beim MPEG-Standard ist dieses bereits geschehen. Geräte wie das Telefon können besser an das Gehör des Menschen angepaßt werden, dadurch kann sich die Klangqualität bei gleichbleibender oder sogar verminderter Datenrate verbessern. Das Potential eines guten Modells vom Gehör wird an diesen Beispielen sichtbar.

Es sind Simulationen erforderlich, wenn die „Güte“ des Modells getestet werden soll, oder wenn es in eine Anwendung integriert wird. Die AG MEDI testet das Perzeptionsmodell als C-Implementierung auf handelsüblichen Rechnern. Somit ist der Hardwareaufwand enorm, wenn man einen späteren Einsatz als Hörhilfe im Sinn hat. Es ist für einen Hörgeräteträger nicht dienlich, einen PC im Rucksack mit sich führen zu müssen. Dieses Vorgehen hat auch schon deshalb keinen Sinn, da die C-Simulation nicht in Echtzeit stattfindet.

Eine Portierung der Software auf Hardware ist notwendig. Hierzu wurde die Projektgruppe „Entwurf eines VLSI-Chips für Hörgeräte“ oder kurz „PG SiliconEar“ auf Initiative der AG MEDI und Prof. Dr. Ing. Wolfgang Nebel ins Leben gerufen, die Teile des Modells implementieren sollte. Die studentischen Teilnehmer waren:

Volkert Barr, Matthias Brucke, Norbert Friese, Uwe Gerken, Burak Gökalan, Frank Poppen, Arne Schulz, Christoph Wetjen und Siegfried Wetjen

Die Betreuer: Dipl.-Phys. Martin Hansen, Dipl.-Ing. Gerd Jochens, Dipl.-Ing. Dirk Rabe, Dipl.-Inform. Bernd Timmermann

Parallel zu dieser Projektgruppe wurde zusätzlich die Studienarbeit „Entwurf einer integrierten Schaltung zur binauralen Verarbeitung frequenzgefilteter akustischer Signale“ an Eike Schmidt vergeben. Hier wurde ein weiterer Teil des Modells, die binaurale Lateralitätsbewertung, in VHDL implementiert.

Bei der Erstellung einer Anforderungsdefinition innerhalb der PG traten Differenzen zwischen den Wünschen aus der AG MEDI und dem technisch realisierbaren auf. Das Ergebnis der PG, in welches das der vergebenen Studienarbeit einfloß, konnte somit kein fertiges Layout von einem VLSI-Chip sein. Hierzu findet der Leser mehr im 4. Kapitel.

Die Folge war eine Weiterführung der Arbeit innerhalb der Projektgruppe SINALCO. SINALCO besteht aus den drei Arbeitsgruppen:

- AG MEDI der Universität Oldenburg
    - •Birger Kollmeier
    - •Martin Hansen
    - •Volker Hohmann
    - •Jürgen Tschorz
- AG EIS der Universität Oldenburg
    - •Wolfgang Nebel
    - •Matthias Brucke
    - •Frank Poppen
    - •Eike Schmidt
- AG IMA der Universität Hamburg
    - •Bärbel Mertsching
    - •Alexander Schwarz
    - •Ole Blaurock

Die AG IMA befaßt sich mit der Realisierung eines weiteren Teils des Perzeptionsmodells in Hardware. Die AG EIS führt die Arbeit der PG SiliconEar weiter. Dazu wurde diese Diplomarbeit „Redesign eines VLSI Chips zur gehörgerechten Signalvorverarbeitung akustischer Signale“ vergeben.

3 Einordnung der Diplomarbeit in die Gesamtthematik

Zum Verständnis des Perzeptionsmodells ist ein Einblick in die Funktionsweise des Gehörs erforderlich. Im Kapitel 3.1 soll die Biologie des Ohrs kurz beschrieben werden. Anschließend werden die so gewonnenen Erkenntnisse in das Perzeptionsmodell transformiert, Kapitel 3.2.

3.1 Das Ohr

Die Funktion des Ohrs ist, Schallwellen zu registrieren und sie in für das Gehirn verwertbare Nervenimpulse zu wandeln. Schall ist eine Druckschwankung, die sich in Luft, Wasser oder anderen Materialien wellenförmig ausbreitet. Es kommt zu einer Energieübertragung von der Schallquelle zum Schallempfänger. In Luft beträgt die Schallgeschwindigkeit ca. 333 m/s.

Vom Menschen wird die Frequenz der Schwingung als Tonhöhe wahrgenommen. Die empfundene Lautstärke hängt monoton mit der Intensität, also der Amplitude der Welle, zusammen. Das Ohr ist nicht in der Lage, Frequenzen außerhalb des Intervalls von 16 Hz bis 20 KHz zu registrieren. Der Frequenzbereich der menschlichen Sprache liegt innerhalb dieses Intervalls, erstreckt sich aber nicht über dessen gesamte Breite. Welche Frequenzen für das Sprachverständnis notwendig sind wird aus Abbildung 1 deutlich. Neben der Tonhöhe und der Lautstärke kann als dritte Eigenschaft die Richtung bestimmt werden. So kann der Mensch den Ursprung einer Schallwelle ausfindig machen. Hierfür reicht ein Sinnesorgan alleine nicht aus. Erst zwei Ohren gewährleisten das Hören in drei Dimensionen. Man nennt diese Eigenschaft „binaurales Gehör“.

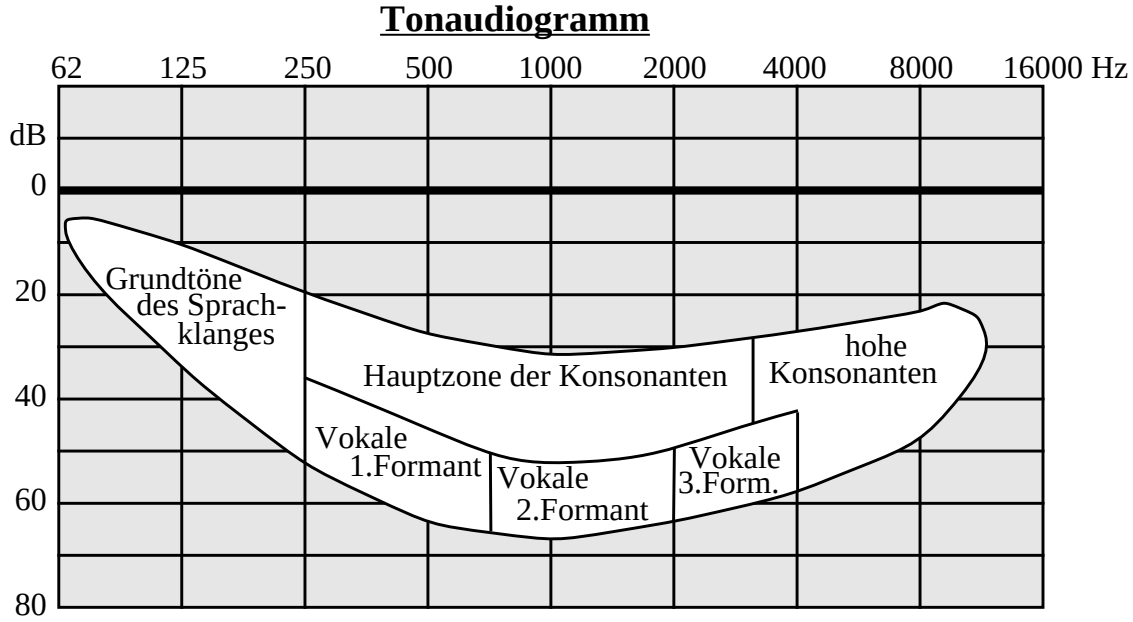


Abbildung 1: Fantsches Sprachfeld (gestellt von der AG MEDI)

Das Ohr des Menschen unterteilt sich in drei Bereiche, das Außen-, Mittel- und Innenohr. Abbildung 2 zeigt diesen Aufbau von links nach rechts gelesen.

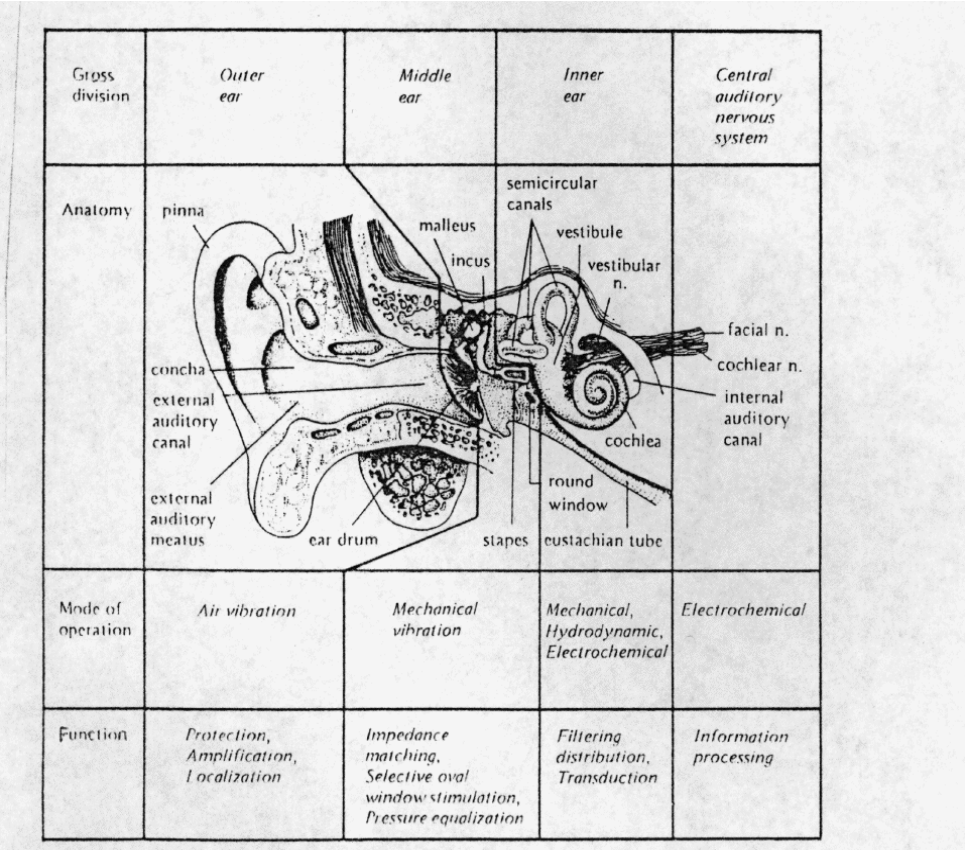


Abbildung 2: Das Ohr des Menschen [4]

Als äußeres Ohr wird die Ohrmuschel und der Gehörgang bis zum Trommelfell bezeichnet. Dieser Teil dient als Schalltrichter, der eintreffenden Schall auffängt. Hier wird die Welle spektral so verändert, daß ein ausgerichtetes Hören im dreidimensionalen Raum möglich wird.

Das Mittelohr besteht aus der Paukenhöhle mit den drei Gehörknöchelchen Hammer, Amboß und Steigbügel und der Eustachischen Röhre. Das Trommelfell bildet die Grenze zwischen Außen- und Mittelohr. Es wird durch eintreffenden Schall in Bewegung versetzt. Die Eustachische Röhre, welche vom Mittelohr zum Nasenrachenraum führt, dient dem Druckausgleich zwischen Außen- und Mittelohr. Nur so kann das Trommelfell ungehindert schwingen. Bei einem schlechten Ausgleich kommt es zu einem „Druckgefühl“ im Ohr, ein bekanntes Phänomen für Flugpasagiere und Taucher.

Mit dem Trommelfell ist der Hammer verbunden. Er nimmt die Luftschwingung auf und überträgt sie als mechanische Schwingung über Amboß und Steigbügel an das ovale Fenster zum Innenohr. Die Gehörknöchelchen garantieren eine effiziente Übertragung der Wellenenergie an die Lymphflüssigkeit im Innenohr und verhindern Reflexionen.

Das Innenohr (Abbildung 3) liegt gut geschützt in den harten Knochen des Felsenbeins. Hier befinden sich die eigentlichen Sinnesorgane. Die Cochlea oder auch Hörschnecke hat beim Menschen zweieinhalb Windungen und ist ca. 32 mm lang. Sie besteht aus einem spiralig gerollten Schlauch, der durch zwei Membrane unterteilt wird. Die Basilarmembran trennt die mit Endolymphe gefüllte Scala Media von der mit Perilymphe gefüllten Scala Tympani. Die Begrenzung zur Scala Vestibuli ist durch die Reissnersche Membran gegeben.

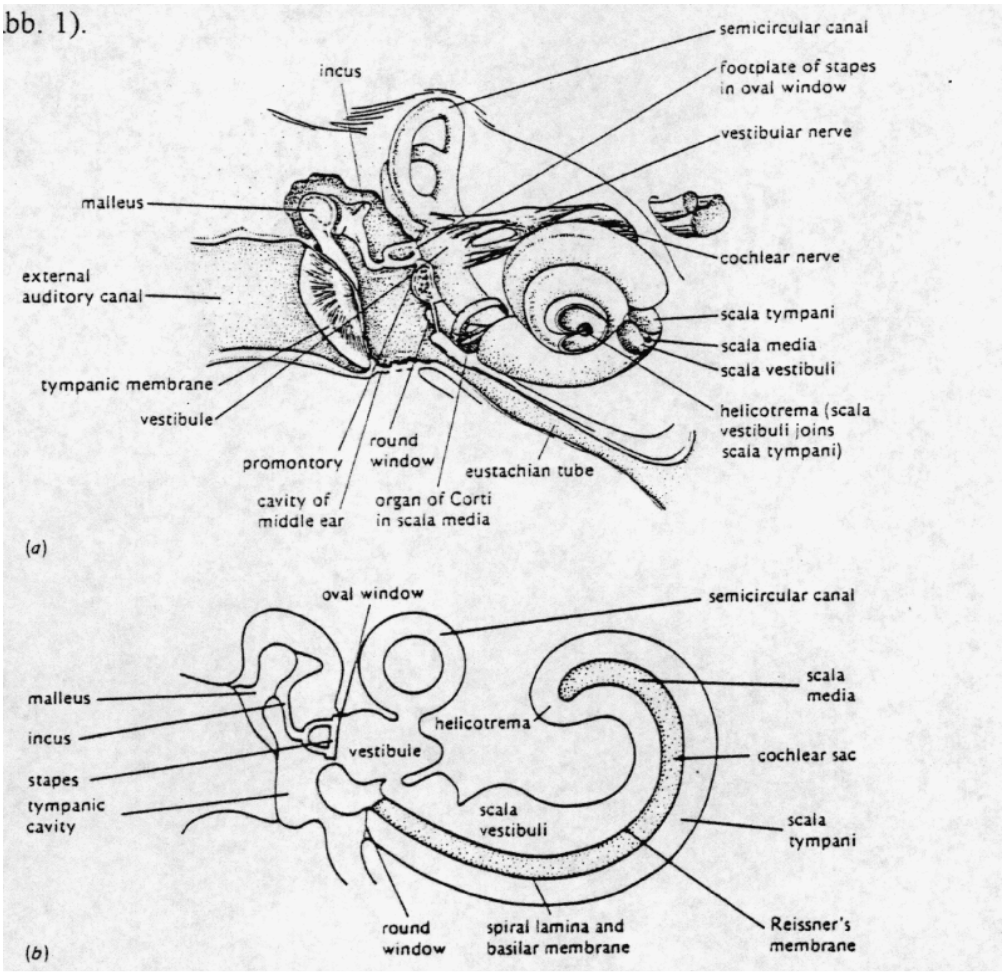


Abbildung 3: Das Innenohr des Menschen [4]

Die Basilarmembran trägt das Cortische Organ, das Hörorgan. Hier sind die Sinneszellen in drei Reihen äußerer Haarzellen und einer Reihe innerer Harzellen, umgeben von Stütz- und Nährzellen, angeordnet. Über den Stereozilien der Haarzellen liegt die Deckmembran und hat mit den Stereozilien der äußeren Haarzellen direkten Kontakt. Die Hörnervfasern gehen von den inneren Haarzellen über die Ganglion Spirale zum Hörnerv. Dieser führt weiter zum Cochlear Nucleus. Von hier aus werden die verschiedenen Anteile der zentralen Hörbahn gekreuzt

über mehrere unterschiedliche Bahnen zur Hirnrinde geführt. Kreuzen heißt, daß Nervenimpulse vom linken Ohr vorwiegend zur rechten und solche vom rechten Ohr vorwiegend zur linken Hirnhälfte übermittelt werden.

Die über das ovale Fenster vom Steigbügel in das Innenohr gelangende Schallschwingung bewegt sich zwischen demselben und dem runden Fenster. Hierbei läuft eine Wanderwelle entlang der Basilarmembran und reizt die Stereozilien der Haarzellen. Die Amplitude dieser Welle wird frequenzabhängig auf verschiedene Bereiche der Membran abgebildet. Wellen mit hoher Frequenz haben ihr Amplitudenmaxima an der Basis, solche mit niedriger Frequenz an der Spitze der Schnecke. Diese Abbildung der Frequenz auf den Ort maximaler Auslenkung auf der Basilarmembran ist annähernd logarithmisch.

Bei der Weiterleitung der Reize werden diese von der zentralen Hörbahn auf dem Weg zum Gehirn vorverarbeitet.

3.2 Das Perzeptionsmodell

Das Perzeptionsmodell versucht die Vorgänge beim Hören so genau wie möglich nachzubilden. Die Genauigkeit ist nicht allein durch physikalische und biologische Vorgänge definiert. Als wesentliche Informationsquelle dienten psychoakustische Versuche mit normalhörenden Personen. In Abbildung 4 ist das Perzeptionsmodell schematisch aufgezeichnet.

Nicht aufgenommen in das Schema wurden das Außen- und Mittelohr. Sie können leicht mit Hilfe eines Analog/Digital Wandlers und einem passenden, vorgeschalteten Bandpaßfilter nachgebildet werden. Gleichzeitig ist ein Tiefpaßfilter notwendig, um Aliasingeffekte zu verhindern.

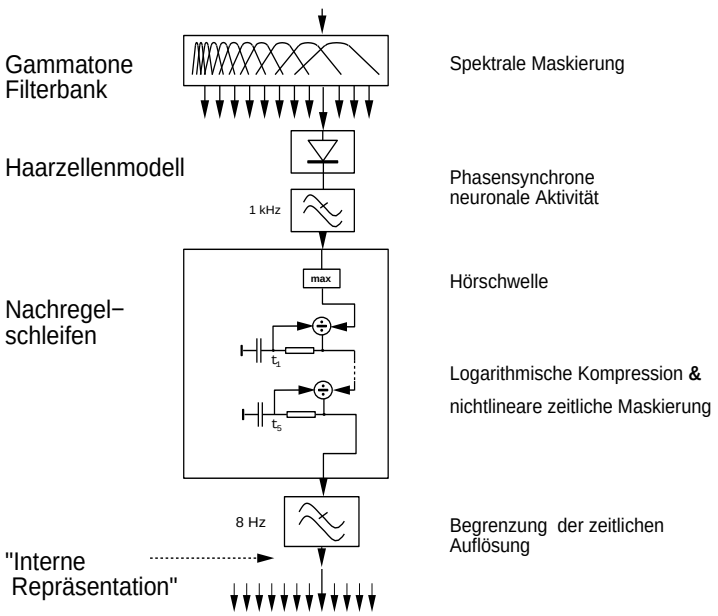


Abbildung 4: Das Perzeptionsmodell [3]

Die Eigenschaften der Basilarmembran wurden mittels einer GammatoneFilterbank (GFB) implementiert. Es handelt sich hierbei um ein Array von 30 Bandpaßfiltern mit Mittenfrequenzen von 73 Hz bis 6,681 KHz. Dieser Bereich deckt, wie aus Abbildung 1 ersichtlich, das Spektrum der menschlichen Sprache ab. In der Abbildung 4 wird angedeutet, daß tiefe Töne, also solche Schallwellen mit niedriger Frequenz, durch Filter mit kleinerer Bandbreite verarbeitet werden. Die Auflösung tiefer Töne ist somit höher. Daher liegen die Filter bei tieferen Mittenfrequenzen dichter verteilt als bei höheren Mittenfrequenzen. Diese Anordnung spiegelt die etwa logarithmische Frequenzabbildung auf die Basilarmembran wieder.

Die 30 bandpaßgefilterten Signale werden gleichgerichtet und tiefpaßgefiltert. Durch den Gleichrichter wird das Verhalten der Haarzellen nachgebildet, nur bei einer Auslenkung in eine Richtung Nervenimpulse abzugeben. Die Tiefpaßfilterung bei 1 KHz spiegelt eine abnehmende Sensitivität der Haarzellen wieder, die Phase des Eingangssignals kodieren zu können.

Die folgenden Nachregelschleifen (NRGL) beschreiben nichtlineare Effekte der Amplitudenkompression und der Kontrastierung des Signals. Diese Effekte treten sowohl in der Basilarmembran, als auch auf den Hörnerven auf. Die GFB bildet lediglich die linearen Eigenschaften der Basilarmembran auf das Perzeptionsmodell ab. Daher sind die NRGL zusätzlich erforderlich. Sie werden auf dem ASIC, welcher in dieser Diplomarbeit behandelt wird, nicht implementiert und sollen daher an dieser Stelle keine weitere Beachtung finden. Die AG IMA der Universität Hamburg bearbeitet diesen Teil des Perzeptionsmodell.

4 Ergebnisse der Projektgruppe

Nach der Implementation des Perzeptionsmodells als Computerprogramm, konnte die AG MEDI seine Tauglichkeit anhand von Simulationsläufen unter Beweis stellen. Die Simulationen wurden aufgrund der Komplexität der Algorithmen jedoch nicht in Echtzeit durchgeführt. Es lag daher nahe, die Softwareimplementation nach Hardware zu portieren. Ein Teil dieser Arbeit wurde durch die PG SiliconEar geleistet. Die Ergebnisse der PG sollen im folgenden Kapitel zusammengefaßt werden. Genauere Informationen sind in [42] zu finden.

Eine 1:1 Überführung des C-Codes in Hardware konnte nicht durchgeführt werden. Er wurde semantikerhaltend transformiert. Gleichzeitig wurde das Zahlenformat intern von 64 Bit Fließkomma auf 24 Bit Festkomma im Zweierkomplement ohne signifikanten Genauigkeitsverlust reduziert. Die Festkommadarstellung deckt das Intervall von $1\text{-}2^{-23}$ bis -1 mit einer Genauigkeit von +/- 2^{-23} ab. Extern, an den Schnittstellen, arbeitet der VLSI-Chip mit 16 Bit-Festkomma im Zweierkomplement. Der Wechsel des Zahlenformats war aus Optimierungsgründen notwendig: „So wenig wie möglich, so viel wie nötig!“

Analysen ergaben, daß eine einkanalige ASIC-Version die maximal zur Verfügung stehende Chipfläche nicht ausfüllt. So wurde eine monaurale Version zugunsten einer binauralen verworfen. Diese Designentscheidung hat zur Folge, daß die 30 Kanäle der GFB doppelt zu implementieren sind.

Es erfolgte eine Analyse der Aufgabe. Der VLSI-Chip wurde in die Blöcke IIF, GFB, BLB, CU und OIF aufgeteilt. Abbildung 5 zeigt das Blockschaltbild des von der PG entworfenen ASIC. Er beschreibt nicht das gesamte Perzeptionsmodell. Nur die GFB wurde realisiert. Der Block BLB ist nachträglich hinzugefügt worden und ist das Resultat einer Studienarbeit „Entwurf einer integrierten Schaltung zur binauralen Verarbeitung frequenzgefilteter akustischer Signale“ [43].

Als Zieltechnologie wurde die ecpd07 Library in 0,7 µm vorgesehen. Der Systemtakt wurde auf 50 MHz festgelegt. Dieses hatte zur Folge, daß die ursprüngliche Samplingfrequenz von 16 KHz auf 16,276 KHz erhöht werden mußte. Der Wert ergibt sich aus dem Systemtakt mit 3072 als ganzzahligem Teiler. Die Filterkonstanten der GFB wurden an die neue Frequenz angepaßt.

Als Signalquellen für den ASIC wurden ein Analog/Digitalwandler und der DSP 32C von AT&T vorgesehen.

Als Senken waren zu Beginn des Projektes mehrere DSP 32C gedacht. Diese Hardware stand der AG MEDI bereits zur Verfügung und war hinreichend bekannt. Während der Dauer des Projektes stellte sich aber heraus, daß die erforderliche Datenrate die Senken überlasten wird. Dieses veranlaßte die AG MEDI, eine andere Systemumgebung zu definieren.

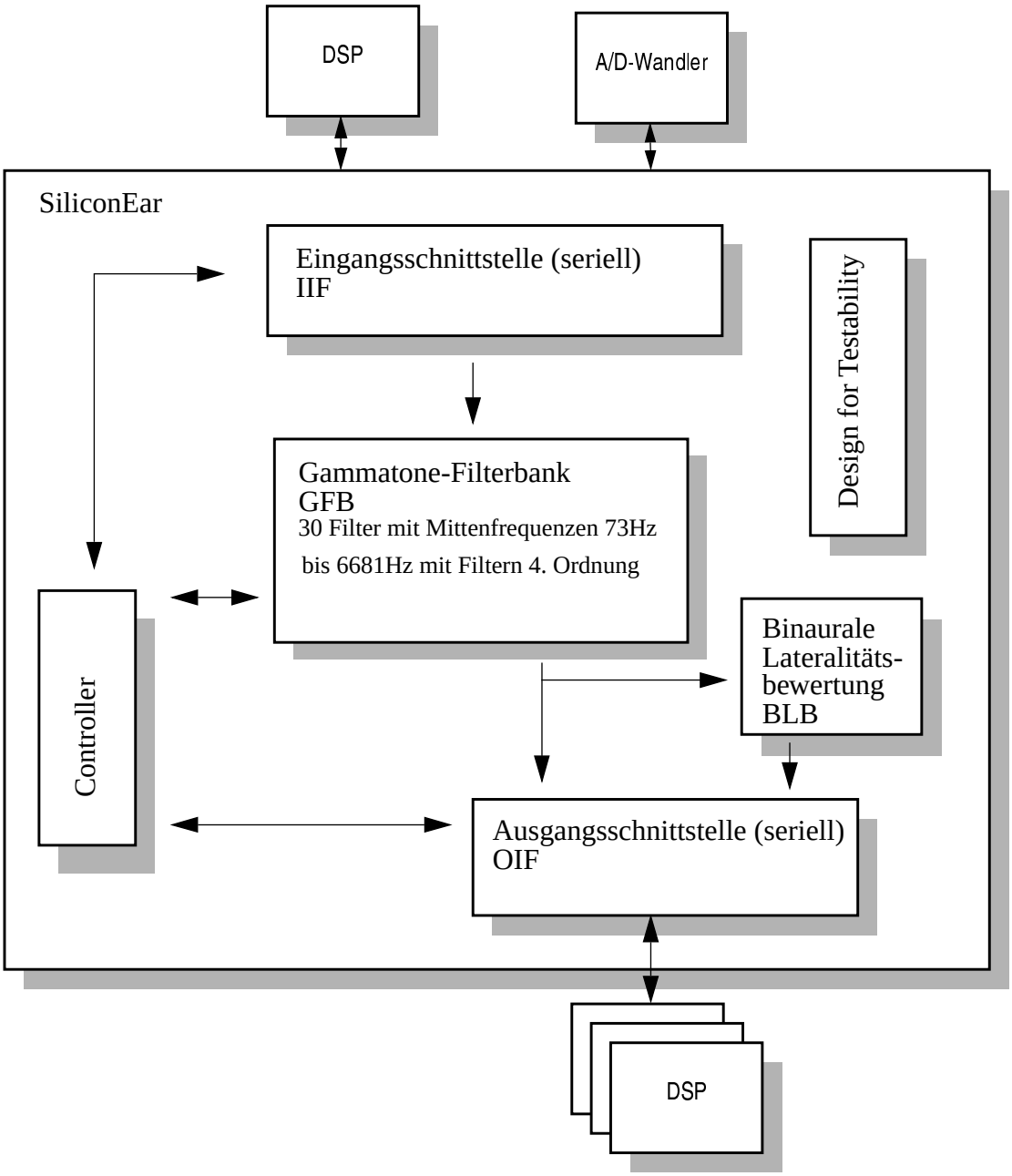


Abbildung 5: Blockschaltbild des ASIC aus der PG SiliconEar

Aufgrund der geringen Erfahrung der Projektgruppenteilnehmer wurden Designentscheidungen getroffen, die sich im Nachhinein als ungünstig für die weiteren Designschritte herausstellten. Die Folge war, daß die Verhaltensbeschreibung nicht maschinell in eine Registertransferbeschreibung überführt werden konnte. So wurden die Schnittstellen der Blöcke zu großzügig dimensioniert. Das Resultat waren Schaltungen mit Registern von 1280 Bit Breite und entsprechenden Multiplexern von 1280 Bit auf 16 Bit. Beim Syntheseversuch war das einzige Ergebnis von Synopsys ein Coredump mit einer Größe von einigen hundert Megabytes.

Die Projektgruppe war aus Zeitgründen nicht in der Lage die Integration von RAM und ROM in den ASIC zu prüfen. Das Vorhalten der Filterkonstanten und -variablen in Registern ist wegen eines zu großen Bedarfs an Chipfläche nicht effizient.

Ein Redesign der Verhaltensbeschreibung bei gleichzeitiger Überführung auf die Registertransferebene ist notwendig und Ziel dieser Arbeit. Im nächsten Kapitel wird auf die erforderlichen Änderungen eingegangen.

5 Änderungen beim Redesign

Die in der PG SiliconEar aufgedeckten Probleme machten Änderungen im Design des ASIC notwendig. Hier ist eine Auflistung der Aufgaben, die innerhalb der Diplomarbeit zu bearbeiten waren.

- **Datensenken**

  Die serielle Schnittstelle des DSP 32C von AT&T bewältigte den Datenstrom am Ausgang des ASIC nicht. Aus diesem Grund wurde als Zielumgebung ein DSPC60x von Texas Instruments gewählt. Seine serielle Schnittstelle kann mit einem Takt von 50 MHz arbeiten. Das entspricht einer theoretischen Datenrate von 47,68 Mbps, die zur Übertragung sämtlicher Daten ausreicht.

  Als weitere Senke wird ein zweiter ASIC von der AG IMA in Hamburg entwickelt. Auf diesem VLSI-Chip sind die NRGL des Perzeptionsmodells implementiert. Auch hier werden die Daten seriell übertragen. Die Frequenz beträgt 12,5 MHz. Die maximale Datenrate ist somit 11,92 Mbps.

- **Protokolle des OIF**

  Angesichts der veränderten Senken, wurden die Protokolle für die Kommunikation zwischen dem ASIC und der neuen Systemumgebung geändert, und das OIF war an diese neuen Protokolle anzupassen. Als folge hiervon wurden die HSS und die LSS spezifiziert.

- **neue Blöcke**

  Um für die LSS des OIF eine Reduktion der Datenrate zu erhalten, wurden weitere Bestandteile des Perzeptionsmodells mit im SiliconEar integriert. Diese sind ein Halbwellengleichrichter und ein Tiefpaßfilter zur Einhüllendenbildung. Ein Block zur Berechnung des gleitenden Mittels aus den Resultaten der BLB ersetzt die Mittelwertbildung nach 128 Werten. Das Übertragungsprotokoll der LSS machte diesen Schritt erforderlich.

- **Redesign der Schnittstellen**

  Die Schnittstellen der Blöcke GFB <-> OIF, GFB <-> BLB und BLB <-> OIF wurden neu überdacht. Die Schnittstellen der ersten ASIC Version bestanden aus Registern mit Breiten weit größer als 1000 Bit. Diese enthielten sämtliche Ergebnisse, die aus einem Sample berechnet wurden. Die nachfolgenden Blöcke verarbeiteten die Teilergebnisse jedoch nicht gleichzeitig, sodas das zur Verfügungstellen sämtlicher Werte zu einem Zeitpunkt überflüssig war. Effizienter ist, Werte „just in time“ zu berechnen und somit die Breite von Register zusammenschrumpfen zu lassen.

- **Integration von RAM & ROM**

  Für einen geringen Verbrauch von Chipfläche und eine geringe Verlustleistung war es erforderlich, Register durch RAM- und die Konstanten durch ROM-Blöcke zu ersetzen.

- **Synthese**

  Bei der Implementation des ASICs auf RT-Ebene mußte eine spätere Synthese im Auge behalten werden. Als Ergebnis dieser Arbeit sollte eine RT-Beschreibung des SiliconEar vorliegen, die maschinell auf die Gatterebene abgebildet werden kann.

- **Wegfall des Blocks CU**

  Bei der Überarbeitung der Schnittstellen wurden Handshakeprotokolle zwischen den Blökken eingefügt, sodas eine übergeordnete Kontrollinstanz nicht mehr gebraucht wurde. Sämtliche Synchronisationen zwischen den Blöcken werden von diesen lokal durchgeführt. Der Block CU aus der PG wurde ersatzlos gestrichen.

Es wurde geprüft, wie die durch die PG SiliconEar erstellte Verhaltensbeschreibung des ASIC für diese Diplomarbeit verwendet werden kann. Dabei stellte sich heraus, daß die Eingriffe besonders in die Struktur der GFB und des OIF zu fundamental waren, als das der bereits vorhandene VHDL-Code weiterverwendbar wäre.

Die zweite Überlegung, die Verhaltensbeschreibung zumindest als Testbench für die Vergleichsdatengewinnung heranzuziehen, wurde ebenfalls verworfen, da Ungereimtheiten im VHDL-Code auftraten. Es befanden sich sogar syntaktische Fehler in der Verhaltensbeschreibung. Es stellt sich die Frage, ob der vorliegende VHDL-Code derjenige war, der von der PG einem Abschlußtest unterzogen wurde, oder ob beim Anfertigen der Sicherheitskopie VHDL-Code älterer Versionsnummer aufgenommen wurde. Eine Rekonstruktion der Endversion hat die inzwischen aufgelöste Projektgruppe nicht erbracht. Aus Mangel an Vertrauen in die Ergebnisse wurde der VHDL-Code aus der PG somit nicht weiter verwendet.

Die C-Testbench der PG erwies sich als zu unflexibel. Es handelt sich um ein nicht modulares, unübersichtliches Konstrukt, welches kaum noch von seinem Programmierer verstanden wurde. Die erste Version der OIF wurde mit in den C-Code integriert, sodas ein einfacher Austausch durch die zweite nicht möglich war. Somit wurde auch der C-Code der PG nicht berücksichtigt, sodas die Diplomarbeit vom Sourcecode her bei „Null“ begonnen wurde.

Es soll hier nicht der Eindruck entstehen, die Projektgruppe hätte keine nutzbaren Ergebnisse erbracht. Im Gegenteil wurden während der Codeerstellung eine Vielzahl Probleme erkannt. Erst in der PG konnte ein geeigneter Algorithmus zur Berechnung der Bandpaßfilter ermittelt werden, der in Hardware implementiert werden kann. Das Ergebnis der Projektgruppe ist kein fertiger VLSI-Chip oder dessen Beschreibung, aber es wurden die „Hotspots“ des Designs aufgedeckt, auf die es sich zu konzentrieren galt. Diese Informationen waren für die folgende Arbeit von essentieller Bedeutung. Abbildung 6 zeigt das abgeänderte Blockschaltbild des im Rahmen dieser Arbeit zu erstellenden ASIC.

