UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Fakultät für
Elektrotechnik,
Informatik und
Mathematik

Konzeption und Bewertung eines Entwicklungsrahmenwerks zur energieoptimierenden Schaltungssynthese

zur Erlangung des akademischen Grades

DOKTORINGENIEUR (Dr.-Ing.)

der Fakultät für Elektrotechnik, Informatik und Mathematik
der Universität Paderborn
vorgelegte Dissertation
von

Nikolaus Voß
Wedel

Referentin:	Prof. Dr.-Ing. Bärbel Mertsching
Korreferent:	Prof. Dr.-Ing. Wolfgang Nebel

Tag der mündlichen Prüfung: 13.07.2010

Paderborn, den 13.07.2010

Diss. EIM-E/260

Abstract

The exponential increase of the integration density of integrated circuits in recent years has made it possible to manufacture information processing systems on a single chip (SoC, System-on-Chip). However, the trend to integrate millions of transistors has also led to higher requirements of electric power. The need for cooling as well as the relatively increasing weight and size of the batteries in embedded systems is a problem which is already a limiting constraint for portable and high performance systems.

There is also a problem on the development side: design methodologies for hardware synthesis are not yet on the same abstraction level as software design methodologies, where a paradigm shift happened with the introduction of object oriented programming. The common specification of hardware for synthesis on the register-transfer level often makes it necessary to transfer a higher level system description by hand what implies difficulties in the verification phase. Literature commonly refers to this as the "design-gap".

This work presents an approach to overcome the above problems by introducing a software framework which provides a design methodology across all abstraction levels from a high level specification of the system through architecture exploration up to the physical chip.

It has a high level of power optimization as a design goal which is achieved by three properties of the framework: Firstly, it uses commercially available, highly optimizing EDA tools as much as possible. Secondly, a module library was integrated into the framework which contains highly optimized digital signal processing modules. Thirdly, a quality measure is an integral part of the framework. The integrated measure allows a far better classification of the qualitiy grade of the output concerning human hearing of the processing system than conventional signal-to-noise ratio based classification. This allows for a energy-saving quantization of the hardware operators.

The framework is evaluated using efficiency examinations for the embedded modules. Further, case studies implementing digital signal processing chains in real hardware are presented.

Zusammenfassung

Durch die exponentielle Steigerung der Integrationsdichte von ICs ist in den vergangenen Jahren der kompakte Aufbau funktional umfangreicher Systeme auf einem Chip möglich geworden. Die Kehrseite der hohen Anzahl der Schaltelemente ist ein stark gestiegener Bedarf an elektrischer Leistung. Der nötige Abtransport der entstehenden Wärme steht dem Aufbau kompakter portabler Systeme ebenso im Wege wie die Bereitstellung der Energie durch großvolumige und schwere Batterien oder Akkus.

Ein weiteres Problem ergibt sich für den Entwurf der Schaltungen, der noch nicht auf dem Stand der Softwaretechnik ist, in der sich mit dem objektorientierten Entwurf ein Paradigmenwechsel ereignet hat: Die übliche Spezifikation auf der relativ hardware-nahen Register-Transfer-Ebene erfordert einen aufwändigen manuellen Transfer von der Systemebene, der die Verifikation des Systems zudem stark erschwert. Dies wird in der Literatur als Entwurfs-Lücke (*Design-Gap*) bezeichnet.

In dieser Arbeit soll ein Lösungsansatz für die vorstehenden Probleme auf Basis eines Software-Rahmenwerks (*Frameworks*) erarbeitet werden, das einen durchgängigen, ebenenübergreifenden Hardwareentwurf mit einer Hardwarespezifikation auf Verhaltensebene ermöglicht.

Dabei wird ein hoher Optimierungsgrad für die Verlustleistung gefordert, der durch drei Design-Charakteristika des Frameworks erzielt werden soll: Erstens nutzt das Framework so weit es geht kommerziell verfügbare, hoch optimierte EDA-Werkzeuge. Zweitens ist eine Modulbibliothek in das Framework integriert, die optimierte Komponenten für die Signalverarbeitung enthält. Drittens verfügt das Framework über ein Qualitätsmaß für Audiosignale, das bessere Vorhersagen über die durch einen menschlichen Hörer empfundene Qualität eines Signals ermöglicht als das gewöhnlich angewendete Signal-Rausch-Verhältnis; Auf diese Weise ist eine energiesparende Quantisierung der Operatoren möglich und die Verifikation wird erleichtert.

Das Software-Framework wird anhand von Beispielen evaluiert, wobei ein Teil der Beispiele Effizienz-Betrachtungen der integrierten Module sind und ein anderer Teil die Umsetzung von Audio-Signalverarbeitungs-Algorithmen als Fallstudien.

Danksagung

An dieser Stelle möchte ich allen herzlich danken, die über viele Jahre hinweg zum positiven Abschluss dieser Arbeit beigetragen haben. Das gilt vor allem für Frau Prof. Dr. Bärbel Mertsching sowie für die Mitglieder des GET-Lab und der Arbeitsgruppe IMA an den Universitäten Hamburg und Paderborn, die mir stets mit Rat und Tat zur Seite standen. Herrn Prof. Dr. Wolfgang Nebel von der Arbeitsgruppe EHS an der Universität Oldenburg danke ich für das Interesse an meiner Arbeit sowie die Übernahme des Zweitgutachtens.

Besonderen Dank möchte ich den an dem Projekt beteiligten studentischen Mitarbeitern Michael Scholz und Felix Klöckner sowie meinen ehemaligen Kollegen Thomas Eisenbach und Frank Schmidtmeier aussprechen, ohne deren Unterstützung diese Arbeit in dieser Form nicht möglich gewesen wäre.

Weiterhin danke ich meinen Mitstreitern im Projekt „PRO-DASP“ an der Universität Oldenburg in den Arbeitsgruppen MEDI (Prof. Dr. Dr. Birger Kollmeier) und EHS (Prof. Dr. Wolfgang Nebel), insbesondere Rainer Huber und Arne Schulz.

Nurdan danke ich für ihre liebevolle Unterstützung.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Symbolverzeichnis

α	Transistor Kanalsättigung
$\mathbf{w}(t)$	Hanning Fensterfunktion
$\mathcal{H}_s(z)$	Systemfunktion
ω_N	Twiddle-Faktor
v	geschätzte Rauschleistung
a_i, b_i, c_i	Filterkoeffizienten
C_L	Mittlere geschaltete Kapazität
D	Verzögerungsoperator
f_{clk}	Taktfrequenz
i	kontextabhängig: Strom oder imaginärer Anteil
I_0	Diodensperrstrom
I_0	Modifizierte Besselfunktion erster Ordnung
I_1	Modifizierte Besselfunktion zweiter Ordnung
I_{sc}	mittlerer Kurzschlussstrom bei Pegeländerungen
I_{leak}	mittlerer Leckstrom
p_t	mittlere Transitionswahrscheinlichkeit
P_{total}	mittlere Gesamtleistung
T_{d}	Gatterverzögerung
V_{t}	Schwellenspannung oder *Threshold-Voltage*
V_{dd}	Betriebsspannung
V_{GS}	Gate-Source Spannung bei FETs
V_{t}	Schwellenspannung (*Threshold-Voltage*)

1. Einleitung

Nach einer Motivation der Problemstellung im Kontext aktueller Entwicklungen in der Schaltungstechnik wird die Einbettung der Arbeit in das DFG-Forschungsprojekt PRO-DASP beschrieben.

In Abschnitt 1.1.2 erfolgt die Beschreibung der Zielsetzung dieser Arbeit, im darauf folgenden Abschnitt findet eine Gliederung statt.

1.1. Motivation

1.1.1. Elektrische Leistung: Grenzen des Wachstums

Das Problem des wachsenden Leistungsbedarfs schaltungstechnischer Systeme wird seit Anfang der 1990iger Jahre mit zunehmender Intensität bearbeitet. Es war in dieser Zeit erstmals möglich, mehrere Millionen Transistoren auf einem Chip zu integrieren[1].

Inzwischen ist die Integration von Milliarden von Transistoren möglich[2].

Obwohl mit den Strukturverkleinerungen, den Versorgungsspannungsabsenkungen und dem Wechsel von der Bipolar-Technik zur CMOS-Technik der Energiebedarf pro Transistor stark vermindert werden konnte, ist der elektrische Leistungsbedarf der Mikroprozessoren beständig gestiegen (Abb. 1.1).

Die hohe Integrationsdichte macht es zudem schwer, die thermische Verlustleistung abzuführen, da die Leistungsdichte an der Oberfläche des Silizium-Dies immer weiter gestiegen ist (Abb. 1.2).

Durch die hohe Integrationsdichte ist es möglich geworden, ganze Systeme auf einem Silizium-Die zu integrieren, sogenannte *System-On-Chips* (SOCs). Dies sind ICs, die mehrere Funktionen vereinen, die früher auf verschiedene Chips verteilt waren, z. B. Allzweck Prozessor, (USB-)Interface, DSP, Takterzeugung usw..

Diese SOCs ermöglichen den Aufbau komplexer System in einem kleinen Volumen und ermöglichen so portable Systeme. Um das Volumen des portablen Geräts nicht durch die erforderliche Kühltechnik unverhältnismäßig zu erhöhen, ist es bei diesen Geräten besonders wichtig, die thermische Verlustleistung zu begrenzen. Weiterhin konnten die portablen Energiespeicher in ihrer Entwicklung nicht mit dem Leistungsbedarf der SOCs mithalten, so dass ein beachtlicher Anteil des Volumens und Gewichts für diesen benötigt wird (siehe Abschnitt 2.3).

[1] Der im Jahr 1993 vorgestellte Intel Pentium Prozessor integriert 3,1 Mio. Transistoren bei einer Strukturgröße von 0,8 μm und einer Versorgungsspannung von 5 V.

[2] Der 2005 erschienene Intel Itanium 2 (Montecito) hat 1,7 Milliarden Transistoren bei einer Strukturgröße von 90 nm.

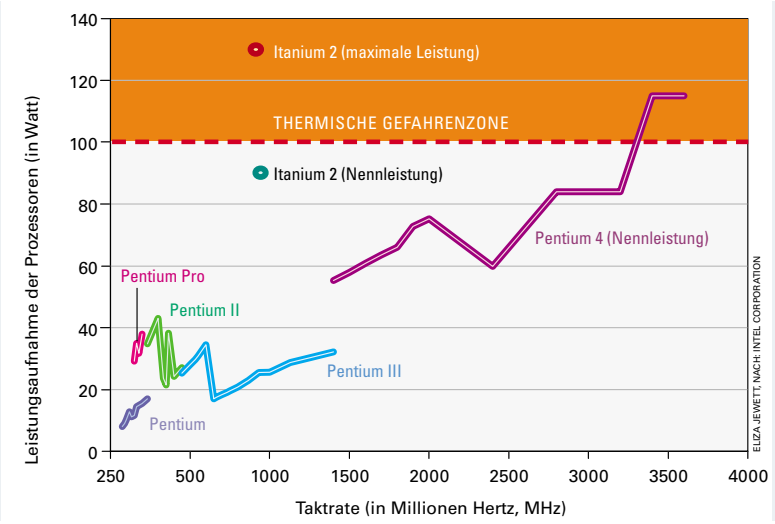


Abbildung 1.1.: Entwicklung der Leistungsaufnahme von Mikroprozessoren mit dem Takt. Der Kühlaufwand ab einer Nennleistung von 100 W ist mit gängigen Kühlsystemen aufgrund der hohen Energiedichte nicht mehr beherrschbar, weshalb im Pentium 4 und Itanium eine Taktabschaltung auf dem Siliziumchip integriert ist, um Überhitzungen zu vermeiden. Für das Design zukünftiger Prozessorgenerationen spielt die Verlustleistung eine entscheidende Rolle. Die hier dargestellten Prozessoren sind alle noch nach der alten Kostenfunktion Geschwindigkeit mal Chipfläche entwickelt, die nun an eine praktische Grenze stößt (aus [147]).

Dies gilt z. B. für aktuelle Handys, bei denen der Volumen- und Masseanteil des Akkus im zweistelligen Prozentbereich liegt, obschon die maximal erzielbare Gesprächszeit nur wenige Stunden beträgt.

Dieses Problem tritt verschärft bei Geräten aus dem medizinischen Bereich auf, deren Energiespeicher nur sehr selten gewechselt werden sollten. Dazu zählen insbesondere Implantate (z. B. Herzschrittmacher, Cochlea-Implantate, Netzhaut-Implantate).

1.1.2. Entwurfslücke

Die nach dem Moore'schen Gesetz[3] gewachsene Integrationsdichte von Siliziumchips hat neben der gestiegenen Rechenleistung und der oben erwähnten Verlustleistungsproblematik noch einen weiteren Effekt:

Die Chip-Entwicklerin und der Chip-Entwickler stehen vor der Aufgabe, im Vergleich zu früher ungleich komplexere Algorithmen auf ungleich viel mehr Transistoren als früher abbilden zu müssen. Die Entwicklungsmethodik hat sich indes seit 20 Jahren nicht grundlegend verändert: In der Regel wird integrierte Schaltungstechnik noch heute als synchroner Entwurf auf der Register-Transfer- (RT-) Ebene mittels der Sprachen VHDL oder Verilog spezifiziert[4]. Dabei ist es notwendig, das Taktverhalten und die Architektur einer Implementierung genau zu spezifizieren, ebenso wie das RT-Verhalten jeder Schnittstelle innerhalb eines hierarchischen Entwurfs.

[3] siehe Abschnitt 2.2

[4] Eine genauere Betrachtung der Sprachen und Entwurfsebenen folgt in Kapitel 2

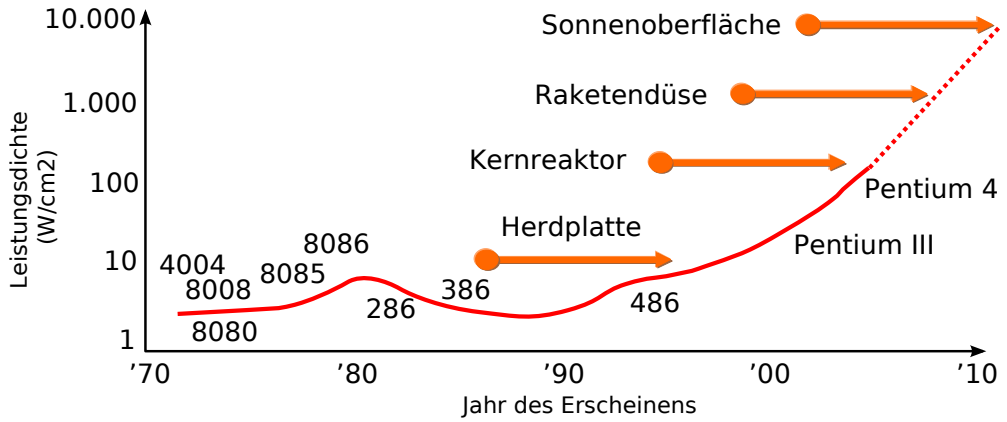


Abbildung 1.2.: Entwicklung der Flächen-Leistungsdichte am Beispiel der Intel Mikroprozessoren. Seit Anfang der 1990iger Jahre ist die Leistungsdichte an der Oberfläche des Silizium-Dies exponentiell gestiegen. Das Moore'sche Gesetz gilt demnach nicht nur für die Integrationsdichte, sondern auch für die thermische Leistungsdichte. Wird der Anstieg bis zum Pentium 4 Prozessor in die Zukunft extrapoliert (gestrichelte Linie), so würde bis 2010 die Leistungsdichte von 10 kW/cm^2 erreicht, was größenordnungsmäßig der Oberflächenleistungsdichte der Sonne entspricht.
Mit den ca. 100 W/cm^2, die der Pentium 4 (Prescott) erreicht, was ungefähr der Leistungsdichte eines Kernreaktors entspricht, scheint jedoch eine Grenze erreicht zu sein, die nicht so leicht durchbrochen werden kann. Jenseits dieses Leistungsbereichs kann die Verlustwärme aufgrund der relativ schlechten Wärmeleitfähigkeit von Silizium nicht mehr schnell genug aus dem Die abgeführt werden, so dass ein stabiler Betrieb nicht mehr möglich ist. Der Pentium D als Nachfolger des Pentium 4 hat in etwa die gleiche Leistungsdichte wie der Prescott (Quelle: Datenblätter für Intel-Prozessoren [74])

Es ist daher kaum verwunderlich, dass die Erstellung und Verifikation neuer Schaltungsarchitekturen einen hohen Zeitaufwand erfordert. Es war deshalb erforderlich, das Design von ICs mit einer hohen Anzahl von Transistoren relativ einfach zu halten. Dies wurde bei Mikroprozessoren[5] durch folgende Maßnahmen erzielt (siehe auch Abb. 1.4):

1. Seit Anfang der 1980iger Jahre werden CPU-Instruktionen nicht mehr direkt durch Schaltungstechnik dekodiert und ausgeführt, sondern die Befehle werden indirekt über den sogenannten *Microcode* spezifiziert und abgearbeitet. Der Mikrocode stellt einen kleinen Satz einfacher Befehle dar, der verwendet wird, um komplexe Befehle zu implementieren und zur Laufzeit abzuarbeiten. Das Hardware-Design der CPUs konnte damit stark vereinfacht werden, allerdings wurden für gleiche Leistung und Geschwindigkeit wesentlich mehr Transistoren benötigt[6]. Die Prozessoren wurden dadurch aus Hardwaresicht weniger komplex und konnten dadurch sehr regelmäßig aufgebaut werden (siehe auch Abb. 1.3).
2. Operationen wie das Holen, Dekodieren und Ausführen eines Befehls, die vormals in einem einzigen Taktzyklus erledigt wurden, werden in Teilschritte unterteilt. Diese Teilschritte werden jeweils in einem Takt durch dedizierte Hardware ausgeführt, so dass diese Schritte gleichzeitig abgearbeitet werden können, sogenanntes *Pipelining*. Gleichzeitig konnte die Zykluszeit verringert und damit der Durchsatz gesteigert werden. Der Pentium 4 (Netburst-Architektur) hat z. B. 20 Pipeline-Stufen.
3. Identische Komponenten wurden mehrfach auf einem Chip integriert. Dies begann mit der doppelten Ausführung der ALU (Berechnungseinheit) beim Pentium, setzte sich fort über einen doppelten *Scheduler* beim Pentium 4[7] und mündet heute in die Vervielfachung des gesamten Prozessors (Dual-/Multicore) beim Pentium D (Abb. 1.3c).
4. Ein großer Anteil der Transistoren wird für den auf dem Silizium-Die integrierten Zwischenspeicher (Cache) verwendet (Abb. 1.3b+c).

Trotz der vorgenannten Architekturverbesserungen ist ein Großteil des Leistungszuwachses moderner Prozessoren auf die Skalierung der Taktfrequenz zurückzuführen (Abb. 1.4). Da hier aber ein Limit erreicht zu sein scheint (siehe Abschnitt 2.2.2), muss für einen weiteren Leistungszuwachs der Anteil der Architektur-Verbesserungen zukünftig gesteigert werden.

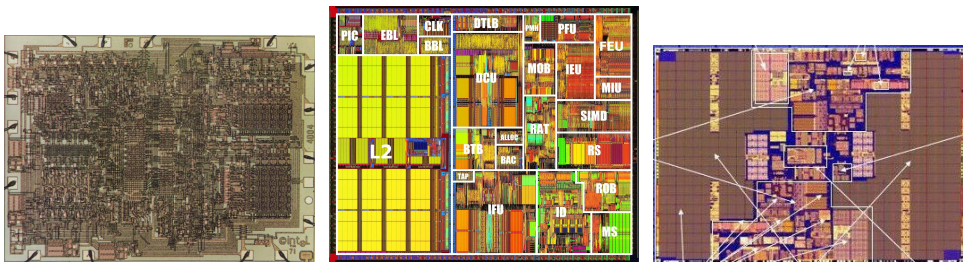
Die Entwurfslücke (Design-Gap)

Zusammengefasst lässt sich sagen, dass es in der Entwicklung integrierter Schaltungen aufgrund der stark gewachsenen Anzahl integrierbarer Elemente eine Entwurfs-

[5] Mikroprozessoren sind die komplexesten Chips die entworfen werden und deshalb ein guter Indikator auch für zukünftige Trends bei ASICs.