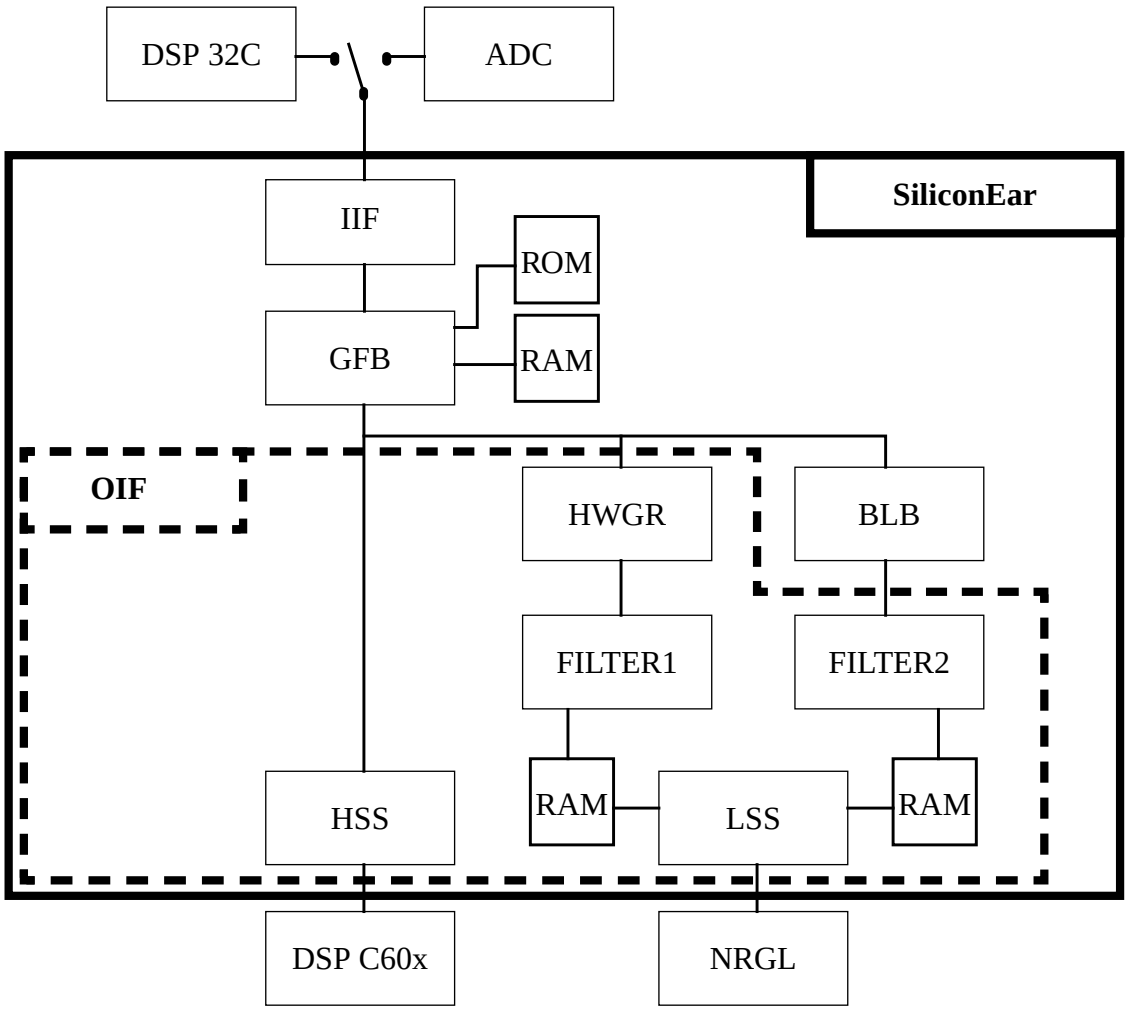


Abbildung 6: Geändertes Blockschaltbild des ASIC

6 Analyse der Operationen im ASIC

Die Berechnungen im SiliconEar sind in einem strengen Zeitrahmen von 61,44 µs (entsprechen einer Frequenz von 16,276 KHz) durchzuführen. Wird als zu optimierender Parameter ausschließlich die Zeit betrachtet, so liegt z.B. die Lösung nahe, alle 30 Bandpaßfilter der GFB parallel zu implementieren. Das ist jedoch nicht zweckmäßig, weil die GFB dann mit der Berechnungsdauer weit unter dem Zeitlimit liegen würde (siehe auch Kapitel 6.2) und danach idle wäre. Vorhandene Zeit und Chipfläche werden nicht effizient genutzt.

Die Fläche ist so weit zu minimieren, bis die Randbedingung, sämtliche Berechnungen mit einer Frequenz von 16,276 KHz durchzuführen, noch erfüllt wird. So wird ein optimales Preis (Platz) Leistungs (Zeit) Verhältnis erreicht.

Eine Analyse der benötigten Operationen pro Zeit soll an dieser Stelle am Beispiel der GFB durchgeführt werden. Das ist sinnvoll, da die GFB die platz- und zeitintensivste Komponente des ASIC ist. Aus diesem Zweck sollen die Gleichungen der Bandpaßfilter vorgestellt werden.

6.1 Funktionsprinzip der GFB

Die GFB ist ein Array bzw. eine Bank von digitalen Filterkanälen. Es handelt sich um Filter mit unendlichen Impulsantwort, sogenannte „Infinite Impuls Response Filter“, kurz IIR vierter Ordnung [10, 11, 12]. Sie arbeiten nach der Formel 1.

$$\tilde{y}[i] = (a \cdot \tilde{x}[i]) + (\tilde{b} \cdot \tilde{y}[i-1]) \qquad \text{4mal iteriert}$$

Formel 1

a und b sind filterkanalspezifische Konstanten. Ihre Werte sind im Anhang B in der Tabelle 6 zu finden. Das Symbol „~“ signalisiert die Komplexwertigkeit der Werte x, y und b. Nach Auflösen der Gleichung in ihren Real- und Imaginärteil ergeben sich die Formeln 2 und 3. Siehe auch die Abbildung 7.

$$Re(\tilde{y}[i]) = (a \cdot Re(\tilde{x}[i])) + (Re(\tilde{b}) \cdot Re(\tilde{y}[i-1])) - (Im(\tilde{b}) \cdot Im(\tilde{y}[i-1]))$$

Formel 2

$$Im(\tilde{y}[i]) = (a \cdot Im(\tilde{x}[i])) + (Re(\tilde{b}) \cdot Im(\tilde{y}[i-1])) + (Im(\tilde{b}) \cdot Re(\tilde{y}[i-1]))$$

Formel 3

Mittels Substitution der Konstanten a und b durch Werte aus Tabelle 6 im Anhang B werden die Berechnungsvorschriften für die 30 Bandpaßfilter gebildet.

In Kapitel 4 wurde der binaurale Designentscheid genannt. Demzufolge ist die GFB zweimal zu implementieren. Die Konstanten sind für den linken und rechten Stereokanal die selben. Deshalb kann eine einmalige Implementation der GFB ausreichend sein, wenn diese mit der doppelten Geschwindigkeit arbeitet, als sie bei einer parallelen Implementation erforderlich wäre. Diese GFB könnte beide Stereokanäle sequentiell berechnen. Dieses Vorgehen birgt großes Potential bei der Einsparung von Chipfläche. Der folgende Abschnitt soll zeigen, daß eine sequetielle Verarbeitung machbar ist.

6.2 Berechnung der maximal zulässigen Zeitdauer einer Operation in der GFB

Die GFB besitzt 30 Bandpaßfilter, die nicht notwendigerweise parallel berechnet werden müssen. Sie operieren vollständig nebenläufig. Eine sequentielle Berechnung ist somit nicht ausgeschlossen, wenn diese den zeitlichen Rahmen von 61,44 µs nicht verläßt. Bei einer Verringerung der Verzögerungszeit einer Bandpaßfilterimplementation um den Faktor 30, kann diese alle Berechnungen der GFB durchführen. Diesem dann programmierbaren Kanal sind die Filterkonstanten a und b als Parameter zu übergeben. Bei einer binauralen Version ist die Verzögerungszeit nochmals zu halbieren.

Es stellt sich die Machbarkeitsfrage: Gelingt es, einen programmierbaren Filterkanal mit 60fach verringerter Verzögerungszeit, im Vergleich zu 60 fixen Kanälen einer parallelen Version der GFB, zu realisieren?

Das zu entscheiden erfordert eine genauere Analyse der Gleichungen aus dem vorangegangenen Unterkapitel. Abbildung 7 zeigt die Struktur der vierfach iterierten Gleichungen 2 und 3. Es sind pro Bandpaßkanal 24 Multiplikationen und 16 Additionen zu berechnen. Die Multiplikationen wurden mit M1 bis M24, die Additionen mit A1 bis A16 durchnumeriert. Zwei Worte, eines für den rechten und eines für den linken Stereokanal, sind bei einer Frequenz von 16,276 KHz zu verarbeiten. Bei einem 50 MHz Takt stehen somit 1536 Zyklen pro Stereosample zur Verfügung (siehe Formel 4).

$$\frac{50\mathrm{MHz}}{16,276\mathrm{kHz} \cdot 2} = 1536 \text{ Takte pro Stereosample}$$

Formel 4

Die Stereosample sind sequentiell von 30 Filterkanälen auszuwerten. Damit bleiben für die Berechnung eines Filters 51 Takte.

Abbildung 8 zeigt, daß prinzipiell ein Scheduling existiert, nachdem alle Multiplikationen sequentiell zueinander berechnet werden können. Dabei werden die Additionen parallel zu den Multiplikationen und sequentiell zueinander bearbeitet.

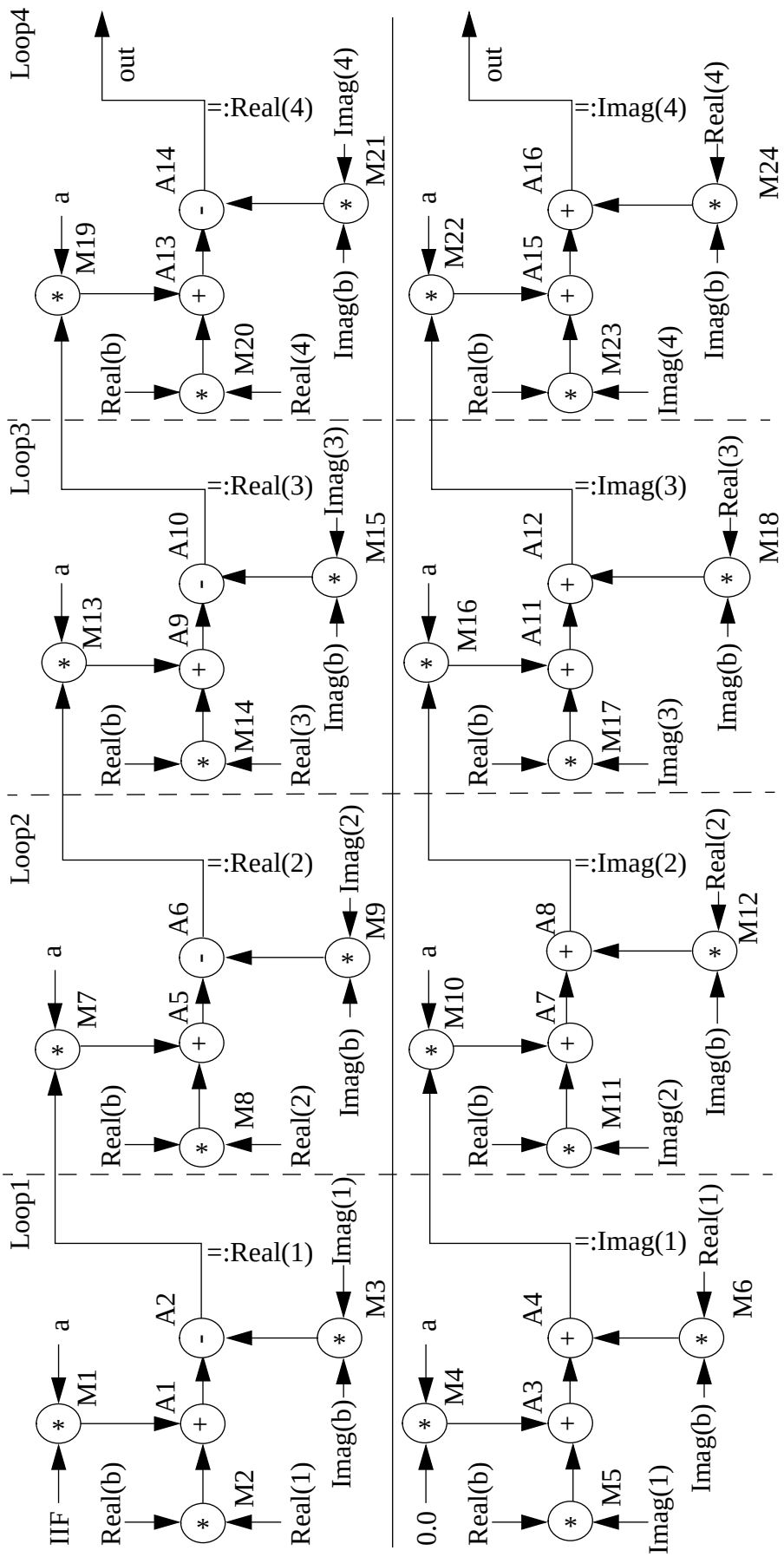


Abbildung 7: Visualisierung der Formeln 2 und 3

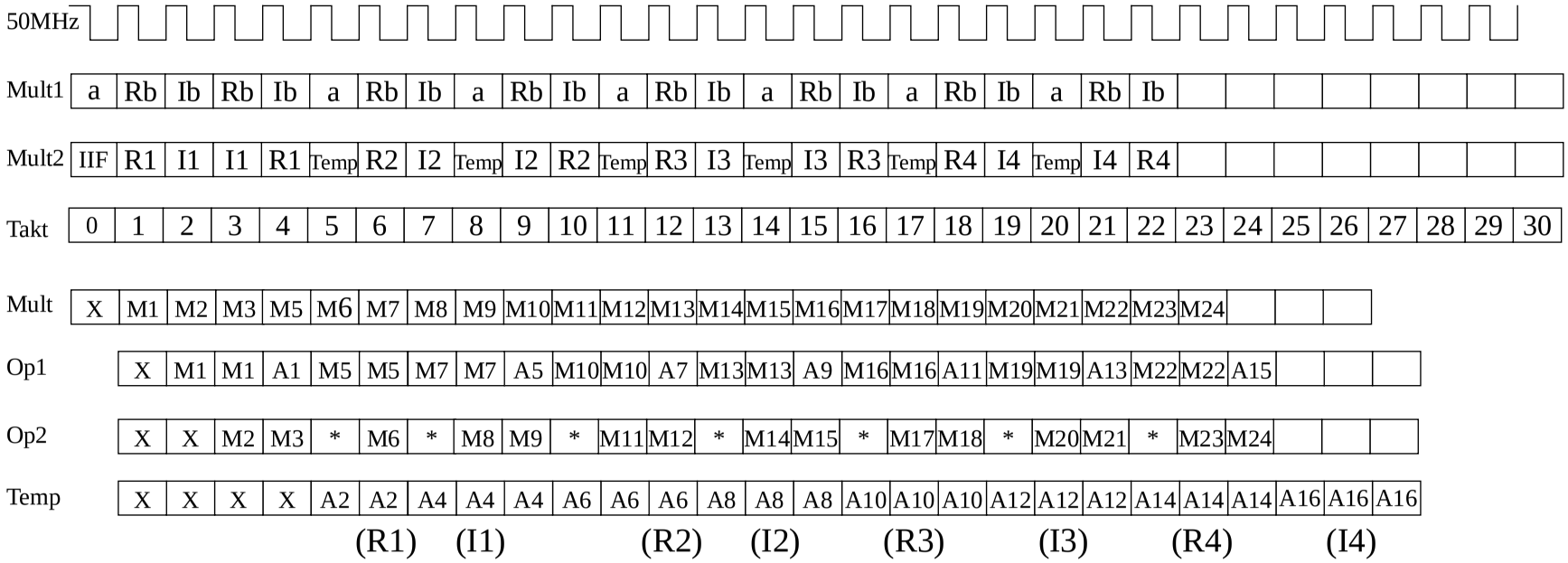


Abbildung 8: Gantdiagramm für zweistufigen Multiplier und Adder

Das Diagramm ist wie folgt zu lesen:
Nach einem Takt wurde vom zweistufigen Multiplizierer die Multiplikation M1 berechnet. Das Ergebnis wird bei steigender Flanke in das Register OP1 übernommen. Im zweitenTakt steht M2 zur Verfügung und wird bei steigender Flanke in das Register OP2 übernommen. Beide Register des Addierers sind geladen und A1 kann berechnet werden. Das Ergebnis wird wieder in das Register OP1 geschrieben. OP2 erhält das Ergebnis von M3 vom Multiplizierer. Der Addierer berechnet im 4.Takt A2. Das Ergebnis, welches gleichzeitig der erste neue Realwert ist, wird in das Register Temp zwischengespeichert. Usw.

Die Multiplikation M4 und damit die Addition A3 brauchen nicht berechnet zu werden, da das Ergebnis konstant Null ist. Es verbleiben 23 Multiplikationen und 15 Additionen. Nach Gleichung 5 ergeben sich 1536 Takte pro Stereosample zu 2 Takten pro Multiplikation.

$$\frac{1536 \text{ Takte pro Stereosample}}{23 \text{ Multiplikationen} \cdot 30 \text{ Kanäle}} = 2 \text{ Takte pro Multiplikation}$$

Formel 5

Das Ergebnis ist folgendermaßen zu interpretieren: Wird genau ein Variablenmultiplizierer implementiert, darf dieser nicht mehr als zwei Takte (bei einem Systemtakt von 50 MHz sind das 40 ns) benötigen, um sämtliche Multiplikation der binauralen GFB durchführen zu können. Ein parallel zum Multiplizierer arbeitender Addierer ist einfach zu erzeugen, da ein Addierer weit weniger Ressourcen verbraucht als ein Multiplizierwerk.

Das 7. Kapitel handelt folglich von der Analyse vorhandener Addierer und Multiplizierer aus den Designware Libraries 1 und 2, kurz DW01 und DW02, von Synopsys.

7 Analyse von Designware Komponenten

Aus Kapitel 6 folgt die Aufgabe, ein Multiplizierwerk und ein Addierwerk zu finden, deren maximale Verzögerungszeit 40 ns nicht überschreitet. Es ist zu beachten, daß in der Berechnung aus dem vorherigen Abschnitt, keine Verzögerungszeiten für Registerumladevorgänge berücksichtigt wurden. Der zeitliche Rahmen von 40 ns, zwei Systemtakten, ist knapp bemessen. Wünschenswert sind Rechenwerke mit einem Durchsatz von einer Operation pro Systemtakt. Bei der dem ASIC zugrundeliegenden Technik ecpd07 soll vom Worst Case ausgegangen werden, so daß von den 20 ns Systemtakt 1,5 ns ecpd07 spezifische maximale Setupzeit abzuziehen sind. Es verbleiben somit nicht mehr als 18,5 ns. Die Latenz spielt eine geringe Rolle bei Berechnungen im Perzeptionsmodell. Der Mensch nimmt Verzögerungen, z.B. zwischen der Bewegung der Lippen eines Sprechers und dem registrieren der Sprache, unterhalb einer millisekunde nicht wahr. Eine millisekunde Verzögerungszeit entspricht 50.000 Takten und wird bei weitem nicht erreicht.

Latenz und Durchsatz sind Begriffe, die im Zusammenhang mit Pipelining in Rechnerarchitekturen fallen. Der Unterabschnitt 7.1 erklärt das Prinzip des Pipelining. Mit diesem Wissen werden in 7.2 Komponenten aus der DW01 und DW02 untersucht.

7.1 Was bedeutet Pipelining?

Pipelining in einer Rechnerarchitektur ist das Äquivalent zur Fließbandarbeit in einem produzierenden Betrieb. Das Produkt ist pro Zeit maximal oft zu generieren. Dabei spielt die dauer der Herstellung des einzelnen Resultats eine untergeordnete Rolle. So verlassen beispielsweise bei einem Fahrzeughersteller pro Stunde 10 neue Wagen die Werkshallen. Diese Zahl wird Durchsatz genannt. Ein einzelnes Auto wird jedoch nicht innerhalb von 6 Minuten erstellt. Die Dauer der Herstellung eines einzelnen Wagens wird Latenzzeit genannt und kann ein vielfaches der Durchsatzzeit betragen. Diese Vorgehensweise kann auch beim Hardwareentwurf angewendet werden:

Eine in Hardware realisierte Funktion $f(x_1,...,x_n)$ habe die Verzögerungszeit T. T_{cycle} sei die Dauer eines Systemtakts Clk, mit dem die zu grundeliegende Hardware arbeitet. Die minimale Setupzeit des Designs sei T_{setup}. Es ergeben sich dann zwei mögliche Fälle:

1.) $T+T_{setup} < T_{cycle}$

Die Berechnung der Funktion $f(x_1,...,x_n)$ wird innerhalb eines Systemtaktes abgeschlossen. Die Werte x_i werden bei steigender Flanke auf Clk geladen und mit der nächsten steigenden Flanke kann das Ergebnis in ein Register geschrieben werden. Es gibt keine Probleme beim Timing des Designs.

2.) $T+T_{setup} > T_{cycle}$

Die Berechnung der Funktion $f(x_1,...,x_n)$ benötigt mehr Zeit, als in einem Takt Clk gegeben ist. Alle k Takte mit k=(T modulo T_{cycle}-T_{setup})+1 kann ein Ergebnis in ein Ausgangsregister geschrieben werden. Der Durchsatz ist somit 1 Ergebnis pro k Takte. Abbildung 9a zeigt den zeitlichen Verlauf von Berechnungen für k=3.

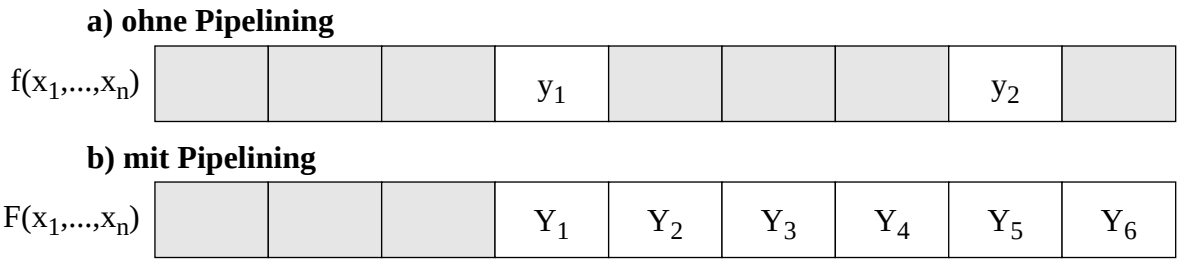


Abbildung 9: Ohne kontra mit Pipelining

Der 2. Fall kann mittels Pipelining optimiert werden, wenn sich f in mehre Teilfunktionen zerlegen läßt (Abbildung 9b). Sei $f(x_1,...,x_n)=f_1(f_2(f_3(x_1,...,x_n)))$ und T_1,T_2,T_3 die Verzögerungszeiten der Teilfunktionen. Weiter seien $T_i+T_{setup}<T_{cycle}$ mit iε{1,...,3}. Dann kann eine semantikerhaltene Umformung nach Abbildung 10 durchgeführt werden. Die Latenzzeit k, in dem Beispiel ist k=3, ist die selbe geblieben. Das Einbringen von Registern entkoppelt die Teilfunktionen voneinander. Während f_2 einen Output von f_3 bearbeitet, kann f_3 die nächste Berechnung beginnen. Nach drei Takten Latenzzeit wird mit jedem weiteren Taktsignal ein Ergebnis geliefert. Der Durchsatz ist auf ein Ergebnis pro Takt gestiegen.

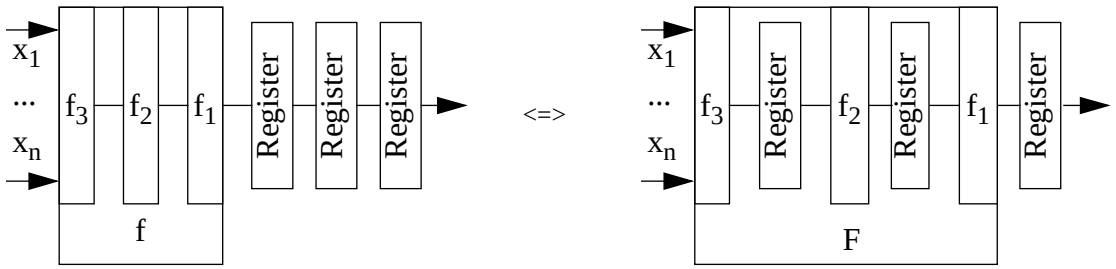


Abbildung 10: Pipelining

7.2 Die Designware Komponenten

Synopsys stellt in den Libraries DW01 und DW02 Komponenten zur Verfügung, die für das Design des ASIC von Interesse sind. Die DW01 enthält Elemente aus der ALU Familie. Hier befinden sich unter anderem die Addierer DW01_add und DW01_addsub. In der DW02 befindet sich die Advanced Math Familie. Betrachtet werden die Multiplizierer DW02_mac, DW02_prodsum1, DW02_mult, DW02_mul2 und die mehrstufigen DW02_mult_n_stage.

Mehrstufig ist synonym für gepipelined. Die Onlinehilfe gibt weitere Informationen zu den Libraries. Zunächst soll in 7.2.1 die Wahl eines Multiplizierers begründet werden. In 7.2.3 wird ein passender Addierer gesucht. Auf Grund der höheren Komplexität im Vergleich zu einem Addierer ist es sinnvoll in dieser Reihenfolge vorzugehen.

7.2.1 DW02

Bei DW02_mult, DW02_mult2 und DW02_mult_n_stage handelt es sich um einfache Multiplizierer der Art A mult B => C. DW02_mac und DW02_prodsum1 realisieren die Funktion (A mult B) plus C => D. Diese Funktion ist ideal für die Realisierung der GFB, wie aus Abbildung 7 bzw. Formel 1 aus Kapitel 6 deutlich wird. In [42] im Kapitel 6.4.1 befinden sich weitere Informationen. Dort wird auch gezeigt, daß diese Komponenten zu langsam sind. Ehe weitere Spekulationen über vermeintlich gute Komponenten angestellt werden, sollen die Designs auf die Technik ecpd07 gemappt werden. Das Ergebnis ist in Tabelle 1 notiert. Beim Erzeugen der Reports zur Ermittlung der Verzögerungszeiten wurde das für die Technologie ecpd07 schlechteste Zeitverhalten gewählt. Die für ecpd07 typischen Zeiten sind etwa halb so groß wie jene vom Worst Case.

Designname DW02_...	Funktion	Verzögerungs-zeit in ns	Platzbedarf in Squares	Platzbedarf in mm^2 (ca.)
mac(csa)	A*B+C	86,49	5.867.717	2,87
mac(wall)	A*B+C	101,35	5.982.605	2,93
prodsum1(csa)	A*B+C	56,98	2.132.768	1,05
mult(csa)	A*B	78,19	4.927.088	2,41
mult2(csa)	A*B	28,67	5.455.208	2.67
mult_2_stage	A*B	18,21	6.697.720	3,28
mult_3_stage	A*B	15,22	7.533.945	3,69
3_stage optimiert	A*B	14,90	7.955.379	3,90
mult_4_stage	A*B	12,95	7.989.482	3,91
mult_5_stage	A*B	12,66	9.089.070	4,45
mult_6_stage	A*B	11,26	10.020.931	4,91
6_stage optimiert	A*B	11,03	10.122.832	4,96

Tabelle 1: DW02 auf ecpd07 mit 24 Bitbreite gemappt, max. delay

Die einzigen Komponenten, die bei der Multiplikation einen Durchsatz von einer Multiplikation pro Takt (20 ns) erreichen, sind die n-stufigen Multiplizierer. Zunächst fiel die Wahl auf den mult_2_stage, da für diesen in Abbildung 8 bereits ein passendes Scheduling dargestellt

Bei der Implementation der GFB mit einem sechsstufigen Multiplizierer werden 47 Takte verbraucht. Das entspricht einer Zeit von 39,16 ns pro Multiplikation bzw. 56,4 µs pro Stereosample. Die Maximalzeit von 61,44 µs wird um 5 µs unterboten.

Abbildung 11: Scheduling für den DW02_mult_6_stage

wurde, und er die vorhandene Zeit optimal bei minimalem Platzverbrauch ausnutzt. Bei der Synthese der GFB (dazu mehr im Kapitel 11) stellte sich heraus, daß das Timing dieser Wahl

zu knapp war. Daher wurde der mult_2_stage gegen den mult_6_stage ausgetaucht. So konnte eine zusätzliche Zeitreserve von 3,96 ns mittels eines um 1,63 mm^2 erhöhten Flächenverbrauchs erkauft werden.

Es stellte sich die Frage: Existiert für den sechsstufigen Multiplizierer ein geeignetes Scheduling? Die Antwort liefert das Gantdiagramm aus Abbildung 11. Hier ist zu erkennen, daß die Parallelität von Multiplikationen und Additionen nicht mehr vollkommen gilt. Die Pipeline des Multiplizierers läuft immer dann leer, wenn die Additionen M7, M13 und M19 wegen der hohen Latenzzeit von sechs Takten noch nicht berechnet wurden. Es ergeben sich 16 Wartetakte. 47 Takte dauert infolgedessen die Berechnung eines Kanals, und somit verstreicht im Durchschnitt pro Multiplikation eine Zeit von 39,2 ns. Die berechnete Zeit von 40 ns wird vollkommen genutzt. In der Berechnung aus Kapitel 6 werden keine Registerumladezeiten bedacht. Diese sind im Scheduling aus Abbildung 11 enthalten. Somit dauert die Berechung eines Kanals 0,94 µs, die Berechnung einer Filterbank 28,2 µs und die eines Stereosamples 56,4 µs. Die Samplingfrequenz von 16,276 kHz läßt eine maximale Zeit von 61,44 µs zu. Es beibt ein Zeitpuffer von 5 µs. Auch die Synthese zeigte, daß der DW02_mult_6_stage eine für die GFB funktionstaugliche Komponente ist. Ein technisches Datenblatt befindet sich im Anhang D. Es wurde der Onlinedokumentation *iview* entnommen.

7.2.2 Multiplizierer als Megazelle

Es wurde die Möglichkeit in Betracht gezogen, den Multiplizierer nicht der DW02 Library zu entnehmen. Stattdessen wurde mit Hilfe des Megacell Compilers (siehe hierzu [45] bzw. Kapitel 8) ein Makroblock aus der ES2 Library erzeugt. Ein Mapping mittels Synopsys ist nicht erforderlich. Der so gewonnene Blockmultiplizierer ist speziell für diese Technologie von ES2 entwickelt worden und sollte daher sehr effizient sein. Die Verzögerungszeit von maximal 41,76 ns disqualifiziert diese Komponente und macht sie für eine Verwendung in der GFB untauglich, wenn nur mit einem Multiplizierwerk gearbeitet werden soll. Ein anderes Design mit zwei oder mehr Makromultiplizierern wäre denkbar, soll in dieser Arbeit aber nicht weiter verfolgt werden. Da in [45] nichts zu mehrstufigen Multiplizierern zu finden ist, werden die Möglichkeiten des Megacell Compilers in dieser Hinsicht nicht weiter verfolgt.

7.2.3 DW01

Die Wahl eines passenden Addierers fällt leichter. Hier soll auch eine Tabelle (Tabelle 2) einen Überblick verschaffen. Die Komponenten wurden mit Defaulteinstellungen und maximalem Mappingaufwand synthetisiert. Auch hier wird ausschließlich der Worst Case für die Verzögerungszeiten betrachtet.

Die einfachen und wenig Fläche verbrauchenden Addierer, die ripple carry adder (rpl), kommen für das Design nicht in Frage, da die Verzögerungszeiten über der Taktdauer von 20 ns lie-

gen. Eine Optimierung durch setzen von zeitlichen Constrains im *es2_design_analyzer* hatte keinen nutzbringenden Effekt. Diese Komponenten belegten ebensoviel Platz wie ein carry look-ahead (cla) oder first carry look-ahead (clf) Addierer und besaßen dennoch eine größere maximale Verzögerungszeit.

Die Berechnung der GFB schließt Subtraktionen mit ein. Die Wahl ist also auf den Addierer/ Subtrahierer addsub gefallen. Welche Version, cla oder clf, verwendet wird, macht keinen wesentlichen Unterschied im Platzbedarf und Zeitverhalten. Im Anhang D befindet sich ein Datenblatt zu dieser Komponente.

Designname DW01_...	Funktion	Verzögerungs-zeit in ns	Platzbedarf in Squares	Platzbedarf in mm^2 (ca.)
add(cla)	A+B	6,93	424.204	0,21
add(clf)	A+B	7,64	422.386	0,21
add(rpl)	A+B	28,85	97.728	0,05
addsub(cla)	A+/-B	8,59	465.146	0,23
addsub(clf)	A+/-B	8,55	490.772	0,24
addsub(rpl)	A+/-B	30,71	138.520	0,07
addsub(rpl) mit Constrain 17ns	A+/-B	17,00	455.952	0,22

Tabelle 2: DW01 auf ecpd07 mit 24 Bitbreite gemappt, max. delay

8 Verwendung von RAM und ROM in der GFB

Der programmierbare Bandpaßkanal, welcher alle 30 Filterkanäle der GFB berechnet, benötigt eine große Zahl Parameter für seine Funktion. Es handelt sich hierbei sowohl um Konstanten, wie auch um Variablen. Die Eingangsskalierung a, der Realteil und der Imaginärteil von b sind Konstanten innerhalb der 30 Bandpaßfilter. Jeder der 30 Kanäle besitzt somit ein eigenes Konstantentripel. Die interne Bitgenauigkeit beträgt 24 Bit (siehe Kapitel 4). Es sind folglich 2160Bit an Konstanten vorzuhalten.

Jeder der 30 Kanäle benötigt vier Iterationsstufen für die Berechnung (siehe Formel 1). Die komplexwertigen Zwischenergebnisse jeder Stufe werden für die Berechnung des nächsten Samples benötigt und können daher nicht verworfen werden. Sowohl Real- als auch Imaginärwert werden mit einer Genauigkeit von 24 Bit berechnet. Aus der binauralen Verarbeitung folgt damit nach Formel 6, daß 11,248 KBit variabler Speicher benötigt wird. Es ist nicht effizient, die variablen Werte in Registern zu halten. Die 24 Bit Register benötigen nach einer Analyse mit dem *es2_design_analyzer* eine Fläche von 142.614,80 Square. Bei der verwendeten Technologie ecpd07 entspricht das einer Fläche von 0.07 mm^2. Alle 480 Register würden eine Fläche von 33,6 mm^2 einnehmen. Darin sind noch keine Multiplexer zur Ansteuerung berücksichtigt. Der Flächenverbrauch ist zu hoch und die Komplexität der Strukturen übersteigt die Fähigkeiten der verwendeten Designtools.

$$2 \text{ imaginär} \cdot 30 \text{ Kanäle} \cdot 4 \text{ Iterationen} \cdot 2 \text{ binaural} \cdot 24 \text{ Bit} = 11,248 \text{ KBit}$$

Formel 6

In der DW03 Library von Synopsys befinden sich mehrere RAMarten, die als Ersatz für die Register in Betracht kommen. Synopsys erstellt die Blöcke der DW03 Library aus Zellen der verwendeten Zieltechnology, also aus UND-Gatterb und ODER-Gattern etc. Diese Art RAM zu verwenden ist dann sinnvoll, wenn nur wenige Werte zwischengespeichert werden sollen. Die DW03 Library hilft bei einer schnellen und einfachen Erstellung. Platzverbrauch und Verlustleistung sind jedoch nicht optimiert. Aus [17] ist zu entnehmen, daß die RAM-Größen der DW03 256 Bits nicht überschreiten sollten. Anstelle dessen wird die Verwendung von Makroblöcken empfohlen.

ES2 stellt den in [43] Kapitel 5 beschriebenen Generator Kernel „Cgenerate“ zur Verfügung. Mit Hilfe dieses C-Programms werden Megacells wie RAM, ARAM, ROM, Dual-Port RAM, FIFO und Multiplizierer erstellt. Interessant für diese Diplomarbeit sind die Unterkapitel 5.4 Compiled ROM Megacells, 5.5 Compiled Multiplier Megacells und 5.6 Compiled Dual-Ported RAM Megacells.

8.1 Einrichten der ES2 Designumgebung

In den nachfolgenden Kapiteln 8.2 bis 8.5 wird die Konfiguration der Designumgebung für die Bibliotheken von ES2 vorausgesetzt. Dieses Kapitel soll beschreiben, welche Umgebungsvariablen zu setzen und welche Shellskripte auszuführen sind. Im Verzeichnis *opt/europractice/es2lib/es2vhdlk.202/notes* befindet sich eine ASCII-Datei, welche weitere Hinweise geben kann.