[6] Z.B. waren die Anfang der 1980er vorgestellten Prozessoren Motorola 68000 (68000 Transistoren, Mikrocode) und Zilog Z8000 (17500 Transistoren, hart-verdrahtet) von ihrer Leistungsfähigkeit ungefähr gleich.

[7] Dieses wurde von Intel als „Hyperthreading“ beworben. Es ermöglicht u. a. eine bessere Auslastung der arithmetischen Einheiten bei unoptimiertem Code.



(a) 1971: Intel 4004 (b) 1999: Intel Coppermine (c) 2005: Intel Montecito

Abbildung 1.3.: Fotos der Siliziumfläche einiger Prozessoren. Der Intel 4004 (a) war noch „hart-verdrahtet“, d. h. hatte keinen Mikrocode, was sich in der unregelmäßigen Struktur äußert.
Der in (b) dargestellt Pentium-III Coppermine sieht wesentlich regelmäßiger aus und lässt sich in wenige deutlich abgrenzbare Funktionseinheiten gruppieren, zudem wird ein großer Anteil der Fläche für Cache-Speicher (L2) verbraucht.
Der Itanium-2 (Montecito) (c) hat 1,7 Milliarden Transistoren und besteht aus zwei identischen „Cores“, die über eine Logik (der Kasten in der Mitte) verbunden sind. Hier dominiert der große Cache-Speicher an den Rändern links und rechts [148].

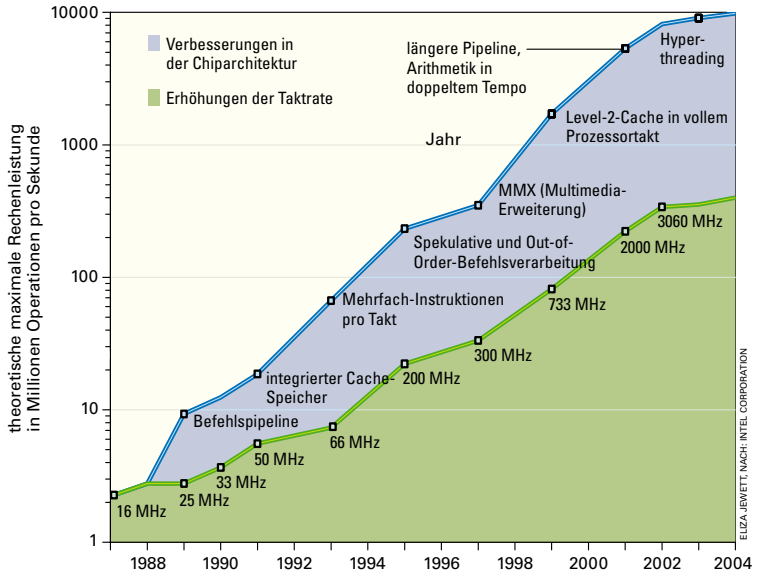


Abbildung 1.4.: Takterhöhung vs. Architekturverbesserungen. Ein Großteil der Leistungssteigerungen bei Prozessoren erwächst aus der Taktskalierung. Die in der Abbildung genannten Architekturverbesserungen sind alle relativ einfacher Natur, wie im Text beschrieben (aus [147]).

krise gibt, die Parallelen zu der Softwareentwurfskrise Ende der 1980iger aufweist; damals sind die Softwareprojekte so komplex geworden, dass die Abbildung auf eine prozedurale Programmiersprache nicht mehr effizient möglich war. Diese Krise wurde inzwischen durch die Softwaretechnik mit der Einführung der objektorientierten („strukturellen“) Programmiersprachen gelöst, die eine abstrakte Kapselung in einer modularen Struktur ermöglicht hat.

Ein solcher Paradigmenwechsel steht für die Hardwareentwicklung noch aus, obgleich seit mehr als zehn Jahren Anstrengungen unternommen werden, diesen Wechsel auf breiter Front zu ermöglichen (siehe Abschnitt 2.7).

1.2. Das PRO-DASP Projekt

Diese Arbeit ist im Rahmen des PRO-DASP Projektes entstanden, das von der DFG im Zeitraum 2000-2004 geförderte wurde. PRO-DASP (*Power Reduction for Digital Audio Signal Processing*) hatte das Ziel, Energie-optimierte Audiosignalverarbeitungssysteme in Hardware zu entwickeln. Das Projekt war in das DFG-Schwerpunktprogramms VIVA (Grundlagen und Verfahren verlustarmer Informationsverarbeitung) eingebettet.

Ziel von PRO-DASP war die Erarbeitung und Formulierung einer Entwurfsmethodik, die eine ebenenübergreifende Optimierung von der System-Spezifikation auf der Verhaltensebene bis hin zum fertigen platzierten und verdrahteten Chip ermöglichen sollte.

In diesem Rahmen sollten hochoptimierte Verarbeitungskomponenten für die digitale Signalverarbeitung erstellt werden, weiterhin sollten verlustleistungsoptimierende Entwurfsentscheidungen auf der Architekturebene erarbeitet und umgesetzt werden.

Als Beispielanwendung sollte eine optimierte Resynthesefilterbank erstellt werden (siehe Kapitel 5).

Das Projekt wurde im Verbund mit zwei Arbeitsgruppen an der Universität Oldenburg durchgeführt:

1. AG MEDI: Die Kernkompetenz der AG MEDI unter der Leitung von Prof. Dr. Dr. Birger Kollmeier ist die interdisziplinäre Forschung auf den Gebieten der Audiologie und des menschlichen Hörens sowie digitalen Hörgeräten und Sprachverarbeitung und -erkennung im Rahmen der Erkenntnisse der medizinischen Physik. Die AG MEDI entwickelte die in Hardware umzusetzenden Algorithmen sowie ein Audio-Gütemaß, das einen Kernbestandteil des in dieser Arbeit entwickelten Entwurfs-Frameworks darstellt.
2. AG EHS: Die Abteilung Eingebettete Hardware-/Software-Systeme unter der Leitung von Prof. Dr. Wolfgang Nebel beschäftigt sich mit der Schnittstelle zwischen Hard- und Software. Der für PRO-DASP ausschlaggebende Arbeitsschwerpunkt ist die Analyse und Optimierung von integrierten Schaltungen auf hoher Abstraktionsebene. Die AG EHS hat mit ORINOCO im Rahmen des Projekts ein Werkzeug zur Verlustleistungsschätzung und -optimierung entwickelt, das in das in dieser Arbeit entwickelte Entwurfs-Framework (Kapitel 4) integriert werden kann (siehe Abschnitt 3.8.).

Die für diese Arbeit durchgeführten Untersuchungen wurden an der Universität Hamburg in der AG IMA (Informatikmethoden für Mikroelektronikanwendungen) unter der Leitung von Prof. Dr. Bärbel Mertsching im Zeitraum von 2000-2003 durchgeführt. Seit April 2003 leitet Frau Mertsching die Arbeitsgruppe GET Lab an der Universität Paderborn, wo das PRO-DASP-Projekt bis 2004 fortgeführt wurde. Die Arbeitsgruppe hatte dabei eine Scharnierfunktion zwischen den beiden anderen Partnerinnen (Abb. 1.5), da sie über ein großes Maß an Erfahrung bei der Entwicklung und Implementierung digitaler Schaltungstechnik verfügt[8].

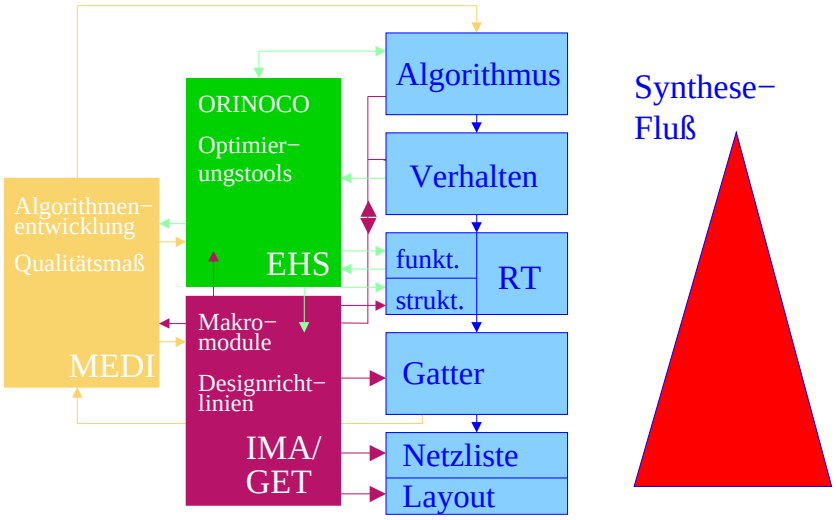


Abbildung 1.5.: Struktur des PRO-DASP-Projekts. Während MEDI das Gütemaß und Audio-Algorithmen entwickelt hat, wurde bei EHS die Entwicklung des High-Level Power-Schätz- und Optimierungswerkzeugs ORINOCO vorangetrieben. Bei IMA/GET wurden die Algorithmen umgesetzt und ebenenübergreifend optimiert.

1.3. Zielsetzung der Arbeit

Die im vorigen Kapitel beschrieben Zielsetzung des PRO-DASP-Projektes ist sehr weit gefasst.

Die hohe Komplexität des viele Abstraktionsebenen übergreifenden modernen Hardwareentwurfs eröffnet zwar auf der einen Seite zahlreiche Ansatzpunkte für eine Energieoptimierung, auf der anderen Seite ist es aufgrund der erläuterten Entwurfslücke schwer, eine durchgängige Methodik von der Spezifikation des Signalverarbeitungssystems bis hin zum fertigen Chip zu entwickeln.

Diese Arbeit umfasst daher sowohl die Erarbeitung einer angepassten Entwurfsmethodik unter Anwendung verfügbarer Komponenten, Werkzeuge und Methodiken; zusätzlich wird eine Erweiterung des sich daraus ergebenden Entwurfsflusses mit eigenen Komponenten vorgenommen, deren Untersuchung aus wissenschaftlicher Sicht ebenfalls interessant ist.

[8] Ein kurze Beschreibung der PRO-DASP-Arbeiten findet sich in [99]

Die aus der Analyse für diese Arbeit herausdestillierte Lösungsstrategie ist die Kapselung eines energieoptimierten Entwurfsflusses in ein *Software-Framework*. Dieses Framework soll die Durchgängigkeit des Flusses garantieren, indem es die Entwicklerin und den Entwickler in eine bestimmte Entwurfsmethodik zwingt.

Weiterhin wurde eine für Signalverarbeitungsalgorithmen konzipierte, mehrstufige Makromodulbibliothek in das Software-Framework integriert. Sowohl die Makrokomponenten als auch die durch das Framework implementierte Methodik werden im Rahmen dieser Arbeit beschrieben und sowohl theoretisch bewertet wie anhand von Fallbeispielen praktisch evaluiert.

Ziel der Arbeit ist, einen Weg aufzuzeigen, wie mit den gegenwärtig in Forschung und Industrie verbreiteten (und hochoptimierenden) EDA-Werkzeugen ein **durchgängiger, auf allen Entwurfsebenen Energie-optimierender High-Level Entwurfsfluss mit integrierter Modulbibliothek** erzielt und praktisch verwendet werden kann. „High-Level" bedeutet eine Spezifikation des zu implementierenden Algorithmus' auf einer höheren Abstraktionsebene als der verbreiteten RT-Ebene. „Durchgängigkeit" bedeutet eine konsistente Berücksichtigung und Verknüpfung aller Abstraktionsebenen während der Implementierung.

1.4. Vorgehensweise und Gliederung der Arbeit

Die Arbeit beginnt mit einem Grundlagenteil (Kapitel 2). Dort werden die in dem entwickelten Framework verwendeten Techniken und Verfahren vorgestellt und in ihre Nutzung innerhalb des Frameworks eingeordnet. Das bedeutet auch, dass die vorgestellten Verfahren innerhalb einer Klasse verglichen und bewertet werden; an jeden Abschnitt schließt sich ein Bewertungsteil an. Weiterhin werden Bezüge sowohl zum Framework, als auch zu aktuellen Arbeiten gemacht, die sich mit diesem Aspekt befassen.

Eine Recherche über internationale Forschungsaktivitäten in Bezug auf Entwicklungssysteme mit einer ähnlichen Zielsetzung erfolgt in Kapitel 3. Dort werden Systeme vorgestellt und gegeneinander abgegrenzt, die vor dem Entwicklungsframework entwickelt wurden und die Struktur und Implementierung des Frameworks maßgeblich beeinflusst haben. Neben einer Bewertung für jedes der vorgestellten Systeme schließt sich ein Resümee an, dass die Eigenschaften der Systeme zusammenfasst und gegen das in dieser Arbeit entwickelte Framework abgrenzt.

Der Kern der Arbeit ist die Vorstellung des Entwicklungs-Frameworks (im folgenden auch oft Software-Framework genannt) in Kapitel 4. Das Kapitel gliedert sich in Abschnitte, die nach einem Überblicksabschnitt jeweils einen Bestandteil des Entwicklungsframework genauer beschreiben. Zum Abschluss des Kapitels werden die wesentlichen Punkte zusammengefasst.

Das Kapitel 5 widmet sich einigen Beispiel-Anwendungen aus der Audiosignalverarbeitung, die die Anwendung des Entwicklungs-Frameworks illustrieren und jeweils einige Aspekte des Frameworks in den Fokus fassen.

In der abschließenden Diskussion wird der Beitrag der Arbeit zusammengefasst und

bewertet. Ferner werden die aufgetretenen Probleme kurz benannt und ein Ausblick auf mögliche Folgearbeiten umrissen.

Im Anhang sind in einem kleinen Glossar einige der verwendeten Fachbegriffe erläutert. Eine Stichwortsuche ist über den davor befindlichen Index möglich.

2. Grundlagen

In diesem Kapitel werden die der Arbeit zugrunde liegenden Prinzipien und Technologien erläutert. Zunächst wird der grundsätzliche Energiebedarf für das Speichern und Verarbeiten von Informationen aus der Physik motiviert. In Abschnitt 2.2 wird ein Überblick über die Technologien für die Hardwareentwicklung mit ihren Vor- und Nachteilen gegeben. Darauf folgt eine Bewertung in der Forschungsphase befindlicher Alternativtechnologien.

Dann folgt mit Abschnitt 2.3 eine kurze Darstellung der aktuellen Situation bei mobilen Energiespeichern.

Die Eingabesprachen für den automatisierten Hardwareentwurf werden in Abschnitt 2.4 vorgestellt.

In Abschnitt 2.5 erfolgt eine detaillierte Analyse des Energiebedarfs der CMOS-Halbleitertechnologie, die derzeit den Massenmarkt immer deutlicher beherrscht.

Die Abstraktionsebenen im Hardware-Entwurf werden in Abschnitt 2.7 dargestellt.

Im folgenden Abschnitt 2.8 wird auf verschiedene Zahlensysteme und Zahlendarstellungen eingegangen.

Der abschließende Abschnitt 2.9 dieses Kapitels skizziert die Theorie der digitalen Audiofilter, die für diese Arbeit eine zentrale Rolle spielen.

2.1. Energiebedarf informationsverarbeitender Systeme

Eine interessante Frage ist, ob Informationsverarbeitung in Form von logischen Operation (die die Grundlage für arithmetische Operationen sind) notwendigerweise mit Dissipation, d. h. Verlustleistung, verbunden ist. Diese Frage ist nicht leicht zu beantworten.

Tatsache ist, dass bis heute keine *theoretische* untere Grenze für den Energiebedarf bekannt ist[1]. Die Theorie der *reversiblen Berechnung* (Reversible Computing) widmet sich der Suche nach einer solchen Grenze. Dabei geht es um physikalisch reversible Prozesse, d. h. Prozesse, die keine Entropie erzeugen und damit keine Dissipation haben.

Eine ausführliche Abhandlung über physikalisch reversible Berechnungstheorien ist z. B. in [45] zu finden.

[1] Die von-Neumann-Landauer Grenze von $k_b T \ln 2$ gilt für nicht-reversible Systeme, also z. B. die heute gängige CMOS-Technik.

2.2. Technologien für den Schaltungsentwurf

In diesem Abschnitt werden für die Realisierung digitaler Schaltungen benutzbare physikalische Technologien beschrieben. Der erste Teil gibt einen Überblick über Halbleitertechnologien, die bereits im kommerziellen Einsatz sind. Der zweite Teil ab Abschnitt 2.2.3 geht knapp auf alternative Entwicklungen ein, die momentan Gegenstand der Forschung sind.

2.2.1. Überblick

Mit einem Welthandelsvolumen von 250 Milliarden Dollar in 2007 [2] ist die Halbleiterbranche innerhalb kurzer Zeit zu einem großen Wirtschaftsfaktor geworden. Dies hatte zur Folge, das erhebliche Summen in die Fortentwicklung und Optimierung der bestehenden Technologien investiert wurden, weshalb neue Konzepte am Markt sehr schwer Fuß fassen können.

Annähernd alle integrierten Schaltungen basieren daher heute auf der Silizium-Halbleitertechnologie, nicht zuletzt weil sie durch die Massenfertigung bei hoher Leistungsfähigkeit sehr preiswert ist. Die Gründe für den Erfolg und den aktuellen Stand der Technik sind unten umrissen.

2.2.2. Halbleitertechnologien

Historischer Überblick

Das Fundament für die Entwicklung der heute in industriellen Prozessen verwendeten Halbleitertechnologie wurde zu Anfang des letzten Jahrhunderts durch die Entdeckung der Halbleitereigenschaften von elementarem Silizium durch Pickard 1906 gelegt. Die sich zu dieser Zeit rasant entwickelnde physikalische Festkörperforschung führte dazu, dass W. Schottky und N. F. Mott 1938 unabhängig eine Theorie zum Metall-Halbleiter-Kontakt veröffentlichten, der phänomenologisch bereits 1874 durch F. Braun beschrieben wurde. Auf Grundlage dieses Effekts wurde 1948 der von W. Shockley, Bardeen und Brattain entwickelte Bipolartransistor vorgestellt, der die Grundlage für die erste integrierte Schaltung bildete, die 1958 von J. S. Kilby bei Texas Instruments entwickelt wurde. Bereits ein Jahr später wurde die Silizium-Planartechnologie bei Fairchild entwickelt, die eine kostengünstige Massenfertigung von Halbleiterbauelementen ermöglichte. 1970 wurde dann von D. Kahn der MOS-Transistor entwickelt, der strukturell einfacher war, von der Geschwindigkeit jedoch lange Zeit nicht mit der Bipolartechnologie mithalten konnte.

Mit der Entwicklung der Mikroprozessoren, die 1974 mit dem 4004 von Intel begann, setzte dann eine beispiellose industrielle Optimierung der siliziumbasierten Entwicklung integrierter Schaltungen ein, die noch heute von dem von Moore aufgestellten Gesetz einer Verdoppelung der Zahl der Transistoren alle 18 Monate geprägt wird [100]. Für den 1997 vorgestellten Pentium-II-Prozessor wurde erstmals aus Verlustleistungsgründen keine Bipolartechnologie mehr eingesetzt, der Chip war ausschließlich in CMOS-Technologie gefertigt.

Silizium

Das Element Silizium mit der Ordnungszahl 14 ist die Basis der Halbleitertechnologie. Es ist nach Sauerstoff das zweithäufigste Element der Erdkruste. Durch die Hintereinanderschaltung von verschiedenen Reinigungstechniken kann es in eine hochreine, einkristalline Form gebracht werden, die nur noch die thermodynamisch bedingten Störstellen, aber keine nachweisbaren Fremdelemente mehr enthält. Es hat damit die höchste jemals in einem industriellen Prozess erreichte Reinheit und eignet sich damit hervorragend für die Herstellung sehr kleiner Strukturen [20].

Trotz der im Vergleich zu anderen Halbleitern relativ schlechten physikalischen Eigenschaften von Silizium wie schlechter Wärmeleitung und niedriger Elektronenbeweglichkeit, ist es mittlerweile gelungen, mehrere hundert Millionen Transistoren bei einem Maximaltakt von mehreren Gigahertz zu realisieren.

So stellt der seit Dezember 2004 erhältliche Pentium 4 mit 3,8 GHz vermutlich für die nächste Zeit den Prozessor mit dem höchsten Takt dar, eine angekündigte Version mit 4 GHz hat Intel Anfang 2005 offiziell abgesagt[2]. Bis heute (September 2008) hat Intel keinen höher getakteten Prozessor im Angebot. Der Prozessor hat eine Strukturbreite von 90 nm und ist aus 125 Mio. Transistoren aufgebaut. Die mittlere Verlustleistung bei Volllast[3] beträgt 115 Watt, die maximale Stromaufnahme liegt bei 119 A [73].

Die Skalierung zu kleineren Strukturen hatte bisher zwei Vorteile:

1. Die Transistordichte nimmt zu, so dass mehr Funktionalität auf gleicher Fläche untergebracht werden kann.
2. Durch die kleineren Strukturen war es nötig, die Versorgungsspannung V_{dd} abzusenken, damit die elektrische Feldstärke im p- bzw. n-Kanal des Transistors sich nicht erhöht[4].

   Dadurch sinkt der dynamische Energiebedarf pro Transistor, der zu V_{dd}^2 proportional ist (siehe Abschnitt 2.5). Damit der Transistor dadurch nicht langsamer wird, muss die Schwellenspannung V_{t} ebenfalls abgesenkt werden.

Letzterer Punkt kehrt sich zunehmend zu einem Nachteil, da die notwendige V_{t}-Skalierung für stark erhöhte Leckströme sorgt (Abb. 2.1). Weiterhin wird die Isolierlage zwischen Gate und Kanal dünner, so dass der resultierende Tunnelstrom einen beträchtlichen Anteil an der statischen Verlustleistung bekommt.

Letztendlich sorgt die Integration von Milliarden von Transistoren trotz der Einsparungen pro Transistor für eine stark gestiegene Gesamtleistungsaufnahme (Abb. 2.2), die durch geeignete Kühlmaßnahmen abgeführt werden muss.

[2] Das P4-Prescott-Design war ursprünglich so konzipiert, dass als Grenzfrequenz sogar 5 GHz angestrebt wurden.

[3] Dieser Wert wird „Thermal Design Power“ (TDP) genannt. Das Kühlsystem muss in der Lage sein, dauerhaft diese Wärmeleistung vom Prozessor abzuführen

[4] Das elektrische Feld E im Kanal zwischen Source und Drain ist von der Kanallänge l und der Drain-Source-Spannung V_{ds} abhängig: $E = V_{ds}/l$, d. h. die Feldstärke steigt mit kürzerem Kanal (kleinerer Strukturbreite). Ist das Feld zu hoch, kann es (bei gesperrtem Transistor) das durch das Verarmungsgebiet zwischen Source und Kanal erzeugte Feld kompensieren, wodurch der Transistor nicht mehr sperren kann.

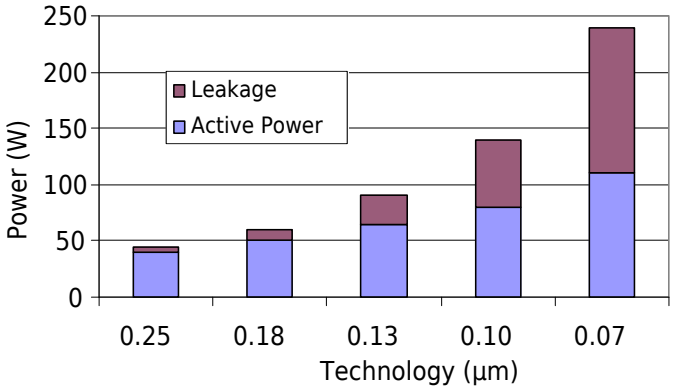


Abbildung 2.1.: Entwicklung von statischer und dynamischer Verlustleistung mit der Strukturgröße. Kleinere Strukturen führen zu einer starken Erhöhung der statischen Verlustleistung. Neben durch die V_t-Absenkung bedingten Unterschwellströmen (siehe Abschnitt 2.5) kommt es zu einem Tunnelstrom durch die sehr dünne Isolationslage zwischen Gate und Kanal (aus [89]).

Mit kleineren Strukturen wird die *Prozessvariabilität* zunehmend zu einem Problem. Der Gültigkeitsbereich der klassischen Elektrodynamik ist mit aktuellen Prozessen bereits verlassen, hier spielen Quanteneffekte eine zunehmende Rolle. Relativ geringe Unterschiede im Prozess haben jetzt große Einflüsse auf Geschwindigkeit und Energiebedarf der Schaltung.

Stand der Technik ist momentan die sogenannte *synchrone CMOS-Logik*. Dabei sind die Logikgatter aus Paaren komplementärer Transistoren aufgebaut (siehe Abschnitt 2.5.1). Zustandswechsel werden dabei durch ein periodisches Taktsignal ausgelöst.

Verbesserungen der Silizium-Technologie

Die Silizium-basierte CMOS-Technologie unterliegt ständigen Verbesserungen, die moderne Chips mit ihren geringen Strukturbreiten und Milliarden von Transistoren erst möglich gemacht haben.

Die wesentlichen Verbesserungen sind im folgenden zusammengefasst[5]:

Silicon-on-Insulator (SOI) Bei diesem Prozess wird an der Grenzschicht zwischen dotiertem und undotiertem Silizium eine Isolationsschicht (typischerweise $Si0_2$) eingebracht, dadurch sinkt die effektive Kapazität an der Grenzschicht zu dem undotierten Trägermaterial aus Silizium (*bulk material*) erheblich. Dies macht Energieeinsparungen im Betrieb von bis zu 50 % bei gleichzeitiger Senkung der Transistorverzögerung möglich [71].

gestrecktes Silizium (*Strained Silicon*) Hier wird durch Einbringen einer Fremdatomlage der Abstand zwischen den Siliziumatomen der funktionalen Lage gestreckt,

[5]Es handelt sich hierbei um eine Auswahl der wichtigsten Verbesserungen. Für die Fertigung der heute gängigen Strukturen von 70 nm waren außerdem unzählige Prozessverbesserungen und -optimierungen nötig [157].

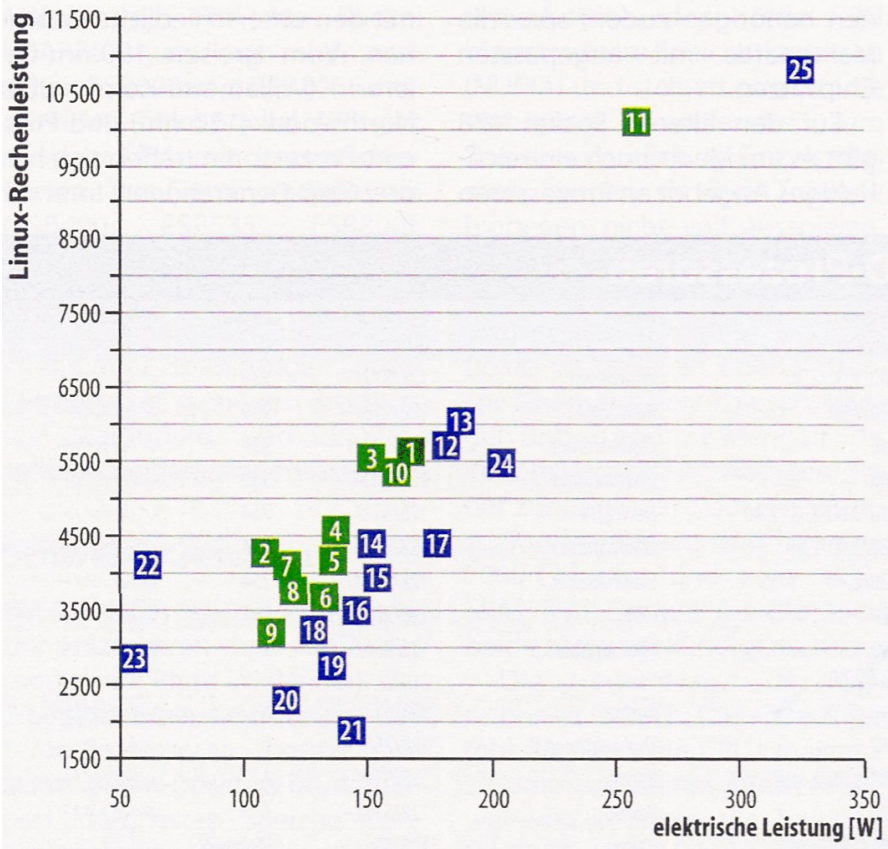


Abbildung 2.2.: Übersicht über die Leistungsaufnahme und Rechenleistung aktueller Mikroprozessor-Systeme. Grün dargestellt sind Prozessoren von AMD, blau von Intel. Erwähnenswert sind die beiden Prozessoren links in der Grafik, der Pentium M (22) bzw. Celeron M (23). Diese haben eine sehr verlustleistungsarme Architektur. Bei 11 und 25 handelt es sich um Zweiprozessor-Systeme, die Werte sind die Verdoppelungen von 10 und 24. 1-4: Athlon 64, 5+6: Athlon XP, 7-9: Sempron, 10: Opteron 250, 11: DualOpteron 250, 12-17: Pentium 4, 19-21: Celeron, 22: Pentium M, 23: Celeron M, 24: Xeon, 25: DualXeon (aus [16]).

wodurch die Elektronenbeweglichkeit um ca. 70 % erhöht wird. Dies führt zu signifikanten Steigerungen der Transistorgeschwindigkeit bei gleichzeitiger Reduktion der ohmschen Verluste [86].

Dielektrika mit hoher Dielektrizitätskonstante (*High-κ-Dielectrics*) werden als Isolationsmaterial zwischen Gate und Kanal eingesetzt. Um die erforderlichen hohen Feldstärken im Kanal zu erreichen, ist es für sehr kleine Strukturen notwendig, eine SiO_2-Isolierlage < 2 nm (2-4 Atomlagen) einzubringen, was einen sehr hohen Tunnelstrom und damit eine hohe statische Verlustleistung zur Folge hat. Um die Isolationslage dicker ausführen zu können, ohne die Feldstärke im Kanal zu senken, muss ein Dielektrikum mit einer höheren Dielektrizitätskonstante als der von Silizium eingesetzt werden. Hier kommt z. B. HfO_2 zum Einsatz, was den Prozess stark verkompliziert, für eine weitere Skalierung aber unausweichlich erscheint [174].

Dreidimensionale Transistorstrukturen ermöglichen den kompakteren Aufbau eines Transistors als die Planartechnologie (2D-Strukturen) bei gleichzeitig geringerer Gate-Kapazität. Ein Beispiel ist der FinFET [126].

Gallium-Arsenid

Gallium-Arsenid ist ein III-V Verbindungshalbleiter. Es hat gegenüber Silizium den Vorteil einer wesentlich höheren Elektronenbeweglichkeit, was Taktfrequenzen von bis zu 250 GHz ermöglicht, ferner ist das Rauschverhalten deutlich besser.

Die Nachteile sind eine wesentlich kostenintensivere Produktion und vor allem die gegenüber Silizium niedrigere Löcherbeweglichkeit, was den Aufbau von p-FETs erschwert, die für die CMOS-Technik benötigt werden.

Deshalb werden GaAs-ICs fast ausschließlich in NMOS-Technik gefertigt, was gegenüber der CMOS-Technik eine wesentlich höhere Ruheleistungsaufnahme zur Folge hat.

Seymour Cray hat in den 1980iger Jahren große Summen in die industrielle Forschung für GaAs-Halbleiter gesteckt, die in die Entwicklung der auf GaAs-Halbleitern beruhenden Cray-3 mündeten. Da die Entwicklungskosten aufgrund der neuen Technologie enorm hoch waren und nur eine einzige Cray-3 verkauft wurde, war Cray 1995 zahlungsunfähig und wurde ein Jahr später von SGI gekauft. Seitdem steckt wieder konventionelle Silizium-Technologie in Cray-Computern.

GaAs-Halbleiter werden heute hauptsächlich in der Hochfrequenztechnik verwendet (Handy- und Satelliten-Kommunikation). Weiterhin wird es (in einer Heterostruktur) in (Laser-)LEDs und Solarzellen verwendet. Für ASICs spielt GaAs dagegen keine Rolle mehr.

2.2.3. Alternativtechnologien

Auf Alternativtechnologien zu der statischen synchronen CMOS-Logik, wie sie in der überwiegenden Mehrheit der produzierten ASICs zum Einsatz kommt, soll an dieser Stelle nur kurz eingegangen werden, da sie für diese Arbeit keine Rolle spielen,

gleichwohl aber interessante Alternativen darstellen, die allerdings allesamt eine völlig andere Entwurfs-Methodik und Energieoptimierung erfordern.

Leichte Abwandlungen der statischen synchronen CMOS-Technik stellen die folgenden Technologien dar:

Asynchrone CMOS-Logik verzichtet auf ein Taktsignal, so dass alle Zustandsübergänge sofort stattfinden und nicht durch Flanken eines globalen Takts ausgelöst werden. Dies ist bei der Steuerung mit erheblichen Problemen verbunden und deshalb Gegenstand der Forschung, allerdings gibt es schon kommerzielle ASICs die dieses Prinzip (zumindest für einige Teile der Schaltung) nutzen [41].

Dynamische Logik und Pass-Transistor-Logik spielt in der aktuellen Low-Power Forschung nur eine untergeordnete Rolle. Dies ist darauf zurückzuführen, dass das Einsparpotenzial durch diese Technik als gering eingeschätzt wird. Ein Überblick und Vergleich mit der CMOS-Technologie auch in Hinblick auf den Energiebedarf findet sich in [25, 127].

Adiabatische Schaltungstechnik ersetzt die gewöhnlich verwendete Versorgungsgleichspannung durch stetige, phasenverschobene Versorgungswechselspannungen um durch Limitierung der Lade-/ Entladeströme die ohmschen Verluste zu minimieren (siehe auch Abschnitt 2.5.1). Eine ausführliche Darstellung findet sich in [121].

In einem für aktuelle Produktionsprozesse noch weniger entwickelten Zustand sind die folgenden Technologien:

Reversible Schaltungstechnik Die sog. reversible Schaltungstechnik geht noch einen Schritt weiter als die adiabatische Schaltungstechnik. Hier soll die eingesetzte Energie komplett wiedergewonnen werden, die in dem reversiblen System nur noch von einem Speicher in den anderen verschoben wird. Möglicherweise ist es auf diese Art möglich, Informationen ohne Energieverlust zu verarbeiten (siehe Einleitung zu diesem Kapitel) [45].

Supraleitende Halbleiter ermöglichen die Reduktion des Energiebedarfs, da keine ohmschen Verluste mehr anfallen. Durch die Entwicklung des Ein-Elektronen-Transistors (*Single-Electron-Transistor*, SET) sind theoretisch zudem sehr kleine Strukturen möglich. SQUIDs (*superconducting quantum interference devices*) werden schon zur Verstärkung sehr kleiner Ströme eingesetzt, sind jedoch für integrierte Schaltungen ungeeignet [28].

Biochemische Informationsverarbeitung Hierbei wird beispielsweise mittels eines DNA-basierten Computers [8] eine parallele Exploration des Suchraums ermöglicht. Damit ist diese Technologie potentiell in der Lage, NP-vollständige Probleme in Polynomialzeit zu lösen.

Quanten-Computer basieren auf QBits, die statt zweier Zustände beliebig viele Zustände zwischen 0 und 1 gleichzeitig annehmen können, weshalb auch mit dieser Art von Computer ein paralleles Arbeiten möglich wäre. Momentan scheitert dieser Ansatz noch an dem Problem, einen kohärenten Zustand über mehrere QBits über eine längere Zeit ($> 10^{-10}\,s$) zu halten und auszulesen [111].

2.2.4. Bewertung

Das Potenzial der Silizium-Technologie erscheint noch längst nicht ausgereizt, wie die erwähnten stetigen Verbesserungen zeigen. Laut ITRS-Roadmap [75] wird das Moore'sche Gesetz die nächsten 15 Jahre Bestand haben.

Das bedeutet sehr wahrscheinlich, dass auch in einigen Jahren die leistungsfähigsten Prozessoren und ASICs noch in der klassischen statischen CMOS-Technik entwickelt und gefertigt werden.

Obgleich die erwähnten, viel versprechenden Alternativen existieren, ist ein Durchbruch für den Massenmarkt in naher Zukunft nicht zu erwarten. Aus diesem Grund konzentriert sich diese Arbeit auf die statische synchrone CMOS-Variante, für die es eine Fülle von Werkzeugen gibt, die ständigen Verbesserungen unterliegen.

2.3. Mobile Energiespeicher

Ein hoher Energiebedarf informationsverarbeitender Systeme bedingt im wesentlichen zwei Problemkomplexe:

1. Die mit der anfallenden Abwärme einhergehenden Kühlungsprobleme und
2. Die ausreichende Versorgung mit Energie, die insbesondere bei mobilen Anwendungen wesentlich sowohl zur Gesamtmasse als auch dem Gesamtvolumen beitragen kann.

Für mobile Anwendungen kommen elektrochemische Energiespeicher zum Einsatz, die im Vergleich mit Primärenergieträgern wie Erdöl (10 KWh/l) eine sehr viel geringere Energiedichte aufweisen (< 0,5 kWh/l). Es ist jedoch möglich, die gespeicherte Energie direkt als elektrischen Strom an einen Verbraucher zu liefern. Energieträger ist ein Elektrolyt (Batterie) oder ein Gas (Brennstoffzelle).

Aus Abb. 2.3 wird ersichtlich, dass die Entwicklung der Energiespeicher nicht mit dem wachsenden Leistungsbedarf der Mikroprozessoren mithalten konnte, trotz mehrerer Innovationen in der Batterietechnik (Abb. 2.4).

Da die Integrationsdichte in der Schaltungstechnik weiter steigt, erscheint es immer attraktiver, portable System zu bauen, da der realisierbare Funktionsumfang bei sinkendem Raumbedarf steigt.

Dies ist heute aufgrund der Limitierungen der Energiespeicherung nicht mehr möglich, ohne den Leistungsbedarf beim Entwurf des Gesamtsystems zu berücksichtigen.