Wichtig sind die folgenden Schritte:

1) Setzen der Umgebungsvariablen: *setenv ES2VHDLK* und *setenv host_type Sun4*.

2) Die *.cshrc* sollte einen Eintrag enthalten, mit dem ein für ES2 benötigtes Skript ausgeführt wird: *source $ES2VHDLK/es2vhdlk.rc*

3) Im Designverzeichnis ist einmalig das Programm *es2init* auszuführen. Es erzeugt die Dateien es*2case.dat, es2proc.dat* und *es2cond.dat.* Mit dem Parameter *-proc ecpd07* wird die Zieltechnologie ecpd07 festgelegt. Mit *-cond ind* werden die Conditions auf Industriestandard gesetzt. *-case max* setzt das Timing der Zellen auf den worst case.

4) Mit *es2cgen* werden Megacells und passende VHDL Verhaltensbeschreibungen erzeugt. Hierzu befindet sich genaueres in den nachfolgenden Kapiteln.

5) Der von *es2cgen* erzeugte VHDL Code benutzt eine Library von ES2. Diese ist Synopsys durch einen Eintrag in der Datei *.synopsys_vss.setup* bekannt zu machen:

   ES2 : /opt/ep/es2/VHDLKIT/synopsys/ES2

Hiernach ist der Megacell Generator einsatzbereit.

8.2 Arbeiten mit dem Megacell Generator

Die Benutzung des Megacell Generators erweist sich als simpel. Das Programm kann mittels *es2cgen* gestartet werden, wenn die Schritte aus 8.1 korrekt durchgeführt wurden. Es erfolgt ein kurzer Dialog mit dem Benutzer:

- Art der zu generierenden Megacell.
- Name der Cell <FILE>.
- Soll BIST eingefügt werden?
- Funktionsparameter wie Wortzahl und Wortbreite etc.

Im Anschluß erzeugt das Programm die Dateien:

- <FILE>.ds: Ein Datenblatt zum erzeugten Block im ASCII-Format
- <FILE>.dat: Ein Konfigurationsfile im Format EDB
- <FILE>.sim: Ein Simulationsfile im Format EDB
- <FILE>.phy: outline and terminals im format EDB

- <FILE>.pfl: Layout als Struktur für den Compiler von Cadence
- <FILE>.edf: BIST Netzliste im Format EDIF
- <FILE>.sgn: Bist Signatur im Format FAST
- <FILE>.vhd: Verhaltensbeschreibung der Zelle in VHDL
- <FILE_p>.vhd: Package welches die Zelle enthält in VHDL
- <FILE>.ttb: Wahrheitstabelle für ein ROM
- <FILE>.prg: Programmiertabelle für ein ROM

Letztere Datei wird nicht vom Programm sondern vom Benutzer erstellt. Es handelt sich um eine ASCII-Datei mit ebenso vielen Zeilen, wie das ROM Worte enthält. Zu Beginn der Zeile, durch ein Leerzeichen vom Wert getrennt, steht die Adresse, an welcher der Wert im ROM zu finden sein wird. Die Adressen können binär oder dezimal, die Werte binär, dezimal oder hexadezimal kodiert sein.

EDB ist ein ES2internes Format und steht für „Electrical Data Base“.

Die Möglichkeiten von BIST sollen an dieser Stelle nicht weiter betrachtet werden. Zur Zeit ist eine Implementation von BIST-Strukturen im SiliconEar nicht vorgesehen.

8.3 Dual-Port RAM Megacells (DPR)

Bei Random Access Memory mit zwei Ports ist die Möglichkeit gegeben, auf zwei Werte im RAM simultan lesend und/oder schreibend einzuwirken. Eine Speicherzelle des RAM ist zusammengesetzt aus 8 bis 10 Standard-CMOS Transistoren. Dadurch ist es möglich, das RAM auf minimaler Fläche mit minimalem Verluststrom zu betreiben. In 8.3.1 wird das I/O-Verhalten von DPR erklärt. Bei der Erstellung des Kapitels wurde auf die Dokumentation [45] zurückgegriffen. In 8.3.2 werden Tests genannt, die zum Verifizieren des DPR durchgeführt wurden.

8.3.1 Darstellung von Signalabhängigkeiten im DPR

Ein DPR mit der Konfiguration Port A read-only und Port B write-only besitzt folgende Pins:

- Die Memory Enable Signale *MEa* und *MEb. ME* ist active high. *MEa* kontrolliert die Lesezyklen auf Port A und *MEb* die Schreibzyklen auf Port B.
- Die Adreßbusse *ADDa<i>* und *ADDb<i>*. Die Breite der Busse beträgt log2(#Worte).
- Den Datainputbuss *DIb<i>*. Hier liegen Werte an, die ins RAM geschrieben werden.
- Den Dataoutputbuss *DOa<i>*. Hier liegen Werte an, die aus dem RAM ausgelesen wurden.
- Power (*VDD*) und Ground (*GND*). Hier wird die Möglichkeit gegeben, das RAM mit einer vom Chipkern getrennten Spannungsquelle zu versorgen. So haben vom RAM erzeugte Spannungsschwankungen keine Auswirkungen auf den Kern eines VLSI-Chips. Wird eine getrennte Spannung nicht gewünscht, so ist Power und Ground explizit

mit den Chipglobalen *VDD* und *GND* zu verbinden. Die Powerleitugen dürfen nicht offen gelassen werden.

Beim Lesen arbeitet das DPR mit einem einfach, beim Schreiben mit einem zweifach geflanktem Takt. Das heißt: Beide Ports A und B besitzen das Memory Enable Signal *MEa* bzw. *MEb.* Bei einer steigenden Flanke auf *ME* wird die am RAM anliegende Adresse übernommen. Alle Änderungen der Adresse ohne steigende Flanke auf *ME* haben keine Auswirkungen.

Nach steigender Flanke auf Signal *MEa* wird ein Lesezyklus gestartet. Mit einer gewissen Verzögerungszeit steht das ausgelesende Wort am Ausgang *DOa* zur Verfügung.

Bei steigender Flanke auf *MEb* wird die Schreibadresse ins RAM übernommen. Erst bei fallender Flanke wird der an *DIb* anliegende Wert ins RAM geschrieben. Da beide Flanken der Clock *MEb* verwendet werden, heißt der Schreibmodus „zweifach geflankt“.

Zum Zeitverhalten der RAM Schnittstellen siehe auch die Abbildungen 12 und 13.

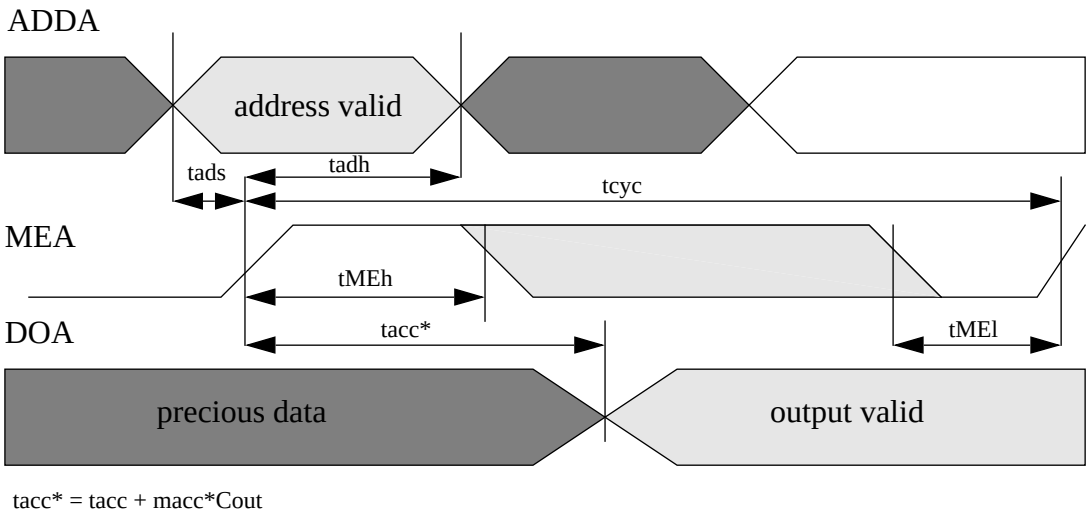


Abbildung 12: Dual-Port RAM Lesezyklus Zeitdiagramm

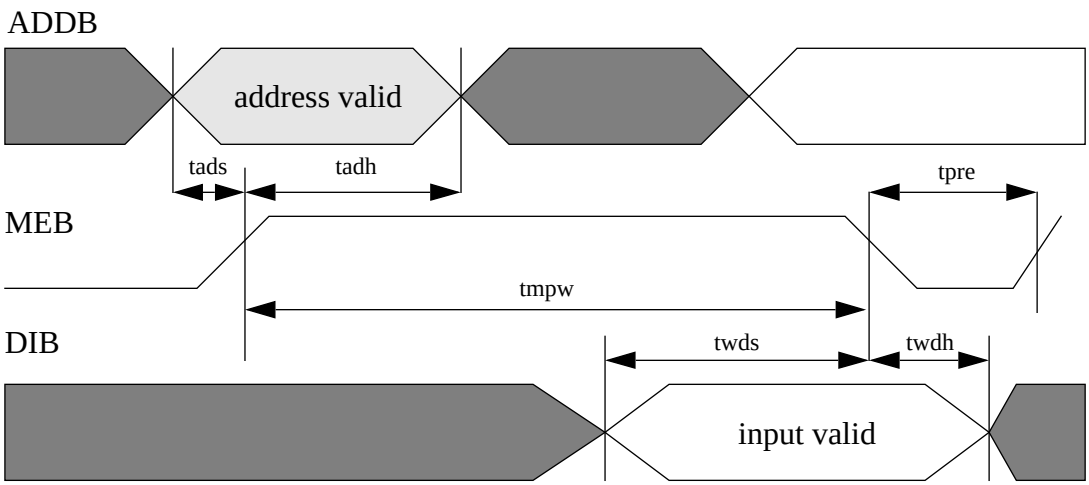


Abbildung 13: Dual-Port RAM Schreibzyklus Zeitdiagramm

8.3.2 Test des Zeitverhaltens

„Vertrauen ist gut, Kontrolle ist besser.“ Um das Zeitverhalten von DPR zu prüfen wurde die Testbench *tb_c_RAM_Block* geschrieben. In ihr wird die von ES2 gestellte Verhaltensbeschreibung für DPR überprüft. Im ersten Teil der Testbench wird das DPR im Takt vom SiliconEar mit 50 MHz angesprochen. Im zweiten Teil wird das DPR mit maximaler Geschwindigkeit betrieben.

Der vom Megacell Compiler generierte Code lieferte zunächst eine Fehlermeldung infolge einer Bereichsüberschreitung des Adressraums, obwohl dieses eigentlich nicht der Fall war. Eine Prüfung der VHDL-Beschreibung ergab, daß der Megacell Compiler offenbar nicht fehlerfrei arbeitet. Im VHDL-Code wurden Generics, welche die Größenangaben des DPR definierten, mit Null substituiert. Somit generiert *cgen* ausschließlich DPR der Größe Null. Jedoch konnte die DPR-Verhaltensbeschreibung manuell korrigiert werden. Beide Tests produzierten dann keine Fehlermeldung mehr. Es wurde gezeigt, daß der Einsatz von DPR im SiliconEar möglich ist.

Im dritten Teil wurden bewußt alle bekannten Setup- und Holdzeiten verletzt, sodas das korrekte Zeitverhalten sichergestellt wurde. Anhand des dritten Tests wurden die folgenden Fehlermeldungen generiert:

- *Error on ‚DPRGFB‘ (‚DPR‘) - port: 1 - Address to ME setup time violation (tADS) - specified: 2.41 NS found: 1.41 NS*
- *Error on ‚DPRGFB‘ (‚DPR‘) - port: 1 - Address hold time violation (tADH) - specified: 2.26 NS found: 1.26 NS*
- *Error on ‚DPRGFB‘ (‚DPR‘) - port: 1 - ME pulse width violation (tMPW) - specified: 7.62 NS found: 6.62 NS*
- *Error on ‚DPRGFB‘ (‚DPR‘) - port: 1 - Data setup time violation (tWDS) - specified: 6.84 NS found: 5.84 NS*
- *Error on ‚DPRGFB‘ (‚DPR‘) - port: 1 - Data hold time violation (tWDH) - specified: 3.01 NS found: 2.01 NS*
- *Error on ‚DPRGFB‘ (‚DPR‘) - port: 0 - Address to ME setup time violation (tADS) - specified: 2.41 NS found: 1.41 NS*
- *Error on ‚DPRGFB‘ (‚DPR‘) - port: 0 - Address hold time violation (tADH) - specified: 2.26 NS found: 1.26 NS*
- *Error on ‚DPRGFB‘ (‚DPR‘) - port: 0 - ME high timing violation (tMEH) - specified: 2.58 NS found: 2.38 NS*
- *Error on ‚DPRGFB‘ (‚DPR‘) - port: 0 - Cycle time violation (tCYC) - specified: 22.14 NS found: 21.14 NS*

Sollte eine dieser Meldungen während der Simulation des SiliconEar generiert werden, wird das DPR nicht richtig angesprochen. Nach der Synthese kann dieses bei zu großen Verzögerungszeiten der Fall sein.

8.4 Erzeugen von Dual-Port RAM Megacells (DPR)

Im SiliconEar werden insgesamt drei DPR Zellen verwendet. In diesem Kapitel soll die Erzeugung von DPR am Beispiel des RAM für die GFB durchgespielt werden.

DPR ist ein Schreib/Lese-Speicher auf den mittels zweier, voneinander unabhängiger Ports zugegriffen werden kann. Für die GFB ist die Konfiguration Port A nur lesende und Port B nur schreibende Zugriffe ausreichend. Obwohl andere Möglichkeiten bestehen, wie Port A oder B lesend und schreibend, sollen diese nicht betrachtet werden. Der Vollständigkeit halber werden hier jedoch alle Konfigurationsparameter angegeben:

- Anzahl der Reihen: 4, ..., 128
- Anzahl der Spalten: 2, ..., 128
- Anzahl der Wörter: 8, ..., 16384
- Anzahl der Bits pro Wort: 1, ..., 64
- Absolute Größe: 8, ... 16384
- Konfiguration von Port A: read/write, read-only
- Konfiguration von Port B: read/write, write-only

Im Folgenden wird der Dialog aufgelistet, der beim Erstellen des fürs SiliconEar verwendeten RAMs geführt wurde. Benutzereingaben sind **Fett** hervorgehoben:

es2cgen dprgfb dpr

-I- Tool ‚es2cgen‘ started.

-I- Spawning megacell generator.

generate_3.1_19Apr96 :

Please specify parameters for megacell dprgfb in process ecpd07

BIST ? (yes/no) (<CR>=yes) : ***no***

Choose the functional configuration :

Port A Read-only Port B Write-only	*(1)*
Port A Read-Write Port B Write-only	*(2)*
Port A Read-only Port B Read-Write	*(3)*
Port A Read-Write Port B Read-Write	*(4)*

---------> ***: 1***

Number of words, port A (readOnly) ? (8...16384): ***240***

Bits per word? (1...64): ***48***

Possible width for port B (writeOnly):

3 6 12 24 48

*Port B Width ? --->**48***

Wished configuration : PortA 240 x 48 - PortB 240 x 48

```
*--*-----------*------------*------------*-----------------*--------*
|  | rows cols |   port A   |   port B   | area (mmxmm)| tacc(ns)|
*--*-----------*------------*------------*-----------------*--------*
| 1 | 120 x  96 |   240 x 48 |   240 x 48 | 2.21 x 2.46 |   10.66  |
*--*-----------*------------*------------*-----------------*--------*
```

Only this aspect ratio is available

Generation completed

-I- Done with megacell generator.

-I- Moving megacell files to ‚generate/dprgfb/‘ dir.

-I- Creating ‚generate‘ directory.

-I- Creating ‚generate/dprgfb‘ directory.

-I- Done with file moving.

-I- Processing ‚generate/dprgfb/dprgfb.sim‘.

-I- Done with ‚generate/dprgfb/dprgfb.sim‘.

-I- Processing ‚generate/dprgfb/dprgfb.dat‘.

-I- Done with ‚generate/dprgfb/dprgfb.dat‘.

-I- Including file ‚generate/dprgfb/dprgfb.pl‘.

-I- Done with ‚generate/dprgfb/dprgfb.pl‘.

-I- Writing VHDL file ‚generate/dprgfb/dprgfb.vhd‘.

-I- Done with VHDL file ‚generate/dprgfb/dprgfb.vhd‘.

-I- Writing VHDL file ‚generate/dprgfb/dprgfb_p.vhd‘.

-I- Done with VHDL file ‚generate/dprgfb/dprgfb_p.vhd‘.

-I- Tool ‚es2cgen‘ done.

Die Tabelle am Ende des Dialogs enthält lediglich einen Eintrag. Bei der Generierung anderer Zellen sind mehrere Einträge denkbar. Durch Auswahl der Zeilennummer kann eine der Alternativen selektiert werden.

Die Parameter des so erzeugten RAMs wurden der Datei *dprgfb.ds* entnommen:

Port A Configuration : readOnly

Words : 240

	Bits per word	*:*	*48*
Port B	*Configuration*	*: writeOnly*	
	Words	*:*	*240*
	Bits per word	*:*	*48*
Aspect Ratio	*Rows*	*:*	*120*
	Columns	*:*	*96*
Bist		*:*	*no*

Technology : ecpd07
Dimensions : 2206.00 x 2463.20 (um x um) (without BIST)
Area : 5.43 sq. mm
BIST Number of equivalent nand2 gates : 0.00
---

powerA AC power, portA, VDD=5.0V, unloaded 7.91 mW/MHz
powerB AC power, portB, VDD=5.0V, unloaded 8.06 mW/MHz

ES2 DPRAM GENERATOR Generator : generate_2.2_15Jun95
Block : dprram Date/Time : 9 Sep 14:29:56 1997

Parameter	*Description*	*Min*	*Typ*	*Max*	*Mil*
Port A :					
mdlhA	*Load dependant rise time*	*0.24*	*0.56*	*1.14*	*1.34 ns/pF*
mdhlA	*Load dependant fall time*	*0.19*	*0.44*	*0.90*	*1.06 ns/pF*
taccA	*Access time*	*2.22*	*5.24*	*10.66*	*12.56 ns*
tcycA	*Cycle time*	*4.62*	*10.88*	*22.14*	*26.09 ns*
tadsA	*Address setup time*	*0.50*	*1.18*	*2.41*	*2.84 ns*
tadhA	*Address hold time*	*0.47*	*1.11*	*2.26*	*2.66 ns*
tMEhA	*MEA pulse width at high*	*0.54*	*1.27*	*2.58*	*3.04 ns*
tMElA	*MEA pulse width at low*	*0.51*	*1.20*	*2.44*	*2.87 ns*
twdsA	*data setup time*	*-*	*-*	*-*	*- ns*
twdhA	*data hold time*	*-*	*-*	*-*	*- ns*
tpreA	*min precharge (after Write)*	*-*	*-*	*-*	*- ns*

trdsA	*WE_A read pulse setup time*	*-*	*-*	*-*	*-*	*ns*
trdhA	*WE_A read pulse hold time*	*-*	*-*	*-*	*-*	*ns*
tmpwA	*min write (Write Cycle)*	*-*	*-*	*-*	*-*	*ns*
twpwA	*min write (Read Modif Write)*	*-*	*-*	*-*	*-*	*ns*
twrsA	*write setup time*	*-*	*-*	*-*	*-*	*ns*

Port B :

mdlhB	*Load dependant rise time*	*-*	*-*	*-*	*-*	*ns/pF*
mdhlB	*Load dependant fall time*	*-*	*-*	*-*	*-*	*ns/pF*
taccB	*Access time*	*-*	*-*	*-*	*-*	*ns*
tcycB	*Cycle time*	*-*	*-*	*-*	*-*	*ns*
tadsB	*Address setup time*	*0.50*	*1.18*	*2.41*	*2.84*	*ns*
tadhB	*Address hold time*	*0.47*	*1.11*	*2.26*	*2.66*	*ns*
twdsB	*data setup time*	*1.43*	*3.36*	*6.84*	*8.06*	*ns*
twdhB	*data hold time*	*0.63*	*1.48*	*3.01*	*3.55*	*ns*
tMEhB	*MEB pulse width at high*	*-*	*-*	*-*	*-*	*ns*
tMElB	*MEB pulse width at low*	*-*	*-*	*-*	*-*	*ns*
tpreB	*min precharge (after Write)*	*2.57*	*6.06*	*12.34*	*14.53*	*ns*
trdsB	*WE_B read pulse setup time*	*-*	*-*	*-*	*-*	*ns*
trdhB	*WE_B read pulse hold time*	*-*	*-*	*-*	*-*	*ns*
tmpwB	*min write (Write Cycle)*	*1.59*	*3.75*	*7.62*	*8.98*	*ns*
twpwB	*min write (Read Modif Write)*	*-*	*-*	*-*	*-*	*ns*
twrsB	*write setup time*	*-*	*-*	*-*	*-*	*ns*

Contentions :

tmcs	*ME (write port) setup time*	*0.95*	*2.23*	*4.55*	*5.36*	*ns*
tmch	*ME (write port) hold time*	*1.04*	*2.45*	*4.99*	*5.88*	*ns*
twcs	*WE_ (write port) setup time*	*-*	*-*	*-*	*-*	*ns*
twch	*WE_ (write port) hold time*	*-*	*-*	*-*	*-*	*ns*
tdcs	*data (write port) setup time*	*0.84*	*1.97*	*4.02*	*4.73*	*ns*
tdch	*data (write port) hold time*	*1.42*	*3.35*	*6.82*	*8.04*	*ns*

BIST or FIFO related :

mclh	*Addr. counter load dep. rise*	*-*	*-*	*-*	*-*	*ns/pF*

mchl	*Addr. counter load dep. fall*	-	-	-	-	*ns/pF*
tctr	*Addr. counter output time*	-	-	-	-	*ns*
tcsu	*Addr. counter setup time*	-	-	-	-	*ns*
tmlh	*ME hold time to CLRZ end*	-	-	-	-	*ns*
trecA	*CLRZ minimum low time*	-	-	-	-	*ns*
trecB	*CLRZ minimum low time*	-	-	-	-	*ns*

Fehlende Einträge stehen nicht für unbekannte, sondern für nicht relevante Werte. Dieses ist durch die konkrete Realisierung der beiden Ports in „nur Eingang“ und „nur Ausgang“ begründet. Die entsprechenden Zeiten für Schreib- bzw. Lesezugriffe sind daher nicht von Bedeutung.

8.5 ROM Megacells

Das in 8.6 erzeugte ROM ist einfach flankengesteuert. Mit einer steigenden Flanke auf dem Signal *ME* wird die Adresse von *ADD* übernommen und der Lesezyklus beginnt. Im selben Moment werden die Ausgangslatches geöffnet, sodas das Leseergebnis so früh wie möglich am Ausgang anliegt. Dadurch bedingt ist zeitweise ein nicht gültiger Wert am Ausgang sichtbar. Ein interner Taktmechanismus erkennt das Ende eines Lesezyklus und schließt die Latches am Ausgang des ROMs. Dadurch ist das Ergebnis ab sofort stabil. Der Adreßeingang wird wieder geöffnet. Die Abbildung 14 zeigt das Zeitverhalten des ROM.

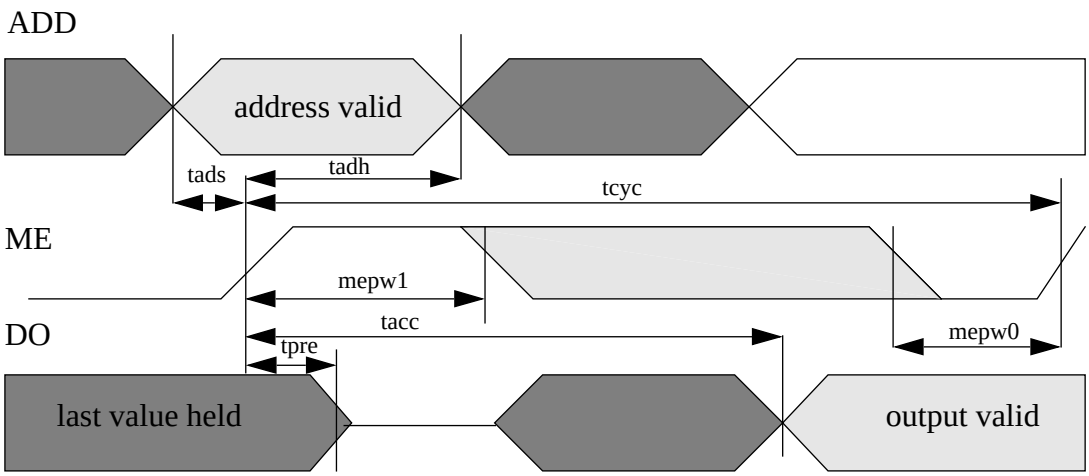


Abbildung 14: ROM Lesezyklus Zeitdiagramm

8.6 Erzeugen von ROM Megacells

Der Megacell Generator bietet mehrere Möglichkeiten für die Generierung von ROM. Beim Ausgang des ROMs kann zwischen einem gepufferten oder einem mit Tristatebuffer gewählt werden. Für die Verwendung im SiliconEar sind Tristatebuffer nicht nötig. Ein Signal OE zur Ansteuerung des Tristatetreibers am Ausgang entfällt somit. Die möglichen Konfigurationsparameter sind:

- Anzahl der Wörter: 9, ..., 16384
- Anzahl der Adreßbits: 4, ..., 14
- Anzahl der Bits pro Wort: 1, ..., 128
- Absolute Größe: 9, ..., 131072 (128k)
- Anzahl der Spalten: 4, ..., 512
- Anzahl der Reihen: 4, ..., 256

Im Folgenden wird der Dialog mit dem Megacell Genertor aufgelistet, der beim Erstellen des ROM für das SiliconEar geführt wurde. Benutzereingaben sind **fett** herforgehoben:

es2cgen rom romgfb

-I- Tool 'es2cgen' started.

-I- Spawning megacell generator.

generate_3.1_19Apr96 :

Please specify parameters for megacell rom in process ecpd07

BIST ? (yes/no) (<CR>=yes) : ***no***

Choose the functional configuration :

with tri-state outputs and OE pin (1)

with buffered outputs (no OE pin) (2)

---------> ***: 2***

Number of words? (9...16384): ***30***

Bits per word? (1...128): ***72***

```
*--------*----------*----------*-------------*---------------*-----------------*
| Choice | tacc(ns) | tcyc(ns) | rows * cols |  size (mm*mm) |area(mm2)|
*--------*----------*----------*-------------*---------------*-----------------*
|  1.    |  13.89   |  27.88   |   8 * 288   |  1.76 *  0.37 |           0.658 |
*--------*----------*----------*-------------*---------------*-----------------*
```

Only this aspect ratio is available

warning: number of physical words rounded to 32

additionnal words will be set to 0.

Expected pattern file is: rom.prg

Specify the address format :

binary (1)

decimal (2)

hexadecimal (3)

*---------> : **2***

Specify the data format :

binary (1)

hexadecimal (2)

*---------> : **1***

Generation completed

Die Datei *romgfb.prg* befindet sich im Verzeichnis */raid1/home/eis/siliconear/frankp/diplomarbeit/vhdl/generate/romgfb* und enthält die Filterkonstanten der 30 Kanäle. Die drei Konstanten von je 24 Bit wurden zu einem Wort von 72 Bit zusammengefaßt. Das ist sinnvoll, da sie immer zusammen ausgelesen werden. Anstelle drei Werte nacheinander auszulesen, können so alle Konstanten für einen Bandpaßkanal in einem Takt gelesen werden. Der Generator rundet die Anzahl der Worte auf 2^5 auf. Die beiden zusätzlichen Worte sind Nullvektoren. Die Parameter des so erzeugten ROMs sind der Datei *romgfb.ds* entnommen:

E S 2 ROM GENERATOR

Data Sheet for Block rom

Generator : generate_2.2_15Jun95

Date/Time : 16 Sep 16:40: 6 1997

Words	*:*	*30*
Bits per word	*:*	*72*
Rows	*:*	*8*
Columns	*:*	*288*
Output buffer	*:*	*buffer*
Pattern File	*:*	*rom.prg*
Address Format	*:*	*decimal*
Data Format	*:*	*binary*
Bist	*:*	*no*

Technology : ecpd07

Dimensions : 1757.50 x 374.40 (um x um) (without BIST)

Area : 0.66 sq. mm

Parameter	*Description*	*Min*	*Typ*	*Max*	*Mil*
tacc	*Access time from ME*	*3.02*	*6.84*	*13.89*	*16.41 ns*
toe	*Output enable time, from OE*	*-*	*-*	*-*	*- ns*
tod	*Output disable time,from OE*	*-*	*-*	*-*	*- ns*
macc	*Load dependent delay*	*0.10*	*0.22*	*0.44*	*0.52 ns/pF*
tcyc	*Cycle time, ME to ME*	*6.07*	*13.72*	*27.88*	*32.94 ns*
tpre	*Time to precharge*	*1.34*	*3.04*	*6.18*	*7.30 ns*
tads	*Address setup time*	*1.51*	*3.41*	*6.93*	*8.19 ns*
tadh	*Address hold time*	*0.20*	*0.45*	*0.92*	*1.09 ns*
mepw0	*ME pulse width low*	*0.90*	*2.04*	*4.14*	*4.89 ns*
mepw1	*ME pulse width high*	*0.90*	*2.04*	*4.14*	*4.89 ns*
power	*AC power, VDD = 5.0V, unloaded*		*8.58*		*mW/MHz*

Fehlende Einträge stehen nicht für unbekannte, sondern für nicht relevante Werte. Dieses ist durch die konkrete Realisierung des ROMs begründet. So fällt durch die Wahl eines gepufferten Ausgangs ohne Tristatetreiber das Signal *OE* weg. Dadurch haben die damit korrelierten Zeiten keine Bedeutung für das ROM.

9 Die neuen Blöcke im ASIC

Die Revision 2 des ASIC SiliconEar erhält aus zwei Gründen vier weitere Funktionsblöcke. Es handelt sich hierbei um den HWGR, einen Tiefpaßfilter, der BLB und einem Block zur Berechnung eines gleitenden Mittelwertes. Die Gründe sind folgende:

1) Es gelang, durch die handoptimierte Version der GFB (siehe 6. und 7.), diese mit einem weit geringeren Flächenverbrauch zu implementieren, als es die Projektgruppe erwartet hatte. Die GFB wird ca. 10 mm^2 Chipfläche von maximal 100 mm^2 einnehmen. Der verbleibende Platz kann zur Implementierung weiterer Funktionen verwendet werden.

2) Anstelle eines Flächenproblems kristallisierte sich jedoch ein neues im Design heraus: Die Übertragung der Daten von dem OIF an die DSP [42]. Die erzeugte Datenrate überschritt das Auffassungsvermögen der seriellen Schnittstellen nachfolgender Hardware. Es galt eine Reduzierung des Datenstroms bei gleichzeitiger Steigerung der Leistung der angeschlossenen Hardware zu erreichen.

HWGR und Tiefpaßfilter sind Bestandteile des Perzeptionsmodells, die ursprünglich auf dem Chip2, den NRGL, realisiert werden sollten. Wie die BLB werden diese Blöcke auf dem SiliconEar implementiert, um eine verbesserte Flächenausnutzung und eine geringere Datenrate an dem OIF zu erzielen (9.1 und 9.2).

Die Entscheidung, die BLB auf Chip1 unterzubringen, wurde bereits in der PG SiliconEar getroffen, mit dem Unterschied, daß dieser Block aufgeteilt wird. Der Kern, die CORDIC-Einheit, behält den Namen BLB (9.3). Die Mittelwertbildung über die Ausgaben des Kerns wird getrennt hierzu implementiert (9.4).

Der Block OIF wurde zwar bereits von der PG SiliconEar spezifiziert, wird aber dennoch in diesem Kapitel aufgeführt, da sich die Spezifikation des Verhaltens so weit änderte, daß keine Ähnlichkeit zur ersten Implementation mehr besteht (9.5). Das OIF Teilt sich in die HSS und die LSS auf, die in eigenen Unterkapiteln (9.5.2 und 9.5.3) abgehandelt werden.

Diese Blöcke sollen hier der Reihe nach kommentiert werden.

9.1 Halbwellengleichrichter

Ursprünglich realisierte der HWGR die Funktion aus Formel 7. Er simuliert das Verhalten der

$$\{\text{if in} < 0.0 \text{ then out} := 0.0 \text{ else out} := \text{in}\}$$

Formel 7

Haarzellen im Perzeptionsmodell, die Ausschläge lediglich in eine Richtung registrieren können. Eine negative Auslenkung im Bezug zu dieser Richtung wird ignoriert. Die AG MEDI

änderte die Spezifikation kurzfristig, als der nachfolgende Block, ein Tiefpaßfilter, definiert wurde. Für seine korrekte Funktion wurde die Schwelle von 0.0 auf 1.0e-5 erhöht. Die aktuelle Version arbeitet aus diesem Grunde nach Formel 8.

$$\{\text{if in} < 0.00001 \text{ then out} := 0.00001 \text{ else out} := \text{in}\}$$

Formel 8

Der HWGR ist ein registerloses Schaltnetz und benötigt somit kein Takt- und kein Resetsignal. Dadurch reduziert sich die Zahl der Signale an den Schnittstellen auf zwei. Diese Signale sind vom Format std_logic_vector(23 downto 0). *SlvIn* bildet die Eingangsschnittstelle, *Slvout* die Ausgangsschnittstelle.

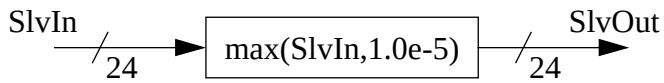


Abbildung 15: Blockschaltbild vom HWGR

9.2 Tiefpaßfilter

Dem HWGR schließt sich ein Tiefpaßfilter an. Dieser dient zur Einhüllendenbildung über die gleichgerichteten GFB-Signale. Der Tiefpaßfilter, oder auch Filter2, simuliert das Trägheitsverhalten der Haarzellen im Perzeptionsmodell. Die Stereozilien bewegen sich auf Grund ihrer Trägheit nicht beliebig schnell und lösen daher hohe Frequenzen schlechter auf als niedrige.

Der Filter2 ist ein Gammatone Filter vierter Ordnung mit einer Grenzfrequenz von 500 Hz. Er arbeitet nach dem selben Algorithmus wie die GFB, also nach Formel 1 aus Kapitel 6. Seine Funktion ist für eine Samplingfrequenz von 16,276 KHz definiert. Die Konstanten sind:

Real(b) = 0.641629
Imag(b) = 0.0
out_scale = 0.0329884

Nähere Informationen zum Tiefpaßfilter befinden sich unter *http://eis.informatik.uni-oldenburg.de/~sinalco/confidential/dokus.html* bzw. *http://eis.informatik.uni-oldenburg.de/~sinalco/confidential/medi/doku/tp_auswahl.ps*.

Es wurde angestrebt, die Struktur der GFB für den Filter 2 zu verwenden. Die AG MEDI suchte einige Wochen nach obigem, zu dieser Struktur passenden Filter. Das Ergebnis kann zwar mit der GFB-Struktur implementiert werden, würde aber einen Mehraufwand von 50% bedeuten. Imag(b) ist null und der Tiefpaß arbeitet demnach ausschließlich realwertig. Die Hälfte der GFB-Struktur würde bei ihrer Verwendung ausschließlich Null-Multiplikationen und Additionen durchführen. Es wurde daher entschlossen eine neue Strukturbeschreibung zu

erstellen. Dieses konnte aus Zeitgründen nicht geschehen, sodas lediglich eine Verhaltensbeschreibung im Verzeichnis *~frankp/dilplomarbeit/vhdl/rtl_filter2* existiert.

Im Übrigen wird von der AG MEDI über eine Verwerfung dieses Filters zugunsten eines anderen nachgedacht. An dieser Stelle ist noch auf eine Entscheidung zu warten.

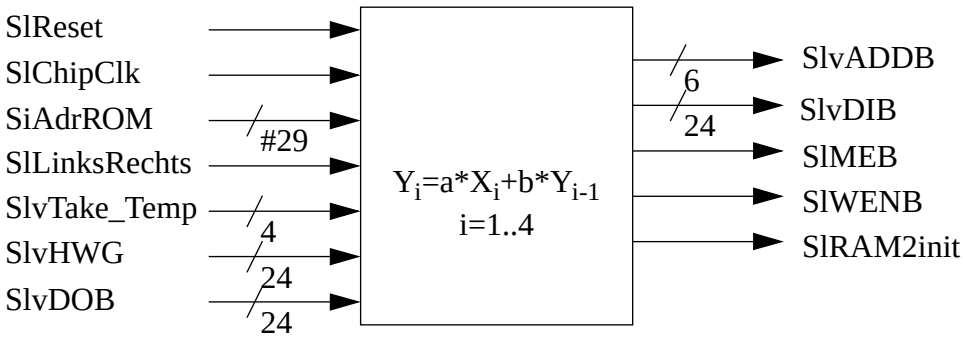


Abbildung 16: Blockschaltbild vom Filter2

Dennoch soll hier auf den derzeitigen Stand der Entwicklung eingegangen werden. Abbildung 16 zeigt die Signale der Ein- und Ausgangsschnittstelle. Zunächst sollen die Signale der Eingangsschnittstelle kommentiert werden:

- *SlChipClk* gibt einen Takt von 50 MHz vor.
- *SlReset* ist active low und synchron zu *SlChipClk*.
- Über den Bus *SlvHWGR* gelangen die Ergebnisse aus der GFB über den HWGR in den Filter.
- Die Gültigkeit von *SlvHWGR* wird mit einem Event auf dem vierbitigen Bus *SlvTake_Temp* signalisiert. Von Bedeutung für den Filter sind die Übergänge: „0110“ -> „0111“ (steht für den Realteil eines Ergebnisses) und „0111“ -> „1000“ (für den zugehörigen Imaginärteil).
- *SiAdrROM* gibt die Nummer des Kanals an, der zur Zeit des Events auf *SlvTake_Temp* auf *SlvHWGR* angelegt wurde.
- *SlLinksRechts* zeigt mit ’high’ die Berechnung des rechten und mit ’low’ die des linken Stereokanals an.
- Über *SlvDOB* gelangen zuvor im RAM gespeicherte Zwischenwerte nach einem Lesezugriff in den Filter zurück.

Hier die Signale der Ausgangsschnittstelle:

- *SlvADDB* gibt eine Adresse im RAM des Tiefpaßfilters an, auf welche ein Zugriff erfolgen soll.
- *SlWENB* ’high’ leitet einen Lesevorgang und ’low’ einen Schreibvorgang auf das RAM ein.

- Mit steigender Flanke auf *SlMEB* wird die Aktion (Lesen oder Schreiben) ausgeführt (sieh auch Abbildungen 12 und 13).
- *SlvDIB* leitet die zu schreibenden Werte in das RAM.
- Nach einem Reset gibt *SlRAM2init* mit 'high' zu erkennen, daß das RAM des Filter2 mit Null initialisiert worden ist.

Der Tiefpaßfilter schreibt seine Werte über den Kanal B in das RAM. Die nachfolgende Komponente, die LSS, bedient sich dieser Werte über den Port A des RAMs. Dieses macht ein entkoppeltes Arbeiten von Producer (Tiefpaßfilter) und Consumer (LSS) möglich. Der einzige kritische Moment ist die Zeitdauer eines Resets, wenn das RAM nicht initialisiert ist. Zu dieser Zeit dürfen von der LSS keine Lesezugriffe erfolgen. Dadurch daß die LSS erst nach steigender Flanke auf *SlRAM2init* gestartet wird, kann dieses vermieden werden.

9.3 BLB

Die binaurale Verarbeitung akustischen Signale durch das SiliconEar ermöglicht die Bestimmung der Richtung, aus der eine Schallwelle eintrifft. Diese Funktion wurde durch die Integration der BLB auf dem ASIC realisiert. Bei der Berechnung der Richtung werden zwei Eigenschaften genutzt:

Schallwellen die zur linken Seite eines Zuhörers eingehen, erreichen dessen rechtes Ohr aufgrund des verlängerten Weges und der endlichen Schallgeschwindigkeit später als das linke. Dieses äußert sich in einer Phasendifferenz zwischen den beiden, von dem Gehör aufgenommenen Signalen.

Zusätzlich hierzu empfängt das rechte Ohr eine im Bezug auf das Linke Ohr abgeschwächte (leisere) Schallwelle, weil es sich im „Hörschatten“ des Kopfes des Zuhörers befindet. Über die Bildung eines Betragsquotienten aus den beiden Signalen links und rechts werden weitere Informationen für das Richtungshören gewonnen.

E. Schmidt hat sich innerhalb einer Studienarbeit [43] mit diesem Thema befaßt, sodas die BLB aus dem Redesign seines Blocks ASS entstand. Die ASS wird im Folgenden Version1 der BLB genannt werden, während die BLB des Redesigns die Versionsnummer 2 erhält. Ein Redesign der ersten Version war aus drei Gründen notwendig:

1) In Kapitel 5 wird eine Änderung der Schnittstelle zwischen GFB und BLB beschrieben. Die BLB ist an das geänderte Verhalten der GFB anzupassen.

2) Durch vierfaches Unterabtasten der BLB-Ergebnisse soll eine Reduzierung der Datenrate um 75% erzielt werden. Dieses Vorgehen wurde in der ersten Version durch die Berechnung eines arithmetischen Mittels über die Ausgaben der BLB verhindert, da die Mittelwerte nur alle 128 Sample aktualisiert wurden.

3) Die Breite der Eingangsschnittstelle ist von E. Schmidt mit 16 Bit implementiert worden. Die GFB führt Berechnungen mit 24 Bit durch. Intern arbeitet die BLB mit Bitbreiten größer 24 Bit. Es ist nicht sinnvoll, die Daten von der GFB von 24 Bit auf 16 Bit zu kürzen, um dann wiederum Berechnungen mit Genauigkeiten größer 24 Bit durchzuführen.

Die Kapitel 9.3.1, 9.3.2 und 9.4 beschreiben das Vorgehen zu den drei genannten Punkten.

9.3.1 Schnittstellen der BLB

Der Version1 standen an der Ausgangsschnittstelle der GFB in einem überdimensionierten Register sämtliche Berechnungsergebnisse zu einem Stereosample zur Verfügung. Aus diesem Register bediente sich die erste Version selbständig mittels eines ebenfalls überdimensionierten Multiplexers. Register und Multiplexer sind in der zweiten Version nicht implementiert (siehe hierzu Kapitel 4). Die GFB berechnet in der zweiten Revision die Werte in einer für die BLB sinnvollen Reihenfolge, sodas nicht **sämtliche** Werte zu **einem** Zeitpunkt zur Verfügung stehen müssen. Die Wahl der zu verarbeitenden Werte wird somit nicht von der BLB, sondern von der GFB vorgegeben. Die Berechnungsreihenfolge entspricht dem Übertragungsprotokoll der HSS aus Kapitel 9.5.2. Abbildung 17 zeigt schematisch den Aufbau der Eingangsschnittstelle (zum Vergleich siehe auch die Abbildung 38 im Anhang A). Bei der Verrechnung eines Stereosample in der GFB werden pro Bandpaßkanal vier Werte erzeugt. Die BLB beobachtet

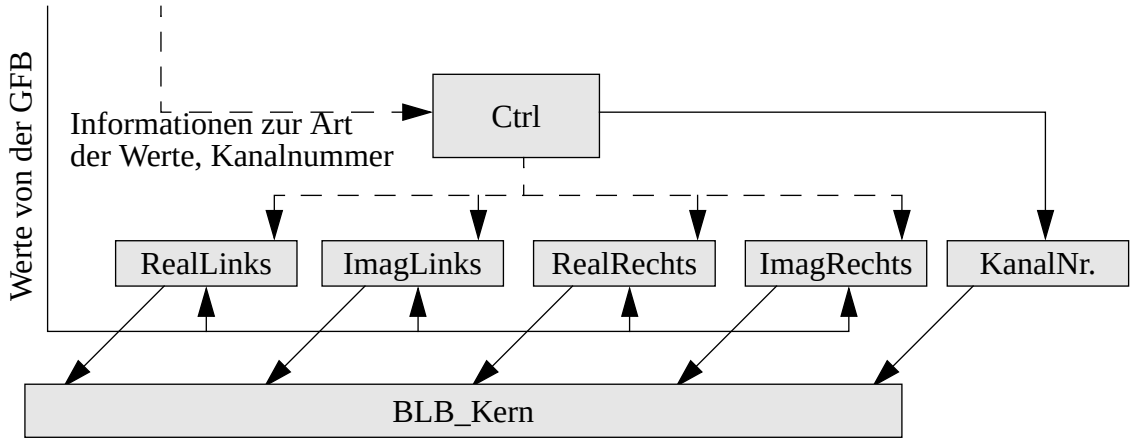


Abbildung 17: Skizze zur BLB Eingangsschnittstelle

den Ausgang der GFB und filtert vier zu einem Kanal gehörenden Werte aus dem Datenstrom aus und legt sie in entsprechenden Registern ab. Ist ein kompletter Satz vorhanden, wird das Quadrupel zusammen mit der aktuellen Nummer des Kanals zum Kern der BLB zur Berechnung der Phasendifferenz und des Betragsquotienten weitergeleitet. Die Ergebnisse werden an einen Filter zur Berechnung eines gleitenden Mittelwertes geleitet. Abbildung 18 zeigt die Signalnamen der BLB-Schnittstelle. Ihre Funktionen an der Eingangsschnittstelle sind:

- *SlChipClk* gibt einen Takt von 50 MHz vor.
- *SlReset* ist active low und synchron zu *SlChipClk*.

- *SlvGFB* bildet einen Datenbus zur Übermittlung von Werten der GFB zur BLB.
- Die Gültigkeit von *SlvGFB* wird durch Events auf *SlvTake_Temp* signalisiert. Von Bedeutung für die BLB sind die Übergänge: „0110“ -> „0111“ (steht für den Realteil eines Ergebnisses) und „0111“ -> „1000“ (für den zugehörigen Imaginärteil).
- *SiAdrROM* gibt die Nummer des Kanals an, der zur Zeit des Events auf *SlvTake_Temp* auf *SlvHWGR* angelegt wurde.
- *SlLinksRechts* zeigt mit ’high’ die Berechnung des rechten und mit ’low’ die des linken Stereokanals an.

Zur Ausgangsschnittstelle der BLB gehören die Signale:

- *SlvP* ist ein 24-bitiger Datenbus, über den eine Block zum Berechnen eines gleitenden Mittelwertes mit Phasendifferenzen beliefert wird.
- *SlvQ* ist das Äquivalent zu *SlvP* für ermittelte Betragsquotienten.
- *SlStart* erklärt die beiden Datenbusse *SlvP* und *SlvQ* mit einer steigenden Flanke für gültig.
- *SiKanal* identifiziert den zu einem Betragsquotienten und einer Phasendifferenz gehörenden Bandpaßkanal der GFB.

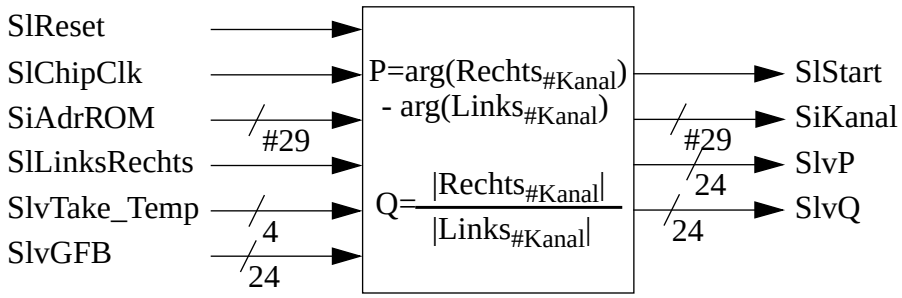


Abbildung 18: Blockschaltbild von der BLB

9.3.2 Anpassung der Bitbreiten

Den Kern der BLB bildet die so genannte CORDIC-Einheit. Sie wurde außerhalb der PG SiliconEar im Rahmen einer Studienarbeit [43] entwickelt. Aufgrund einer falschen Bitbreite an der Eingangsschnittstelle ist ein Redesign dieses Blocks durchzuführen. Nach Aussage von E. Schmidt wurde die Implementierung verifiziert und der erzeugte VHDL-Code ist synthetisierbar. Es ist daher effizient, die vorliegende Arbeit zu modifizieren und nicht den Kern neu zu implementieren. Aus zwei Gründen entstehen jedoch Probleme beim Verständnis des Quellcodes:

Die VHDL-Implementierung der CORDIC-Einheit hält sich nicht an das Stylesheet der PG SiliconEar. Weiterhin ist die CORDIC Einheit ein komplexes Modul.

Es wird die Möglichkeit nicht ausgeschlossen, daß E. Schmidt die Modifikation des Kerns vor-

nehmen wird. Daher und aus Zeitmangel soll eine Anpassung nicht in dieser Diplomarbeit vorgenommen werden. Es wurde lediglich eine Dummyfunktion generiert, welche ein Testen des ASIC erlaubt. Näheres zum Dummy ist unter Kapitel 12.2. zu finden.

9.4 Gleitende Mittelwertbildung

Für die Berechnung eines Mittelwertes aus den Phasendifferenzen und den Betragsquotienten wurde in der ersten Revision ein einfaches arithmetisches Mittel über 128 Werte gebildet. Der Nachteil dieses Verfahrens liegt auf der Hand: Ein gültiges Resultat liegt nur alle 128 Werte vor. Die Implementierung einer gleitenden Mittelwertbildung wurde von E. Schmidt in der Diplomarbeit „Entwurf einer integrierten Schaltung zur binauralen Verarbeitung frequenzgefilteter akustischer Signale“ [43] aufgrund der Vielzahl von Registern, die zum Vorhalten von Zwischenwerten nötig sind, verworfen. Aus zwei Gründen ist diese Entscheidung neu zu überdenken:

1) In [43], Kapitel 4.1 wurde für die Bildung des gleitenden Mittels ein IIR 4.Ordnung angegeben. Tatsächlich reicht ein Filter 1.Ordnung aus [44], sodas nur ein viertel der Zwischenwerte im Vergleich zu einem Filter 4.Ordnung vorzuhalten sind. Die Berechnungsvorschrift für den Filter1 gibt Formel 9 wieder (vergleiche mit Formel 1).

2) Anstelle von Registern kann das in Kapitel 8 vorgestellte RAM verwendet werden.

$$y[i] = (0,03 \cdot x[i]) + (0,97 \cdot y[i-1])$$

Formel 9

Wie bei der Implementierung des Blocks zur gleitenden Mittelwertbildung vorgegangen wurde, gibt das Kapitel 10 wieder. Abbildung 19 zeigt die Schnittstelle dieses Blocks, deren Signale im Folgenden beschrieben werden. Zunächst die Signale der Eingangsschnittstelle:

- *SlChipClk* gibt einen Takt von 50 MHz vor.
- *SlReset* ist active low und synchron zu *SlChipClk* (50 MHz).
- *SlvP* ist ein 24-bitiger Datenbus, über den der Block zum Berechnen eines gleitenden Mittelwertes mit Phasendifferenzen aus der BLB beliefert wird.
- *SlvQ* ist das Äquivalent zu *SlvP* für ermittelte Betragsquotienten.
- *SlStart* erklärt die beiden Datenbusse *SlvP* und *SlvQ* mit einer steigenden Flanke für gültig.
- *SiKanal* identifiziert den zu einem Betragsquotienten und einer Phasendifferenz gehörenden Bandbaßkanal der GFB.

Über die Ausgangsschnittstelle kann auf das RAM der gleitenden Mittelwertbildung zugegriffen werden. Die Funktion der Signale ist:

- *ADDA* gibt eine Adresse im RAM an, auf welche ein Lesezugriff erfolgen soll.

- Mit steigender Flanke auf *MEA* wird ein Lesezugriff auf die Adresse *ADDA* ausgeführt (sieh auch Abbildung 12).
- An *DOA* liegt das aus Adresse *ADDA* ausgelesene Wort nach einer definierten Verzögerungszeit auf einer steigenden Flanke von *MEA* an.
- *SlRAMreseted* zeigt nach dem Einleiten eines Resets mit 'high' den Abschluß des Initialisiervorgangs des RAMs mit Nullwerten an.

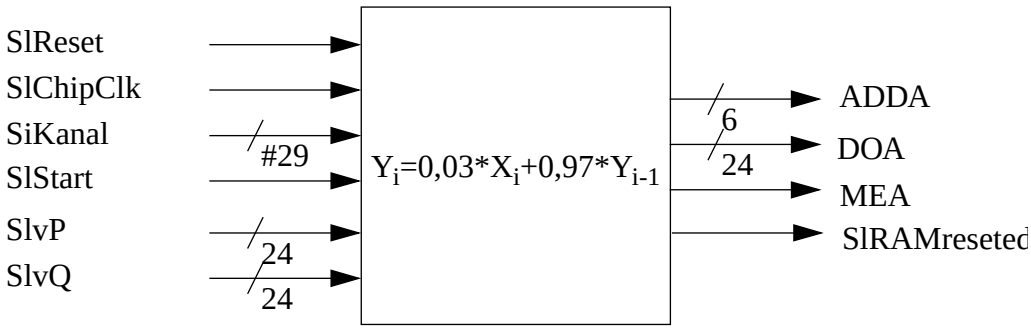


Abbildung 19: Blockschaltbild vom Filter1

9.5 Revision 2 des Output Interface

Die erste Version der OIF aus der Projektgruppe SiliconEar wurde vollständig verworfen. Es hatte sich herausgestellt, daß das in [42] vorgestellte Framekonzept nicht praktikabel ist. Die Entwicklung der zweiten Version mußte daher von Grund auf neu erfolgen. Zunächst mußte eine neue Systemumgebung spezifiziert werden, welche den Datenstrom verarbeiten kann. Zum einen sind das die NRGL und zum anderen ein neuer DSP von Texas Instruments, der DSP C60x. Der Anschluß dieser Komponenten findet jeweils über eine eigene Schnittstelle statt, welche parallel arbeiten. Beide Geräte, DSP und NRGL, können also gleichzeitig am SiliconEar betrieben werden, der DSP an der HSS und die NRGL an der LSS. Das Trennen des Stroms (siehe Abbildung 20) in zwei separate Pfade liegt in einem unterschiedlichen Verwendungszweck begründet.

Zum einen soll die Resynthese des Signals ermöglicht werden, um den ASIC z.B. als Hörhilfe nutzen zu können. Die HSS muß dazu in der Lage sein, die volle, von der GFB generierte Information zu übertragen. Die HSS wird in Kapitel 9.5.2 näher erörtert.

Zum anderen soll das Perzeptionsmodell (Kapitel 3.2) vollständig in Hardware implementiert werden. Den NRGL, die dem SiliconEar folgen, genügt $^1/_4$ der von der GFB generierten Sample, weil die Datenströme bereits von dem HWGR, dem Tiefpaßfilter und der BLB vorverarbeitet werden. Die LSS, die als Schnittstelle zwischen dem SiliconEar und den NRGL fungiert, überträgt daher mit einer verminderten Datenrate. Näheres zur LSS befindet sich in Kapitel 9.5.3.

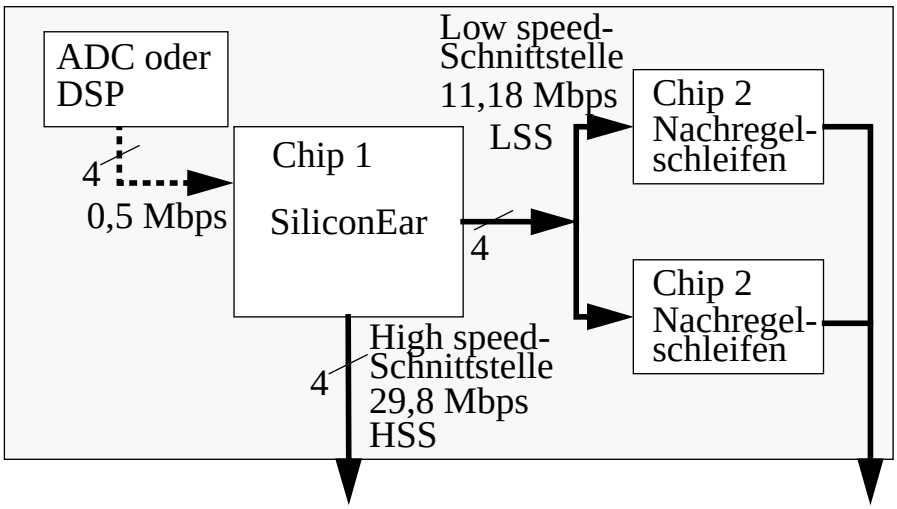


Abbildung 20: OIF aus globaler Sicht

Eine wesentliche Änderung im Vergleich zur ersten Version der OIF ist hervorzuheben: Die zweite OIF arbeitet nicht mit einem Handshakeprotokoll. Zuvor wurde eine korrekte Übernahme von Worten durch den DSP von der OIF kontrolliert. Die Leitung „Wait“ in Richtung DSP zum ASIC [42] wurde nicht implementiert. Somit kann ein Buffer-Overrun auf Seiten des DSP nicht erkannt werden. Das Panicsignal des SiliconEar verliert seine Funktion und wird ebenfalls nicht implementiert. Im Folgenden soll auf die beiden Schnittstellen eingegangen werden.

9.5.1 Schnittstellen des OIF

Abbildung 21 zeigt die Schnittstellen des OIF. Die Ausgangsschnittstellen HSS und LSS des OIF und somit auch die des ASIC werden in separaten Unterkapiteln 9.5.2 und 9.5.3 behandelt. An dieser Stelle erfolgt die Aufzählung der Eingangssignale:

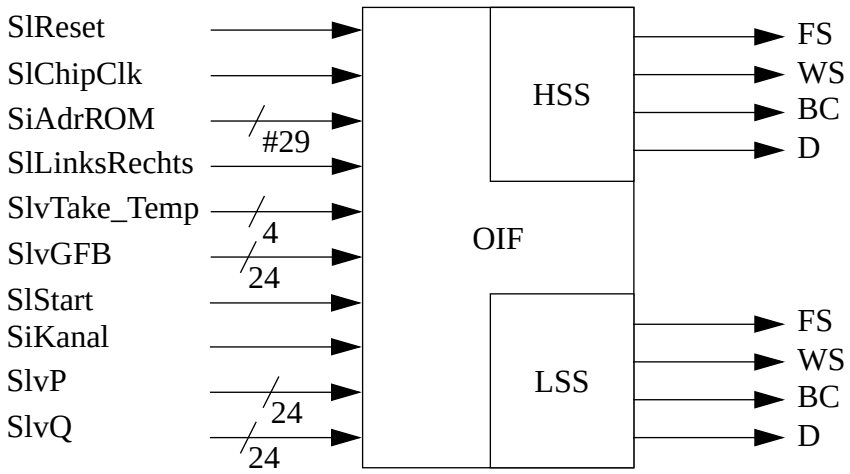


Abbildung 21: Blockschaltbild vom OIF

- *SlChipClk* gibt einen Takt von 50 MHz vor.
- *SlReset* ist active low und synchron zu *SlChipClk*.
- Über den Bus *SlvGFB* gelangen die Ergebnisse aus der GFB zum OIF.
- Die Gültigkeit von *SlvGFB* wird mit einem Event auf dem vierbitigen Bus *SlvTake_Temp* signalisiert. Von Bedeutung für die OIF sind die Übergänge: „0110“ -> „0111“ (steht für den Realteil eines Ergebnisses) und „0111“ -> „1000“ (für den zugehörigen Imaginärteil).
- *SiAdrROM* gibt die Nummer des Kanals an, der zur Zeit des Events auf *SlvTake_Temp* auf *SlvGFB* angelegt wurde.
- *SlLinksRechts* zeigt mit ’high’ die Berechnung des rechten und mit ’low’ die des linken Stereokanals an.
- *SlvP* ist ein 24-bitiger Datenbus, über den die OIF mit Phasendifferenzen aus der BLB beliefert wird.
- *SlvQ* ist das Äquivalent zu *SlvP* für ermittelte Betragsquotienten.
- *SlStart* erklärt die beiden Datenbusse *SlvP* und *SlvQ* mit steigenden Flanke für gültig.
- *SiKanal* identifiziert den zu einem Betragsquotienten und einer Phasendifferenz gehörenden Bandbaßkanal der GFB.

9.5.2 HSS

Die HSS dient als Schnittstelle zum DSP C60x. Dieser DSP von Texas Instruments arbeitet mit dem selben Protokoll wie der schon aus [42] bekannte DSP32C von AT&T. Die Übertragungsfrequenz beträgt 50 MHz. Abbildung 22 zeigt die Signalwechsel während einer Übertragung auf den vier Leitungen FS, WS, BC und D. Die Kommunikation erfolgt in eine Richtung, nämlich vom ASIC zum DSP. Auf D werden Daten seriell übertragen. Bei den drei übrigen Signalen handelt es sich um synchronisierende Clocks. Sie werden aus der Chipclockfrequenz von 50 MHz mittels Division durch Zweierpotenzen generiert.

- FS: Framesync dient der Synchronisation auf Frameebene. Ein Frame besteht aus 120 Worten (zum Protokoll siehe Tabelle 3). Mit fallender Flanke wird die Übertragung eines Frame gestartet. Die Frequenz könnte somit $^1/_{120}$ der von WS betragen (26,04KHz). Der Samplingtakt des SiliconEar von 16,276 KHz ist jedoch einzuhalten. Es entsteht eine „Übertragungspause“ von 23,04 ns, die zwischen zwei Frames einzuhalten ist.
- WS: Wordsync dient der Synchronisation auf Wortebene. Ein Wort besteht aus 16 Bits. Mit einer fallenden Flanke wird eine Übertragung initiiert. Der Takt des Signals beträgt demnach maximal $^1/_{16}$ der Bitclock. Das entspricht einer Frequenz von 3,125 MHz. Dieses ist jedoch nicht die implementierte Frequenz. Die im vorherigen Punkt genannten 23,04 ns werden gleichmäßig zwischen den einzelnen Worten eines Fra-

mes verteilt. Dadurch wird der GFB für die „Just-In-Time-Berechnung“ von Werten mehr Zeit zur Verfügung gestellt. Die implementierte Frequenz für WS ist demnach 1,95 MHz.

•BC: Die Bitclock dient der bitweisen Synchronisation. Die Frequenz entspricht der Chipclock, also 50 MHz. Mit einer fallenden Flanke wird ein Bit auf D übertragen. Daraus resultiert eine max. Datenrate von 47,68 Mbps, die jedoch nicht erreicht wird (siehe hierzu unter FS und WS).

Word 0	Word 1	Word 2	Word 3	...	Word 116	Word 117	Word 118	Word 119
R-l-0	I-l-0	R-r-0	I-r-0	...	R-l-29	I-l-29	R-r-29	I-r-29

R/I: Realteil/Imaginärteil
l/r: linke/rechte Seite
0-29: Frequenzkanalnummer

Tabelle 3: Protokoll der HSS

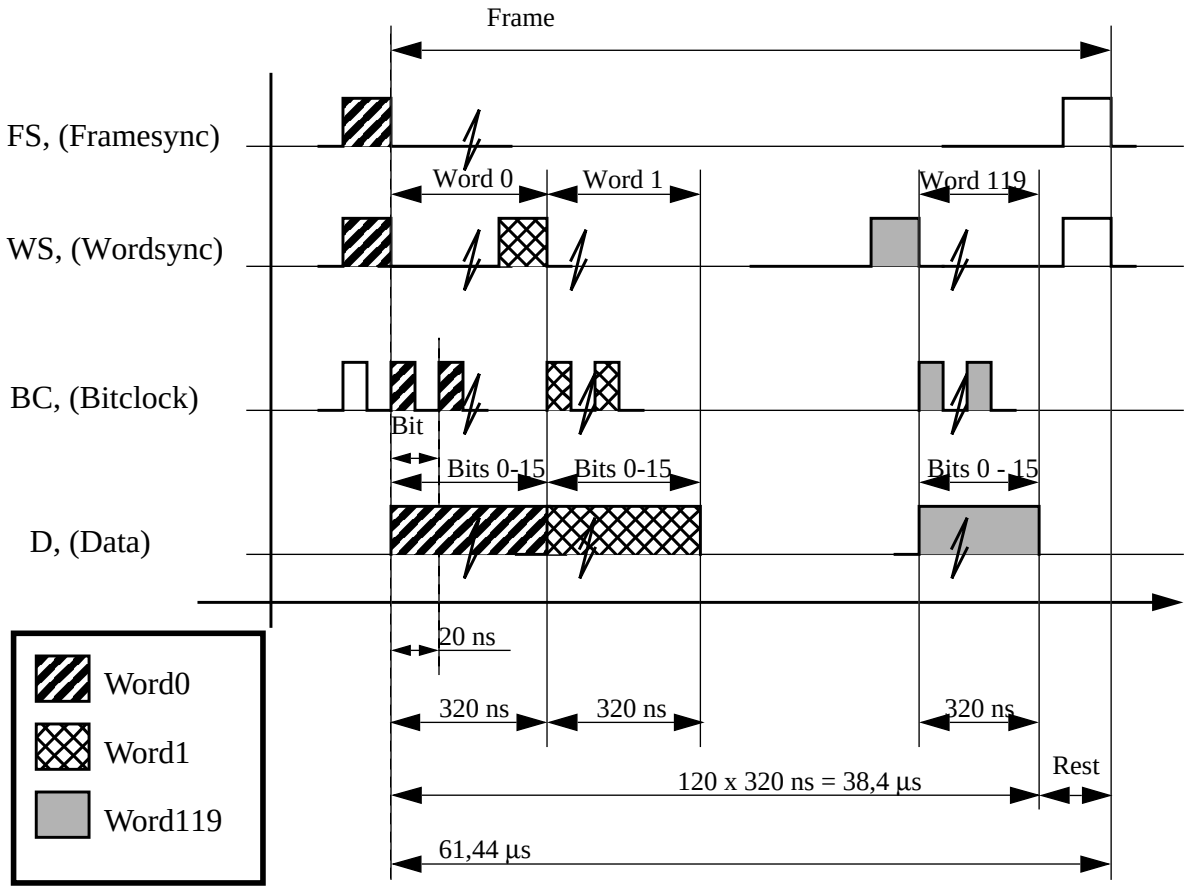


Abbildung 22: Protokoll der HSS

9.5.3 LSS

An die LSS werden die NRGL angeschlossen. Das Protokoll wurde dem eines DSP angeglichen. Dieses ist daher sinnvoll, da die NRGL sämtliche Signale an einen nachfolgenden DSP weiterleiten, nachdem sie „freie Plätze“ in einem Frame (siehe Tabelle 4) durch Werte substituiert haben. Die LSS besitzt die gleichen Leitungen, mit den selben Funktionen wie die HSS. Lediglich die Taktraten sind unterschiedlich. Abbildung 23 zeigt die Abhängigkeiten bei den Signalübergängen.

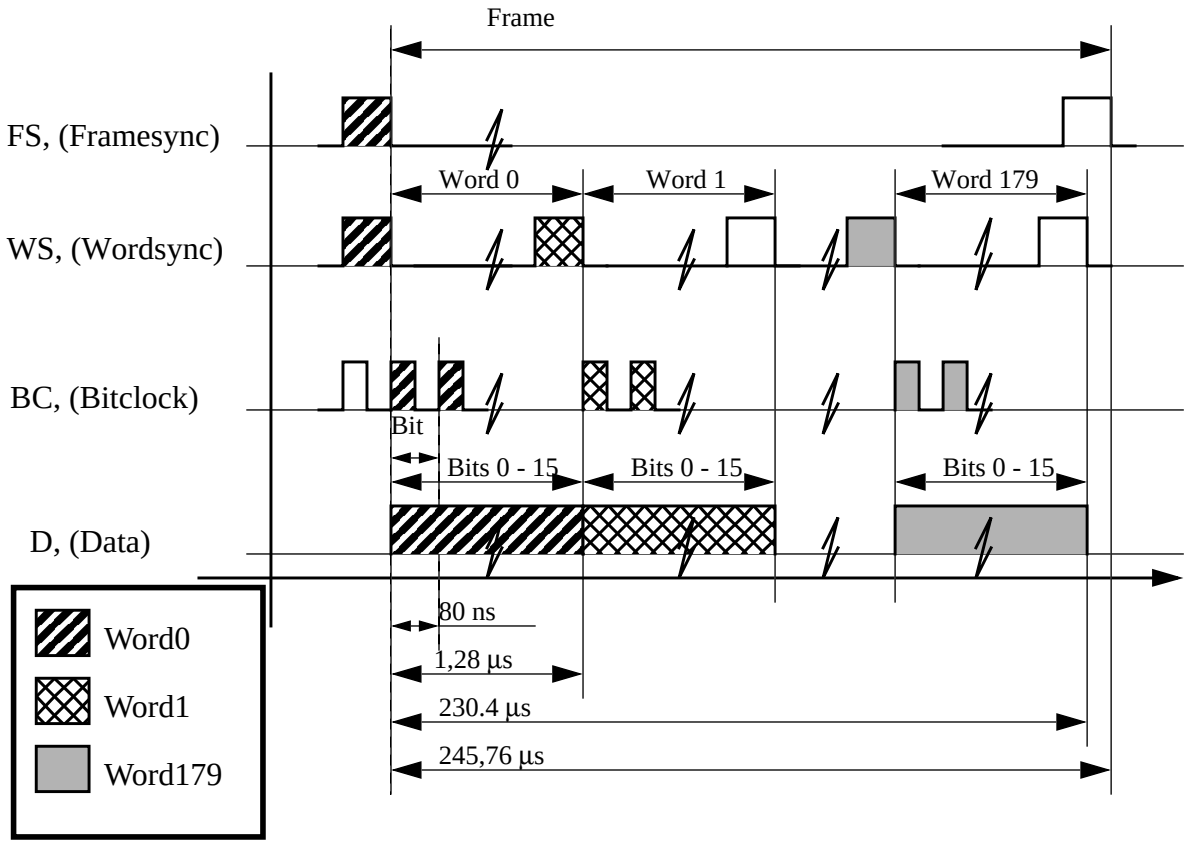


Abbildung 23: Protokoll der LSS

Auf D erfolgt die serielle Bitübertragung. Es verbleiben die drei Clocks:

- FS: Framesync dient der Synchronisation auf der Frameebene. Mit einer fallenden Flanke wird das erste Bit des ersten Wortes eines Frames übermittelt. Dieses besteht aus 180 Worten (siehe dazu Tabelle 4). Die maximale Frequenz beträgt somit 4,34 KHz. Es ist jedoch der Takt von $^1/_4$ der Samplingrate (vierfache Unterabtastung) einzuhalten. Daraus folgt für FS eine Taktfrequenz von 4,069 KHz. Es Entsteht eine „Übertragungspause“ von 15,36 ns, die zwischen zwei Frames einzuhalten ist.
- WS: Wordsync dient der Synchronisation auf Wortebene. Mit einer fallenden Flanke auf WS wird das erste Bit eines 16Bit-Wortes übertragen. Die maximale Frequenz beträgt somit 781,25 KHz. Ein Zeitüberschuß von 15,36 ns (siehe unter FS) wird jedoch zwi-

schen den einzelnen Wortübertragungen verteilt, sodas die implementierte Frequenz für WS 732,421 KHz beträgt.

- BC: Synchron zur Bitclock werden die Bits der vierfach unterabgetasteten Werte übertragen. Die Frequenz ist $^1/_4$ der HSS Bitclock, also 12,5 MHz. Mit jeder fallenden Flanke wird ein Bit gesendet. Daraus ergibt sich eine maximale Datenrate von 11,92Mbps. Diese kann in der Praxis jedoch nicht erreicht werden (siehe FS und WS).