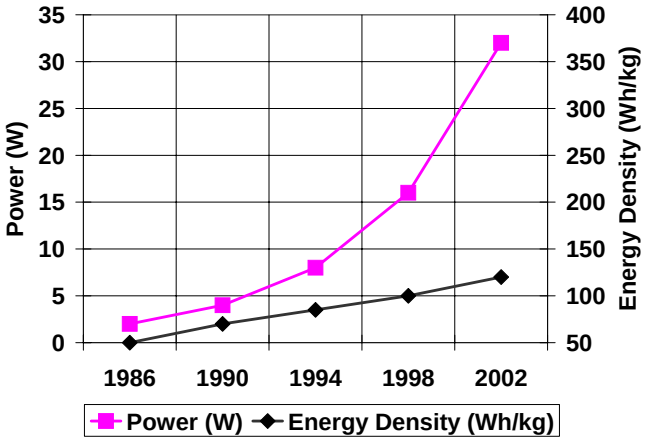


Abbildung 2.3.: Entwicklung der gravimetrischen Leistungsdichte von elektrochemischen Energiespeichern im Vergleich zu der Entwicklung des Energiebedarfs von Mikroprozessoren. Der Abstand zwischen der aus der Batterie zur Verfügung gestellten Energie und der Leistungsaufnahme der Mikroprozessoren ist in der Vergangenheit kontinuierlich gestiegen, inzwischen (2006) scheint bei etwa 120 W eine Grenze erreicht zu sein. Um akzeptable Laufzeiten zu erreichen, wurde es notwendig, spezielle Low-Power Architekturen für Prozessoren zu entwickeln (z. B. den Pentium M). Vormals kamen in Laptops selektierte Desktop-Prozessoren zum Einsatz, die mit einer geringeren Versorgungsspannung betrieben wurden (aus [15]).

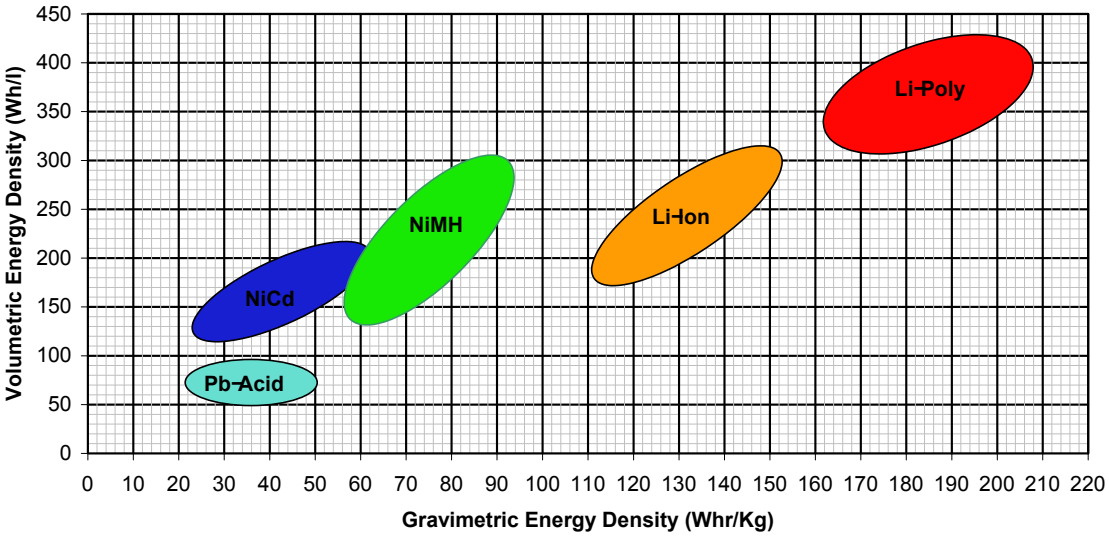


Abbildung 2.4.: Entwicklung der gravimetrischen und volumetrischen Leistungsdichte von elektrochemischen Energiespeichern. Auf Lithium-Ionen und Nickel Metall-Hydrid Technologie beruhende Batterien haben Nickel-Cadmium Zellen inzwischen vollständig verdrängt. Für Laptops werden gerne Li-Ionen Akkus eingesetzt, da sie bei gleicher Energiedichte leichter sind als NiMH-Akkus. Ferner weisen sie für den Alltagseinsatz angenehmere Lade- und Entladeeigenschaften auf, z. B. haben sie keinen „Memory-Effekt“.
Vergleichbar gute Eigenschaften haben Lithium-Polymer-Akkus, die noch höhere volumetrische und gravimetrische Energiedichten aufweisen. Sie werden aufgrund des hohen Preise überwiegend in Kleingeräten wie z. B. Handys eingesetzt (aus [65], S. 6).

2.4. Eingabesprachen für den automatisierten Hardwareentwurf

Im folgenden werden die für den automatisierten Hardwareentwurf gebräuchlichen Eingabesprachen beschrieben. Nach den „Klassikern“ VHDL und Verilog werden die designierten Nachfolger SystemC und System-Verilog beschrieben, die als Sprach-Standards explizit für die Hardware-Synthese entworfen wurden (Abschnitt 2.4.2).

Im letzten Kapitel vor der Bewertung werden noch einige Beispiele[6] für alternative Eingabesprachen gegeben, für die bestehende, nicht für die Hardware-Synthese oder -Simulation entworfene Sprachen übernommen oder abgewandelt wurden. Diese sind in der Regel für einen bestimmten Anwendungsfall optimiert oder ausschließlich einsetzbar.

2.4.1. VHDL und Verilog

Die Sprachen VHDL und Verilog wurden zunächst als Eingabesprachen für die Simulation von Gatternetzlisten entwickelt.

Verilog wurde von Phil Moorby 1984 entworfen, ein Jahr später brachte er die erste Implementierung in Form eines kommerziellen Simulators heraus. Die Besonderheit gegenüber anderen Simulationssprachen war, dass sich nicht nur Netzlisten (strukturelles Verilog) sondern auch „Verhaltensbeschreibungen“ (funktionales Verilog) auf Register-Transfer- oder der gänzlich ungetakteten Verhaltensebene beschreiben und simulieren ließen. Es war damit möglich, zu der heute üblichen Aufteilung in ein Testblock und Logikblock in der gleichen Beschreibungssprache überzugehen.

Die Zahl der Anwender der Sprache wuchs so stark, dass Synopsys 1988 den DesignCompiler vorstellte, der Verilog-RT-Beschreibungen in Gatternetzlisten übersetzen konnte. Der DesignCompiler stellte das erste kommerzielle allgemeine RT-Synthesewerkzeug dar. Dies ließ die Verilog-Verbreitung nochmals sprunghaft ansteigen.

Da Verilog eine proprietäre Sprache war, für die auf der einen Seite Lizenzgebühren kassiert wurden und die auf der anderen Seite nicht offiziell standardisiert war, wurde VHDL[7] 1987 von der IEEE standardisiert. Diese Sprache wurde bereits Ende der siebziger Jahre mit finanzieller Beihilfe des US Department of Defense (DoD) entwickelt, konnte sich durch das Aufkommen von Verilog aber nicht weit verbreiten. Durch die Standardisierung bekam VHDL einen Popularitätsschub.

Dadurch sah sich Cadence, der damalige Inhaber der Verilog-Lizenzrechte, genötigt, die Sprache ebenfalls einer Öffnung zu unterziehen. Erst 1995 wurde Verilog zum IEEE-Standard.

Da sich beide Sprachen nur in der Syntax unterscheiden, von ihrer Mächtigkeit und Konzeption aber sehr ähnlich sind, ist die Bevorzugung der einen oder anderen Spra-

[6] Aufgrund der verfügbaren Menge an synthesefähigen Eingabesprachen in Kombination mit einem speziellen Synthesewerkzeug für spezielle Anwendungsfälle kann hier leider kein vollständiger Überblick gegeben werden.

[7] VHSIC (Very High Speed Integrated Circuit) Hardware Description Language

che Geschmackssache. Interessanterweise hat sich hier eine geografische Abgrenzung gebildet: In den USA wird Verilog, in Europa VHDL bevorzugt.

Im Gegensatz zu VHDL hat Verilog inzwischen eine Erweiterung erfahren, die eine eingeschränkte Systemmodellierung ermöglicht (s.u.).

2.4.2. SystemC und System-Verilog

Um die Herausforderungen der immer komplexer werdenden Entwürfe besonders im Bereich der Eingebetteten Systeme und SOCs (*System-on-Chip*) und der damit verbundenen Entwurfslücke meistern zu können, wird versucht, im Softwareentwurf längst verwendete Techniken wie Wiederverwendung und Objektorientierung in die Hardwarewelt zu importieren. Da es sich bei den klassischen HDLs um prozedurale Programmiersprachen handelt, ist dies nicht ohne Änderungen möglich, weshalb man entweder, wie im Fall von SystemC[6], auf eine komplett andere Programmiersprache ausweichen musste oder aber die Sprache modifizieren musste, wie bei System-Verilog geschehen.

SystemC ist eine von der „Open SystemC Initiative“ (OSCI) spezifizierte C++-Klassenbibliothek, die mit dem Ziel entwickelt wurde, einen abstrakten einheitlichen Systementwurf elektronischer Systeme zu ermöglichen. Es sollte damit möglich sein, Systeme, die aus unterschiedlichen Hardware- und Software-Blöcken bestehen, in einer einheitlichen Sprache zu entwickeln. Der Test und die Verifikation wird dabei durch zahlreiche bestehende C++-Bibliotheken, z. B. für mathematische Funktionen und komplexe Ein-/Ausgabe, gegenüber konventionellen Entwurfstechniken stark erleichtert.

Die Entwicklung erfolgt dabei mehrgleisig: Auf der einen Seite wird die Version 1 als direkter Ersatz für die klassischen HDLs entwickelt, die synthetisierbare Operatoren und Datentypen analog zu VHDL und Verilog bietet. Diese SystemC-Variante wird immer noch weiterentwickelt und wurde schon im März 2000 mit dem SystemStudio von CoCentric kommerziell unterstützt. Auch ist ein Entwurf und die automatische Synthese auf Verhaltensebene möglich [42].

Parallel wird SystemC in der Version 2 weiterentwickelt. Diese Variante hat keine eingebauten synthesefähigen Datentypen mehr, sondern die Synthese basiert auf einer Struktur aus Kanälen, Schnittstellen und Ports. Weiterhin wird eine Ereignis-gesteuerte Modellierung unterstützt, womit erstmals ein abstrakter System-Entwurf möglich wird.

In der Version 3, für die es noch keine lauffähige Implementierung gibt[8], soll die Modellierung von Betriebssystemen sowie der integrierte Entwurf eingebetteter Soft- und Hardware unterstützt werden [149].

Bei System-Verilog handelt es sich um eine Verilog-Erweiterung, die das abstrakte Modellieren von Systemen vereinfacht, u. a. durch eine Automatisierung der Testbench-Erstellung und eine API zur Anbindung an C-Programme [125].

[8] Stand September 2008

2.4.3. Andere Eingabesprachen

Die zunehmenden Anforderungen an eine schnelle Datenpfadsynthese, die insbesondere durch die immer leistungsfähiger und kostengünstiger werdenden FPGAs verschärft werden, haben verschiedene Hersteller veranlasst, Lösungen zu entwickeln, die nicht auf etablierten Sprachstandards für die Hardwaresynthese basieren. Diese liegen von den Möglichkeiten zwischen den klassischen HDLs VHDL/Verilog und den zukünftigen Systembeschreibungssprachen SystemC und System-Verilog. Der Einsatzzweck ist auf eine schnelle Datenpfadsynthese beschränkt, d. h. komplexe Kontrollflüsse können mit den auf diesen Sprachen basierenden Werkzeugen nicht synthetisiert werden.

Es sollen hier exemplarisch drei Werkzeuge umrissen werden:

1. Der Module Compiler von Synopsys[151]. Der Module Compiler von Synopsys wurde für die schnelle Datenpfadsynthese entworfen. Seine Eingabesprache ist eine Mischung von Verilog und C. Die Syntax lehnt sich an Verilog an, jedoch ist ein Präprozessor vorhanden, d. h. es ist z.B. möglich Makros zu definieren. Ebenfalls aus C wurde die Möglichkeit übernommen, Funktion zu definieren. Der Module Compiler wird inzwischen nicht mehr weiterentwickelt, da Synopsys mit der SystemC-Synthese ähnliche Möglichkeiten bietet.
2. Auf reinem C basiert das Synthesewerkzeug der Firma ImpulseC [117]. Es bietet eine API, mittels derer eine explizite Parallelität zwischen Funktionen realisiert werden kann. Weiterhin kann über `#pragma`-Statements der in der Toolchain folgende HDL-Compiler begrenzt beeinflusst werden, z. B. kann so ein Pipelining veranlasst werden. Das Synthesewerkzeug generiert eine HDL-Beschreibung in VHDL oder Verilog, die dann durch einen entsprechenden Compiler weiterverarbeitet werden muss.
3. Als Beispiel für ein Werkzeug, das MATLAB/Simulink als Eingabesprache nutzt, wird hier der *Xilinx System Generator for DSP* aufgeführt, der in Kapitel 3.5 detaillierter beschrieben wird. Dieses System ist auf digitale Signalverarbeitung optimiert, komplexe Kontrollpfade lassen sich mit diesem Werkzeug nicht synthetisieren.

2.4.4. Bewertung

Die klassischen prozeduralen Hardware-Beschreibungssprachen Verilog und VHDL sind nicht geeignet, um eine schaltungstechnische Implementierung sowie Test und Verifikation auf Systemebene zu modellieren bzw. durchzuführen. Allenfalls eine Modellierung auf Verhaltensebene ist möglich. Es gibt jedoch keine standardisierten Schnittstellen, um z. B. eine Testbench-Integration mit C/C++ oder Java-Programmen zu ermöglichen. Zwar existiert eine C-Schnittstelle für Verilog, diese ist jedoch herstellerspezifisch implementiert und entspricht nicht mehr dem Stand der Technik der modularen Softwareentwicklung.

Weiterhin gibt es Ansätze der Hersteller von EDA-Werkzeugen, mit proprietären Erweiterungen im Synthese-Prozess einige der Limitierungen dieser Sprachen abzumildern, wie z. B. der Mechanismus der *Synthetic Libraries* (siehe Abschnitt 4.5.1) von Synopsys,

der eine abstrakte Schnittstellendefinition ermöglicht oder der Gebrauch von `#pragma`-Anweisungen zur Compiler-Steuerung, die als Kommentare getarnt sind.

Die Sprachen SystemC (in der heute verbreiteten Version 1[9]) und System-Verilog können die Defizite ebenfalls nur unzureichend beseitigen, da im Falle von System-Verilog die klassische überholte Sprachsyntax erhalten blieb und im Falle von SystemC die Klassenbibliothek so eng an alte Entwurfsmuster angelehnt wurde, dass die Verbesserungen gegenüber VHDL zumindest in der Version 1 bis auf die Testbench-Einbindung als vernachlässigbar einzustufen sind. Eine Unterstützung für die Version 2 von SystemC ist bis heute bei kommerziellen Synthesewerkzeugen nicht umfassend vorhanden.

Neuere Ansätze lösen sich deshalb komplett von diesen Beschreibungsmöglichkeiten. So basiert die in [87] skizzierte Methodologie auf einer Integration in Microsofts .NET-Framework mit den Eingabesprachen C# und C++. Das bereits verfügbare kommerzielle Werkzeug zur Behavioral-Synthese *Catapult SL* der Firma Mentor Graphics [3] unterstützt ebenfalls reines C++ als Eingabesprache.

Einen anderen Ansatz verfolgen die in Abschnitt 3.5 exemplarischen genannten Sprach-/Synthesewerkzeugkombinationen, die eine optimierte Synthese mit alternativen, nicht explizit für den Hardwareentwurf geschaffenen Eingabesprachen ermöglichen. Es handelt sich dabei allerdings nicht um Allzweck-Systeme, sondern die Anwendungen sind auf bestimmte Domänen beschränkt.

Aufgrund der Verfügbarkeit von Werkzeugen für die Behavioral-Synthese nutzt das in der vorliegenden Arbeit vorgestellte Entwicklungsframework Behavioral-VHDL als Eingabesprache und die von Synopsys entwickelten Methoden zur Umgehung einiger Limitierungen dieser Sprache. Basis der Simulation und Modellierung sind aber C++-Prototypen, die bei Verfügbarkeit eines entsprechenden Werkzeugs mit geringen Änderungen auch direkt zur Synthese verwendet werden könnten (siehe Kapitel 4).

[9] Für die Simulation ist SystemC Version 2, für die Synthese Version 1 gebräuchlich (Stand September 2008)

2.5. Energiebewertung für die CMOS-Technologie

Da alle modernen industriellen Fertigungsprozesse für die Chip-Herstellung auf der CMOS-Technologie basieren, ist die Energiebewertung und -Optimierung hier von besonderem praktischem und kommerziellen Interesse. Letztlich ist die CMOS-Technologie auch die Grundlage für den in dieser Arbeit entwickelten Entwurfsfluss. Im folgenden wird daher auf die physikalischen Grundlagen und die Prinzipien der Energieschätzung und Optimierung eingegangen.

Um wichtige Begriffe wie *adiabatische* oder *reversible* Schaltungstechnik konsistent erklären zu können, wird im ersten Unterabschnitt (2.5.1) knapp und keineswegs vollständig auf die physikalischen Grundlagen eingegangen. Es dient auch als Auflistung der für diese Arbeit verwendeten Annahmen und Näherungen zur Bestimmung des Energieverbrauchs von CMOS-Schaltungstechnik.

2.5.1. Physikalische Grundlagen

Der elektrische Leistungsbedarf einer CMOS-Schaltung setzt sich aus zwei verschiedenen Anteilen zusammen, der dynamischen Leistung, die im Falle einer Zustandsänderung anfällt, und der statischen Leistung, die durch unerwünschte Leckströme bedingt ist:

$$P_{\text{total}} = P_{\text{dynamic}} + P_{\text{static}} \tag{2.1}$$

Dynamische Verlustleistung

Vereinfacht betrachtet man ein CMOS-Transistorpaar als Schalter. Wenn der Schalter betätigt wird, muss die angeschlossene Leitung, die als einfaches *RC*-Glied modelliert wird, auf die Betriebsspannung V_{dd} aufgeladen werden. Dabei modelliert die Kapazität C im wesentlichen die Gate-Kapazitäten der angeschlossenen CMOS-Paare (siehe Abb. 2.5).

Die zum Aufladen dieser Kapazität benötigte Energie E_{tr} ist dabei die aufintegrierte Leistung, die für den Aufladevorgang des Kondensators C benötigt wird:

$$E_{\text{tr}} = \int_{t=0}^{\infty} P(t)\,\mathrm{d}t = V_{\text{dd}} \int_{t=0}^{\infty} i(t)\,\mathrm{d}t \tag{2.2}$$

Der geometrische Parameter Kapazität ist dabei über die gespeicherte Ladungsmenge Q pro Spannung U definiert. Q entspricht dem aufintegrierten Strom $i(t)$:

$$C \equiv \frac{Q}{U} = \frac{1}{U} \int i(t)\,\mathrm{d}t \tag{2.3}$$

Der Strom $i(t)$ ist dabei durch die Maschenregel zu

$$i(t) = \frac{V_{\mathrm{dd}} - U(t)}{R} \tag{2.4}$$

bestimmbar (vergl. Abb. 2.5). Löst man die Definition von C in Gl. (2.3) nach U auf und setzt dies in Gl. (2.4) ein, so gelangt man zu der linearen Integralgleichung

$$i(t) = \frac{1}{R}\left(V_{\mathrm{dd}} - \frac{1}{C}\int i(t)\,\mathrm{d}t\right) \quad , \tag{2.5}$$

die durch Differentiation zu der linearen Differentialgleichung

$$\dot{i}(t) = -\frac{1}{RC}i(t) \tag{2.6}$$

wird. Mit dem Lösungsansatz

$$i(t) = A\exp(Bt) \tag{2.7}$$

und der Randbedingung $I(0) = V_{\mathrm{dd}}/R$ gelangt man zur Lösung

$$i(t) = \frac{V_{\mathrm{dd}}}{R}\exp\left(-\frac{t}{RC}\right) \tag{2.8}$$

Setzt man (2.8) in die Energiegleichung (2.2) ein und wertet das Integral in den Grenzen 0 und ∞ aus, so erhält man für die zu Umladung nötigen elektrischen Energie E_{tr}

$$E_{tr} = V_{\mathrm{dd}}\int_{t=0}^{\infty} i(t)\,\mathrm{d}t = CV_{\mathrm{dd}}^2 \tag{2.9}$$

Da die im Kondensator gespeicherte Energie

$$E_C = \frac{1}{2}CV_{dd}^2 \tag{2.10}$$

beträgt[10], geht die Hälfte der Energie durch Dissipation im Widerstand R verloren (unabhängig von dem tatsächlichen Wert von R). Genau diesen Anteil versucht die *Adiabatische Schaltungstechnik* zu reduzieren[11]. Die *Reversible Schaltungstechnik* möchte auch noch die im Kondensator C gespeicherte Energie zurückgewinnen.