Die Worte eines Frames werden kurz vor dem Versenden aus dem RAM des Filter1 oder Filter2 (gleitende Mittelwertberechnung und Tiefpaßfilter) ausgelesen. Da sich das Versenden eines kompletten Frames über einen Zeitraum erstreckt, in dem die GFB vier Stereosample verarbeitet, sind die Worte des Frames nicht zu einem einzigen Stereosample gehörig. Seien S_0 bis S_4 Stereosample, die in dieser Folge von dem IIF eingelesen werden. Werde S_1 zu dem Zeitpunkt von der GFB bearbeitet, an dem die LSS einen neues Frame beginnt, FS also eine fallende Flanke aufweist. Dann sind Word0 und Word1 aus S_0 errechnet worden, da die neuen Werte zu S_1 zunächst von der GFB, BLB und den beiden Filtern zu bearbeiten sind. Aufgrund von Verzögerungszeiten stehen die alten Werte von S_0 beim Zugriff noch im RAM. Word2 bis Word61 werden aus S_1 erzeugt, da diese Ergebnisse rechtzeitig vor der Übertragung ermittelt werden können. Word62 bis Word121 werden aus S_2, Word122 bis Word177 aus S_3 und Word178 und Word179 sowie Word0 und Word1 des nächsten Frames aus S_4 errechnet. Dieses Verhalten ist unproblematisch, da es bei der Verarbeitung der Daten durch die NRGL ausschließlich auf den korrekten Abstand der Sample zueinander ankommt. Das soll z.B. heißen: Word122 bis Word177 werden immer aus Stereosampeln der Folge S_i mit $i \in \{4 \cdot x + n | \forall x \in \mathbf{N}_0\}$, wobei n=3 ist, generiert. Für das Intervall [178 ..179] und [0 ..1] gilt n=0. Für Intervall [2 ..61] gilt n=1 und für [62 bis 121] n=2.

Word 0	Word 1	Word 2	Word 3	Word 4	Word 5	...	Word 174	Word 175	Word 176	Word 177	Word 178	Word 179
A-l-0	DL-0	leer	A-r-0	DP-0	leer	...	A-l-29	DL-29	leer	A-r-29	DP-29	leer

A: Amplitudenwert (4-fach unterabgetastet)
DL: Amplitudenquotient
DP: Phasendifferenz
l/r: linke/rechte Seite
0-29: Frequenzkanalnummer

Tabelle 4: Protokoll der LSS

Achtung: Die Intervalle werden sich verschieben!

Das heißt: Wenn nach Implementation der BLB, der gleitenden Mittelwertbildung und dem Tiefpaßfilter auf Gateebene Verzögerungszeiten bekannt sind, werden die Worte erst zu späteren Zeitpunkten berechnet sein. So wird zum Beispiel Word122 mit Sicherheit im Intervall von S_2 liegen. Wie gravierend die Verschiebungen sind, wie groß also die Verzögerungszeiten der Blöcke sein werden, kann zu diesem Zeitpunkt nicht erkannt werden. Probleme bei der Verarbeitung der Werte in den NRGL entstehen nicht. Es ist sichergestellt, daß ein Wordi (für iε[0..179]) immer im selben Intervall steht, also daß die Folge eines Wordi immer einen Abstand von vier besitzt. Lediglich der Wert von n kann sich noch ändern. Die Abbildung 24 versucht die letzten beiden Absätze zu visualisieren.

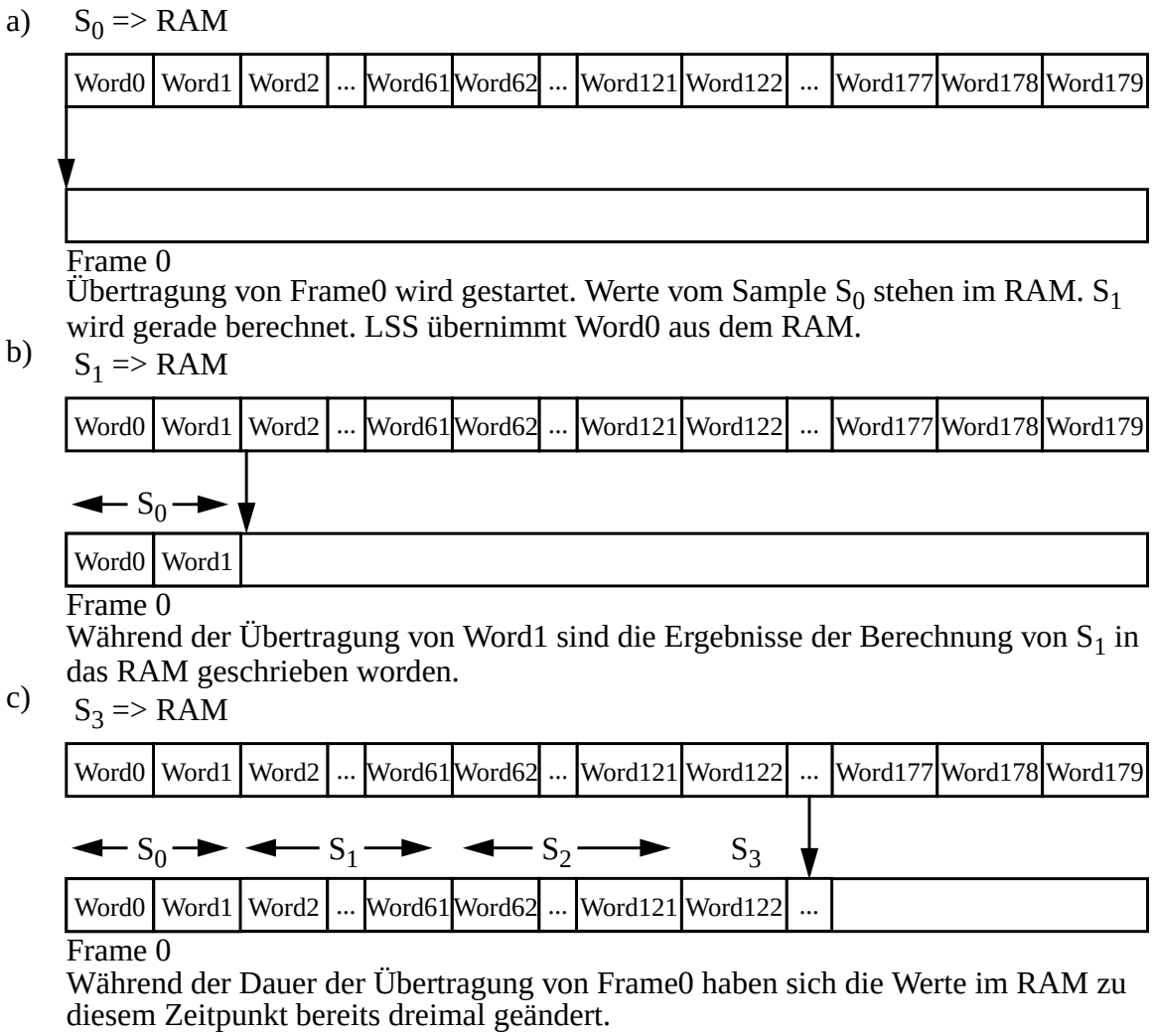


Abbildung 24: Details im Frameformat der LSS

10 Analyse und Implementierung der gleitenden Mittelwertbildung

Die Spezifikation der gleitenden Mittelwertbildung durch die AG MEDI fand noch rechtzeitig statt, um innerhalb dieser Arbeit analysiert und implementiert werden zu können. Es handelt sich um einen realwertigen IIR-Filter erster Ordnung. Die Berechnungsvorschrift ist im vorangegangenen Kapitel unter Formel 9 zu finden. In diesem Abschnitt sollen Überlegungen zum Speicher- (10.1), Zeit- (10.2) und Platzverbrauch (10.3) der Implementierung angestellt werden.

10.1 Analyse des Speicherbedarfs

Der Filter besitzt zwei Konstanten, die Eingangsskalierung dfir := 0,03 und die Filterkonstante diir := 0,97. Beide Werte sind mit einer Genauigkeit von 24 Bit vorzuhalten. Das ergibt 48 Bit konstante Daten, die keine Implementation von ROM erforderlich machen.

Bei den variablen Werten verhält es sich anders. Aus Formel 9 wird ersichtlich, daß jeweils der letzte Wert einer Berechnung vorzuhalten ist. Zu berechnen sind die Mittelungen über die Phasendifferenzen und Betragsquotienten der 30 GFB-Kanäle. Das ergibt 60 Worte, die mit einer Genauigkeit von 24 Bit vorzuhalten sind. Für die gleitende Mittelwertbildung ist somit eine RAMgröße von 1,44 KBit erforderlich.

Mit dem Megacell Generator wurde ein DPR der Größe 64 x 24 Bit generiert. Die Anzahl von 64 Worten folgt aus dem Aufrunden der benötigten Zahl 60 zur nächsten Zweierpotenz. Im Folgenden sind die technischen Daten zum generierten RAM abgebildet:

Generator : generate_3.1_19Apr96
Date/Time : 12 Dec 15:00:14 1997

Port A		
	Configuration :	*readOnly*
	Words :	*64*
	Bits per word :	*24*
Port B		
	Configuration :	*read/write*
	Words :	*64*
	Bits per word :	*24*
Aspect Ratio		
	Rows :	*32*
	Columns :	*48*
Bist :		*no*

Technology : ecpd07
Dimensions : 1252.40 x 897.20 (um x um) (without BIST)

Area : 1.12 sq. mm

BIST Number of equivalent nand2 gates : 0.00

---

powerA AC power, portA, VDD=5.0V, unloaded 1.83 mW/MHz

powerB AC power, portB, VDD=5.0V, unloaded 1.98 mW/MHz

---------- page 1/2 ----------

ES2 DPRAM GENERATOR Generator : generate_3.1_19Apr96

Block : dprblb Date/Time : 12 Dec 15:00:14 1997

Parameter	*Description*	*Min*	*Typ*	*Max*	*Mil*
Port A :					
mdlhA	*Load dependant rise time*	*0.24*	*0.56*	*1.14*	*1.34 ns/pF*
mdhlA	*Load dependant fall time*	*0.19*	*0.44*	*0.90*	*1.06 ns/pF*
taccA	*Access time*	*1.71*	*4.02*	*8.18*	*9.64 ns*
tcycA	*Cycle time*	*3.09*	*7.28*	*14.82*	*17.47 ns*
tadsA	*Address setup time*	*0.51*	*1.19*	*2.43*	*2.86 ns*
tadhA	*Address hold time*	*0.46*	*1.07*	*2.19*	*2.58 ns*
tMEhA	*MEA pulse width at high*	*0.51*	*1.20*	*2.44*	*2.87 ns*
tMElA	*MEA pulse width at low*	*0.51*	*1.20*	*2.44*	*2.87 ns*
twdsA	*data setup time*	*-*	*-*	*-*	*- ns*
twdhA	*data hold time*	*-*	*-*	*-*	*- ns*
tpreA	*min precharge (after Write)*	*-*	*-*	*-*	*- ns*
trdsA	*WE_A read pulse setup time*	*-*	*-*	*-*	*- ns*
trdhA	*WE_A read pulse hold time*	*-*	*-*	*-*	*- ns*
tmpwA	*min write (Write Cycle)*	*-*	*-*	*-*	*- ns*
twpwA	*min write (Read Modif Write)*	*-*	*-*	*-*	*- ns*
twrsA	*write setup time*	*-*	*-*	*-*	*- ns*
Port B :					
mdlhB	*Load dependant rise time*	*0.24*	*0.56*	*1.14*	*1.34 ns/pF*
mdhlB	*Load dependant fall time*	*0.19*	*0.44*	*0.90*	*1.06 ns/pF*
taccB	*Access time*	*1.71*	*4.02*	*8.18*	*9.64 ns*
tcycB	*Cycle time*	*3.09*	*7.28*	*14.82*	*17.47 ns*
tadsB	*Address setup time*	*0.51*	*1.19*	*2.43*	*2.86 ns*
tadhB	*Address hold time*	*0.46*	*1.07*	*2.19*	*2.58 ns*
twdsB	*data setup time*	*0.65*	*1.53*	*3.11*	*3.67 ns*
twdhB	*data hold time*	*0.51*	*1.21*	*2.46*	*2.90 ns*

tMEhB	*MEB pulse width at high*	*0.51*	*1.20*	*2.44*	*2.87 ns*
tMElB	*MEB pulse width at low*	*0.51*	*1.20*	*2.44*	*2.87 ns*
tpreB	*min precharge (after Write)*	*0.99*	*2.34*	*4.75*	*5.60 ns*
trdsB	*WE_B read pulse setup time*	*0.00*	*0.00*	*0.00*	*0.00 ns*
trdhB	*WE_B read pulse hold time*	*0.00*	*0.00*	*0.00*	*0.00 ns*
tmpwB	*min write (Write Cycle)*	*0.95*	*2.24*	*4.57*	*5.38 ns*
twpwB	*min write (Read Modif Write)*	*1.09*	*2.56*	*5.21*	*6.14 ns*
twrsB	*write setup time*	*0.00*	*0.00*	*0.00*	*0.00 ns*
Contentions :					
tmcs	*ME (write port) setup time*	*0.62*	*1.46*	*2.97*	*3.50 ns*
tmch	*ME (write port) hold time*	*0.70*	*1.64*	*3.35*	*3.94 ns*
twcs	*WE_ (write port) setup time*	*0.63*	*1.49*	*3.02*	*3.56 ns*
twch	*WE_ (write port) hold time*	*0.70*	*1.64*	*3.35*	*3.94 ns*
tdcs	*data (write port) setup time*	*0.22*	*0.51*	*1.04*	*1.23 ns*
tdch	*data (write port) hold time*	*0.95*	*2.24*	*4.57*	*5.38 ns*
BIST or FIFO related :					
mclh	*Addr. counter load dep. rise*	*-*	*-*	*-*	*- ns/pF*
mchl	*Addr. counter load dep. fall*	*-*	*-*	*-*	*- ns/pF*
tctr	*Addr. counter output time*	*-*	*-*	*-*	*- ns*
tcsu	*Addr. counter setup time*	*-*	*-*	*-*	*- ns*
tmlh	*ME hold time to CLRZ end*	*-*	*-*	*-*	*- ns*
trecA	*CLRZ minimum low time*	*-*	*-*	*-*	*- ns*
trecB	*CLRZ minimum low time*	*-*	*-*	*-*	*- ns*

---------- *page 2/2* ----------

Port B des DPR ist zum Schreiben und Lesen konfiguriert. Ausschließlich über diesen Port kommuniziert der Filter mit dem RAM. Auf Port A werden lediglich die Lesezugriffe von der LSS auf das RAM ausgeführt. Dabei arbeiten beide Ports unabhängig voneinander. Dieses hat den Vorteil, daß Filter und LSS das RAM nutzen können, ohne daß eine Synchronisation zwischen diesen Blöcken erforderlich wäre. Allein bei der Durchführung eines Resets hat die LSS auf den Zeitpunkt zu warten, an dem das RAM vom Filter initialisiert werden konnte. Dieses Ereignis wird der LSS über das Signal *SlRAMreseted* mitgeteilt.

Die Implementation des DPR als Schnittstelle zwischen Filter und LSS erbringt einen hohen Modularitätsgrad. Die LSS kann ausgetauscht werden, ohne daß Änderungen am Block zur gleitenden Mittelwertberechnung erfolgen müssen. Beim Testen können die Komponenten leicht durch Testbenches ersetzt werden.

10.2 Analyse der Operationen

Eine Analyse der Operationen soll klären, ob sämtliche Berechnungen des gleitenden Mittels innerhalb der Samplingfrequenz von 16,276 KHz durchgeführt werden können. Die Frage kann guten Gewissens bereits im Vorfeld mit „ja“ beantwortet werden, weil es sich bei dem Filter um **einen** IIR **erster** Ordnung handelt. Die Berechnung aller **30** IIR **vierter** Ordnung der GFB kann mit der Samplingfrequenz durchgeführt werden und diese sind um ein vierfaches komplexer. Da beide Blöcke absolut parallel zueinander arbeiten, sollte ein genügend großer Zeitraum zur Verfügung stehen.

Die GFB benötigt laut Abbildung 11 für die Berechnung eines Kanals 47 Takte bei 50 MHz Systemtakt. Erst wenn ein linker und ein rechter Kanal mit gleicher Frequenzkanalnummer berechnet wurden, kann die BLB Werte an das gleitende Mittel liefern. Da diese Vorgänge, wie schon oben genannt, parallel geschehen können, stehen der Mittelwertbildung 94 Takte für die Berechnung eines Phasendifferenz- und Betragsquotientwertes zur Verfügung. Pro Wert ergibt das eine maximale Verzögerungszeit von 940 ns, die nicht überschritten werden darf.

Wie aus Formel 9 ersichtlich sind drei Operationen, eine Addition und zwei Konstantenmultiplikationen, zur Berechnung des IIR erforderlich. Das ergibt eine maximale Verzögerungszeit von 313 ns pro Operation. Diese Zeitspanne ist mehr als ausreichend, da selbst die langsamsten Komponenten der Designware Library nicht mehr als ca. 100 ns Verzögerungszeit besitzen.

10.3 Implementierung der Konstantenmultiplizierer

Bei der Implementierung kann, als Folge von der in 10.2 errechneten maximalen Verzögerungszeit, das Augenmerk gänzlich auf die Optimierung bezüglich der Chipfläche gelegt werden. Der Parameter Zeit ist für dieses Design unkritisch. Der einfache Multiplizierer *mult(csa)* aus der DW02_Library mit einer Verzögerungszeit von 78,19 ns (siehe Tabelle 1) könnte alle Multiplikationen der gleitenden Mittelwertberechnung durchführen. Er verbraucht dabei eine Chipfläche von ca. 4.927.088 Squares. Das entspricht bei der verwendeten 0,7 µ-Technologie in etwa einem Platz von 2,41 mm^2.

Da es sich bei den aufgelisteten Komponenten der DW02_Library um Variablenmultiplizierer handelt, sollten sie nicht notwendigerweise Verwendung finden. Die Multiplikationen werden ausschließlich mit zwei konstanten Werten durchgeführt. Konstantenmultiplizierer sind somit die bessere Lösung, da sie in der Regel eine geringere Fläche als Variablenmultiplizierer belegen.

Konstantenmultiplizierer werden in den Designware Libraries nicht angeboten. Es wurde daher die Entscheidung getroffen, diese manuell als Shift&Add-Multiplizierer zu realisieren. Das folgende Kapitel handelt vom Aufbau eines solchen Multiplizierer.

10.3.1 Shift&Add-Multiplizierer

Shift&Add-Multiplizierer verwirklichen die Berechnung einer Multiplikation nach dem in Abbildung 25 visualisierten Algorithmus. Es ist zu erkennen, daß eine Multiplikation von zwei n-stelligen Binärzahlen durch n-1 Additionen realisiert werden kann. Die Architektur eines solchen variablen Shift&Add-Multiplizierers kann wie in Abbildung 26 gezeigt implementiert werden. Die Zahl der notwendigen Additionen reduziert sich mit jeder Null-Bitstelle im ersten Multiplikanten. Da die Multiplikation kommutativ ist, ist es effizient, denjenigen Wert als ersten Multiplikanten zu definieren, der die meisten Null-Bitstellen besitzt.

Konstante K := 100010
Variable X := $x_5 ... x_0$ mit $x_i \in [0,1]$

K mult X = 1 0 0 0 1 0 mult $x_5 ... x_0$

	x_5	x_4	x_3	x_2	x_1	x_0						
+	0	0	0	0	0	0						
+		0	0	0	0	0	0					
+			0	0	0	0	0	0				
+				x_5	x_4	x_3	x_2	x_1	x_0			
+					0	0	0	0	0	0		
	y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0

<=>

K mult X = 1 0 0 0 1 0 mult $x_5 ... x_0$

	x_5	x_4	x_3	x_2	x_1	x_0						
+						x_5	x_4	x_3	x_2	x_1	x_0	
	y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0

Abbildung 25: Beispiel Binärmultiplikation

Ist einer der Multiplikanten konstant, so kann dessen Wert fest in die Struktur des Multiplizierers implementiert werden. Additionen können wegfallen, wenn die Konstante Null-Bitstellen besitzt. Dieses vereinfacht und beschleunigt die Multiplikation.

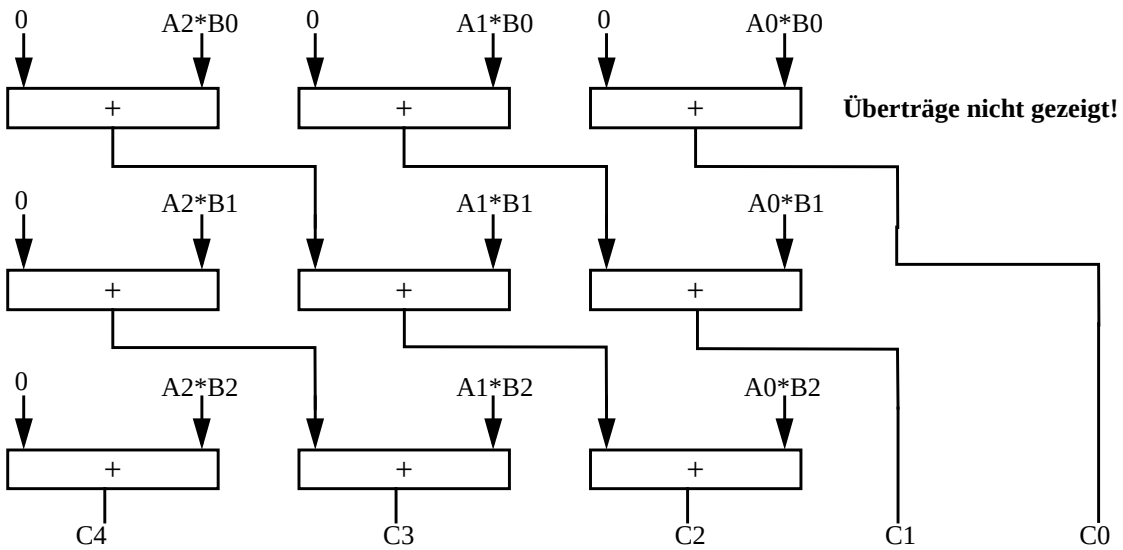


Abbildung 26: Architektur Shift&Add-Multiplizierer aus [18]

10.3.2 dfir- und diir-Multiplizierer

Für die Implementierung der Konstantenmultiplizierer sind die Dezimalzahlen dfir und diir in ein binäres Zweierkomplement zu konvertieren. Die Werte sind:

- dfir = $0{,}03_{10} = 000000111101011100001010_2$
- diir = $0{,}97_{10} = 011111000010100011110101_2$

Der Wert der Binärzahl im Zweierkomplement ergibt sich aus der Formel 10 nach [18]. Dabei ist auf das richtige Format zu achten. Das MSB steht an linker Stelle.

$$Wert_{10} = -b_{23} \cdot 2^0 + \sum_{i=1}^{23} b_{23-i} \cdot 2^{-i}$$

Formel 10

dfir besitzt 14 Null-Bitstellen. Es sind somit neun 48Bit-Additionen für die Realisierung der Funktion mult_dfir(b_{23},...b_0) erforderlich. Für die Funktion mult_diir(b_{23},...b_0) sind, analog ermittel, zwölf 48Bit-Additionen zu berechnen.

Im Folgenden ist der Algorithmen zur Berechnung von mult_dfir() in einem Pseudocode angegeben:

```
function mult_dfir(b[23..0]);
   SHIFT[47..42] := NULL;
   SHIFT[41..18] := b[23..0];          **=>Multiplikation von Bmit Bit 17 von dfir
   SHIFT[17..0] := NULL;
   TEMP := SHIFT;
   shift_right(SHIFT,1);               **Verschiebt Bits von SHIFT eine Stelle
                                       **=> Multiplikation von Bmit Bit 16 von dfir
   TEMP := TEMP + SHIFT;               **Teilprodukt eins wird zum zweiten addiert
   shift_right(SHIFT,1);               **=> Multiplikation von Bmit Bit 15 von dfir
   TEMP := TEMP + SHIFT;               **Teilprodukt drei wird aufaddiert
   ...                                 **analog
   shift_right(SHIFT,2);               **=> Multiplikation von Bmit Bit 1 von dfir
return(TEMP + SHIFT);                  **Rückgabe des Multiplikationsergebnisses
```

Die Berechnung von mult_dfir() erfolgt analog. Die Funktionen können ausschließlich positive Werte verarbeiten. Negative Zahlen sind vor dem Funktionsaufruf mit -1 zu multiplizieren. Das Ergebnis der Funktionen ist entsprechend erneut umzuwandeln. Das Negieren von Binärzahlen im Zweierkomplement erfolgt durch „Kippen“ sämtlicher Bitstellen links der rechtesten eins (siehe [18]). Beispiel: $-0{,}375_{10} = \mathbf{110}100_2$,negiert: $\mathbf{001}100_2 = 0{,}375_{10}$. Die Implementierung ist trivial.

Durch diese Art der Realisierung der Multiplikationen entstehen 21 weitere Additionen. Einschließlich der Addition aus Formel 9 sind somit 22 Additionen durchzuführen. Die maximale Verzögerungszeit darf nach 10.2 940 ns betragen. Pro Addition stehen also maximal 42,73 ns zur Verfügung. Die Folgerung ist: Das gleitende Mittel kann mittels **eines** Addierers, z.B. dem add(cla) aus der DW01_Library mit 6,93 ns Verzögerungszeit (siehe Tabelle 2), realisiert werden.

Ein Variablenmultiplizierer benötigt laut Tabelle 1 ca. 2,41 mm^2. Die Realisierung der Shift&Add-Struktur erfordert eine Fläche von 0,54 mm^2 (ohne einen Addierer. Dieser wird auch für die Implementierung der Struktur mit einem Variablenmultiplizierer benötigt). Daraus folgt eine Flächeneinsparung von 22,4%. Hierdurch bestätigt sich der Vorteil der Konstantenmultiplizierer gegenüber einem Variablenmultiplizierer.

11 Synthese

Nach der Erstellung einer Verhaltensbeschreibung erfolgt nach ausführlichen Testreihen das Mapping des Designs auf eine Hardware/Technologie. Zur Klärung dieses Vorgangs, soll im Unterkapitel 11.1 der Designflow beschrieben werden. Es wird auf verschiedene Abstraktionsebenen beim Design und die Compiler, welche die Übergänge von weniger konkreten zu konkreteren Ebenen ermöglichen, eingegangen. In 11.2 soll dieses Wissen auf den speziellen Fall SiliconEar Anwendung finden.

11.1 Abstraktionsebenen

Moderne Chipdesigns werden immer komplexer, bei immer kürzeren Produktionszyklen. Mehrere Millionen Transistoren werden auf immer kleineren Flächen untergebracht. Mit dem Plot des Layouts einer CPU auf Transistorebene könnten Reichstage verpackt werden. Kein Mensch ist in der Lage, in einem solch komplexen System einen Überblick zu erlangen. Abstraktionen sind notwendig. Die Transistorebene ist bereits eine solche Abstraktionebene. Anstatt die physikalischen Vorgänge in Halbleitermaterialien zu betrachten, werden diese auf das Wesentliche, die Eigenschaften eines Transistors, reduziert. Gruppen von Transistoren werden wiederum zu Gattern zusammengefaßt, usw. Im Folgenden soll eine Beschreibung von der Ebene maximaler Abstraktion bis zur physikalischen Ebene stattfinden.

11.1.1 Algorithmische Ebene

Die Verhaltensbeschreibung ist die Ebene maximaler Abstraktion (in dieser Arbeit) und wird algorithmische Ebene genannt. Weder später verwendete Hardware noch Verzögerungszeiten sind relevant. Die Funktion einer Schaltung wird mittels nebenläufiger Algorithmen in Programmiersprachen wie z.B. C, Verilog oder VHDL implementiert. Verilog und VHDL wurden speziell für die Entwicklung von Hardware erdacht und vereinfachen das Implementieren von nebenläufigen Vorgängen, so daß die Verwendung dieser Sprachen der praktikablere Weg ist. Das Verhalten von Hardware wird mit Konstrukten dieser höheren Programmiersprachen simuliert. So kann z.B. eine Case-Anweisung das Verhalten einer FSM beschreiben.

Die Ziele, die auf dieser Ebene erreicht werden sollen, sind die erstmalige Spezifikation eines Systems und die Verfeinerung der Algorithmen zur optimierung von Parametern wie Flächen,- Leistungs- und Zeitverbrauch.

Mittels geeigneter Compiler findet der Übergang zur nächsten Ebene statt. Die Designtoolumgebung von Synopsys bietet für den Schritt zur Registertransferebene den Behavioral Compiler.

11.1.2 Registertransferebene

Beim Übergang zur Registertransferebene sind die Algorithmen der Verhaltensbeschreibung in Funktionsblöcke zu zerlegen. Diese Blöcke werden ihrerseits zunächst in ihrem Verhalten beschrieben und schließlich heruntergebrochen, bis das Design zuletzt aus einfachen Bauteilen wie Addierern, Multiplizierern, Shiftregistern, usw. besteht. So wird z.B. eine FSM durch ihre Zustandsregister beschrieben. Bei diesem Vorgang werden die Blöcke in Komponenten mit Registern (synchron) und registerlose (asynchron) unterteilt. Erstmalig werden Taktsignale in das Design integriert. Die Blockschaltbilder aus dem Anhang A geben ein Beispiel. Sie zeigen das Design des SiliconEar auf dieser Designebene.

Der Übergang zur nächsten Ebene erfolgt mittels des *es2_design_analysers*. Dieser Vorgang wird Synthese genannt. Im Idealfall werden erst an dieser Stelle der Entwicklung Entscheidungen zur zugrundeliegenden Hardware oder zum Timing im Design getroffen. Das SiliconEar wird auf die Technik ecpd07 gemappt. Die kleinstmöglichen Strukturen betragen somit 0,7µm.

11.1.3 Gatterebene

Auf der Gatterebene werden (in diesem Fall von ES2) Gatter zur Verfügung gestellt. Hierbei handelt es sich um UND-Gatter, ODER-Gatter, Multiplexer, Flipflops, Latches, usw., also um ein System von Gleichungen zwei- oder mehrwertiger Logik. ES2 bestimmt die Art und Weise des Aufbaus der Gatter. Zwar ohne dem „Kunden Designer“ Informationen hierüber zukommen zu lassen, jedoch gibt ES2 Parameter wie minimale, normale und maximale Verzögerungszeit und Flächenverbrauch jedes einzelnen Gatters bekannt. Mittels dieser Werte spiegeln Simulationen auf der Gatterebene erstmals das Zeitverhalten des Designs wieder. Auch sind Aussagen über den Flächenverbrauch möglich. Diese können nur Abschätzungen sein, da die Gatter noch nicht in räumlicher Position zueinander stehen und somit die Verdrahtungswege und - längen nicht bekannt sind. Das Ziel des Designs auf dieser Ebene ist die optimierung des Zeitverhaltens und der benötigten Chipfläche. Bei der Analyse des Zeitverhaltens ist das Taktschema auf seine Realisierbarkeit hin zu untersuchen.

Für das SiliconEar konnte der vollständige Übergang auf diese Ebene aus in Kapitel 11.2 genannten Gründen noch nicht erfolgen.

11.1.4 Layout

Aus der Gatterebene wird mittels eines Tools des Softwarehauses Cadence ein Layout erstellt. Das bedeutet, daß die Funktionsblöcke räumlich zueinander angeordnet und verdrahtet werden. Dieser Vorgang wird stark durch die Zieltechnologie beeinflußt. So gilt es, minimale Abstände zwischen Leiterbahnen einzuhalten. Dieses gelingt um so einfacher, je mehr Verdrahtungslagen die Zieltechnologie bietet. Es gilt auch Scanpfade einzufügen, um die Testbarkeit des ASIC gewährleisten zu können. Es sind hierfür Testpattern zu generieren, die eine

möglichst große Menge an Produktionsfehlern im VLSI-Chip aufdecken.

Für den Übergang zu den nächsten Ebenen existieren keine Tools für den Designer. Diese Schritte werden für den „Kunden“ von ES2 durchgeführt.

11.1.5 Transistorebene

Auf der Transistorebene werden einzelne Gatter wie Latches und Inverter auf Gruppen von Transistoren einer Technologie abgebildet. So entsteht ein System von diskreten Gleichungen mehrwertiger Logik mit Wertebereichen wie TRUE, FALSE, usw. (siehe auch [18]).

11.1.6 Physikalische Ebene

Auf der physikalischen Ebene werden nicht diskrete, logische Werte, sondern kontinuierliche Spannungen und Ströme betrachtet. Es entsteht ein System von Differentialgleichungen. Dabei können Transistoren auf unterschiedlichste Weise realisiert werden (MOS Transistor, Bipolar Transistor, siehe auch [18]).

11.2 Synthese des SiliconEar

Zu Beginn lag eine von der PG SiliconEar erstellte Beschreibung des ASIC auf der algorithmischen Ebene vor. Hätte mit Hilfe des Behavioral Compilers hieraus eine Beschreibung auf der RT-Ebene generiert werden können, so hätte das den Aufwand für diese Diplomarbeit vermindert. Aus in Kapitel 4 und 5 genannten Gründen konnte dieser Weg nicht gegangen werden. Hinzu kommt, daß der Behavioral Compiler zu Beginn dieser Arbeit noch nicht zur Verfügung stand. Es wurde daher entschieden, die RT-Beschreibung des ASIC „von Hand“ zu erstellen. Dieses ist, so weit es möglich war, geschehen und die Beschreibung liegt getestet vor (siehe Kapitel 12).

Abbildung 27 zeigt den Entwurfsraum, der bei der Entwicklung eines ASIC durchschritten wird. In diesem Kapitel wird der Weg von der RT-Ebene zur Gatterebene (gestrichelter Pfeil ind Abbildung 27) kommentiert.

Bei der Synthese sind grundsätzlich zwei Vorgehensweisen denkbar: Bottom Up und Top Down. Der Leser betrachte hierzu den Graphen aus Abbildung 49 im Anhang E. Die Wurzel des Baums bildet die Entität *c_Siliear.* Bei einer Top Down-Analyse würde mit der Synthese von *c_Siliear* begonnen werden. Dieses Vorgehen ist insoweit unvorteilhaft, da ein Testen des Resultats erst nach Abschluß der Synthese aller weiteren Komponenten im Baum möglich ist. Ist das Ergebnis nicht zufriedenstellend, so sind alle Schritte der Synthese erneut durchzuführen. Die Entscheidung ist auf die Bottom-Up-Synthese gefallen. Eine Entität kann nach der Synthese in die RT-Beschreibung eingesetzt und ihr Verhalten verifiziert werden. Bei den Blättern im Baum von Abbildung 49 handelt es sich zudem um Beschreibungen mit geringer Komplexität. Die Synthese dieser Komponenten sollte daher einen leichten Einstieg in den Umgang

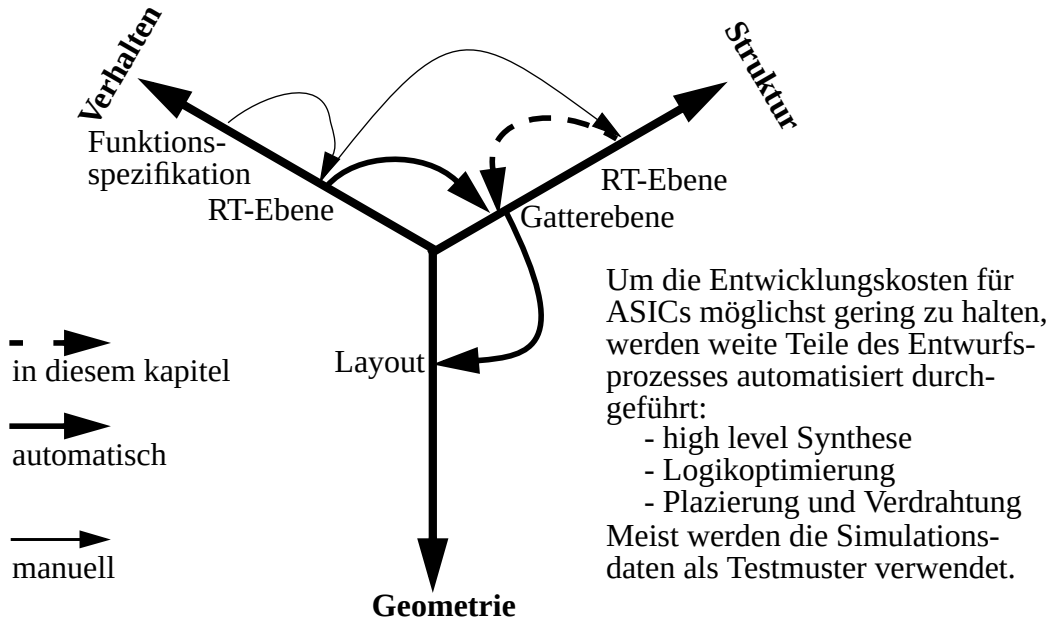


Abbildung 27: Entwurfsmethode ASIC [18]

mit den Synthesetools bereiten. Begonnen wurde mit den Komponenten *c_Register, c_Mux16, c_Mux* und *c_Shiftreg*. Ihre Synthese erwies sich als unproblematisch:

1. *es2_design_analyzer* starten
2. VHDL-Code laden
3. Menü: Tools-Designoptimization-Map Design High-OK

Das Verhalten des synthetisierten VHDL-Codes ist semantisch korrekt. Somit kann die nächst höhere Komponente *c_RegisterCluster* auf die gleiche Weise gemappt werden. Das gesamte IIF und die HSS können so synthetisiert und verifiziert werden.

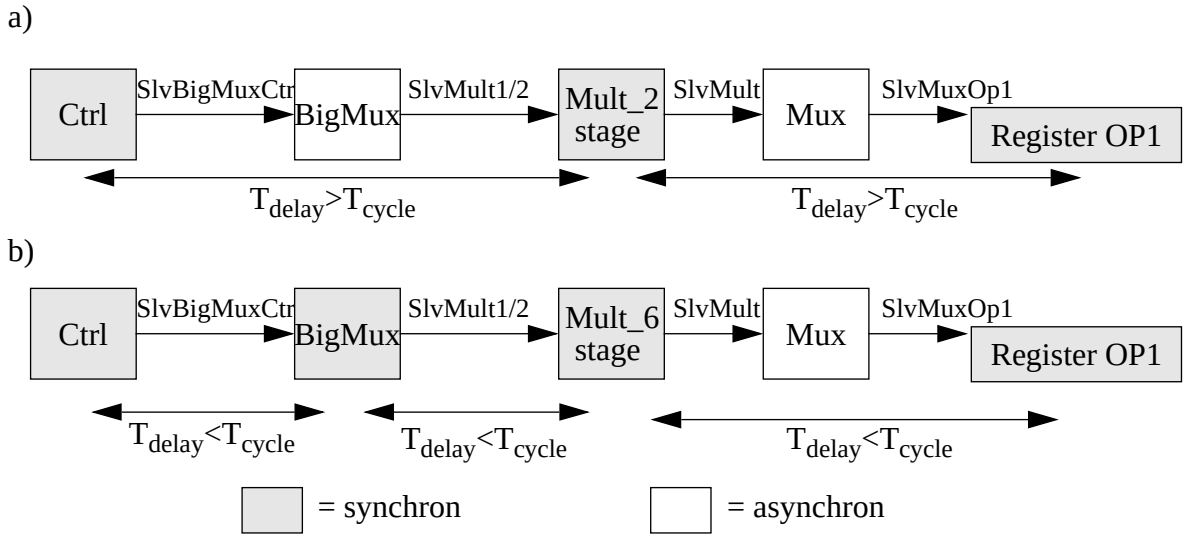


Abbildung 28: Retiming im GFB-Kanal

Bei der Synthese der GFB sind Schwächen im Design auf der RT-Ebene entdeckt worden. Ursprünglich war im Design des programmierbaren GFB-Kanals ein zweistufiger Multiplizierer vorgesehen. Die durch eine Analyse der Schaltung ermittelten kritischen Pfade mit maximaler Verzögerungszeit werden in Abbildung 28a gezeigt (vergleiche mit Abbildung 35 im Anhang A). Bei dem verwendeten Systemtakt von 50 MHz muß eine Verzögerungszeit zwischen zwei Registern von weniger als 20 ns eingehalten werden. Die Setupzeit für die verwendete Technologie ecpd07 beträgt 1,5 ns, so daß insgesamt nur 18,5 ns Verzögerungszeit erlaubt sind.

Pfad1 erstreckt sich von den Zustandsregistern des GFB-Kanal-Controllers über einen großen Multiplexer (BigMux) zu den Eingangsregistern der ersten Stufe des zweistufigen Multiplizierers. Ursprünglich war BigMux als asynchrone Komponente implementiert. Die maximale Verzögerungszeit in diesem Pfad wurde damit jedoch um 12 ns überschritten. Daher wurden am Ausgang von BigMux Register eingefügt, welche den Pfad1 aufteilen. Siehe hierzu die Abbildung 28b. Dadurch gelangen Werte für den Multiplizierer einen Takt später als zuvor an dessen Eingang. Der Controller mußte entsprechend angepaßt werden.

Eine Zeitüberschreitung im 2.Pfad, vom Multiplizierer über einen Multiplexer in das Register des ersten Operanden des Addierers/Subtrahierers, wurde durch den Austausch des zweistufigen durch einen sechsstufigen Multiplizierer verhindert. Dieses hatte erneut ein Umschreiben des Verhaltens des Controllers zur Folge (siehe Kapitel 7.2.1). Mit diese beiden Anpassungen wurden die Timingprobleme im GFB-Kanal gelöst.

Die Synthese des Controllers der GFB zeigte eine Timingverletzung bei der Ansteuerung des RAM-Blocks auf. Er (der Controller) enthielt eine selbstgeschriebene Funktion zur Adressumwandlung von Integerwerten nach Standard-Logic-Vektoren, die eine maximale Verzögerungszeit von 18 ns besaß. Bei der Simulation erzeugte dieses Zeitverhalten einen Fehler in der Ansteuerung des RAM:

> *Error on ‚DPRGFB‘ (‚DPR‘) - port: 1 - Address to ME setup time violation (tADS) - specified: 2.41 NS found: 2 NS.*

Ein Austausch der Funktion durch die Funktion *CONV_STD_LOGIC_VECTOR(from,size)* aus dem Package *STD_LOGIC_ARITH* behob das Fehlverhalten.

An dieser Stelle soll erwähnt werden, daß der vom Megacell Compiler generierte VHDL-Code zu den ROM- und RAMblöcken (siehe Kapitel 8) nicht zu synthetisieren ist. Es handelt sich hierbei um Verhaltensbeschreibungen. Die notwendigen Beschreibungen, für im Designflow tiefer liegende Ebenen, werden vom Megacell Compiler erstellt und nicht vom Design Compiler (Kapitel 8.2).

Bei der Synthese der nächsthöheren Komponente, also der Datei *c_GFB*, bricht der Designanalyzer mit einem *Fatal Error* ab. Es kommt zum Absturz der Software. Der Fehler ist reproduzierbar, tritt also **immer** dann auf, wenn die GFB gemappt wird. Dieses hat eine Synthese bis Dato vereitelt. Die Fehlermeldung ist folgende:

Fatal: Internal system error, cannot recover.
Error code=6
You can use SOLV-IT to search for articles related to this error.
If you wish to use SOLV-IT, please complete the following steps:

1. Copy and paste everything between the dotted lines
into the body of an email message
2. Replace <your solvit_id_number> with your 7 digit solvit id number
3. Please email the message to solvit@synopsys.com

----------cut below this line----------
start:

id:<your solvit_id_number>
getfatal:

Release = ‚v3.5a‘ Architecture = ‚sparc‘ Program = ‚design_analyzer‘

‚13896988 13897468 15185372 -277534180 -277928256 -276881848 15139380 9806872 9808100 9790884 6402084 3502584 3435056 1106688 12393636 12382908 560864 563036 563140 84084 84596 332124 14703388 14251260 14839192 14739468 14742064 14742176 14743024 13144 9048 8300‘

end:
----------cut above this line----------

Nachdem eine Email wie oben beschrieben übermittelt wurde, schickte Synopsys folgende Antwort:

„Please send a description of what you were doing at the time of the fatal (tool crash), any setup files used as well as script files, the log file and the hdl code of the design. Once the test-case information is sent, an engineer will investigate the problem and contact you.“

Die gewünschten Informationen dürfen Synopsys nicht ohne weiteres zugesandt werden. Der HDL-Code ist Eigentum der Universität Oldenburg bzw. der AG MEDI und der AG EIS. Das Vorgehen zu diesem Thema ist noch klärungsbedürftig und wird nicht mehr innerhalb dieser Diplomarbeit ermittelt werden können.

Der Crash des Tools kann vermieden werden, wenn im Design tiefer liegende, also bereits synthetisierte, Komponenten mit dem Attribut „don’t touch“ versehen werden. Die Simulation des Syntheseergebnisses liefert hiernach aber **falsche** Werte! Das Verhalten von RT-Beschreibung und GATE-Beschreibung divergiert. Eine mögliche Begründung können Timingprobleme sein, weil das Tool wegen der Attribute „don’t touch“ kein Retiming durchführen kann. Eine genaue Untersuchung des Problems wird nicht vorgenommen. Es soll zunächst auf eine Lösung für den Softwarefehler im Designtool gewartet werden. Möglicherweise entfällt dann dieses Problem.

Ziel der Arbeit ist nicht die Generierung einer korrekten Beschreibung des ASIC auf der Gate-Ebene, sondern der Übergang von der algorithmischen zur RT-Ebene. Der Code soll synthetisierbar sein, nicht aber notwendigerweise synthetisiert vorliegen. Oben genannte Komplikationen verhinderten den Abschluß der vollständigen Synthese innerhalb der Zeit dieser Arbeit. Weil die Blöcke BLB und Tiefpaßfilter noch nicht auf RT-Ebene vorliegen, konnten noch keine Syntheseversuche zu diesen Komponenten unternommen werden.

Die Generierung einer fehlerfreien Beschreibung auf Gate-Ebene wird im Anschluß an diese Diplomarbeit stattfinden (siehe Kapitel 15).

12 Testbench

Entwickler von Schaltungen, Software oder Designs aller Art sind sich über die Wichtigkeit von ausführlichen Testreihen einig. Ein stark vereinfachtes Beispiel soll dieses verdeutlichen:

Beim Bau einer Ariane Trägerrakete wird bei der Produktion einzelner Komponenten höchste Präzision verlangt. Jede einzelne Komponente sei zu 99,99% fehlerfrei. Werden zwei dieser Teilsysteme zu einem größeren Untersystem verarbeitet, so sinkt die Funktionsgarantie auf $99{,}99^2$%. Würde die Rakete aus 10.000 Bestandteilen zusammengesetzt werden, so ist ein fehlerfreier Start nur zu 36,78% garantiert. Keine Versicherung wäre bereit einen solchen Start decken zu wollen. Also selbst bei einer Funktionsgarantie der Einzelteile von 99,99% kann einem System erst nach einigen erfolgreichen Einsätzen vertraut werden.

Beim Schaltungsdesign wächst die Zahl der möglichen Testfälle exponentiell mit der Zahl der Zustände des Systems. Vermeintlich unwesentliche Fälle müssen daher außer Acht gelassen werden. Die Einteilung in „wesentlich“ und „unwesentlich“ ist bei komplexen Strukturen schwer zu entscheiden. Weiterhin benötigen Simulationen, insbesondere auf der Gatterebene, viel Zeit, die zum Ende eines Projektes schnell knapp wird. Für den letzten Aktionspunkt, dem Testen, sollte daher mehr Zeit eingeplant werden, als vielleicht vorerst notwedig erscheint.

Dieser Abschnitt, welcher sich mit dem Abschlußtest des SiliconEar befaßt, soll nicht auf die zahlreichen Testbenches eingehen, die zu einzelnen Komponenten des SiliconEar erstellt wurden. Wie am Beispiel gezeigt, sagt die Funktionstüchtigkeit der einzelnen Komponenten wenig über die des Ganzen aus. Die „kleinen“ Testbenches sind Nebenprodukt der Implementationsphase und können verworfen werden. Eine Dokumentation hierzu anzufertigen wäre nicht nutzbringend. Von Interesse ist das Ein- und Ausgangsverhalten des ASIC. Stimmen die Werte am Ausgang des Chips, also an dem OIF, bei definierten Eingabewerten mit den erwarteten überein, so ist die Schaltung semantisch korrekt im Sinne der AG MEDI. Die Verifizierung des Designs erfolgt somit durch Testpattern und Vergleich der Ausgaben mit Kontrollwerten.Die Kontrollwerte werden in zwei Schritten generiert:

- Zu der C-Implementation des Perzeptionsmodells aus der Physik, welches mit Floatingpointzahlen arbeitet, ist eine C-Version in Fixpointdarstellung erarbeitet worden. Die Fixpointversion wird durch den Vergleich ihrer Ausgaben mit denen der Floatingpointversion verifiziert. Dabei sind die auftretenden Quantisierungsfehler zu berücksichtigen.
- Die Ergebnisse des VHDL-Codes werden ihrerseits mit denen der C-Fixpointversion verglichen. Auch hier sind Abweichungen zu berücksichtigen, da die C-Testbench nicht wirklich mit Fixpointwerten rechnet. Der Code wurde einer reinen Fixpointimplementierung durch Abschneiden der Bitstellen größer 24 nach jeder Operation lediglich angenähert. Bei den Multiplikationen und Additionen handelt es sich weiterhin um Floatingpointfunktionen. Diese besitzen eine höhere Bitgenauigkeit als die Komponenten der Designware Libraries.

Der Übersicht halber zeigt die Abbildung 29 die Verzeichnisstruktur der Testumgebung. Im Verzeichnis Eingabesample befinden sich die ASCII-Listen der Eingaben. *imp0.99* enthält einen Einheitsimpuls. Die Datei *sprache* enthält Sprachsample, welche von der AG MEDI zur Verfügung gestellt wurden. In jeder Zeile der Datei steht ein Sample in binärer Form. Eine Sampledatei für eine Sekunde zu simulierende Echtzeit besitzt somit 16276 Zeilen.

Die Ausgaben werden in das Verzeichnis Ausgangssample in die Dateien *HSS_Out.dat* und *LSS_Out.dat* geschrieben. Im Verzeichnis *test* befinden sich die C-Vergleichswertgeneratoren. Das für den Abschlußtest relevante Verzeichnis ist der Zweig von *c-SiliEar.* In den folgenden Unterkapiteln soll der c-Vergleichswertgenerator genauer dokumentiert werden. Ab 12.6 wird der VHDL-Teil besprochen.

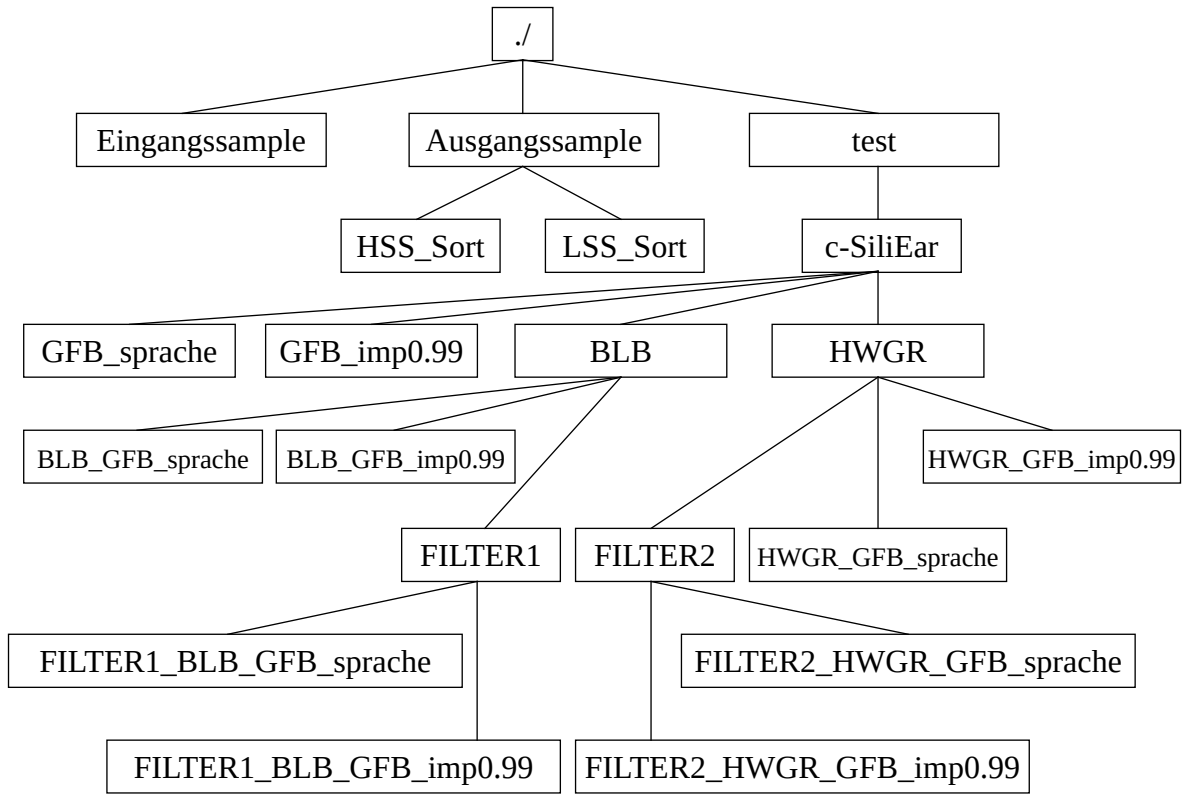


Abbildung 29: Testverzeichnisbaum

12.1 c-gfb.cc

In *c-SiliEa*r befindet sich der Vergleichswertgenerator für die GFB. *c-gfb.cc* enthält den Quellcode. Aufgerufen wird das Programm mit der Befehlssyntax *c-gfb <file> n*. <file> gibt die Sampleinputdatei an. Mit n wird der zu berechnende Kanal ausgewählt. n kann somit Werte im Bereich von 0 bis 29 annehmen. Zur vereinfachten Erzeugung sämtlicher GFB-Werte zu einem Einheitsstoß und einer Sprachsampledatei werden die Dateien *imp0.99* bzw. *sprache* verwendet. Es handelt sich um Batchjobs. Die generierten Vergleichswerte werden in die entsprechenden Unterverzeichnisse *GFB_imp0.99* und *GFB_sprache* verschoben.

c-gfb arbeitet nach der einfachen Formel 1 aus Kapitel 6. Die Quantisierung der Zahlen erfolgt nach jeder Operation im Code durch eine Umwandlung derselben in einen 24-Bitvektor und anschließender Rückwandlung nach Float. Dabei werden alle Bits nach dem 24. abgeschnitten. Es wird ein systematischer Fehler bei der Addition und Multiplikation im Code gemacht, da diese Operationen mit der Genauigkeit von Float arbeiten. Somit können *c-gfb* und *vhdl-gfb* nicht absolut gleiche Werte liefern. Es reicht eine Überdeckungsgenauigkeit von 16 Bit, da nur daß Ausgangsverhalten verifiziert wird. 16 Bit ist die I/O-Genauigkeit des ASIC, im Gegensatz zu 24 Bit im Kern. Das entspricht einer Abweichung von maximal +/-3.05e-5.

Der Code ist übersichtlich und durch bloßes „Hinsehen“ leicht zu verifizieren. Zusätzlich wurden die Impusantworten der 30 Bandpaßfilter mit denen von der AG MEDI gestellten verglichen. Unter Berücksichtigung der Quantisierungsfehler sind beide Ergebnisse übereinstimmend. Nach Satz1 (siehe 12.6) sind die *c-gfb* und die von der AG MEDI in C implementierte GFB des Perzeptionsmodells semantisch identisch. Somit kann *c-gfb* zur Generierung von Vergleichswerten herangezogen werden.

12.2 c-blb.cc

Im Unterverzeichnis BLB wird sich der C-Code zur Simulation der gleichnamigen Komponente befinden. Bis zum Abschluß dieser Diplomarbeit konnte dieser Block des ASIC nicht implementiert werden (siehe 9.3.2). Ein VHDL-Dummycode schleift die Werte von der GFB durch, sodas sich in den Verzeichnissen *BLB_GFB_sprache* und *BLB_GFB_imp0.99* die selben Antworten befinden, wie sie von *c-gfb* erzeugt werden.

12.3 c-FILTER1.cc

Der Code *c-filter1.cc* wurde nach dem selben Verfahren wie der von *c-gfb.cc* erstellt. Es ist sichergestellt, daß die Semantik mit der von der AG MEDI geforderten übereinstimmt. Der Aufruf erfolgt über das Kommando *c-filter1 <Eingabefile>*, wobei „Eingabefile“ in den Datenverzeichnissen von 12.2 steht. Der Einfachheit wegen existieren die Batchjobs *filter1_BLB_GFB_imp0.99* und *filter1_BLB_GFB_sprache*, mit denen die Verarbeitung eines kompletten BLB-Datensatzes durchgeführt wird. Die Ergebnisse gelangen in die Datenverzeichnisse mit dem Präfix *FILTER1*.

12.4 HWGR

Die Daten aus den Unterverzeichnissen von *c-gfb* werden dem HWGR als Eingabe zugeführt. Die bearbeiteten Werte gelangen entsprechend ihres Ursprungs in die Verzeichnisse *HWGR_GFB_imp0.99* bzw. *HWGR_GFB_sprache*. Die Testbench des HWGR besteht aus einem einfachen awk-Programm:

awk ‘{if ($1>0.00001) print $1; if ($1<=0.00001) print „0.00001“ }‘
EINGABEFILE > AUSGABEFILE

Alle Werte größer 1e-5 passieren den HWGR, der Rest wird durch 1e-5 ersetzt. Eine ausführliche Hilfestellung zu awk befindet sich in den Manpages. Die Berechnung eines kompletten Datensatzes wird mittels der Batchjobs *hwgr_gfb_imp0.9* und *hwgr_gfb_sprache* durchgeführt.

12.5 c-FILTER2.cc

Dieses Unterverzeichnis beherbergt, analog zu FILTER1, c-filter2. Der Aufbau ist der gleiche wie in 12.3.

12.6 Der VHDL-Teil der Testbench

Im Designverzeichnis *~frankp/diplomarbeit/vhdl* befinden sich mehrere Batchdateien mit dem Präfix *„makesiliear“*. Sie ermöglichen eine automatisierte Analyse sämtlich notwendiger VHDL-Codefiles. Eine Erweiterung des Präfix um z.B. *_gfb_syn* sagt aus, daß die RT-Beschreibung des ASIC analysiert wird, die gfb jedoch durch eine Beschreibung auf Gate-Ebene realisiert wird. Es ist sicherzustellen, daß sich alle benötigten Files in den Arbeitsverzeichnissen *rtl, rtl_BLB, rtl_Filter1* und *rtl_Filter2* befinden. Sie sind gegebenenfalls mittels des RCS-Befehls *co* im jeweiligen Verzeichnis auszuschecken.

Nach vollständiger und fehlerfreier Analyse wird der Debugger *vhdldbx* **aus dem Designverzeichnis heraus** gestartet. Aus der Defaultlibrary wird das Design *CFG_TB_C_SILIEAR* ausgewählt. Hiernach wird mit *run 1230000000* eine Simulation gestartet. Vom Verhalten des ASIC werden so 12,3 ms Realzeit berechnet. In dieser Zeitspanne werden 200 Stereosample verarbeitet. Der Zeitraum ist ausreichend, da die Impulsantworten der Bandpaßfilter am Ende der Simulation abgeklungen sind. Das heißt, alle weiteren Ergebnisse der GFB sind Nullwerte, und eine verlängerte Simulationszeit verbessert die Beobachtung von Impulsantworten somit nicht (siehe Anhang C). Wichtig ist in diesem Zusammenhang ein Satz über LTI-Systeme, wie die Bandpaßfilter welche sind:

Satz1: Ein lineares, zeitinvariantes System ist durch seine Impulsantwort eindeutig bestimmt. [9].

Der Code der Testbench *tb_c_SiliEar.vhd* aus dem Verzeichnis *vhdl/rtl* verwendet die Komponenten mit den Namen *c_HSS_Drain* und *c_LSS_Drain*. Sie sind dem OIF des ASIC nachgeschaltet und simulieren das Verhalten von DSPs. Die von ihnen entgegengenommenen Werte werden in dem Verzeichnis Ausgangssample abgelegt. Die Dateien sind *HSS_Out.dat* und *LSS_Out.dat*. Ursprünglich war vorgesehen worden, eine größere Zahl (ca. 200) an Dateien zu generieren, in die die Werte nach Frequenzkanalnummer und Art (imaginär, real, Phase, Quotient) während der Simulation einsortiert werden sollten. Synopsys ist jedoch nicht in der Lage mehr als 100 Dateien gleichzeitig zu öffnen. Das Aussortieren der Werte aus den Dateien *.dat* hat folglich nach der Simulation zu geschehen. Dieses wird mittels awk-Programmen erreicht.

In jeder Zeile von *c_HSS_Drain* und *c_LSS_Drain* steht ein Berechnungsergebnis. Ein Kürzel zu Beginn der Zeile gibt an, um welchen Wert es sich handelt. „Real-L-10“ steht für einen Realteil aus der HSS, welcher vom Bandpaßfilter 10 aus einem linken Sample berechnet wurde. „Amplitude-R9“ zeigt einen Amplitudenwert der LSS an, welcher aus dem rechten Filterkanal 9 der GFB ermittelt worden ist. Die Kürzel „BetragN“ und „PhaseN“ sind analog zu interpretieren.

Nach dem Start des Batchjobs *SORT2REAL* aus dem Verzeichnis *Ausgangssample* werden Dateien wie *Imag-R-11.bin* in die Verzeichnisse *HSS_Sort* und *LSS_Sort* geschrieben. Sie enthalten die Werte nach Kürzel sortiert. Nach dem Umwandeln der Binärwerte zu Realzahlen erscheint jeweils eine weitere Datei mit dem Postfix *.real* im Dateinamen. Für den letzten Schritt des Abgleichs ist in das Verzeichnis *HSS_Sort* bzw. *LSS_Sort* zu wechseln. Hier befinden sich die Dateien *compare_HSS* und *compare4step_LSS*. Diese Batchjobs führen den Abgleich mit den Werten aus den Verzeichnissen von *c-Siliear* durch. Unstimmigkeiten werden sowohl auf dem Bildschirm ausgegeben, als auch in Dateien mit den Endungen *.compared* geschrieben.

Compare4step ist ein C-Programm, welches jeden **vierten** der generierten Vergleichswerte aus den Verzeichnissen *FILTER1_BLB_GFB_sprache*, *FILTER1_BLB_GFB_imp0.99*, *FILTER2_HWGR-_GFB_sprache* und *FILTER2_HWGR_GFB_imp0.99* mit der Antwort der LSS vergleicht. Dieses ist durch die Unterabtastung um den Faktor vier in der LSS begründet. Wenn *compare4step_LSS* und *compare_HSS* keine Fehlermeldungen erzeugen, ist die simulierte ASIC-Beschreibung korrekt im Sinne der AG MEDI.

Tabelle 5 gibt einen Überblick über die durchgeführten Testkombinationen. Alle Simulationen wurden für einen Zeitraum von 12,3 ms (Simluationszeit) durchgeführt. In dieser Zeit wurden 200 Stereosample eingelesen und zu ca. 30.000 Ausgabewerten verrechnet. Der Test der Registertransferbeschreibung erzeugt eine Fehlermeldung beim Abgleich mit den Vergleichswerten. Der 16. Wert der Impulsantwort des 28. Kanals besitzt eine Abweichung von 3.2031e-05 zum Vergleichswert. Das entspricht einem Fehler an Bitstelle 16 des Wortes (LSB). Alle nachfolgenden Werte befinden sich im quantisierungsbedingten, akzeptablen Bereich +/- 2^{-15} um den Vergleichswert.

Bei dem im Test verwendeten Impuls handelt es sich um eine positive Spitze mit einer Amplitude von 0,99. Wird der Stoß in negative Richtung gegeben, so tritt der Fehler nicht auf. Eine Sequenz von Sprachsampeln erzeugt diese Anomalie ebenfalls nicht. Der Fehler liegt in der Quantisierung mit Abschneiden der Bitstellen größer 24 erklärt und kann vernachlässigt werden. Die Begründung ist folgende:

Das SiliconEar wird im finalen Betrieb Sprachsample verarbeiten. Beim Test mit Sprache

arbeitet die RT-Beschreibung korrekt. Einheitsimpulse kommen in der „realen Welt“ der Audioverarbeitung nicht vor. Sie sind Extremfälle. Weiterhin ist ein Fehler in Bitstelle 16 derart klein, daß dieser vom Menschen nicht als störend oder qualitätsmindernd empfunden wird. Alle weiteren Werte bleiben innerhalb des Toleranzbereichs, und daher ist ein „OK“ am Ende der ersten Zeile der Tabelle 5 gerechtfertigt.

IIF		GFB		GFB Kanal		DW01 & DW02		Filter1		HSS		LSS		Verify
rtl	syn	rtl	syn	rtl	syn	rtl	syn	rtl	syn	rtl	syn	rtl	syn	
x		x		x		x		x		x		x		OK
	x	x		x		x		x		x		x		OK
x		x		x		x		x			x		x	OK
x		x		x			x	x		x			x	false
	x		x		x		x		x		x		x	...

Tabelle 5: durchgeführte Tests

Im zweiten Test aus Zeile zwei der Tabelle 5 wurde die RT-Beschreibung des IIF durch diejenige auf Gatterebene ersetzt. Dieser Test verläuft, bis auf den bereits genannten unproblematischen Fall, fehlerfrei. Das IIF liegt getestet als Version auf Gatterebene vor.

Beim dritten Test wird die HSS auf Gatterebene getestet. Auch hier bleibt es bei dem einzelnen, vernachlässigbaren Fehler. Die HSS liegt somit ebenfalls korrekt synthetisiert vor.

Bei der Synthese der GFB treten Schwierigkeiten auf, die durch ihre Struktur bedingt (siehe Kapitel 11) und durch Fehler in der Designtoolsoftware von Synopsys begründet sind. Es liegt eine Version auf Gatterebene vor. Der VHDL-Code ist also synthetisierbar. Die Simulation dieses Codes zeigt, daß die Semantik nicht mit derjenigen der RT-Beschreibung übereinstimmt. Es werden keinen sinnvollen Werte von der Testbench erzeugt. Um weitere Testreihen durchzuführen, ist zunächst die Synthese abzuschließen. Dieses kann im Rahmen dieser Diplomarbeit aus Zeitgründen nicht erfolgen.

Als Resultat dieser Arbeit kann semantisch korrekter VHDL-Code zur Beschreibug der RT-Ebene des SiliconEar vorgewiesen werden, der synthetisierbar ist, dessen Syntheseergebnis jedoch nicht das gewünschte Verhalten beschreibt.

13 Zusammenfassung

Dieses Kapitel soll einen Überblick über die Ergebnisse und offenen Punkte der Diplomarbeit verschaffen. Unterpunkt 13.1 faßt den derzeitigen Stand der Dinge zusammen. In 13.2 wird darauf eingegangen, was innerhalb dieser Arbeit nicht erreicht wurde.

13.1 Was wurde erreicht?

Die Systemumgebung im Datenpfad hinter dem SiliconEar wurde neu definiert. Die DSP32C von AT&T sind durch den DSPC60x von Texas Instruments und den NRGL der AG IMA ersetzt worden.

Durch die geänderte Systemumgebung war eine Anpassung des OIF erforderlich. Dieses wurde durch eine Implementierung von zwei voneinander getrennt arbeitenden Schnittstellen erreicht. Die Protokolle dieser Schnittstellen sind in Zusammenarbeit mit der AG MEDI und der AG IMA definiert worden. Der DSPC60x wird an der HSS betrieben. Die NRGL werden mit der LSS verbunden. Der Informationsfluß der Schnittstellen ist gerichtet. Daher mußte auf die Implementierung einer Fehlererkennung verzichtet werden. Die Datenrate an der HSS beträgt 29,80 Mbps, die an der LSS beträgt 11,18 Mbps. DSP und NRGL können gleichzeitig am ASIC betrieben werden.

Komponenten der Designware Library von Synopsys wurden synthetisiert und auf Flächen- und Zeitverbrauch hin untersucht. Die Ergebnisse wurden analysiert und aufbereitet. Zum Design passende Multiplizierer und Addierer bzw. Subtrahierer wurden ausfindig gemacht.

Die Ergebnisse der PG SiliconEar wurden gesichtet. Die vorliegende Verhaltensbeschreibung vom ASIC wurde als „für diese Diplomarbeit nicht effizient nutzbar“ eingestuft. Dennoch konnten viele Informationen und Ideen aus der Arbeit der PG gewonnen und umgesetzt werden.

Die Beschreibung der GFB wurde auf der RT-Ebene neu implementiert. Dabei erfolgte ein Redesign, um eine „just-in-time-Berechnung“ zu realisieren. Es wurde der Zeit-, Flächen- und Speicherverbrauch der GFB analysiert. Die ermittelten Werte sind durch „Handoptimierungen“ zu einem Optimum geführt worden. Die Verwaltung von Variablen in Registern wurde durch Integrieren von RAM ersetzt. Dazu war zunächst eine Einarbeitung in den Megacell Compiler von ES2 notwendig.Es wurde ROM zum Vorhalten der Filterkonstanten erstellt. Geeignete RAM- und ROM-Blöcke wurden erzeugt. Das Zusammenspiel von GFB, RAM und ROM wurde getestet.

Um eine Reduktion der Datenrate von 29,90 Mbps auf 11,18 Mbps zu erreichen, wurden neue Blöcke in das Design integriert. Es handelt sich hierbei um:

- ... die Beschreibung des HWGR. Sie liegt getestet auf Gatterebene vor.
- ... die Ein- und Ausgangsschnittstelle der BLB. Sie sind neu definiert und implementiert worden. Die Beschreibungen liegen getestet auf RT-Ebene vor.
- ... die gleitende Mittelwertberechnung. Sie ist spezifiziert und auf RT-Ebene implementiert worden. Dazu wurde eine Analyse zur Ermittlung des Zeit-, Flächen- und Speicherverbrauchs durchgeführt. Ein geeigneter RAM-Block wurde für das Design generiert.
- ... die Einhüllendenbildung. Sie ist spezifiziert und auf der algorithmischen Ebene implementiert worden. Eine Analyse zur Ermittlung des Speicherverbrauchs wurde durchgeführt. Ein geeigneter RAM-Block wurde erzeugt.

Die Schnittstellen zwischen den Blöcken des ASIC wurden optimiert. Reset und Kommunikation erfordern keine übergeordnete Kontrollinstanz mehr. Die CU (siehe [42]) wurde aus dem Design entfernt.

Der HSS-Pfad des ASIC liegt vollständig auf RT-Ebene beschrieben vor und wurde getestet. Die HSS wurde synthetisiert und auf Gatterebene getestet.

Der LSS-Pfad des ASIC liegt teilweise auf RT-Ebene vor. Der bereits implementierte Teil wurde getestet.

13.2 Welche Punkte stehen offen?

Der Filter zur Einhüllendenbildung wurde von der AG MEDI neu spezifiziert. Es ist eine Analyse zur Ermittlung des Flächen-, Zeit- und Speicherverbrauchs des Blocks durchzuführen, um ein Implementieren der RT-Beschreibung vorzubereiten. Die RT-Beschreibung ist zu synthetisieren und das Design ist zu testen.

Die Bitgenauigkeit der CORDIC-Einheit der BLB ist anzupassen. Das Zusammenspiel von BLB und gleitender Mittelwertbildung ist zu verifizieren, und die BLB ist zu synthetisieren. Das Verhalten der BLB auf Gatterebene ist zu verifizieren.

Es ist zu untersuchen, wie der Softwarefehler im *es2_design_analyzer* zu beheben oder zu umgehen ist, um die GFB synthetisieren zu können. Hiernach ist das Verhalten des gesamten ASIC auf die Gatterebene zu überführen und im Anschluß hieran zu testen.

Warnings, die bei der Synthese von Komponenten ausgegeben wurden sind zu analysieren und eventuell durch Modifizieren des VHDL-Codes bei zukünftigen Syntheseversuchen zu unterbinden. Die meisten dieser Warnungen beziehen sich auf „nicht verbundene Signale“ und können ignoriert werden. Es handelt sich hierbei um offene Leitungen, die durch Bitbreitenanpassungen (24 auf 16 Bit) entstehen.