Weiterhin nimmt man an, dass bei jedem Schaltvorgang eine durch den Kurzschlussstrom I_{sc} bedingte Kurzschlussleistung $V_{\mathrm{dd}} \cdot I_{\mathrm{sc}}$ benötigt wird (siehe Abb. 2.5), die durch die endlich steilen Flanken zustande kommt, während derer beide CMOS Transistoren gleichzeitig leiten.

[10] Eine gute Herleitung findet sich z. B. in [37].

[11] Die ohmsche Verlustleistung ist proportional zu I^2, deshalb werden die Umladeströme reduziert. Dies gelingt durch eine nicht-konstante Versorgungsspannung V_{dd}, die dem Signal nur um eine kleine Phasendifferenz Φ vorauseilt und die Differenz $V_{\mathrm{dd}}(t) - U(t)$ in Gl. (2.4) minimiert.

Somit beträgt die dynamische Verlustleistung

$$P_{\text{dynamic}} = p_t \cdot f_{clk}(C_L \cdot V_{\text{dd}}^2 + I_{\text{sc}} \cdot V_{\text{dd}}) \tag{2.11}$$

Dabei ist p_t die mittlere Transitionswahrscheinlichkeit und f_{clk} die Taktfrequenz der CMOS-Schaltung, C_L steht für die mittlere geschaltete Kapazität und I_{sc} ist der mittlere Kurzschlussstrom bei Pegeländerungen.

Der Kurzschlussstrom I_{sc} trägt üblicherweise mit 5-20% zur dynamischen Verlustleistung bei [25] und wird für gewöhnlich vernachlässigt.

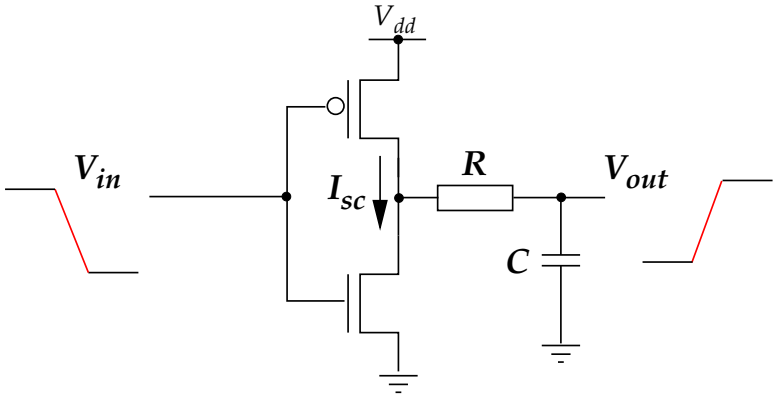


Abbildung 2.5.: CMOS Transistorpaar. Dargestellt ist ein Inverter, der aus einem NMOS- und einem PMOS-Transistor besteht. Komplexere CMOS-Gatter sind stets aus solchen Paaren aufgebaut. Das CMOS-Paar wird für die Leistungsberechnung als idealer Schalter aufgefasst. Die Leitung am Ausgang des Inverters wird durch einen Kondensator C modelliert, der über den Leitungswiderstand R an den Gatterausgang angebunden ist. Durch die Modellierung der Leitung als RC-Glied sind die Flanken der Signale nicht beliebig steil, sondern weisen ein Zeitverhalten auf (rot markiert). Dies hat zur Folge, dass beide Transistoren des CMOS-Paares für eine gewisse Zeit leiten, so dass im Umschaltmoment ein Kurzschlussstrom I_{sc} fließen kann.

Statische Verlustleistung

Für die statische Verlustleistung nimmt man einen mittleren Leckstrom I_{leak} an, der über die gesamte Zeit fließt und somit eine Verlustleistung von

$$P_{\text{static}} = I_{\text{leak}} V_{\text{dd}} \tag{2.12}$$

ergibt. Der Strom I_{leak} setzt sich dabei aus zwei Anteilen zusammen. Zum einen werden die im Herstellungsprozess verwendeten Isolationslagen immer dünner, so dass es insbesondere zwischen der Gate-Elektrode und dem Kanal zu einem Tunnelstrom kommt (siehe Abschnitt 2.2.2). Zum anderen sperren die verwendeten Transistoren im ausgeschalteten Zustand nicht vollständig, so dass ein unerwünschter Drain-Source-Strom fließen kann, der sogenannte Unterschwell- oder *Sub-Threshold*-Strom (siehe Abb. 2.6).

Dieser Effekt ist stark temperaturabhängig, da mit der Erhöhung der Temperatur verstärkt störstellenunabhängige Elektron-Loch-Paare gebildet werden (*Eigenleitung*)[12].

Für 3,3 V-Prozesse ist der Leckstrom noch vernachlässigbar (< 1% der Gesamtleistung), durch die Strukturverkleinerungen und Betriebsspannungssenkungen kann die durch den *Sub-Threshold*-Strom bedingte Verlustleistung jedoch dominieren [157].

Da der Aspekt der Schwellenspannung auch für Energiesparmaßnahmen durch V_{dd}-Absenkung von Interesse ist, folgt hier eine kurze Behandlung.

Wie in [137] ausgeführt, ist die Schaltzeit eines Gatters oder Gatterverzögerung T_d von der Gate-Übersteuerung (*Gate-Overdrive*) $V_{\mathrm{dd}} - V_{\mathrm{t}}$ abhängig (V_{t} wird als Schwellenspannung oder *Threshold-Voltage* bezeichnet):

$$T_d = \frac{KV_{\mathrm{dd}}}{(V_{\mathrm{dd}} - V_{\mathrm{t}})^{\alpha}}, \tag{2.13}$$

Dabei ist K eine Prozess-Konstante und $1 \leq \alpha \leq 2$, die von der Transistor-Geometrie abhängige Kanal-Sättigung. Es ist daher sinnvoll, durch entsprechende Dotierung des Kanals eine niedrige Schwellenspannung V_{t} einzustellen, da der Transistor dadurch schneller schaltet.

Bei gesperrtem Transistor ($V_{GS} < V_{\mathrm{t}}$) fließt jedoch der oben erwähnte *Sub-Threshold*-Strom, der in [137] für n-Kanal FETs wie folgt genähert wird:

$$I_{\mathrm{DS}|V_{GS}<V_{\mathrm{t}}} = I_0 \exp(-\alpha) \exp\left(\frac{V_{GS} - V_{\mathrm{t}}}{S}\right) \quad , \tag{2.14}$$

wobei an einem gesperrten n-Kanal FET bei gängigen CMOS-Technologien eine Gate-Source-Spannung $V_{GS} = 0$ anliegt.

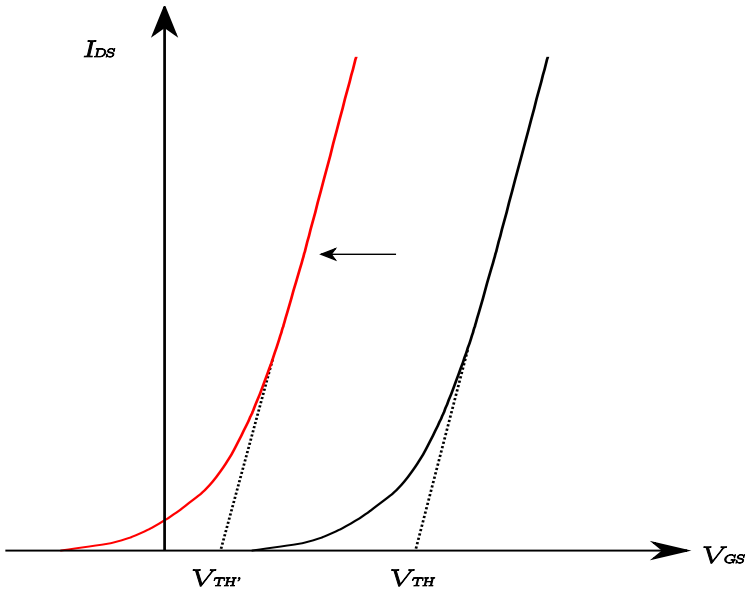
Dabei ist I_0 der Dioden-Sperrstrom und S ein Prozessparameter. Für den eingeschalteten Transistor gilt laut der sog. α-Näherung

$$I_{\mathrm{DS}|V_{GS}>V_{\mathrm{t}}} = I_0 (S\alpha)^{-\alpha} \, (V_{GS} - V_{\mathrm{t}})^{\alpha} \tag{2.15}$$

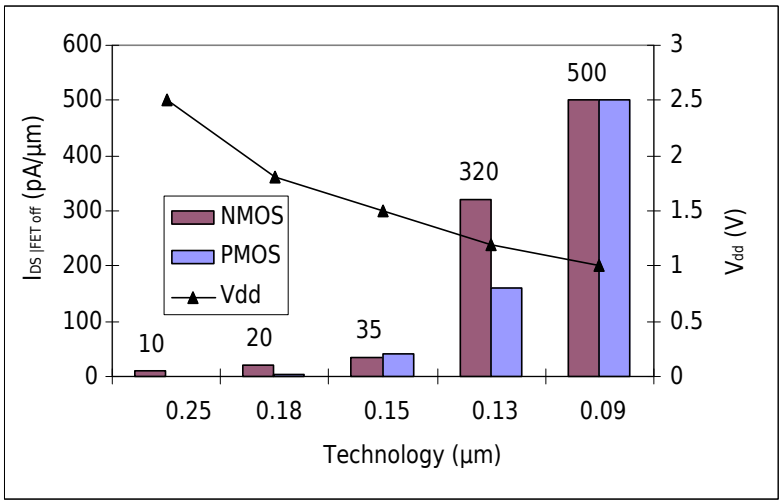
Dort herrscht also näherungsweise ein lineare Abhängigkeit von der Gatespannung. Die Formeln (2.14) und (2.15) sind in Abb. 2.6 visualisiert. Man erkennt dort, dass eine Absenkung von V_{t} zu einem erhöhten Strom durch den Transistor führt, wenn dieser abgeschaltet ist ($V_{GS} = 0$).

Eine empirische Grenze für aktuelle Prozesse scheint bei $V_{\mathrm{t}} \approx 0,5\,V$ zu liegen [58], danach steigt der Leckstrom so stark an, dass die dynamische Verlustleistung gegenüber der statischen vernachlässigbar ist.

[12] Die schwingenden Atomrümpfe übertragen kinetische Energie auf ihre Elektronen. Ist diese hinreichend groß, können Valenzelektronen in das Leitungsband gelangen und tragen damit zu unerwünschten Leckströmen bei. Die Eigenleitung steigt etwa exponentiell mit der Temperatur. Gleichzeitig sinkt die Ladungsträgerbeweglichkeit durch die verstärkten Gitterschwingungen, so dass sowohl Schaltzeit als auch Dissipation ansteigen.



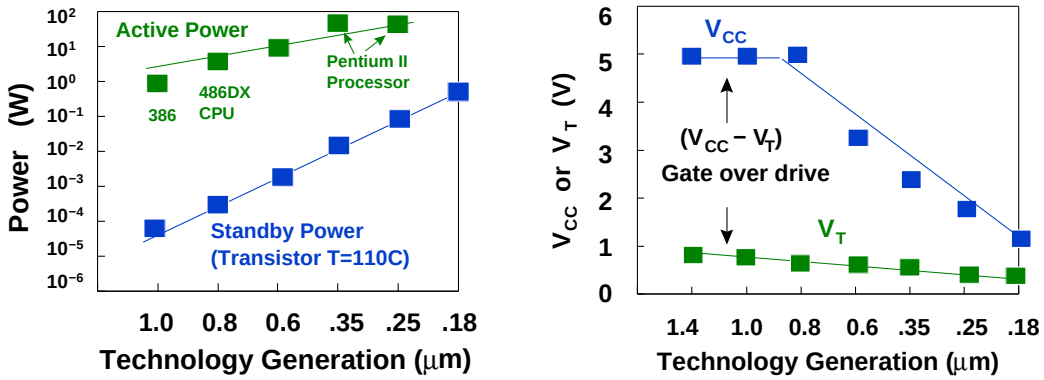
(a) I_{DS}-V_{GS}-Kennlinie



(b) I_{DS}-V_{dd}

Abbildung 2.6.: Unterschwellenstrom bei MOS-Transistoren. In (a) gezeigt ist der Drain-Source-Strom (I_{DS}) in Abhängigkeit der Gate-Source-Spannung (V_{GS}) bei zwei verschiedenen Schwellenspannung V_{TH} und $V_{TH'}$. I_{DS} wurde für $V_{GS} < V_t$ nach der Näherungsformel (2.14) ermittelt, der lineare Bereich ergibt sich aus Formel (2.15).
(b) zeigt I_{DS} bei abgeschaltetem Transistor in Abhängigkeit der Versorgungsspannung V_{dd}. Durch das erforderliche Absenken der Versorgungsspannung V_{dd} bei geringeren Strukturbreiten muss auch V_{TH} abgesenkt werden. Daraus ergibt sich ein zunehmender Unterschwellenstrom.

Da aufgrund des Strukturverkleinerung und Minimierung der dynamischen Verlustleistung die Betriebsspannung V_{dd} in den letzten Jahren immer weiter abgesenkt wurde, musste die Schwellenspannung V_t ebenfalls angepasst werden, um nach Gl. (2.13) eine hohe Schaltgeschwindigkeit zu erzielen. Deshalb, und aufgrund des Gate-Tunnelstroms, ist der Anteil der statischen Verlustleistung stark angestiegen und liegt bei Mikroprozessoren aktuell etwa gleichauf mit der dynamischen Verlustleistung (Abb. 2.7).



(a) Statische und dynamische Verlustleistung bei Mikroprozessoren

(b) Entwicklung der Gate-Übersteuerung

Abbildung 2.7.: Entwicklung der dynamischen und statischen Verlustleistung. (a) zeigt, dass die durch Leckströme bedingte Verlustleistung moderner Prozessoren nur noch etwas mehr als eine Größenordnung von der dynamischen Verlustleistung entfernt ist, beim 80386 waren dies noch ca. fünf Größenordnungen. Die statische Verlustleistung ist somit nicht mehr vernachlässigbar.
Die zusätzliche Leistung ist im wesentlichen durch eine Absenkung der Schwellenspannung V_t bedingt, was aus (b) deutlich hervorgeht (aus [59])

2.5.2. Prinzipien der Energiebewertung

Für eine korrekte Bewertung des Leistungs- bzw. Energiebedarfs einer integrierten Schaltung auf Basis statischer CMOS-Technik muss die im Abschnitt 2.5.1 beschriebene Gesamtleistung als Summe der dynamischen und statischen Verlustleistung berechnet werden (Formel (2.1)).

Die genauesten Werkzeuge gehen dabei so vor, dass für jede Zustandsänderung in dem platzierten und verdrahteten System die dynamische Verlustleistung jeder einzelnen Standardzelle mit Zustandsänderung aufsummiert wird. Zusätzlich wird die durch die Verdrahtung bedingte Verlustleistung anhand der Länge des jeweiligen Netzes (und dem damit assoziierten *RC*-Wert) berechnet. Die statische Verlustleistung wird abhängig vom logischen Zustand jeder Standardzelle ermittelt. Das im Framework verwendete Werkzeug PrimePower (siehe Abschnitt 4.2.3) leistet dies, man bezeichnet diese Art der Schätzung als Ereignis-basierte Verlustleistungsschätzung.

Dafür ist es notwendig, dass

1. der zu schätzende Chip platziert und verdrahtet (als Floorplan) vorliegt,

2. eine Simulation mit korrekten (repräsentativen) Eingabedaten möglich ist
3. und dass die Standardzell-Bibliothek vom Hersteller sehr genau powercharakterisiert ist.

Da die Erfüllung dieser Bedingungen in der Entwicklung üblicherweise einen hohen zeitlichen Aufwand bedeutet, behilft man sich in der Praxis mit mehr oder minder starken Vereinfachungen:

1. Der Prozesshersteller macht sich bei Standardprozessen (Betriebsspannung 3,3 V und höher) generell nicht die Mühe, eine Bibliothek genau zu charakterisieren. Meist gibt es keine Charakterisierung für die statische Verlustleistung und die dynamische ist oft nicht zustandsabhängig.
2. Statt jedes Ereignis in einer Simulation für die Leistungsschätzung einzeln zu erfassen, wird durch den Simulator eine Statistik für jeden Knoten der Schaltung ermittelt. Das Schätzwerkzeug wertet dann nur noch die mittlere Schaltaktivität aus und berechnet daraus den Leistungsbedarf. Ein im Framework verwendetes Werkzeug dieser Art ist DesignPower bzw. der PowerCompiler von Synopsys (Abschnitt 4.2.2).
3. Zu groben vergleichenden Schätzungen wird die resultierende Schaltaktivität auf algorithmischer Ebene abgeschätzt. Da für die meisten Systeme die dynamische Verlustleistung (Gl. (2.11)) dominiert, ist dies ein legitimes Vorgehen. Es ist die Basis aller algorithmischen Transformationen zur Energieeinsparung wie sie einige der in Kapitel 3 vorgestellten Systeme realisieren und sie durch die Module des Entwicklungsframeworks (Abschnitt 4.6) umgesetzt wurden. Man vertraut darauf, dass sich die theoretischen Einsparungen auf algorithmischer Ebene bis zum physikalischen Chip propagieren. Wie in Abschnitt 4.6.4 gezeigt wird, ist dies leider im vollen Umfang selten und kann sich sogar ins Gegenteil verkehren[13].

   Weiterhin gibt es Ansätze, die eine schnelle Verlustleistungsschätzung auf Verhaltensebene durch eine Architekturexploration ermöglichen, z. B. [79, 21, 110, 92, 80, 97, 144]. Diese sind allesamt noch im Forschungsstadium.

2.6. Energieoptimierung für die CMOS-Technologie

Das grundsätzliche Ziel der Energieoptimierung von CMOS-Schaltungstechnik ist die Reduktion der statischen und dynamischen Verlustleistung. Dieses Ziel kann auf allen Entwurfsebenen vom informationsverarbeitenden System bis zur Dimensionierung des zugrundeliegenden physikalischen Prozesses verfolgt werden.

Im folgenden werden etablierte Standardverfahren für die Verlustleistungsoptimierung auf der RT-Ebene und darunter vorgestellt. Dies sind im einzelnen

1. Propagierung von Konstanten
2. Operandenisolierung

[13] Dies liegt an der Komplexität des Weges zwischen Algorithmus und Chip, der mehrere, z. T. nichtdeterministische Optimierungsschritte umfasst.

3. Clock-Gating
4. Glitch-Eliminierung
5. Absenken der Versorgungsspannung

2.6.1. Propagierung von Konstanten

Bei der Propagierung von Konstanten werden Konstanten in arithmetischen Operationen ausgenutzt, um die Struktur der Operatoren zu vereinfachen. Multiplizierer können so in Shift-Adder-Strukturen transformiert werden, was die Komplexität und damit den Energiebedarf der Operation reduziert. Voll-Addierer können ebenfalls vereinfacht werden.

Propagierung von Konstanten ist eine Technik, die moderne Synthese-Systeme wie der DesignCompiler von Synopsys automatisch beherrschen.