14 Fazit

Die Ziele der Diplomarbeit konnten im Großen und Ganzen erreicht werden. Lediglich kleine Teile der Gesamtbeschreibung des ASIC auf RT-Ebene konnten nicht implementiert werden. Fehler in den Designtools, Zeitmangel und fehlende Spezifikationen waren die Ursachen hierfür. Es wird jedoch einfach sein, die fehlenden Teile zukünftig in das Design zu integrieren. Sämtliche Schnittstellen wurden bereits implementiert und getestet. Im günstigsten Fall sollte einfaches „plug and run“ möglich sein.

Die Beschreibung des ASIC realisiert letztendlich mehr Bestandteile des Perzeptionsmodells, als zuvor erwartet wurde. Zu Beginn der PG SiliconEar sind Überlegungen zur Implementierung einer Zweichiplösung, Master und Slave, zur Realisierung der binauralen Verarbeitung akustischer Signale angestellt worden. Der Vorschlag, beide Kanäle auf einem ASIC zu realisieren, schien bei 60 Bandpaßfiltern mit 180 Konstanten und 480 Variablen auf einer maximalen Fläche der Dies von 100 mm^2 zunächst nicht möglich. Wie diese Diplomarbeit aber zeigt, reichen 10 mm^2 für die Realisierung der GFB aus.

So wie der Platz- und Zeitbedarf für die Berechnungen in der GFB überschätzt worden sind, ist die aufkommende Datenrate unterschätzt worden. Die Daten am Ausgang des ASIC sinnvoll zu verwerten, war ein Problem, welches von der PG zunächst unerkannt blieb. Die Lösung lieferte die Zeit von selber: Die Leistung von DSPs erhöhte sich im Laufe der Entwicklung des ASIC beträchtlich. Selbst Übertragungsraten von 47,68 Mbps sind beim derzeitigen Stand der Technik ohne weiteres realisierbar.

Die PG SiliconEar und diese Diplomarbeit haben gezeigt, daß die Implementierung des Perzeptionsmodells in Hardware in absehbarer Zeit realisiert werden kann. Es besteht die Aussicht, den Floorplan des ASIC innerhalb des nächsten Jahres einer Fab zukommen zu lassen und einen Chip fertigen zu lassen.

15 Aussichten

In diesem Kapitel werden die Schritte aufgezählt, die bis zum fertigen Layout vom SiliconEar noch durchzuführen sind. Die hier genannten Punkte werden im Rahmen weiterer Arbeiten zu einem Abschluß geführt werden.

Zunächst sind die in 13.2 genannten offenen Punkte zu schließen. Dabei ist die Beschreibung auf Verletzungen von Design-Rules hin zu untersuchen. Gegebenenfalls sind Modifikationen am VHDL-Code der RT-Beschreibung vorzunehmen.

Liegt das Design final auf Gatterebene vor, sind für Signale, welche später an Pins des Chipgehäuses angeschlossen werden sollen, Pads einzufügen. Dabei ist zu entscheiden, ob zwei Spannungsversorgungen, Vdd und Gnd, zum getrennten Anschluß von Chipkern, RAM und ROM verwirklicht werden sollen bzw. müssen oder ob eine ausreicht.

Es sind Strukturen wie Testpfade zu erzeugen, die ein Testen der Schaltung ermöglichen. Hierzu sind passende Testpattern zu generieren. Es sind die Fragen zu beantworten: Werden genügend Fehler beim Test abgedeckt? Wie kann das Design verändert werden, um eine bessere Fehlerabdeckung zu erreichen? In welchem Format haben Testpattern vorzuliegen, um von der Fab akzeptiert zu werden?

Weitere Designschritte werden nicht unter Synopsys, sondern unter Cadence vorgenommen werden. Die Gatelevelbeschreibung ist daher in ein von Cadence importierbares Format zu konvertieren. Es ist die Frage zu klären, wann dieser Schritt von Synopsys nach Cadence unternommen werden soll, vor oder nach dem Einfügen von Pads und/oder Scanpfaden? Welches der beiden Designtools bietet die bessere Unterstützung bei der Generierung von Testpattern?

Die bis hier aufgezählte Vorgehensweise ist durch die Implementierung der Beschreibung auf RT-Ebene vorgegeben. Die Struktur der Schaltung wird durch die detaillierte VHDL-Codierung in ein starres Korsett gepreßt. Es verbleiben den Designtools von Synopsys nur wenige, zu optimierende Parameter. Es wäre interessant, das Verhalten des Chips auf der algorithmischen Ebene für die Bearbeitung mit dem Behavioral Compiler neu zu implementieren. Es bestünde dann die Möglichkeit, weitere Strukturbeschreibungen maschinell zu erstellen und diese mit der handerzeugten zu vergleichen. Man könnte die Frage beantworten: Welche Multiplizierer verwendet das Tool? Wie wird RAM erzeugt? Wie stark vereinfacht der Behavioral Compiler die Entwicklung von ASICs und wie gut sind dessen Resultate?

Bei der rasanten Entwicklung von immer leistungsfähigeren Prozessoren oder DSPs stellt sich die Frage, ob das Perzeptionsmodell als ASIC realisiert werden muß/soll, oder ob in nächster Zeit ein standard DSP auf dem Markt sein wird, welcher die gleiche Leistung erbringen kann. Die Entwicklung eines ASIC macht dann Sinn, wenn dieser seinen Größenvorteil z.B. bei einer Hörgeräteimplementierung gegenüber einem DSP ausspielen kann.

16 Literatur

Modell zur gehörgerechten Verarbeitung

[1] Tchorz, J. u.a.: "Gehöhrgerechte Merkmalsextraktion zur robusten Spracherkennung in Störgeräuschen". In "Fortschritte der Akustik - DRGA '96" (1996)

[2] Hansen, M.; u.a.: "Objektive Sprachqualitätsvorhersage mittels einer gehörorientierten Vorverarbeitung". In "Fortschritte der Akustik - DRGA '96" (1996)

[3] Dau, T.; u.a.: "A quantitive model of the effective signal processing in the auditory system: I. Model Structure". Submitted to JASA. (1996)

[4] William A. Yost, Fundamentals of Hearing

Allgemeine digitale Signalverarbeitung

[5] Stearns, Samuel D.: "Digitale Verarbeitung analoger Signale" (1994)

[6] Zölzer : "Digitale Signalverarbeitung" (Teubner-Verlag)

[7] Oppenheim, A.V.;Schafer, R.W.: "Discrete-Time Signal Processing". (Prentice Hall, 1989)

[8] Rabiner, L.R.; Schafer, R.W.: "Digital Processing of Speech Signals" (Prentice Hall, 1978)

[9] Professor Jensch, Univ. Oldenburg: „Skript zur Vorlesung Mustererkennung“ (1996)

Signalverarbeitung in der GammaTone-Filterbank

[10] Holdsworth, J.; et. al.: "Implementing a GammaTone Filter Bank" (1988)

[11] Solbach, L.: "The complex-valued continous wavelet transform as a preprocessor for auditory scene analysis.“ (1996) Download ps-file

[12] Hansen, M.; Hohmann, V.: „Implementation der Gammatone-Filterbank“ (1996)

Schaltungsstrukturen zur digitalen Siganlverarbeitung

[13] Pirsch, P.: "Architekturen der digitalen Signalverarbeitung". Teubner (1996)

[14] Potkonjak, M., u.a.: "Multiple Constant Multiplications: Efficient and Versatile Framework and Algorithms for Exploring Common Subexpression Elimination". IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, Vol. 15, No. 2, Feb. 1996.

[15] Tietze, U.; Schenk, C.: "Halbleiter-Schaltungstechnik" (Springer, 1993) 10.Aufl.

Entwurf integrierter Schaltungen / Design-Methoden

[16] c't Heft 5/ 96: "Vom Sand zum Chip, ASIC-Entwicklung heute" (1996)

[17] Ecker, W.: "Neue Verfahren für den Entwurf digitaler Systeme mit Hardwarebeschreibungssprachen". Shaker Verlag (1995)

[18] Professor Nebel, Univ. Oldenburg: „Vorlesung Entwurf Integrierter Schaltungen“ (1996)

VHDL

[19] Peter J. Ashenden: "The VHDL Cookbook - First Edition" (1990)
[20] Synopsys OnLine-Doku (Aufruf iview) - Methodology Notes
[21] VHDL-Kurzbeschreibung der Univ. Hamburg
[22] Lehmann; Wunder; Selz: „Schaltungsdesign mit VHDL“
[23] VHDL-Ecke: Serie in der Elektronik x/1996
[24] Eggers: „Entwurf digitaler Systeme mit Hardwarebeschreibungssprachen“
[25] Professor Nebel, Univ. Oldenburg: Vorlesung Rechnerstrukturen
[26] Professor Kowalk, Univ. Oldenburg: Vorlesung Rechnernetze

Design For Low Power

[27] Nagendra, Ch.; et. al.: "Design Tradeoffs in High Speed Multipliers and FIR Filters". (Aus der WWW-Homepage von Nagendra erhältlich)
[28] Nagendra, Ch.; et. al.: "Power-Delay Characteristics of CMOS Adders". (Aus der WWW-Homepage von Nagendra erhältlich)
[29] Nagendra, Ch.; et. al.: "Low Power Tradeoffs in Signal Processing Hardware Primitives". (Aus der WWW-Homepage von Nagendra erhältlich)
[30] Nagendra, Ch.; et. al.: "Area-Time-Power Tradeoffs in Parallel Adders". (Aus der WWW-Homepage von Nagendra erhältlich)

Datenblätter

[31] Information Manual, AT&T, WE DSP32C, Digital Signal Processor,
[32] Datenblatt: AK5343 ASAHI KASEI

Seminarausarbeitungen der Projektgruppe Silicon Ear

[33] DFW II - Logikentwurf (Christoph Wetjen)
- Grundlagen zum IC-Entwurf mit DFW II der Fa. Cadence und dem ES2 Design Kit
- Eingabe der logischen Schaltung (Schematic Entry)
- Empfehlungen und Richtlinien beim Schaltungsentwurf mit ES2 Libraries
- Erzeugung testbarer Schaltungen und Testmustergenerierung mit DFW II

[34] DFW II - Placement & Routing (Siegfried Wetjen)
- Floorplanning
- Layout-Extraktion und Backannotation
- Routing-Verfahren
- Layout-Verifikation
- Bonding

[35] Synopsis (Frank Poppen)
- Schaltungssimulation und -synthese

[36] Projektplanung und -durchführung (Norbert Friese)
- Grundlagen zur Projektplanung und -durchführung

- Das Projektplanungstool "Superprojekt"
- Mittel zur Projektkontrolle (Milestone-Diagramme, Trend-Charts,...)

[37] Low Power Design - Grundlagen (Volkert Barr)
- Grundlagen zur Verlustleistung bei digitalen CMOS-Schaltungen
- Ansätze zur Verlustleistungsreduzierung

[38] Low Power Design - Architekturen (Matthias Brucke)
- Low Power Design von arithmetischen Einheiten

[39] Grundlagen der digitalen Signalverarbeitung (Arne Schulz)
- Übertragungseigenschaften zeit- und wertdiskreter Systeme
- digitale Filter (einfache Beispiele)
- optimierte Filterbänke
- Fast Fourier Transformation

[40] Funktionsweise und Aufbau des Innenohres - Technische Realisation der Basilarmembranfunktion (Burak Gökalan)
- Außenohr, Innenohr, Frequenz-Orts-Transformation auf der Basilarmembran
- Filtercharakteristiken der Basilarmembran, Nachbildung dieser Eigenschaften
- Übertragungseigenschaften der FFT, Frequenz-Abtast-Filterbank
- logarithmische Transformation der Intensitäten

[41] Analog-digital / digital-analog Wandlung und perzeptive Kodierung von akustischen Signalen (Uwe Gerken)
- A/D-Wandler, suksessive Approximation und Sigma-Delta-Wandlung
- D/A-Wandler, Oversampling
- Codierung zur Datenreduktion mit Hilfe psychoakustischer Prinzipien

Dokumente zur Projektgruppe SiliconEar

[42] Interner Bericht - Endbericht der Projektgruppe „Entwurf eines VLSI-Chips für Hörgeräte“ [EIS-97-0020-G-P]

[43] Interner Bericht - Studienarbeit „Entwurf einer integrierten Schaltung zur binauralen Verarbeitung frequenzgefilteter akustischer Signale“ (Eike Schmidt)

[44] Laufende Dokumentation des Projekts „Integrierte Cochlea“

Dokumente von ES2

[45] ES2 ecpd07 Library Databook

Anhang A:

Im Anhang A befinden sich die Strukturzeichnungen des SiliconEar. Einen Überblick bieten die Abbildungen 30 und 31. In den Abbildungen 32 und 33 wird das InputInterFace visualisiert. Die Abbildungen 34 und 35 zeigen die innere Struktur der GammaToneFilterbank.

Die Clock SlChipClk und das Signal SlReset sind der Übersichtlichkeit wegen nicht immer eingezeichnet worden. Alle getakteten Blöcke, die diese Signale benötigen sind jedoch grau hinterlegt. Die Schaltnetze wie z.B. Multiplexer sind weiß hinterlegt. Durchgezogene Pfeile in den Abbildungen geben den Datenfluß, gestrichelte Pfeile den Kontrollfluß an.

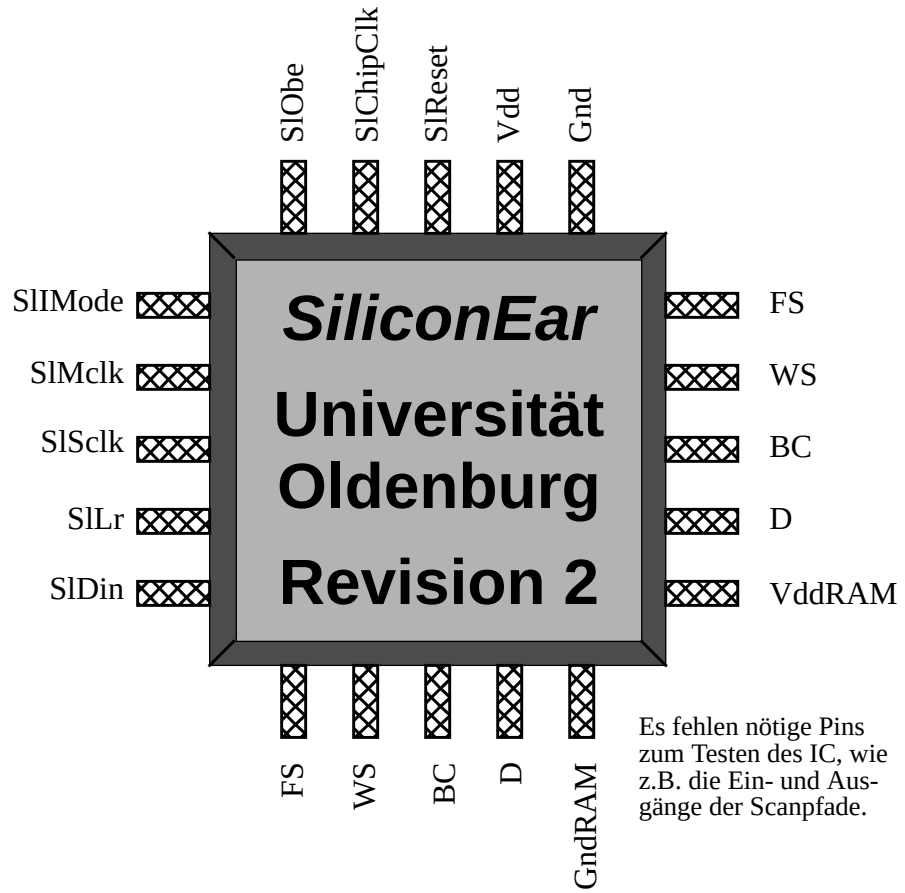


Abbildung 30: Der Chip SiliconEar

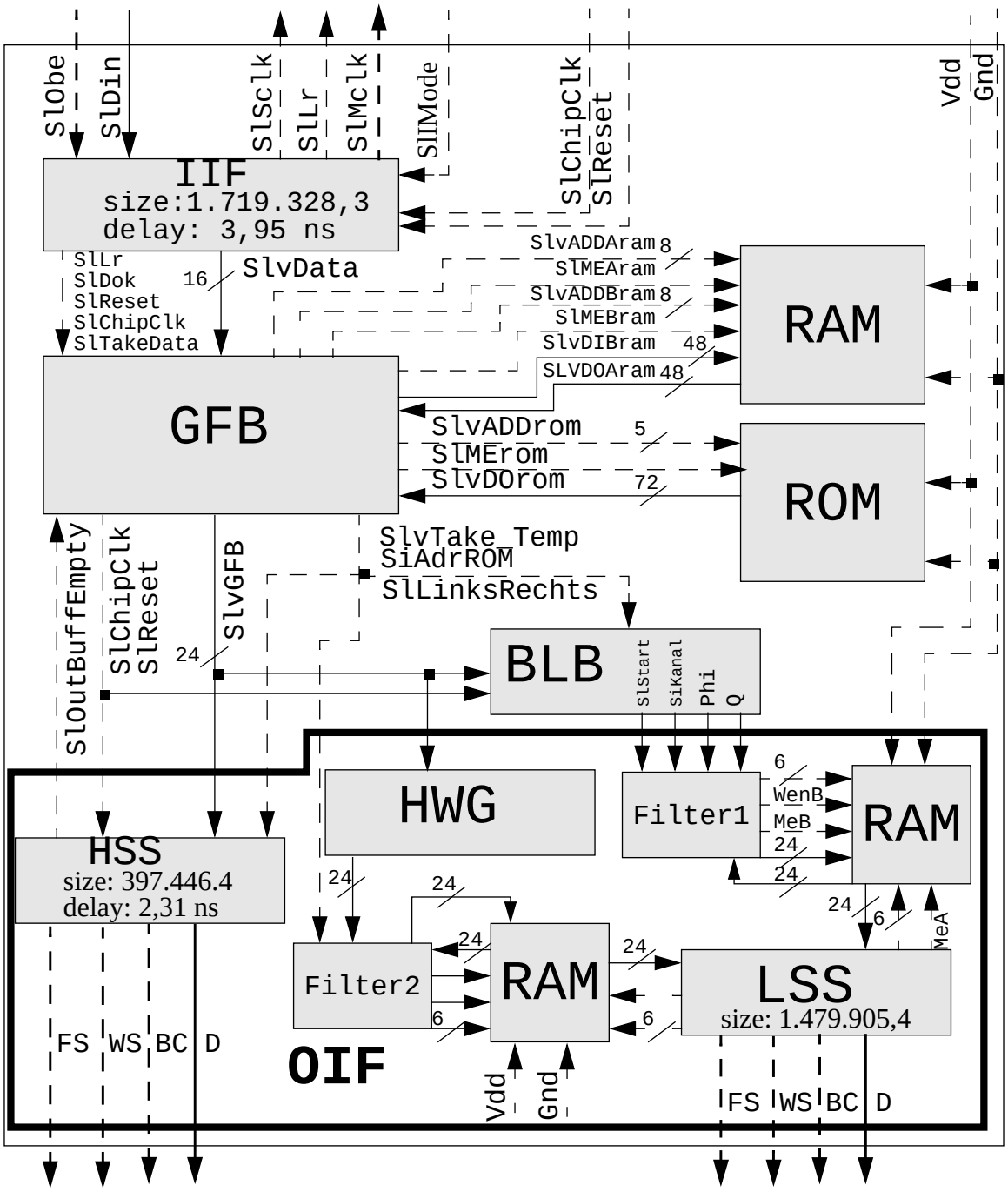


Abbildung 31: Die Struktur des SiliconEar

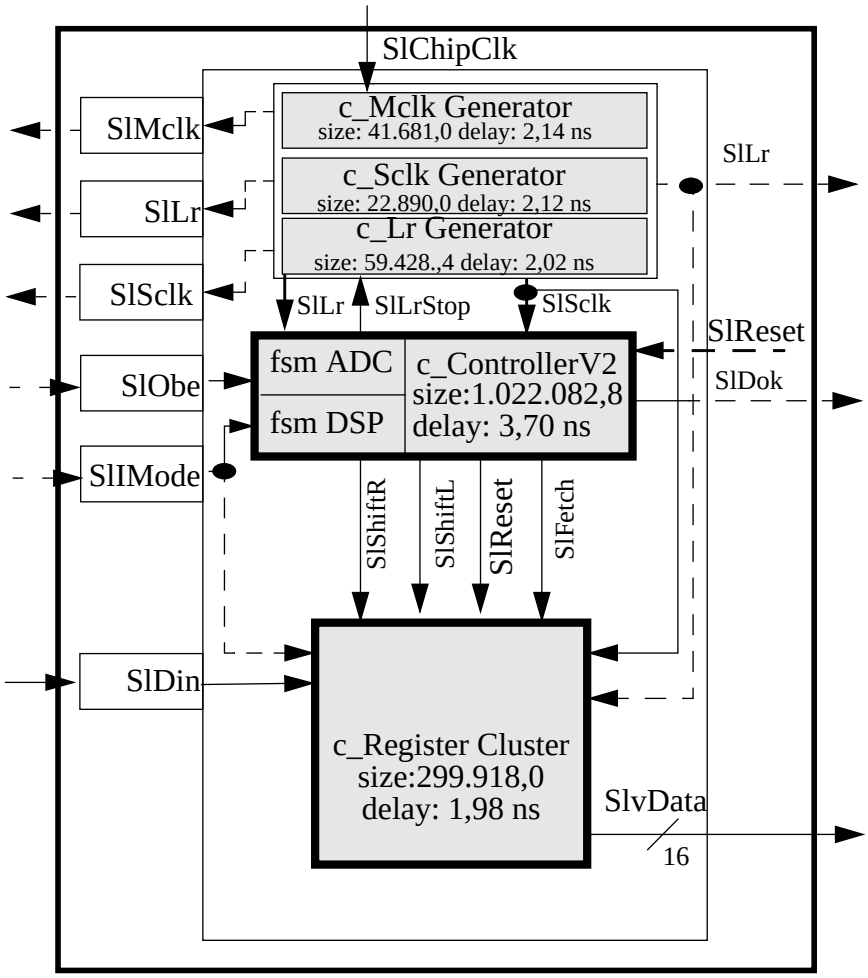


Abbildung 32: Die Struktur des InputInterFace (IIF)

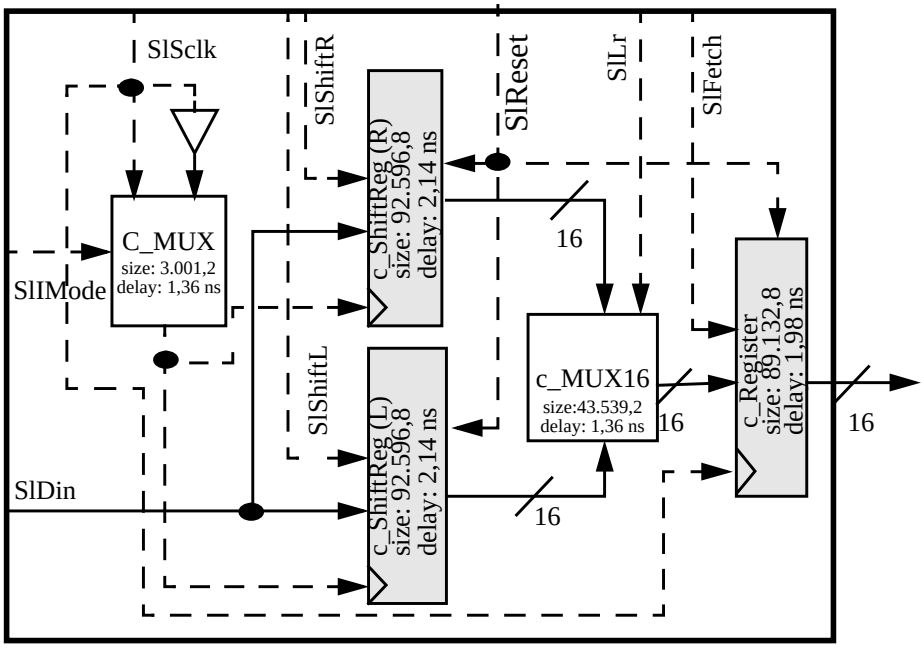


Abbildung 33: Die Struktur des c_RegisterCluster

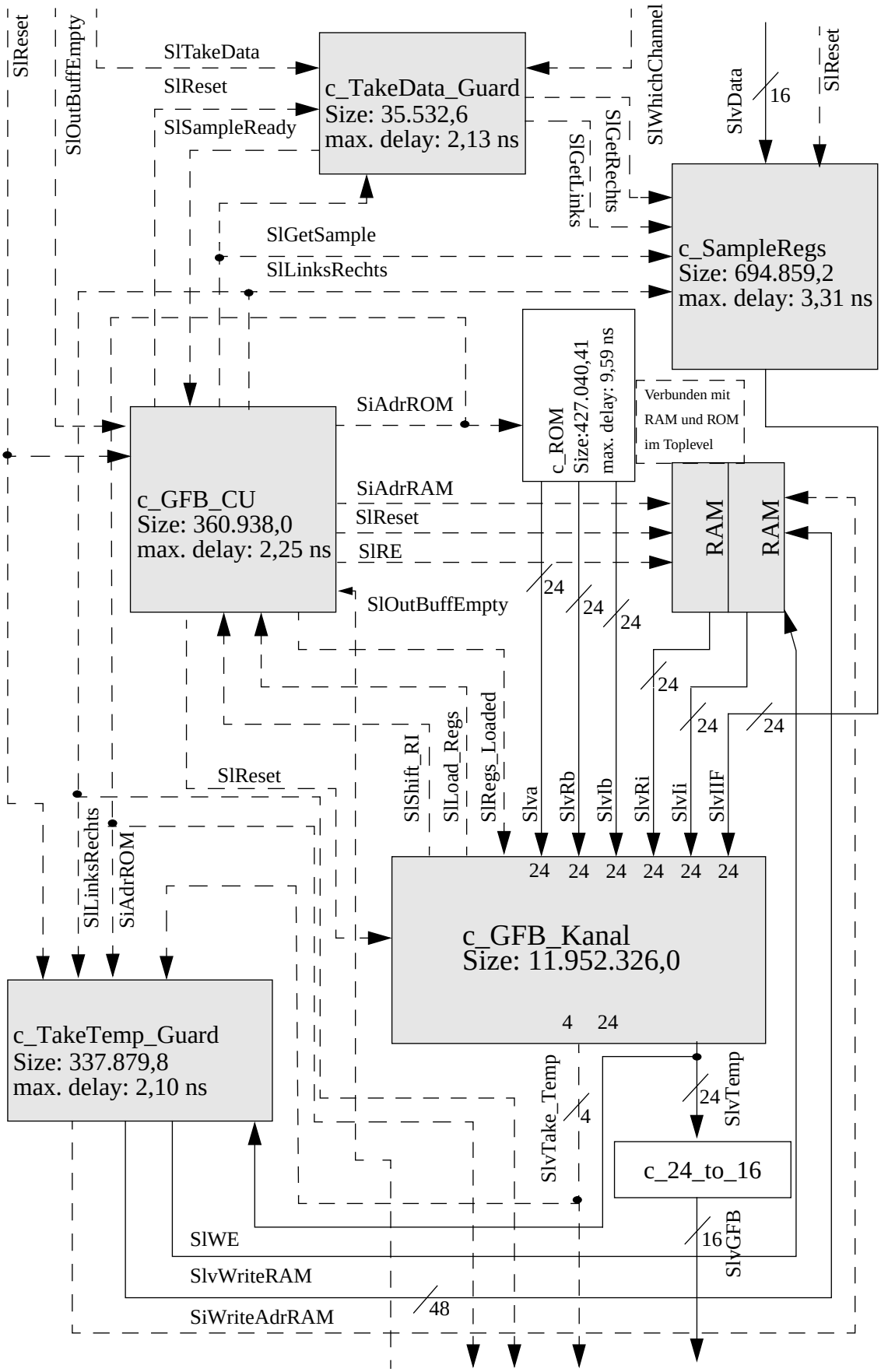


Abbildung 34: Die Struktur des Blocks GFB

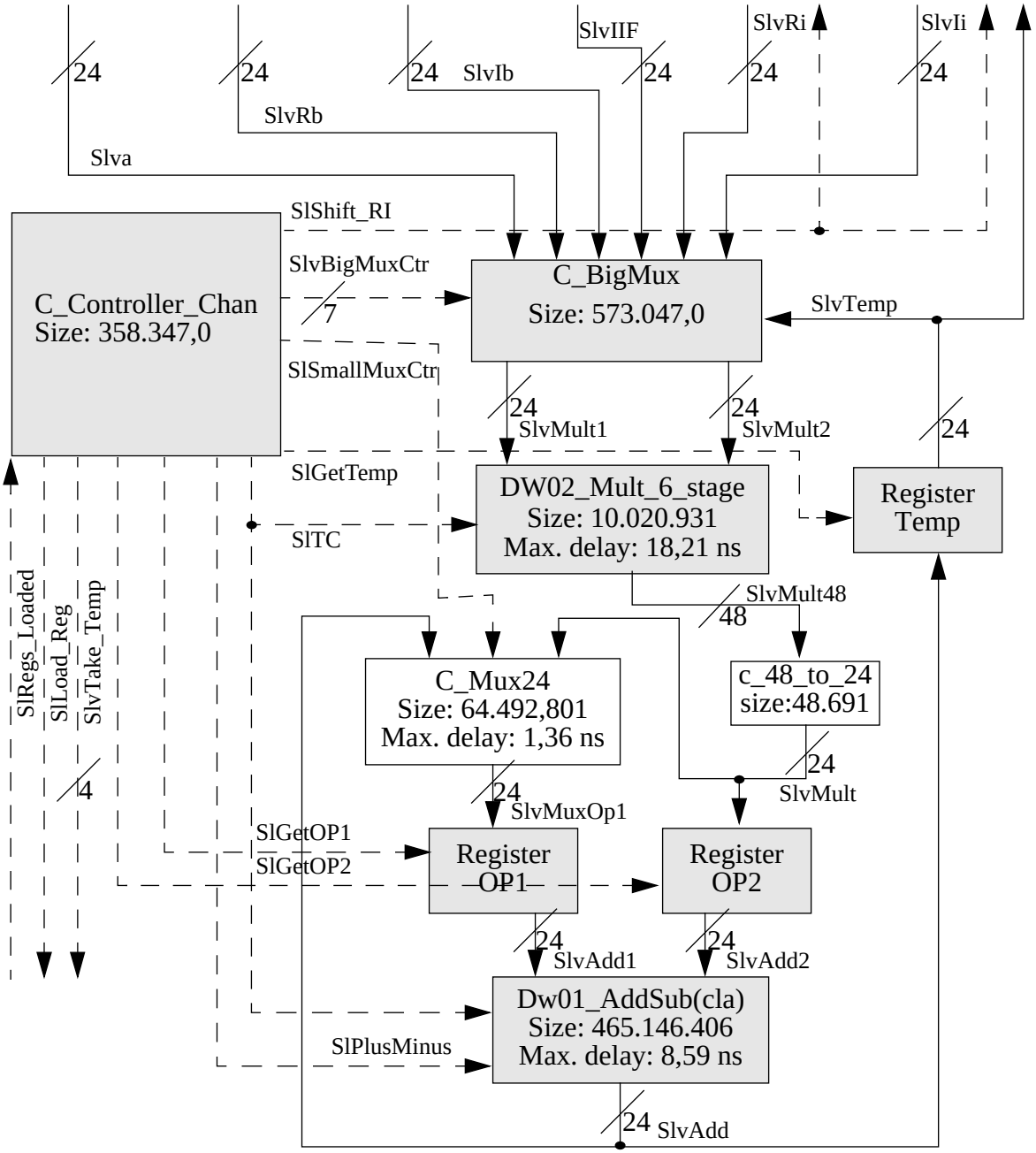


Abbildung 35: Die Struktur des GFB_Kanals

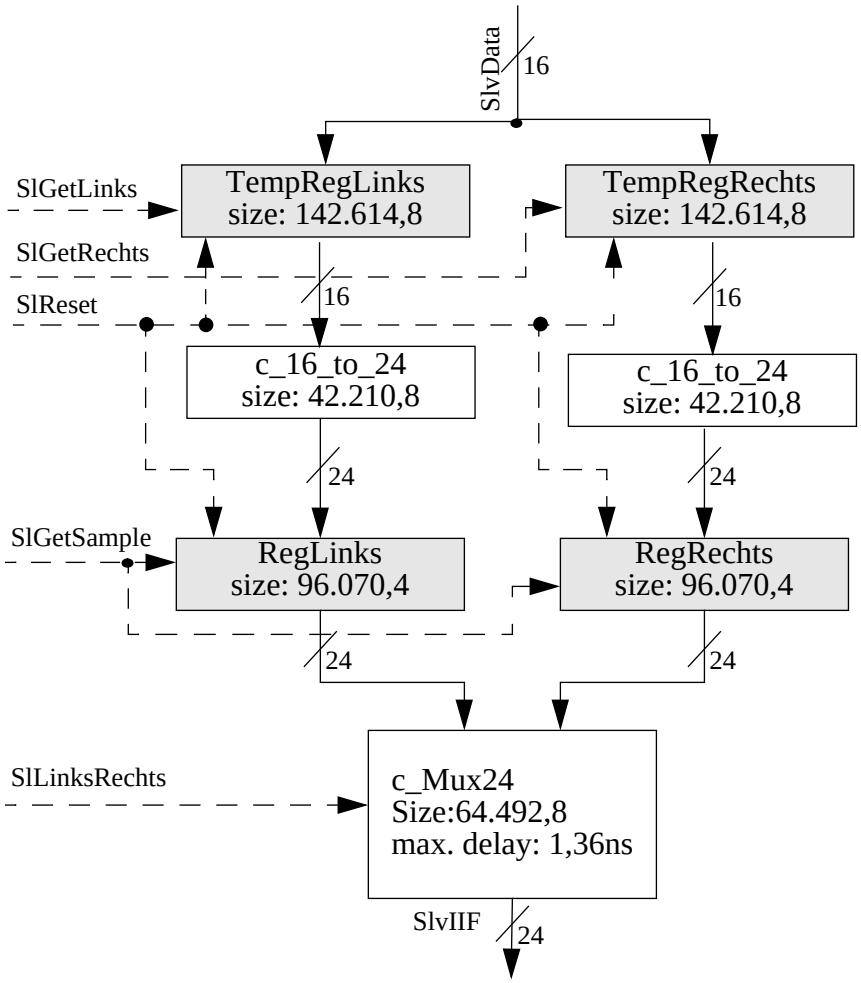


Abbildung 36: Die Struktur von c_SampleRegs

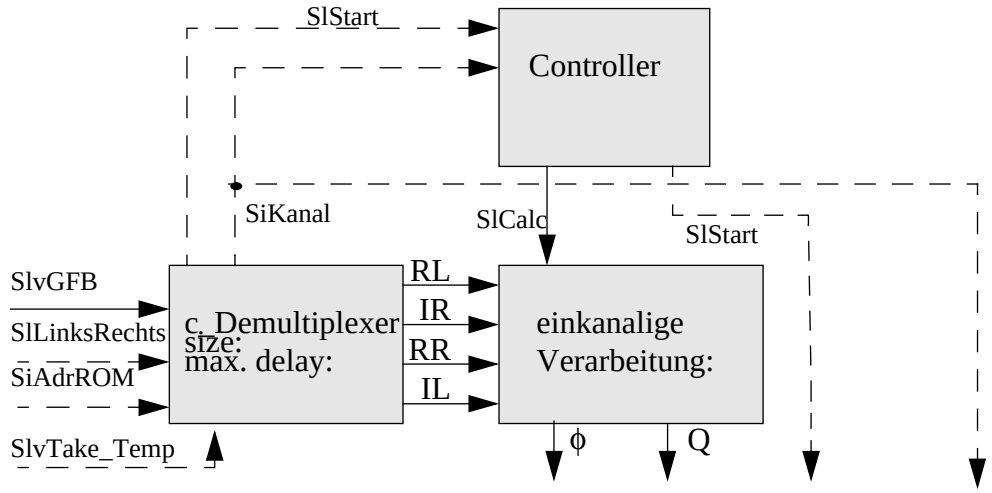


Abbildung 37: Die Struktur der BinauralenLateralitätsBewertung (BLB)

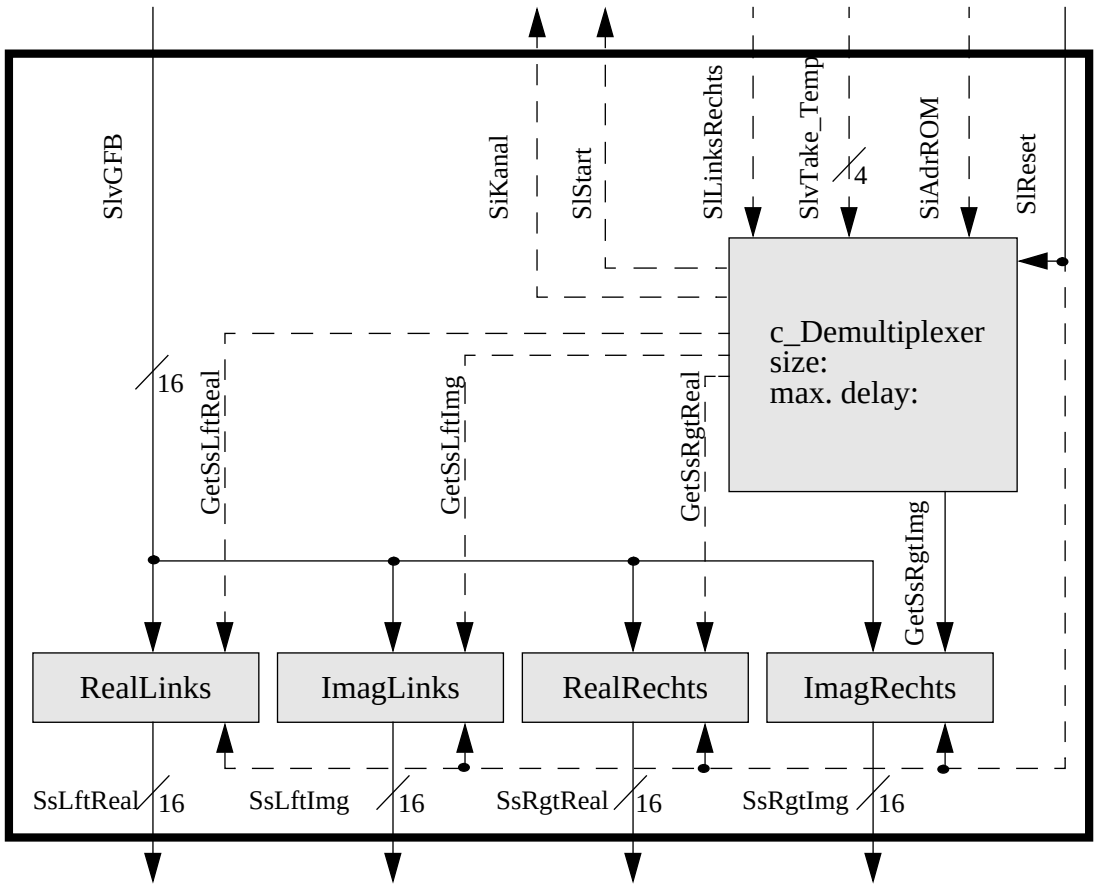


Abbildung 38: Die Struktur des c_Demultiplexer

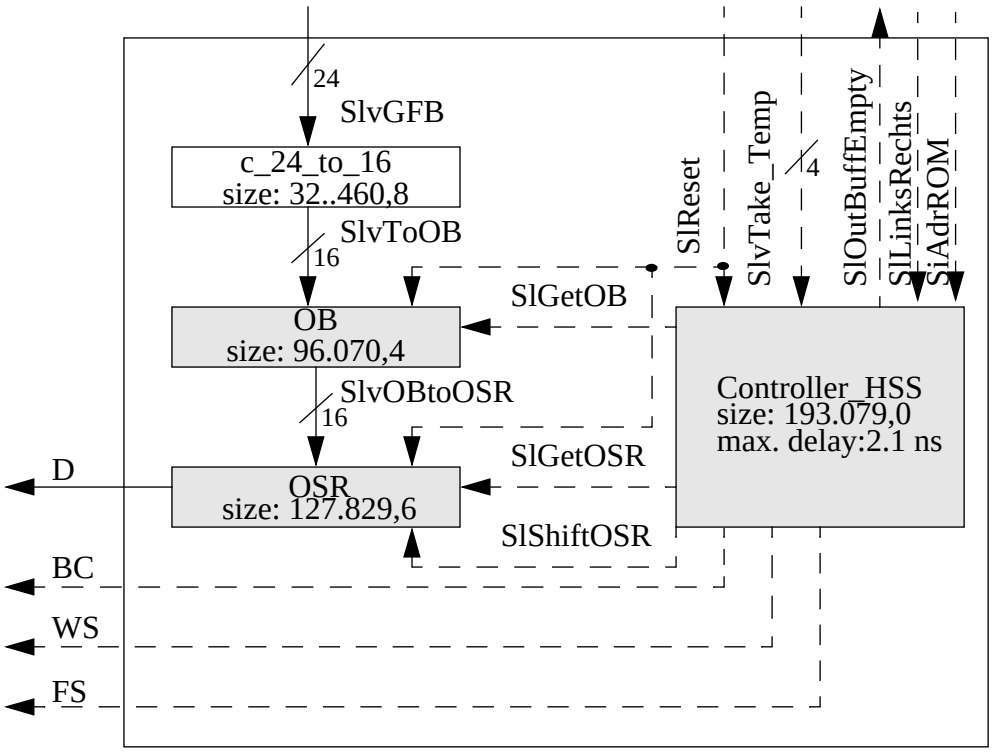


Abbildung 39: Die Struktur der HighSpeedSchnittstelle (HSS)

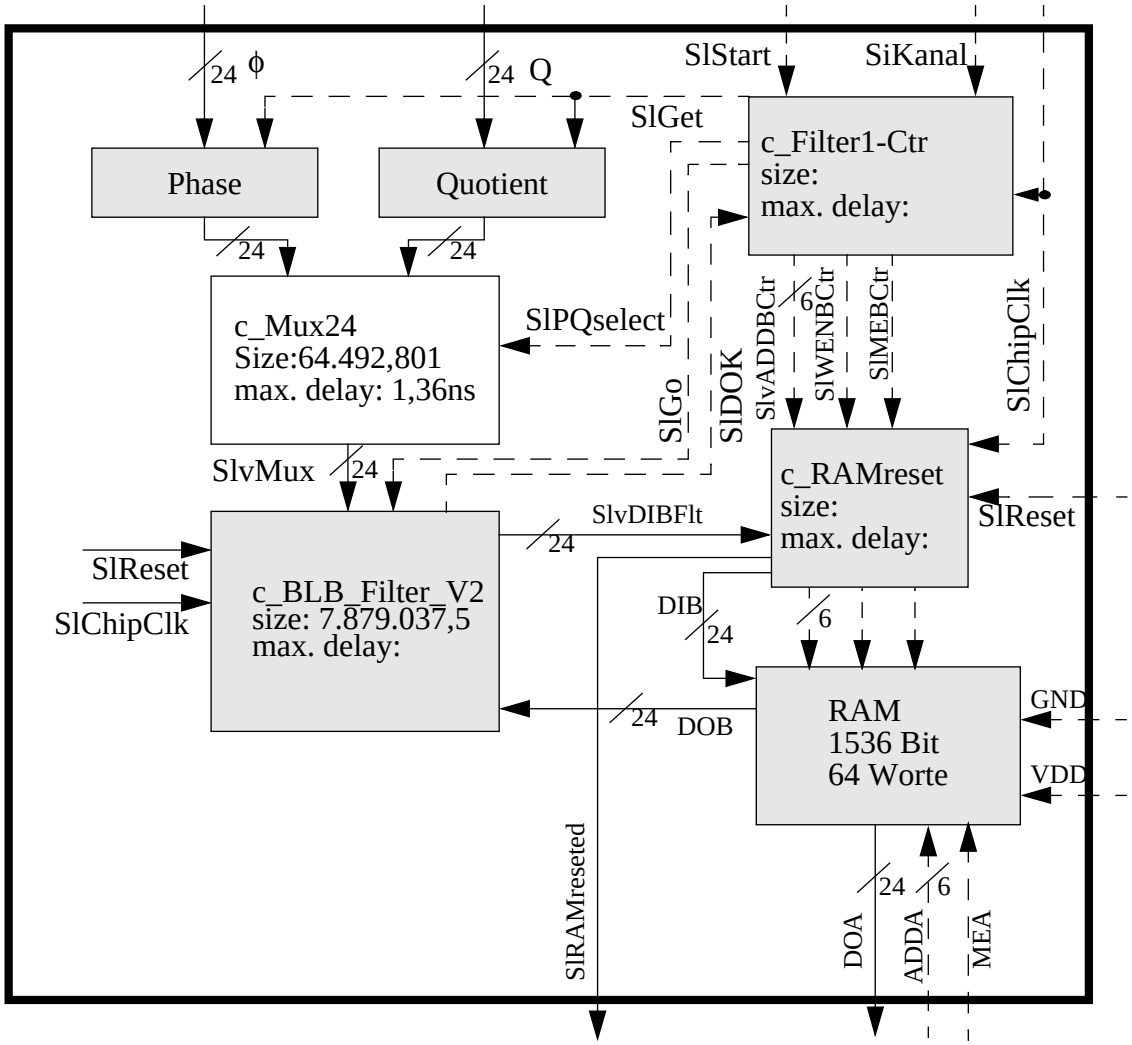


Abbildung 40: Die Struktur des Filter1

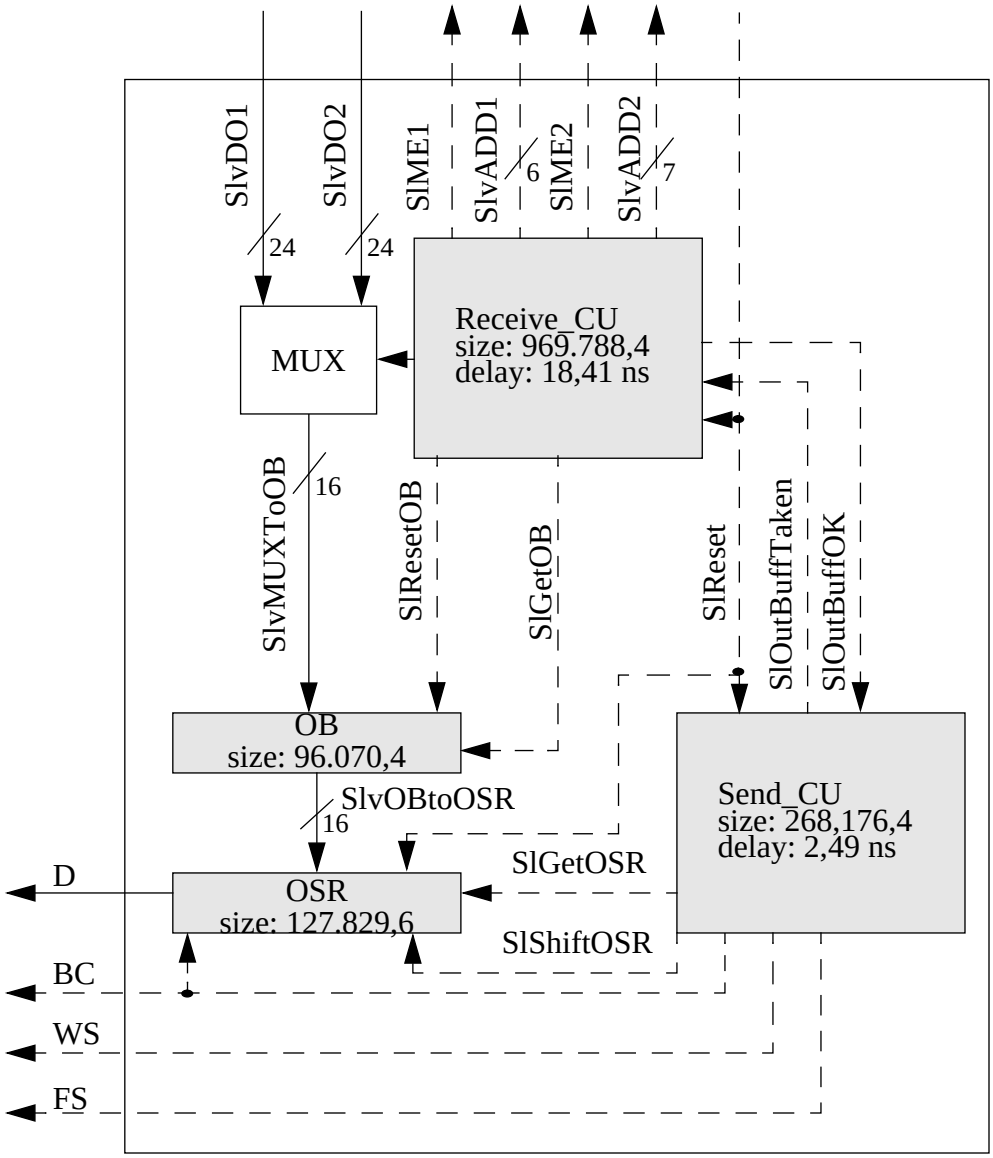


Abbildung 41: Die Struktur der LSS

Anhang B:

Filter kanal	Mitten frequenz [Hz}	Eingangs-Skalierung	Realteil der Konstanten a	Imaginärteil der Konstanten a
0	73,2370	1.27388872E-02	9.86866563E-01	2.79086113E-02
1	107,6663	1.41804384E-02	9.84968166E-01	4.09624230E-02
2	146,0196	1.57838091E-02	9.82652918E-01	5.54504576E-02
3	188,7442	1.75668511E-02	9.79826426E-01	7.15196982E-02
4	236,3383	1.95493088E-02	9.76372860E-01	8.93285810E-02
5	289,3568	2.17530036E-02	9.72150213E-01	1.09046543E-01
6	348,4179	2.42020290E-02	9.66984588E-01	1.30852917E-01
7	414,2105	2.69229591E-02	9.60663365E-01	1.54934911E-01
8	487,5017	2.99450699E-02	9.52927080E-01	1.81484288E-01
9	569,1460	3.33005702E-02	9.43459886E-01	2.10692264E-01
10	660,0957	3.70248428E-02	9.31878478E-01	2.42741951E-01
11	761,4112	4.11566910E-02	9.17719485E-01	2.77797476E-01
12	874,2739	4.57385881E-02	9.00425461E-01	3.15988658E-01
13	1000	5.08169229E-02	8.79329894E-01	3.57389773E-01
14	1140,0554	5.64422368E-02	8.53642080E-01	4.01990611E-01
15	1296,0734	6.26694402E-02	8.22433361E-01	4.49657586E-01
16	1469,8733	6.95579995E-02	7.84627285E-01	5.00082332E-01
17	1663,4817	7.71720780E-02	7.38997702E-01	5.52714908E-01
18	1879,1563	8.55806151E-02	6.84180953E-01	6.06678857E-01
19	2119,4120	9.48573193E-02	6.18711281E-01	6.60666045E-01
20	2387,0505	1.05080550E-01	5.41092431E-01	7.12811197E-01
21	2658,1925	1.15319873E-01	4.58350882E-01	7.56685798E-01
22	3017,3147	1.28701532E-01	3.44100361E-01	8.00472337E-01
23	3387,2900	1.42275952E-01	2.23125841E-01	8.28194060E-01
24	3799,4325	1.57148629E-01	8.75492049E-02	8.38292056E-01
25	4258,5483	1.73412941E-01	-6.04332343E-02	8.24374908E-01
26	4769,9909	1.91161679E-01	-2.16221799E-01	7.79402054E-01
27	5339,7242	2.10484929E-01	-3.71952859E-01	6.96408728E-01
28	5974,3920	2.31467454E-01	-5.15688790E-01	5.69830980E-01
29	6681,3948	2.54185493E-01	-6.30981631E-01	3.97619742E-01

Tabelle 6: Konstanten der GFB bei einer Ausgangsskalierung von 2

Anhang C:

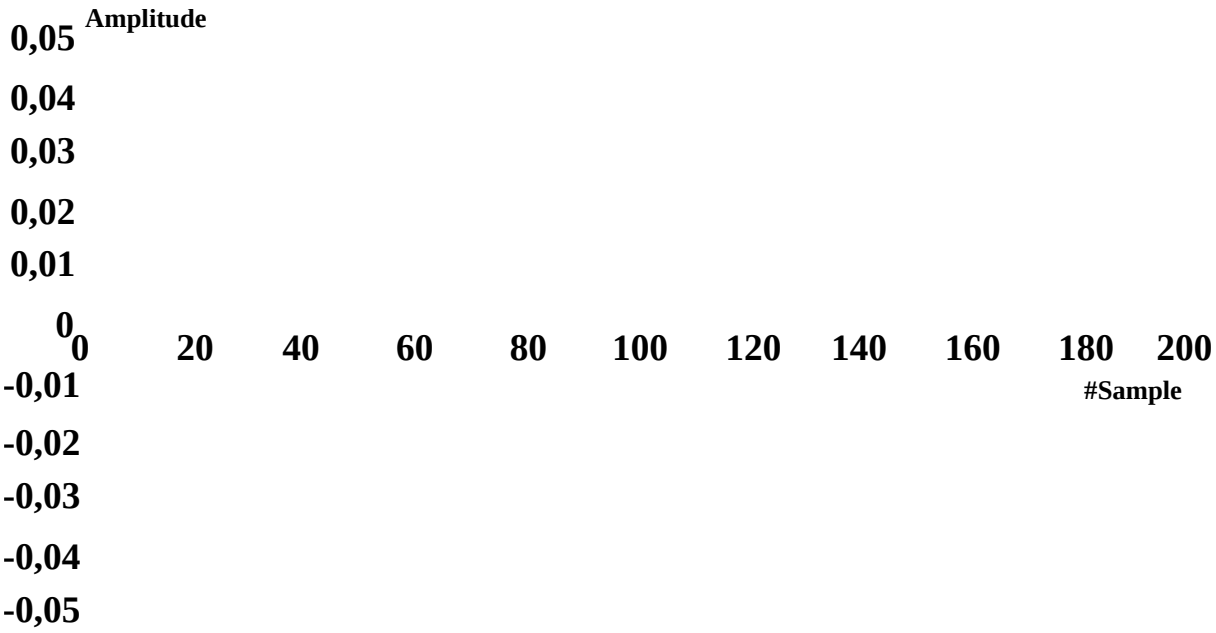


Abbildung 42: Realteil der Impulsantwort des Bandpaßfilters 13

Abbildung 43: Realteil der Impulsantwort des Bandpaßfilters 14

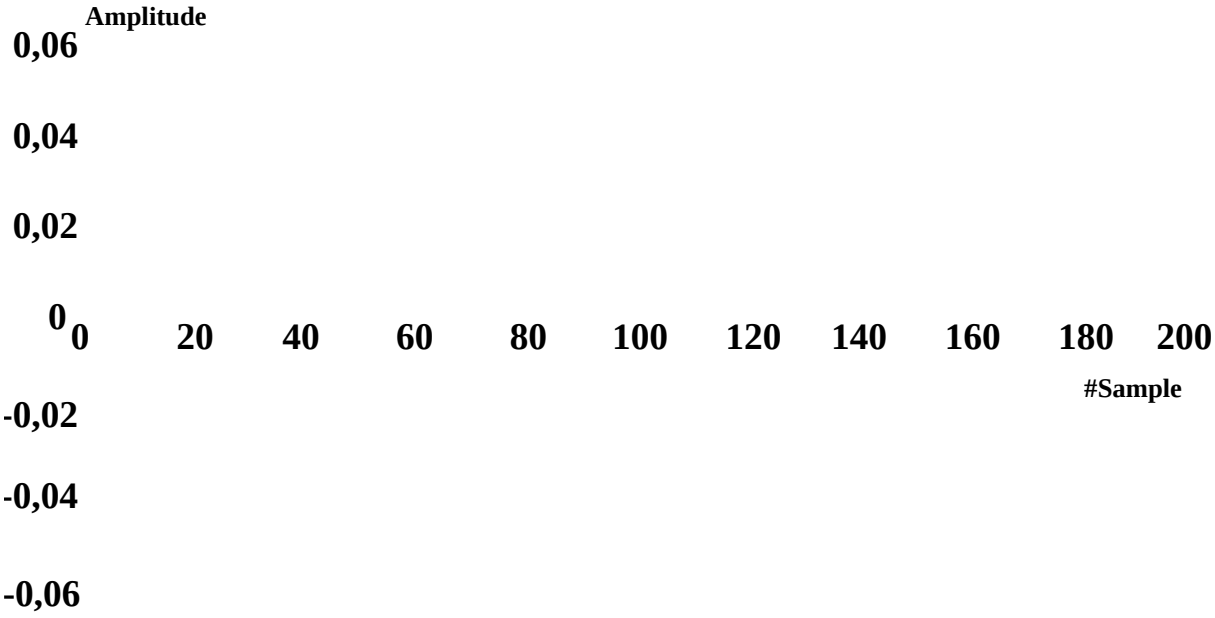


Abbildung 44: Realteil der Impulsantwort des Bandpaßfilters 15

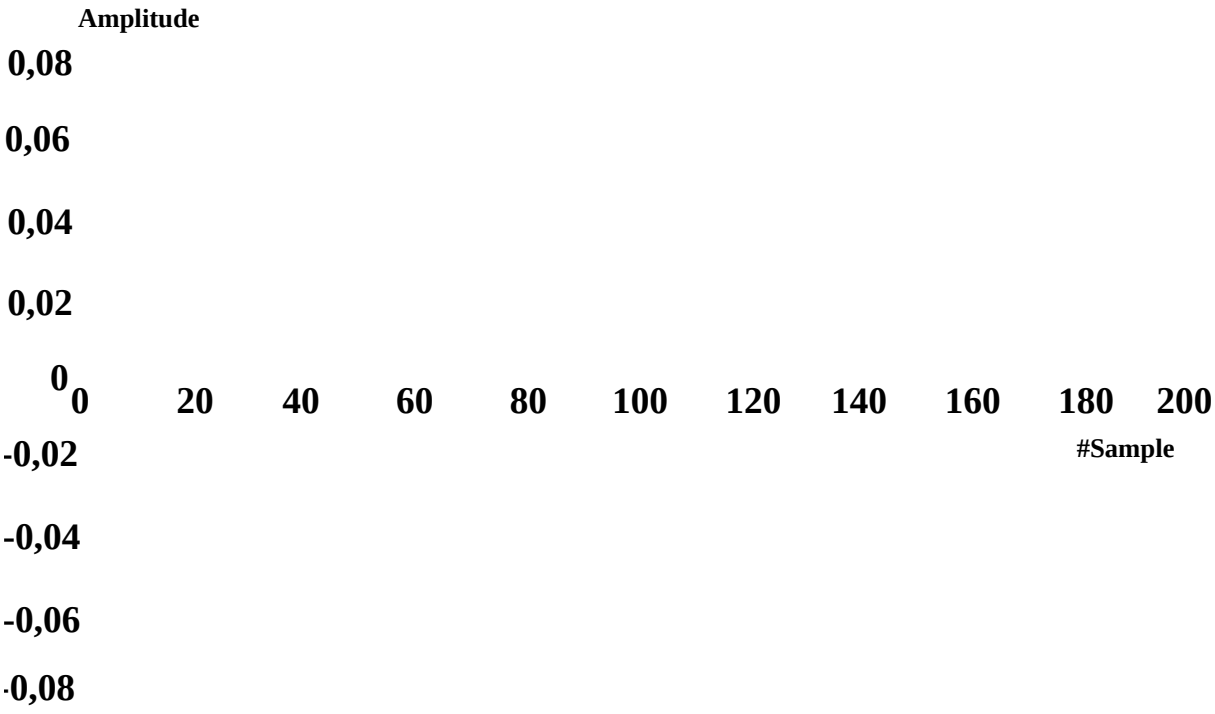


Abbildung 45: Realteil der Impulsantwort des Bandpaßfilters 16

Abbildung 46: Realteil der Impulsantwort des Bandpaßfilters 17

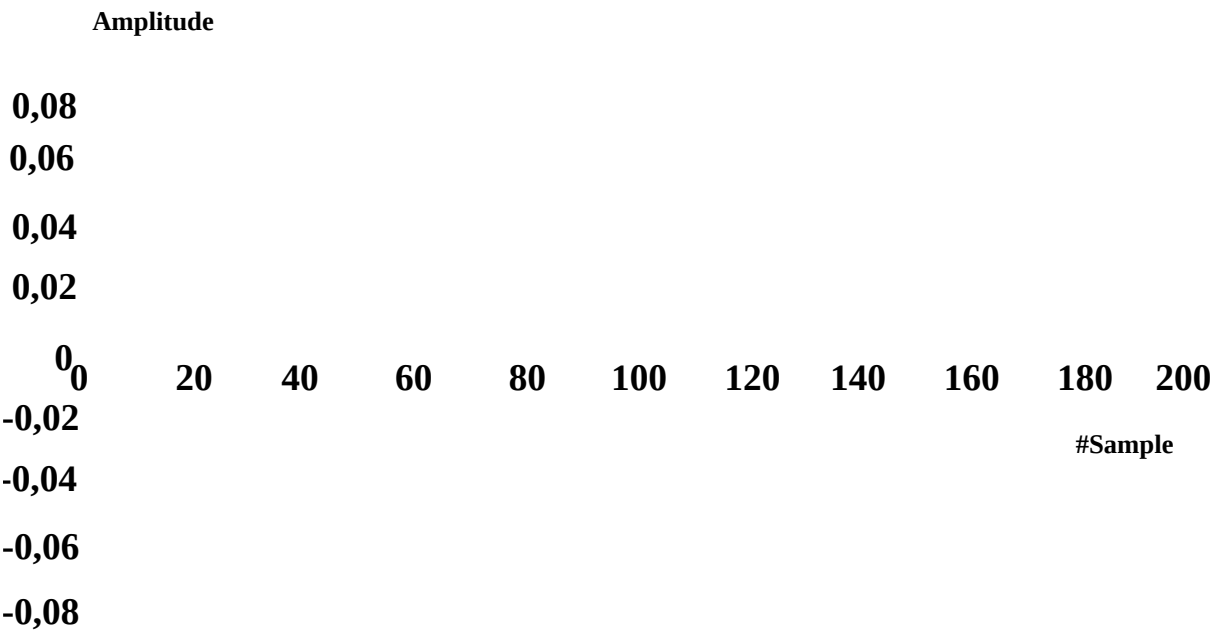


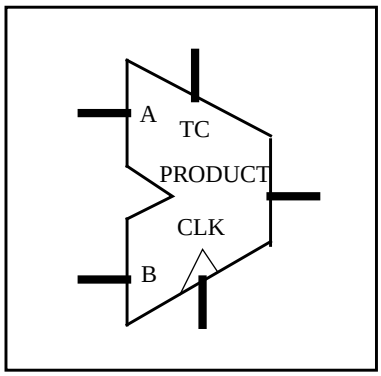
Abbildung 47: Realteil der Impulsantwort des Bandpaßfilters 18

Anhang D: DW02_mult_6_stage

Six-Stage Pipelined Multiplier

Vorzüge:

- parameterisierte Wordlänge
- vorzeichenbehaftete und vorzeichenlose Multiplikation
- zweistufige Pipelinearchitektur
- automatisches pipeline retiming
- vom Behavioural Kompiler erschließbar
- VHDL Simulationsmodel vorhanden
- Verilog Simulationsmodel vorhanden



Beschreibung

Der DW02_mult_6_stage is ein zweistufiger Multiplizierer mit Pipelinearchitektur. Er multipliziert den Operanden A mit B und erzeugt ein Produkt PRODUCT mit einer Latenszeit von fünf Taktzyklen auf CLK. Das Konrollsignal TC legt fest, ob die Eingabe und Ausgabe vorzeichenlos (TC ist 'low') oder vorzeichenbehaftet (TC ist 'high') ist.

Automatisches Pipelineretiming sichert ein optimales Plazieren von Pipelineregistern innerhalb des Multiplizierers, um einen maximalen Durchsatz zu erhalten.

Pinname	Größe	Typ	Funktion
A	A_width	Eingang	Multiplikator
B	B_width	Eingang	Multiplikant
TC	1	Eingang	Zweierkomplement-kontrolle
CLK	1	Eingang	Clock
PRODUCT	A_width + B_width	Ausgang	Produkt A * B

Tabelle 7: Pinbeschreibung DW02_mult_6_stage

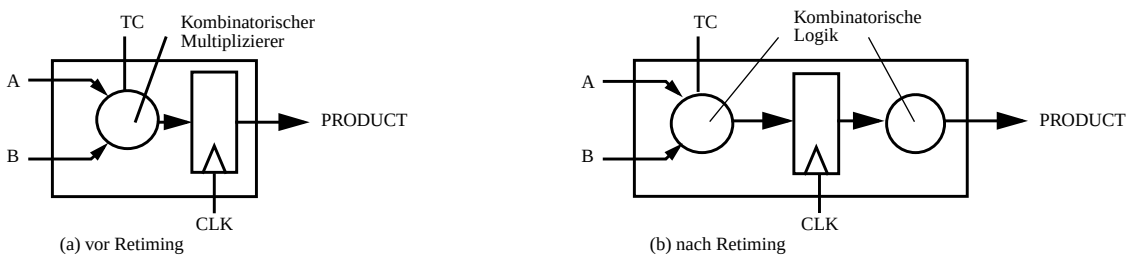
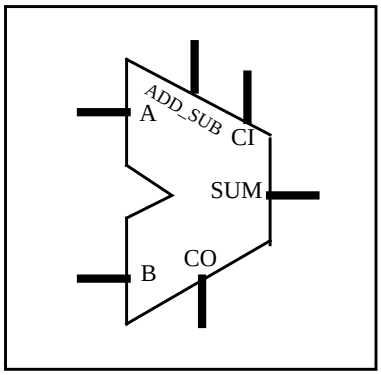


Abbildung 48: Beispiel: Blockschaltbild des mult_2_stage

DW01_addsub
Addierer-Subtrahierer

Vorzüge:

- parameterisierte Wordlänge
- Carry-in- und Carry-out-Signale
- mehrfache Implementation
- VHDL Simulationsmodel vorhanden
- Verilog Simulationsmodel vorhanden



Beschreibung

Der DW01_addsub ist ein Addierer-Subtrahierer mit zwei Eingängen. Er addiert oder subtrahiert zwei Operanden A und B mit einem carry-in CI zum Ausgang SUM mit einem Übertrag CO, welcher active high ist. Der Eingang ADD_SUB legt die Art der Operation fest. Wenn ADD_SUB ’low’ ist wird addiert. Wenn ADD_SUB ’high’ ist wird subtrahiert.

Pinname	Größe	Typ	Funktion
A	width	Eingang	Wert A
B	wdth	Eingang	Wert B
CI	1	Eingang	Carry-in
ADD_SUB	1	Eingang	Addition wenn 0 Subtraktion wenn 1
SUM	width	Ausgang	Summe/Differenz
CO	1	Ausgang	Carry-out

Tabelle 8: Pinbeschreibung DW01_addsub

Anhang E:

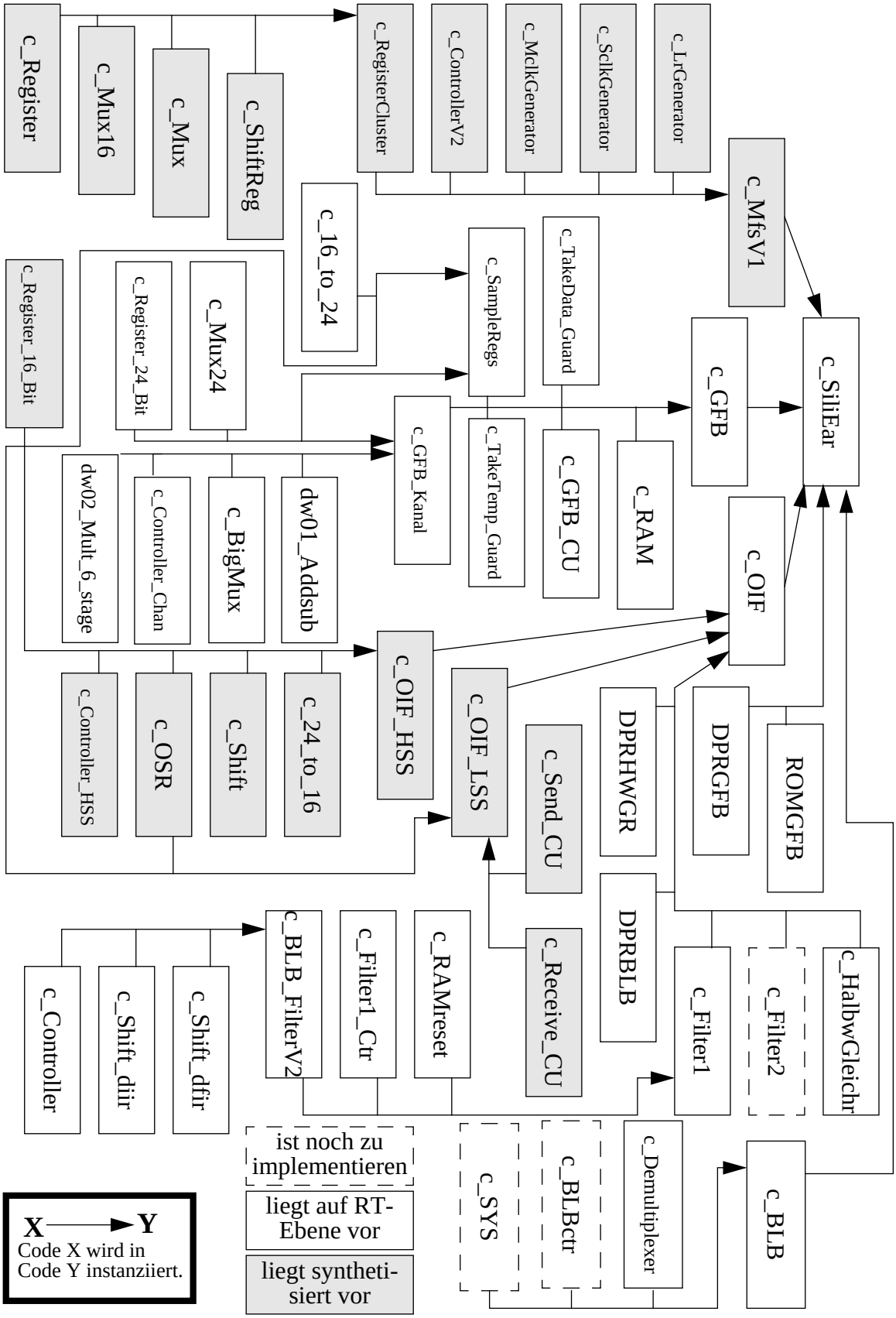


Abbildung 49: Relationengraph der Komponenten

Entity	Lines of Code	Entity	Lines of Code
c_SiliEar	390	c_MfsV1	141
c_LrGenerator	141	c_SclkGenerator	70
c_MclkGenerator	106	c_ControllerV2	353
c_RegisterCluster	182	c_ShiftReg	52
c_Mux	38	c_Mux16	39
c_Register	47	c_Register_16_Bit	49
c_GFB	393	c_RAM	140
c_TakeData_Guard	103	c_GFB_CU	284
c_SampleRegs	227	c_TakeTemp_Guard	2026
c_GFB_Kanal	334	c_16_to_24	57
c_Mux24	39	c_BigMux	144
c_Register_24_Bit	49	c_Controller_Chan	511
c_OIF	393	c_OIF_LSS	255
c_Send_CU	328	c_Receive_CU	3050
c_OIF_HSS	195	c_24_to_16	65
c_Shift	115	c_OSR	64
c_Controller_HSS	176	c_HalbwGleichr	39
c_Filter1	226	c_RAMreset	73
c_Filter1_Ctr	366	c_BLB_Filter_V2	220
c_shift_dfir	98	c_shift_diir	106
c_Controller	439	c_BLB	164
c_Demultiplexer	193	tb_c_SiliEar	237
Gesamt **ohne** Subtestbenches			12.717

Tabelle 9: Lines of Code

17 Erklärung

Hiermit versichere ich, daß ich die vorliegende Arbeit selbständig verfaßt und keine anderen als die angegebenen Hilfsmittel und Quellen verwendet habe.

_____

Datum, Unterschrift