2.6.2. Operandenisolierung

Ein Problem in komplexeren Schaltungen ist die Tatsache, dass nicht alle Schaltungsteile in jedem Takt-Zyklus benötigt werden. Die unbenötigten Teile werden gewöhnlich aber ständig getaktet und die (ungültigen) Daten werden auch lokal verarbeitet, was zu unnötiger Schaltaktivität führt.

Ein früher Ansatz, diese Schaltaktivität zu eliminieren, war die Einführung der Operandenisolierung. Dabei wird vor Schaltungteile im Datenpfad, die nicht immer gebraucht werden, ein Multiplexer eingesetzt, der das Ausgangssignal dieses Schaltungsteils auf den Eingang zurückkoppeln kann (Abb. 2.8). Dadurch wird der unbenutzte Datenpfad hinter dem Multiplexer abgetrennt, da sich die Pegel hier nicht mehr ändern. Bei einer Spezifikation der Schaltung auf Behavioral-Ebene[14] ist dem Synthese-Werkzeug bekannt, welche Teile des Datenpfades wann benötigt werden, so dass die Kontroll-Logik für den Multiplexer automatisch generiert werden kann. Operandenisolierung wird daher seit längerem vom Synopsys Behavioral Compiler unterstützt. Auf RT-Ebene kann redundante Logik durch die Benutzung von `default`-Zuweisungen[15] für den Compiler markiert werden, so dass auch der DesignCompiler eine Operandenisolierung durchführen kann. Ein Dateneingang des Multiplexers wird dann mit dem konstanten `default`-Vektor vorbelegt, so dass im Gegensatz zu Abb. 2.8 kein Register inferiert werden muss.

Zu den Nachteilen der Operandenisolierung zählen neben der erforderlichen Logik (die einfach aus der FSM abgeleitet wird) die erforderlichen Multiplexer für jedes Bit, die sowohl selbst Energie benötigen als auch eine zusätzliche Verzögerung in den Datenpfad einbringen, den kritischen Pfad mithin verlängern können.

Der Vorteil gegenüber dem im folgenden vorgestellten Clock-Gating ist, dass das Takt-Signal selbst nicht beeinflusst wird.

[14] Verhaltensebene, siehe auch Abschnitt 2.7.2

[15] Dabei wird einer Variable ihr alter Wert zugewiesen, wenn sie nicht benötigt wird. Dies ist ohnehin guter Stil bei der Codierung auf RT-Ebene, da es sonst u. U. zur Inferenz unerwünschter Register kommen kann.

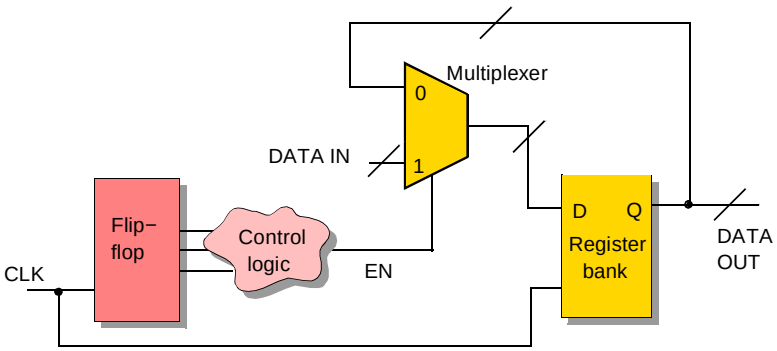


Abbildung 2.8.: Operandenisolierung einer Register-Bank. Die Register-Bank wird ständig mit dem Taktsignal „CLK“ versorgt. Über eine Feedback-Schleife durch den Multiplexer wird mit jedem Taktsignal der alte Registerinhalt übernommen. Der auf den Multiplexer folgende Schaltungsteil wird dadurch abgekoppelt und unnötige Transitionen vermieden. Wenn die Kontroll-Logik den Multiplexer über das „EN“-Signal auf den Dateneingang „DATA IN“ schaltet, wird die Isolierung aufgehoben.
Gibt es im Code eine `default`-Zuweisung für den Datenvektor, wird der Eingang `0` des Multiplexers mit dieser Konstante belegt, so dass zur Abtrennung kein Register notwendig ist.

2.6.3. Clock-Gating

Da bei den üblichen synchronen Schaltungen jedes Flipflop mit dem gleichen Takt versorgt wird, haben die Taktnetze bei größeren Designs vergleichsweise hohe Kapazitäten und damit einen hohen Strombedarf. Eine naheliegende Möglichkeit, die Belastung durch gerade nicht benötigte Bereiche des Designs zu verringern, besteht nun darin, den Takt selektiv für bestimmte Sub-Komponenten zu bestimmten Zeiten abzuschalten. Man bezeichnet dieses Vorgehen als „Clock-Gating“.

Da zu diesem Zweck in die betreffende Taktleitung mindestens ein zusätzliches Bauelement eingefügt werden muss, ergibt sich durch dieses zwangsweise eine Verzögerung des Taktsignals, was zu Timing-Problemen in synchronen Schaltungen führen kann, da jetzt positive Hold-Zeiten erforderlich sind. Aus diesem Grunde ist es notwendig, dass die verwendeten Synthesewerkzeuge Clock-Gating direkt unterstützen.

Alle gängigen Synthesewerkzeuge inferieren synchrone Registerbänke; das bedeutet, dass mit jeder Taktflanke *alle* Register einer Registerbank angesprochen werden, obwohl der Registerinhalt nur bei einer u.U. kleinen Teilmenge der zur Bank gehörigen Register geändert wird. Auf die anderen Register wird der eigene Ausgang über einen Multiplexer zurückgekoppelt (siehe Abb. 2.8). Dies impliziert Schaltaktivität und damit Energiebedarf sowohl im Multiplexer als auch in der Registerbank, die keine Auswirkungen auf den Zustand des Systems hat und damit überflüssig ist. Gleiches gilt für komplette Registerbänke, deren Eingänge zu einem bestimmten Taktzyklus ungültig sind, so dass der Ausgang ohne Clock-Gating über einen Multiplexer auf den Eingang zurückgekoppelt werden muss.

Diese unnötige Schaltaktivität kann durch das Einfügen z. B. eines Latches und eines Gatters, die das globale Taktnetz von der Registerbank entkoppeln, erheblich reduziert werden (Abb. 2.9). So werden ebenfalls auf den Auswahl-Leitungen des Multiplexers

ggf. auftretende Glitches (siehe Abschn. 2.6.4) von der Registerbank entkoppelt. Der Multiplexer entfällt, was bei großen Registerbänken zu erheblichen Platz-Vorteilen und damit auch zu einer geringeren Ruheleistung führt.

Auf der anderen Seite muss die durch diese beiden Schaltelemente bedingte Verzögerungszeit von dem Synthesewerkzeug berücksichtigt werden. Weiterhin wird auch das Taktsignal selbst durch das zusätzliche Gatter verzögert, weshalb an der Registerbank eine positive Haltezeit auftritt. Konventionelle Synthesewerkzeuge gehen immer von einer vollständig synchronen Schaltung aus, d. h. die Haltezeit der Signale an den Registern ist Null. Moderne Synthesewerkzeuge wie z. B. der PowerCompiler von Synopsys sind daher in der Lage, ein skriptgesteuertes automatisches Einfügen von Clock-Gating durchzuführen, dass die auftretenden Verzögerungszeiten berücksichtigt[16].

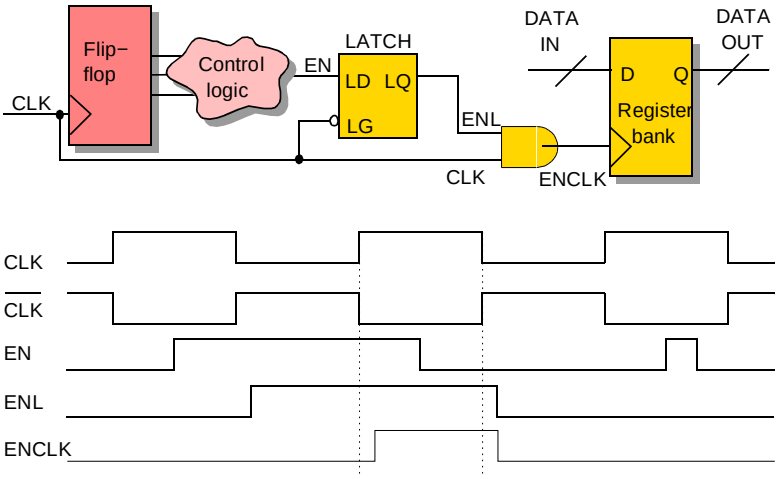


Abbildung 2.9.: Register-Bank mit Clock-Gating. Der Multiplexer ist hier im Unterschied zu Abb. 2.8 durch ein Active-Low-Latch und ein Und-Gatter ersetzt worden. Dies bringt mehrere Vorteile mit sich: Der Datenausgang der Register-Bank ist kapazitiv geringer belastet (niedrigerer Fan-Out), Glitches der „EN“-Leitung (im Diagramm rechts angedeutet) werden vom Register abgekoppelt und das globale Taktnetz „CLK“ wird vom Register abgekoppelt. Bei großen Register-Breiten ist der Platzbedarf für den Multiplexer größer als für die Gating-Logik. Zu den Nachteilen zählen positive Hold-Zeiten für die Register-Signale durch das verzögerte Taktsignal (im Diagramm durch die gestrichelten Linien angedeutet) und bei kleinen Registerbänken ein größerer Platzbedarf.

2.6.4. Glitch-Eliminierung

Glitches sind zusätzliche Schaltvorgänge in Gattern, die durch unterschiedliche Signallaufzeiten in parallelen Signalpfaden entstehen.

[16] Dies ist ein komplexer Vorgang, da in der RT-Spezifikation von einer vollständigen Synchronität ausgegangen wird, d. h. die Taktflanke an allen flankengesteuerten Einheiten früher ankommt als jede in diesem Taktzyklus vorkommende kombinatorische Pegeländerung. Für die durch Clock-Gating abgetrennten Schaltungteile ist dies nicht mehr der Fall, was bedeutet, dass die Register-Daten über den Beginn der Taktflanke stabil sein müssen (eine positive Hold-Zeit haben). Bis 2003 waren zur Implementierung von Clock-Gating noch aufwändige und schwer validierbare manuelle Änderungen am Quellcode wie etwa in [146] beschrieben nötig.

Sie lassen sich durch Strukturen eliminieren, die solche parallelen Signalpfade vermeiden, z. B. durch ein Ausbalancieren der Pfade. Diese Art der Optimierung kann durch den PowerCompiler von Synopsys automatisch erfolgen, wodurch der Energiebedarf je nach der durchschnittlichen logischen Tiefe um 30-70% gesenkt werden kann [18].

2.6.5. Absenken der Versorgungsspannung

Da die dynamische Verlustleistung vom Quadrat der Versorgungsspannung abhängt (Gl. (2.11)), bringt eine V_{dd}-Absenkung einen vergleichsweise großen Effekt. Nach dem α-Modell (Gl. (2.13)) hat dies jedoch eine erhöhte Schaltzeit der Transistoren zur Folge.

Diese erhöhte Schaltzeit führt normalerweise dazu, dass die Schaltung nicht mehr funktionieren wird. Daher ist eine oder mehrere der folgenden Maßnahmen notwendig:

1. Die Schwellenspannung V_t muss abgesenkt werden, um die gleiche Transistorverzögerung wie beim Betrieb mit der vollen Versorgungsspannung zu erzielen (Gl. (2.13)). Dies hat höhere Leckströme zur Folge (Gl. 2.14), die die Einsparungen bei der dynamischen Verlustleistung schnell zunichte machen können. Der Hersteller wählt üblicherweise einen Kompromiss zwischen V_{dd} und V_t. Eine V_t-Skalierung kann dennoch mit Vorteilen verbunden sein, wenn z. B. der Betriebstemperaturbereich[17] eingeschränkt wird [58].

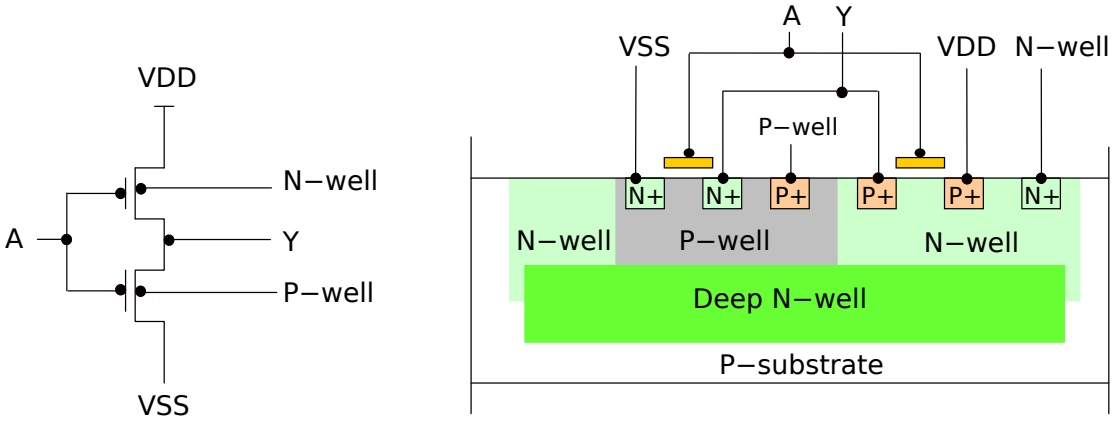
2. Der kritische Pfad muss verkürzt werden. Dies kann automatisch durch die Vorgabe einer höheren Sollfrequenz bei der Synthese der Schaltung geschehen. Im Betrieb wird die Schaltung dann mit einer weit geringeren Taktfrequenz betrieben, was einen gewissen Spielraum für eine V_{dd}-Absenkung schafft. Eine Geschwindigkeitsoptimierung der Schaltung ist somit immer auch eine mögliche Energieoptimierung [161].

3. Die Schaltung hat mehrere V_{dd}-Domänen. Für schnelle Logik verwendet man Standardzellenvarianten mit niedrigem V_t (und hoher Ruhestromaufnahme), für die langsame Logik kommen Standardzellenvarianten mit hohem V_t zum Einsatz [77]. Der PowerCompiler von Synopsys kann eine solche Partitionierung automatisch durchführen, wenn eine Dual-V_t-Bibliothek zum Einsatz kommt; dies ist jedoch ein nicht-triviales Optimierungsproblem [11].

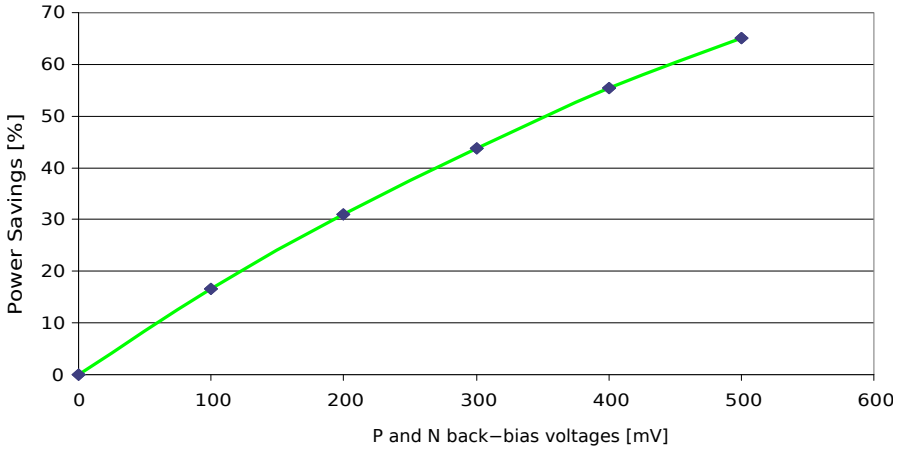
4. Durch spezielle Gestaltung der Standardzellen ist es möglich, eine dynamische V_t-Skalierung im Betrieb vorzunehmen [81]. Durch eine zusätzliche analoge Spannung wird der Kanal vorgespannt, dadurch wird V_t in bestimmten Grenzen einstellbar. Das Prinzip ist in Abb. 2.10 dargestellt.

   Dies erfordert allerdings neben der Herstellung dieser speziellen Gatter auch eine Steuerung (die auch Energie benötigt) [120]. Der Efficeon-Prozessor der Firma Transmeta verwendet einen solchen Mechanismus, um Software-gesteuert Teile des Chips abschalten oder in ihrer maximalen Taktgeschwindigkeit skalieren zu können.

[17] Aufgrund der Eigenleitung von Silizium steigen die Leckströme mit steigender Betriebstemperatur exponentiell an (siehe auch Abschn. 2.5.1).



(a) CMOS-Struktur mit V_t Biasing



(b) Einsparungen durch Biasing

Abbildung 2.10.: Technologische Minimierung der Unterschwellströme. Durch das Vorspannen des p- und n-Kanals mit analogen Spannungen kann V_t beeinflusst werden. Es ist somit möglich, Schaltungsteile, die gerade nur langsam getaktet werden, durch eine V_t-Erhöhung zu einem geringeren Leckstrom zu verhelfen. Wird wieder die maximale Taktfrequenz benötigt, wird V_t entsprechend abgesenkt. In Abb. (b) ist die Reduktion der Ruhestromaufnahme in Abhängigkeit der Vorspannung aufgetragen, Abb. (a) zeigt den schematischen Aufbau einer solchen Schaltung.

2.6.6. Bewertung

Dynamische Verlustleistung

Die beschriebenen Methoden stellen alle effektive Mittel zur Senkung des Energiebedarfs einer auf statischer CMOS-Logik basierenden Schaltung dar.

Die Operandenisolierung wurde weitgehend durch das Clock-Gating verdrängt. Das wahlweise Abschalten des Taktes stellt eine wirkungsvolle Methode dar, einzelne, vorübergehend nicht benötigte Schaltungsteile vom Haupttakt abzukoppeln. Ergebnisse dazu finden sich in Abschnitt 5.2.4. Es ist mittlerweile zu eine Standardmethode geworden, die von allen Synthesewerkzeugen der großen Hersteller unterstützt wird.

Die Propagierung von Konstanten kann relativ selten genutzt werden, ist, wenn sie zum Einsatz kommt, aber sehr effektiv und seit längerem automatisch unterstützt.

Eliminierung von Glitches birgt ein erstaunlich hohes Einsparpotenzial, ist aber ein schwieriges Optimierungsproblem, das inzwischen von kommerziellen Werkzeugen beherrscht wird.

Statische Verlustleistung

Alle oben genannten Methoden senken im wesentlichen die dynamische Verlustleistung. Aufgrund der tendenziell zunehmenden statischen Verlustleistung ist es wichtig, auch dieses Problem im Auge zu haben. Neben Verbesserungen auf technologischer Ebene (Abschnitt 2.2.2) kann von den oben erwähnten Methoden nur eine V_{dd}-Skalierung in Verbindung mit einer V_{t}-Anpassung helfen, die nebenbei auch ein effektives Mittel zur Reduktion der dynamischen Verlustleistung ist.

Eine V_{dd}-Skalierung wird bereits durch den PowerCompiler unterstützt, der MTCMOS[18]-Standardzellbibliotheken nutzen kann, um den Schaltkreis in Bereiche mit hohem V_{t} und Bereiche mit niedrigem V_{t} partitionieren zu können. Da für diese Arbeit keine MTCMOS-Bibliothek zur Verfügung stand, konnte leider nicht evaluiert werden, wie gut der PowerCompiler dieses Optimierungsproblem lösen kann.

Eine reine Absenkung der Versorgungsspannung auf Werte unterhalb der Herstellerspezifikation ist auf nicht-empirischer Basis unseriös, da die Hersteller-Charakterisierung einer Bibliothek weder die dann geltenden Gatterverzögerungen noch den Energieverbrauch liefert. Theoretische Abschätzungen werden in der Literatur auf Basis des α-Modells gemacht (Abschnitt 2.5.1), dieses stellt jedoch nur eine Näherung dar[19].

Im Entwicklungs-Framework (Kapitel 4) wird neben Architektur-Optimierungen eine Kombination aller oben genannten Optimierungsstrategien eingesetzt. Eine V_{dd}-

[18] *Multiple Threshold Voltage CMOS*

[19] Weiterhin ist es schwierig, anhand eines algorithmischen Modell auf den minimalen kritischen Pfad der resultierenden Hardware zu schließen, der für die Abschätzung der minimalen Versorgungsspannung ebenfalls wichtig ist. Dieser lässt sich in der Regel nur durch die Durchführung einer realen Synthese abschätzen, da hier Heuristiken und nicht-deterministische Optimierungsverfahren zum Einsatz kommen. Die Literaturangaben zur maximalen Energieeinsparung einer Transformation unter Berücksichtigung einer Versorgungsspannungsabsenkung lassen sich deshalb nicht exakt nachvollziehen, da sie letztlich auch zieltechnologieabhängig sind.

Skalierung wird durch eine Geschwindigkeitsoptimierung beim Compile-Vorgang ermöglicht, die über das benötigte Maß hinausgeht; die möglichen Einsparungen werden jedoch aufgrund der oben genannten Schwierigkeiten nicht bewertet. Alle in dieser Arbeit genannten konkreten Energieziffern sind direkt aus der Standardzellen-basierten Verlustleistungsschätzung entnommen.

2.7. Ebenen für die Hardwareentwurf

Für den Entwurf komplexer schaltungstechnischer Systeme, die viele Millionen Transistoren auf einem Chip vereinen und zudem kontrolliert mit der Umwelt interagieren sollen, ist ein strukturiertes Vorgehen unerlässlich. Im nächsten Abschnitt wird diese Entwurfssytematik beschrieben.

2.7.1. Vorgehen beim Hardwareentwurf

Um den Entwurf sowie den Test und die Validierung komplexer Systeme übersichtlich zu halten, werden diese im allgemeinen auf einem möglichst hohen Abstraktionsniveau konzipiert. Bei der Implementierung arbeitet man sich dann Schritt für Schritt nach unten, bis man bei dem platzierten und verdrahteten Chip landet. Ein solches Vorgehen wird als *Top-Down-Design* bezeichnet[20]. Die verschiedenen Abstraktionsniveaus sind im nächsten Abschnitt beschrieben.

Um den Entwurfs- und Verifikationsaufwand gering zu halten, wird heute zunehmend auf fertige Bausteine zugegriffen, mit denen sich ein System wesentlich schneller aufbauen lässt als bei dem klassischen Top-Down-Vorgehensweise. Diese Bausteine nennt man IP-Blöcke[21]; dies können in der Softwaretechnik (Klassen-)Bibliotheken sein. In der Hardwareentwicklung sind dies meistens sog. *Cores*, damit bezeichnet man üblicherweise fertig verdrahtete Chip-Module, die ein genau definiertes I/O-Timing einhalten. Zunehmend werden aber auch Blöcke auf höheren Abstraktionsebenen vermarktet, von denen auch in dem im Rahmen dieser Arbeit entworfenen Framework gebraucht gemacht wird (siehe Abschnitt 4.6.2 und Abschnitt 4.7.2). Das Framework verfügt auch über eine eigene Modulbibliothek mit Modulen unterschiedlicher Abstraktionsebenen. Diese wird in Abschnitt 4.6 beschrieben.

2.7.2. Entwurfsebenen

Üblicherweise werden beim Entwurf elektronischer Schaltkreise fünf verschiedene Abstraktionsstufen unterschieden [170], nämlich die Systemebene, die algorithmische Ebene, die Register-Transfer-Ebene, die Logik-Ebene sowie die Schaltkreis-Ebene. Da die Abgrenzung nicht immer eindeutig, für die Einordnung des Frameworks aber wichtig ist, werden die Ebenen an dieser Stelle kurz definiert:

Systemebene Auf dieser Ebene werden die grundlegenden Schnittstellen und Eigenschaften des informationsverarbeitenden Gesamtsystems beschrieben. Hier wird insbesondere die eventuelle Anbindung an externe Module spezifiziert und eine grobe Partitionierung des Systems vorgenommen, z. B. in Hardware- und Software-Blöcke.

[20] Der Begriff kommt aus der Softwaretechnik und wurde mit dem Aufkommen strukturierter Programmiersprachen geprägt [178]. Das Vorgehen gründet auf der Beobachtung, dass die Implementierung eines komplexen Systems auch im Detail stark von den Anforderungen an das Gesamtsystem geprägt ist.

[21] *Intellectual Property Core*. Vorgefertigtes, meist kommerzielles Hardwaremodul, das nicht im Quelltext, sondern einer strukturell nicht einsehbaren Form vorliegt.

Algorithmische Ebene/ Verhaltensebene Hier erfolgt die Beschreibung eines Systems oder Teilsystems durch nebenläufige Algorithmen. Man unterscheidet hier zwei Sichtweisen: Auf der einen Seite gibt es die strukturelle Sicht, die aus miteinander über (unterschiedliche) Schnittstellen kommunizierenden Blöcken zusammengesetzt ist[22]. Auf der anderen Seite gibt es die Verhaltenssicht, in der das System direkt als Algorithmus mit Variablen und Operatoren dargestellt wird. Dies ist z. B. eine Beschreibung in einer Hochsprache wie C++, BC-VHDL (Behavioral Compiler-VHDL, siehe Kap. 4) oder auch ein Ablaufdiagramm.

Register-Transfer-Ebene Während die vorigen Ebenen allgemein für informationsverarbeitende System gelten, erfolgt ab dieser Ebene eine Spezialisierung auf den Entwurf elektronischer Schaltungen. Auf der Register-Transfer-Ebene (auch RT-Ebene oder RTL (*Register-Transfer-Level*) genannt) wird eine Schaltung durch arithmetische Operationen und den Transfer der verarbeiteten Operanden zwischen Registern beschrieben. Es wird hier eine weitere Unterteilung in *strukturellen RT-Code* und *funktionalen RT-Code* vorgenommen. Die funktionale Beschreibung ist die Spezifikation über einen endlichen Automaten (FSM, *Finite State Machine*), die strukturelle Sicht beschreibt die Schaltung mit ihren kompletten Ressourcen wie Registern, Multiplexern oder arithmetischen Einheiten.

Eine funktionale Beschreibung ist heute als Eingangsbeschreibung für die Hardwaresynthese üblich, da solche Beschreibungen inzwischen automatisch auf die strukturelle RT-Ebene überführt werden können, d. h. *synthetisierbar* sind.

Logikebene Auf dieser Ebene wird eine digitale Schaltung durch logische Verknüpfungen und Verzögerungszeiten bestimmt. In der strukturellen Darstellung wird eine Schaltung durch einen Gatterschaltplan dargestellt. In der funktionalen Sicht erfolgt die Darstellung durch Boolesche Ausdrücke. Eine Simulation auf dieser Ebene wird mit rein digitalen Werten durchgeführt. Die Beschreibung liegt dann meist als VHDL- oder Verilog-Gatternetzliste vor.

Schaltkreisebene Auf der Schaltkreisebene werden die Gatter auf Logikebene in ihre Bestandteile zerlegt, d. h. in Transistoren, Widerstände, Induktivitäten und Kapazitäten mit jeweils gatterspezifischen Werten. Eine Simulation auf dieser Ebene berücksichtigt demnach die physikalischen Eigenschaften detaillierter und wird mit Analog-Simulatoren wie Hspice durchgeführt.

Jede beschriebene Ebene kann nach Verhalten, Struktur und Geometrie kategorisiert werden. Eine populäre Übersicht über die Entwurfsebenen in den verschiedenen Kategorien bietet das Gajski-Walker-Diagramm (Abb. 2.11).

[22] Im allgemeinen hat die gewählte Darstellung keinen Bezug zur späteren Hardwarepartitionierung und kommt ohne zeitliche Detaillierungen wie Takt oder Rücksetzsignale aus. Die in der Signalverarbeitung beliebte Modellierung mit Simulink (siehe Abschnitt 3.5, Abb. 3.3) ist daher eine Abstraktionsebene tiefer anzusiedeln, da die dort verwendeten Blöcke ein definiertes Taktverhalten haben

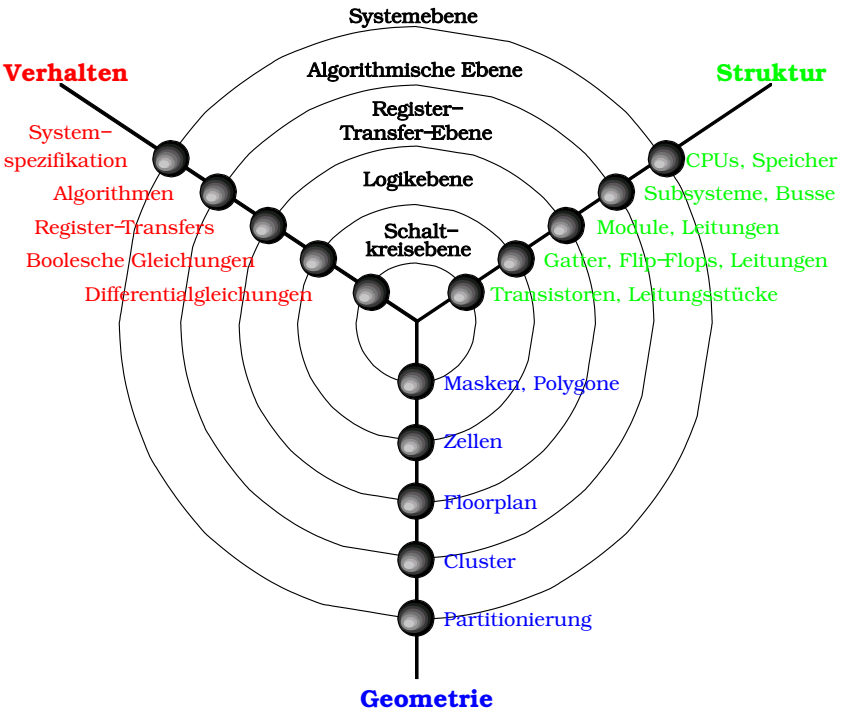


Abbildung 2.11.: Gajski-Walker-Diagramm. Die verschiedenen Entwurfsebenen von der Systemebene bis zur Schaltkreisebene sind hier durch konzentrische Kreise dargestellt. Die Ebenen werden dann aus Verhaltenssicht, Struktursicht und Geometrie erläutert. Nach [170].

2.7.3. Entwurfsablauf

Klassischer Entwurfsablauf

Der klassische Entwurfsablauf mit den dazugehörigen Tätigkeiten und Verifikationsmöglichkeiten für den VHDL-basierten Hardwareentwurf ist in Abb. 2.12 gezeigt.

Die Spezifikation erfolgt dabei auf der Systemebene. Hier kann in den meisten Fällen keine automatisierte Verifikation erfolgen, die Überprüfung findet hier von Hand statt[23]. Die nächste Tätigkeit ist die Verfeinerung des Entwurfs, die in eine Verhaltensbeschreibung auf algorithmischer Ebene mündet. Dies kann ein Flussdiagramm, aber auch z. B. ein C++-Programm sein. An dieser Stelle ist daher u.U. eine automatische Verifikation möglich.

Auf jeden Fall erfolgt nun eine händische Umsetzung in eine Hardwarebeschreibungssprache wie SystemC, VHDL oder Verilog, die üblicherweise auf RT-Ebene angesiedelt ist. Ab dieser Ebene kann eine Verifikation mittels Simulation durchgeführt werden und der Übergang auf die jeweils tiefere Ebene erfolgt automatisiert.

[23] Immer mehr an Bedeutung gewinnen hier Modell-basierte Entwurfsmethoden, die eine Verifikation durch Simulation der Umgebung ermöglichen. Dies bezeichnet man als *Model-based Design*. Bekanntestes Werkzeug für dieses Verfahren ist Simulink.

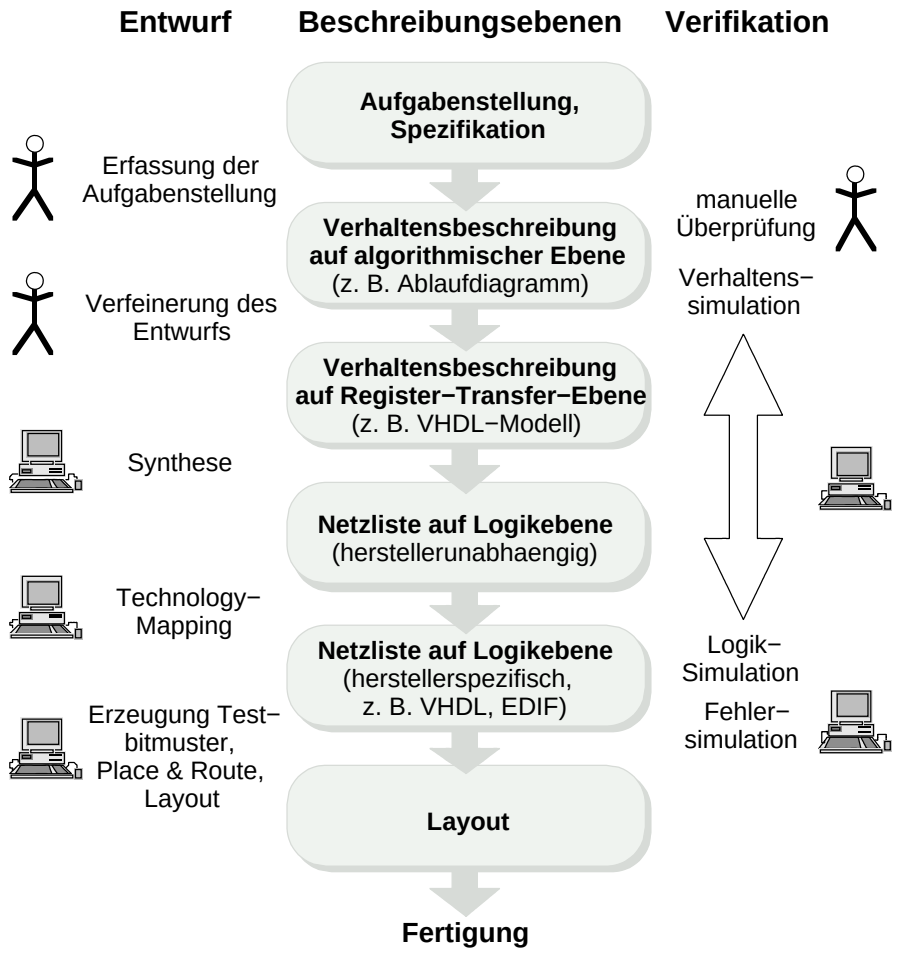


Abbildung 2.12.: Entwurfsablauf für den VHDL-Entwurf. Aus [88].

Entwurf mit High-Level Synthese

Unter High-Level-Synthese oder Behavioral-Synthese versteht man eine automatische Synthese aus einer Spezifikation auf algorithmischer oder Verhaltens-Ebene. Konkret bedeutet dies, dass keine händische Umsetzung des Verhaltensentwurfs in einen Entwurf auf RT-Ebene stattfindet, sondern ein Werkzeug diese Umsetzung übernimmt.

Obwohl erste Ansätze schon in den 60iger Jahren existierten [47], hat dieser Ansatz erst in den 80igern Einzug in industrielle Prozesse erhalten. Dabei kamen immer spezialisierte Systeme zum Einsatz, die z. B. für Aspekte der digitalen Signalverarbeitung [93] oder Prozessor-Synthese [115] geeignet waren.

Werkzeugen für die High-Level-Synthese, die ähnlich wie RT-Synthese-Werkzeuge an keine spezifische Domäne gebunden waren, blieb der (industrielle) Durchbruch jedoch bis heute verwehrt. Als einziges Produkt dieser Kategorie existierte bis 2004 der Behavioral Compiler der Firma Synopsys. Erst in jüngster Zeit wendet sich die Industrie wieder der Behavioral-Synthese zu. Eine genauere Beschreibung der Grundlagen der High-Level-Synthese erfolgt in Abschnitt 4.5.1.

2.7.4. Bewertung

Die aus der Literatur bekannten und in Abschnitt 2.7.1 beschriebenen, für den Hardwareentwurf relevanten Entwurfsebenen sind sowohl für den klassischen, auf der RT-Synthese beruhenden Entwurf gültig als auch für einen abstrakteren Ansatz, der in Abschnitt 2.7.3 angerissen wurde, sinnvoll, weshalb sie hier relativ ausführlich beschrieben wurden.

2.8. Zahlensysteme für die Audioverarbeitung

Die Wahl des Zahlensystems, d. h. der mathematischen Repräsentation der Zahl und der Abbildung auf eine digitale Repräsentation kann für den Energiebedarf einer Schaltung von entscheidender Bedeutung sein. Dies wird schon aus der Vielzahl der aktuellen Veröffentlichungen ersichtlich, die teilweise von Energieeinsparungen > 50 % berichten, ohne die Signalverarbeitungsqualität herabzusetzen, z. B. in [12, 118, 171, 177, 176, 160, 159, 96, 23].

In der Praxis ist es allerdings oft so, dass sich das Zahlensystem nicht ohne weiteres ändern lässt, da ein zur Hardwareimplementierung anstehender Algorithmus in der Regel auf Gleitkommaberechnung basiert und auch so qualifiziert wurde. Ein Wechsel des Zahlensystems erfordert also umfangreiche Requalifikationen, daher ist es ratsam, sich schon zu Beginn eines Projektes darüber Gedanken zu machen.

Im folgenden werden die in dem Entwurfs-Framework verwendeten Zahlensysteme vorgestellt. Dies sind im einzelnen:

1. Logarithmisches Zahlensystem
2. Residuenbasiertes Zahlensystem
3. Gleitkomma-Zahlensystem
4. Festkomma-Zahlensystem

Anschließend wird eine Bewertung vorgenommen.

2.8.1. Logarithmisches Zahlensystem

Bei der logarithmischen Zahlendarstellung (*LNS, Logarithmic Number System*) wird anstelle der Zahl x deren Logarithmus zur Basis b gespeichert: $l = \log_b(x)$. Da der Logarithmus einer Zahl betragsmäßig immer größer als Null ist, wird zusätzlich eine Null-Darstellung sowie ein Vorzeichen gespeichert, meist in Form von Null-Flag z und Vorzeichen-Flag s. l wird dabei als Festkommazahl gespeichert.

Diese Darstellung hat gegenüber einer linearen Zahlendarstellungen zwei Vorteile:

1. Werte um die Null werden feiner quantisiert als größere Werte, der *relative Quantisierungsfehler* ist über den ganzen Darstellungsbereich fast konstant.
2. Die Multiplikation reduziert sich auf eine Addition, die Division auf eine Subtraktion. Die LNS-Potenzierung reduziert sich auf eine Multiplikation (siehe Tab. 2.1).

In [113, 160] zeigen die Autoren, dass sich gegenüber einer Festkommadarstellung eine deutlich geringere Schaltaktivität für ein unkorreliertes Eingangssignal ergibt.

Dem gegenüber steht für bestimmte Operationen ein erheblicher Aufwand:

Wie aus Tabelle 2.1 ersichtlich, wird die Berechnung einer Addition komplizierter, da hierzu die Berechnung des Logarithmus' nötig ist. Dies wird meist über eine Approximation auf Grundlage einer Tabelle gelöst [14]. Die Potenzierung kann bei der Wahl der Basis b zu 2 durch eine einfache *shift*-Operation implementiert werden, die zwar strukturell einfach ist, aber (bei variabler Länge) mehrere Taktzyklen benötigt.

Lineare Operation	LNS-Operation
$Z = X \pm Y = b^x \pm b^y = b^x(1 \pm b^{y-x})$	$z = x + \log_b(1 \pm b^{y-x})$
$Z = X \cdot Y = b^x b^y = b^{x+y}$	$z = x + y$
$Z = X \div Y = b^x / b^y = b^{x-y}$	$z = x - y$
$Z = X^m = (b^x)^m$	$z = x \cdot m, m \in \mathbb{Q}$

Tabelle 2.1.: LNS-Operationen für $x = log_b(X), y = log_b(Y), z = log_b(Z)$. Links steht zunächst die lineare Operation, dann wird gemäß obiger Vorschrift substituiert. In der rechten Spalte stehen die zugehörigen LNS-Operationen. Multiplikation, Division und Potenzierung wird vereinfacht, Addition und Subtraktion wird jedoch sehr aufwendig.

Unter Energieaspekten lohnenswert ist die Verwendung dieses Zahlensystems daher nur, wenn auf einfache Additionen weitgehend verzichtet wird. Dies erfordert im allgemeinenen tiefgreifende Veränderungen am zu implementierenden Algorithmus, kann dann aber hohe Einspareffekte zeigen. In [102] beschreiben R.E. Morley et al. ein digitales Hörgerät, das auf eine LNS-Arithmetik basiert und im Vergleich zu anderen Geräten etwa 90% weniger Strom verbraucht. In [23] wird der generelle Einfluss des Zahlensystems auf Audiosignalverarbeitung untersucht, auch hier werden mit LNS große Einsparungen erzielt. Auf einer FPGA-Plattform konnten ebenfalls Einspareffekte gezeigt werden [96].

2.8.2. Residuenbasiertes Zahlensystem

Bei der Darstellung im residuenbasierten Zahlensystem RNS (*Residue-Number-System*) werden Divisionsreste der zu repräsentierenden Zahl zu vorher festgelegten Moduln M gespeichert. Die Gesamtheit der Moduln

$$M = \{m_1, m_2, ..., m_N\} \tag{2.16}$$

wird auch *Basis* genannt. Für eine injektive mathematische Abbildung der Residuen $I(n)$ einer Zahl n

$$n \rightarrow \{\langle n \rangle_{m1}, \langle n \rangle_{m2}, ..., \langle n \rangle_{mN}\} \qquad \text{mit } \langle n \rangle_m \equiv n \bmod m \tag{2.17}$$

auf die darzustellende natürliche Zahl $n \in \mathbb{N}$, müssen die Elemente der Basis paarweise koprim sein, d. h. sie dürfen keinen Primteiler gemeinsam haben. Der Wertebereich ergibt sich damit durch das Produkt aller Moduln[24]. Die Kodierung der Residuen $\langle n \rangle_m$ erfolgt meist in binärer Stellenwertkodierung, ist grundsätzlich aber beliebig.

Der große Vorteil des RNS ist die Vereinfachung der Operationen *Addition, Subtraktion* und *Multiplikation*. Für zwei RNS-Zahlen I und J, die Repräsentationen der natürlichen

[24] Die genaue Theorie dieser Darstellung kann man in vielen Lehrbüchern über die Zahlentheorie nachlesen, eine kurze Einführung findet sich z. B. in [154].

Zahlen i und j sind, ist die Rechenvorschrift für obige Operationen

$$\begin{aligned} P &= I \otimes J \\ &= \{\langle\langle I\rangle_{m1} \otimes \langle J\rangle_{m1}\rangle_{m1}, \langle\langle I\rangle_{m2} \otimes \langle J\rangle_{m2}\rangle_{m2}, \ldots, \\ &\quad \langle\langle I\rangle_{m2} \otimes \langle J\rangle_{mN}\rangle_{mN}\}. \end{aligned} \tag{2.18}$$

Die Residuen zu jedem Basiselement werden also unabhängig voneinander verknüpft, damit ist die Länge des kritischen Pfades in Hardwarearchitekturen im Gegensatz zu anderen Zahlendarstellungen unabhängig von der Länge der Basis.

Die Nachteile dieses Zahlensystem sind die folgenden:

1. Für die Darstellung negativer Zahlen muss entweder eine fester Offset vereinbart werden oder das Vorzeichen muss gesondert gespeichert werden.
2. Um Zahlen aus $\mathbb{Q}$ darstellen zu können, muss ein fester Faktor vereinbart werden, mit dem die Zahl zu multiplizieren bzw. zu dividieren ist (vergleichbar mit der Position des Kommas in einem Stellenwertsystem, siehe 2.8.4).

   Eine schnelles Shiften der Zahl wie in der Festkommadarstellung, welches in der digitalen Signalverarbeitung häufig nötig ist, um Wertebereichsanpassungen zu machen, ist im RNS nicht möglich. Dazu müsste der Skalierungsfaktor als eine Potenz aller Reste darstellbar sein, was der Forderung nach der paarweisen Koprim-Eigenschaft der Basiselemente widerspricht.
3. Überläufe des Darstellungsbereichs sind sehr schwer zu erkennen, da Werte außerhalb dieses Bereiches auf einen Wert innerhalb des Darstellungsintervalls abgebildet werden.
4. Die Format-Umwandlung in eine n-näre Stellenwertkodierung und zurück ist relativ aufwendig. Durch geschickte Wahl der Basis lässt sich der Aufwand jedoch stark reduzieren [158].

Für die digitale Signalverarbeitung erweisen sich die letzten drei Nachteile als entscheidend.

Trotzdem gibt es auch in jüngerer Zeit immer wieder Forschungsansätze, die auf RNS basieren, da die Vorteile gerade für große Wertebereiche, die eine äquidistante Skalierung erfordern, sehr verlockend sind [141, 118, 158, 173, 22, 32].

Die Zahlendarstellung wurde in das Framework integriert, um die Quantisierungseigenschaften in einem frühen Entwurfsstadium bewerten zu können.

2.8.3. Gleitkomma-Zahlensystem

Bei der Gleitkomma-Darstellung wird eine Zahl z durch eine vorzeichenbehaftete Mantisse m multipliziert mit einer Potenz zu einer festen, ganzzahligen Basis b dargestellt:

$$z \equiv m \cdot b^e \quad m \in \mathbb{Q},\ e \in \mathbb{Z} \tag{2.19}$$

Der Exponent e wird dabei so gewählt, dass die Mantisse m im Wertebereich $[1, b[$ liegt. Diese, sog. normalisierte Darstellung ist eindeutig. Mantisse und Exponent werden

üblicherweise in binärer Stellenwertkodierung gespeichert, die Basis b ist in der Regel 2.

Bei den in der Computertechnik weit verbreiteten IEEE 754-Gleitkommazahlen wird das Vorzeichen der Zahl zudem getrennt gespeichert und es gibt eine Sonderdarstellung z. B. für die Null [57].

Werte um den Nullpunkt werden in der halblogarithmischen Darstellung besonders fein quantisiert, zu den Grenzen des Wertebereichs wird der Quantisierungsabstand immer größer. Die relative Auflösung ist näherungsweise konstant (siehe Abb. 2.13).

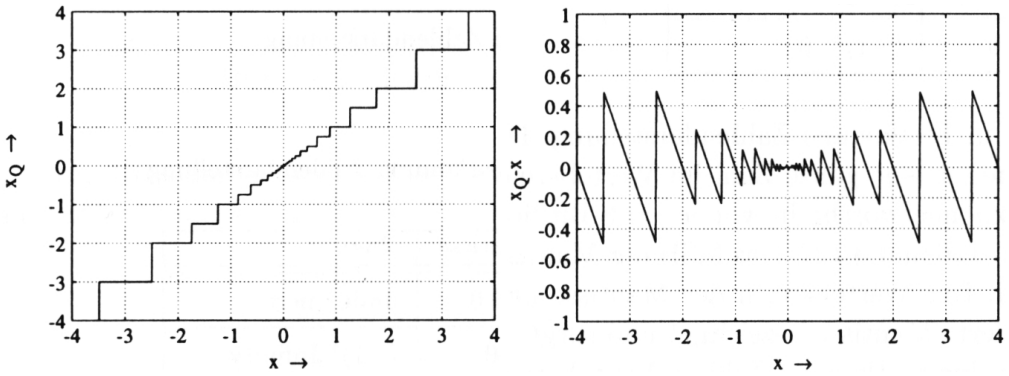


Abbildung 2.13.: Quantisierung und Auflösung von Gleitkommazahlen. Links dargestellt ist die Quantisierungskennlinie für eine Mantissen- und Exponentenbitbreite von 3 Bit, rechts der absolute Fehler (aus [184]).

Die einfachen arithmetischen Verknüpfungen für zwei Gleitkommazahlen $X = (m_x, e_x)$ und $Y = (m_y, e_y)$ sind wie folgt definiert (o.E.: $e_x \geqslant e_y$):

$$\begin{aligned} X \;+\; Y \;&\equiv\; (m_x\, b^{e_x - e_y} + m_y)\, b^{e_y} \\ X \;-\; Y \;&\equiv\; (m_x\, b^{e_x - e_y} - m_y)\, b^{e_y} \\ X \;\cdot\; Y \;&\equiv\; (m_x \cdot m_y)\, b^{e_x + e_y} \\ X \;\div\; Y \;&\equiv\; (m_x \div m_y)\, b^{e_x - e_y} \end{aligned} \tag{2.20}$$

Man sieht, dass die Operationen in zwei Schritte unterteilt werden können: Auf der einen Seite Exponentenoperationen wie das Vergleichen und Subtrahieren/Addieren, auf der anderen Seite Festkomma-Mantissenoperationen. Aufgrund dieser Zweiteilung eignen sich Gleitkomma-ALUs sehr gut für eine Pipeline-Implementierung. Ein beispielhafte Beschreibung anhand der MC68040-ALU findet sich in [70].

2.8.4. Festkomma-Zahlensystem

In der Festkomma-Darstellung wird eine Zahl z als N-stellige Folge von ganzzahligen Koeffizienten c_i darstellt, wobei jede Position n in der Folge eine feste Wertigkeit b^n hat.

Der Wert ergibt sich als Summe über alle Koeffizientenprodukte:

$$z \equiv \sum_{n=0}^{N} c_n \, b^{n+k}, \quad c_i < b \tag{2.21}$$

Der Wertebereich ist hier b^N. Der feste Offset k bestimmt dabei die Position des Kommas, für $k = 0$ sind nur ganze Zahlen darstellbar. Die Abstände der Zahlen auf dem Zahlenstrahl sind äquidistant, d. h. Festkommazahlen sind linear quantisiert (Abb. 2.14).

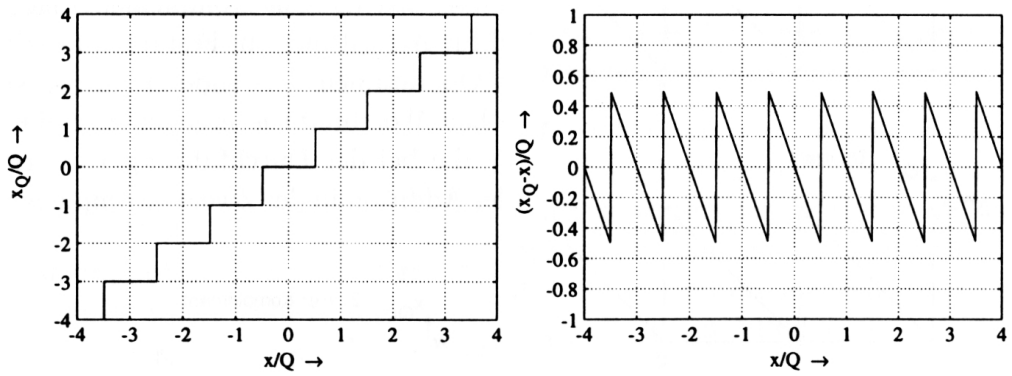


Abbildung 2.14.: Quantisierung und Auflösung von Festkommazahlen. Links dargestellt ist die Quantisierungskennlinie für eine Bitbreite von 3 Bit, rechts der absolute Fehler (aus [184]).

Für die Darstellung von negativen Zahlen kann ein Vorzeichenbit vereinbart werden (*Sign-Magnitude-Darstellung*) oder die Kodierung der Zahl erfolgt als sog. Zweierkomplement. Dies hat den Vorteil, das die Additionsarithmetik das Vorzeichen nicht beachten muss [142]; dies führt i.A. zu einfacheren Hardwarearchitekturen, kann unter energetischen Gesichtspunkten aber ungünstiger sein als die Vorzeichendarstellung.

Sowohl die Vorzeichen-Darstellung als auch die Zweier-Komplement-Darstellung ist im Entwicklungs-Framework integriert.

2.8.5. Bewertung

Aufgrund der einfachen Struktur ist für Hardwarearchitekturen in ASICs im Bereich digitaler Signalverarbeitung die Festkomma-Arithmetik vorherrschend. Auch bei Mittelklasse-DSPs mit 16- oder 32-Bit Architektur wie z. B. dem Blackfin-Prozessor [1] fehlt in der Regel eine Hardwareeinheit für Fließkommaarithmetik, so dass die meisten Software-Architekturen z.Zt. ebenfalls noch in Festkomma-Arithmetik implementiert werden. Dies bringt z. T. massive Probleme bei der Quantisierung durch Skalierung der Zahlen mit sich, die aber z. T. automatisiert werden kann [98].

Prozessoren mit Vielzweck-Bestimmung wie z. B. PC-Prozessoren verfügen bereits seit Mitte der Neunziger über eingebaute Koprozessoren für die Fließkommaverarbeitung.

Andere Zahlensystem wie LNS und RNS tauchen in der wissenschaftlichen Literatur zwar in jüngerer Zeit nicht zuletzt aufgrund der Energieproblematik immer wieder

auf, fristen aber ein Nischendasein. Obwohl für diese System einige Demonstrator-ASICs bereits geschaffen wurden (z. B. [12, 13, 101]) überwiegen letztlich die Nachteile, insbesondere in der Entwurfs- und Qualifikationsphase für den zu implementierenden Algorithmus, da u. a. die Vermeidung von Bereichsüberläufen (die bei Fließ- und Festkommazahlen leicht festgestellt werden können) hohe Anforderungen an den Entwickler stellt.

Das Fließkommaformat ist in der Softwarewelt bereits fest etabliert und erscheint sehr viel einfacher beherrschbar als die alternativen Systeme. Mit zunehmender Modularisierung und Strukturverkleinerung wird es immer einfacher und kostengünstiger, Fließkomma-Einheiten auch in ASICs zu integrieren und scheint angesichts der Vorteile bei Berechnung und Konvertierung die Exoten RNS und LNS immer weiter an den Rand zu drängen.

2.9. Digitale Audiofilter

Dieser Abschnitt bietet einen knappen Überblick über die digitale Verarbeitung insbesondere von Audiosignalen. Nach einer Einführung in die mathematischen Grundlagen werden geeignete mathematische Darstellungsformen der Filter diskutiert. Davon ausgehend wird dann auf schaltungstechnische Realisierungsmöglichkeiten eingegangen, die in Abschnitt 4.6 wieder aufgegriffen werden.

Da es in dieser Arbeit um digitale Schaltungstechnik geht, beschränken sich die Ausführungen auf den zeitdiskreten Bereich. Auf Probleme bei dem notwendigen Übergang vom zeitdiskreten in den zeitkontinuierlichen Bereich wie Abtastung und Rekonstruktion wird nicht eingegangen, da der eigentliche Filterentwurf zur Algorithmusspezifikation gehört und hier nicht weiter vertieft werden kann. Der Entwurf solcher Systeme wird aber in vielen Standardwerken zur Systemtheorie diskutiert (z. B. [46]).

Da *Filtertransformationen* im Entwurfs-Framework eine große Rolle spielen, wird die Problematik der Stabilität und Quantisierung digitaler Filter angerissen.

Trotz der Kürze des Kapitels wurde versucht, eine konsistente, wenn auch sehr geraffte Darstellungsform zu finden.

2.9.1. Grundlagen

Ein *Digitalfilter* ist ein System $\mathcal{S}$, das eine diskrete Eingangssignalfolge in eine Ausgangssignalfolge $y[k] \equiv \mathcal{S}(x[k])$ umformt (Abb. 2.15). Diskrete, zeitinvariante und lineare Systeme[25] führen bei mathematischen Beschreibungsversuchen auf eine Abhängigkeit des Ausgangs y zum Zeitpunkt k vom Eingangssignal x der Form

$$y[k] \equiv -\sum_{i=1}^{n} a_i\, y[k-i] + \sum_{l=0}^{m} b_l\, x[k-l] \tag{2.22}$$

Das Ausgangssignal hängt dabei von einer Linearkombination von Ausgangssignalen zu früheren Zeitpunkten (linker Term) sowie einer Linearkombinationen von Eingangssignalen zum jetzigen ($x[k]$) und zu früheren Zeitpunkten (rechter Term) ab. Die Zeitinvarianz zeigt sich dabei in den konstanten Koeffizienten a_i und b_i. Bei (2.22) handelt es sich um eine *lineare Differenzengleichung*. In kontinuierlichen Systemen entspricht dies einer linearen Differentialgleichung mit konstanten Koeffizienten.

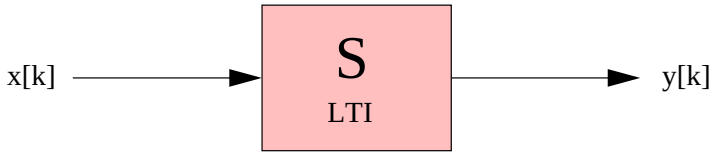


Abbildung 2.15.: Diskretes LTI-System. Aus einer Eingangsfolge $x[n]$ wird eine Ausgangsfolge $y[n]$. Ein solches System bezeichnet man als digitales Filter.

[25] Solche System werden als LTI-Systeme bezeichnet, für Linear Time-Invariant. Dazu zählen natürlich auch die zeitkontinuierlichen Systeme, mit denen wir uns hier nicht befassen wollen.

Das Ausgangssignal zu einem beliebigen diskreten Zeitpunkt t_d lässt sich durch rekursive Auswertung von Gleichung (2.22) bestimmen und eignet sich damit ideal für eine Berechnung mit Digitalrechnern.

FIR- und IIR-Filter

Wenn alle Koeffizienten a_i verschwinden, d. h. das Ausgangssignal nicht mehr von früheren Ausgangssignalen, sondern nur noch von den Eingangssignalen abhängt, nennt man solche Strukturen *FIR-Filter*[26], d. h. Filter mit endlicher Impulsantwort. Der Name besagt, dass das Ausgangssignal nach Anregung des Systems mit einem diskreten δ-Impuls unabhängig von den Koeffizienten b_i nach endlicher Zeit verschwinden muss.

Sind nicht alle Koeffizienten a_i Null, kann das System u. U. als Reaktion auf einen δ-Impuls noch nach unendlich langer Zeit ein nicht verschwindenden Ausgang haben, deshalb nennt man solche Systeme *IIR-Filter*[27], d. h. Filter mit unendlicher Impulsantwort.

2.9.2. Alternative Beschreibungsformen

Die Beschreibung eines Systems mit der Gleichung (2.22) eignet sich zwar sehr gut für eine numerische Berechnung und ist deshalb der wichtigste Grundbaustein digitaler Filter, jedoch liegt keine algebraische Beschreibungsform vor, mit der sich eine Systemanalyse oder -synthese durchführen lässt.

Um alternative Darstellungsmöglichkeiten zu finden, ist es deshalb sinnvoll, die Signale auf einen Vektorraum abzubilden, in dem das Verhalten des Systems $\mathcal{S}$ durch einen Homomorphismus $\mathcal{M}_s$ beschrieben werden kann, was im folgenden kurz skizziert werden soll.

Ordnet man jedem Signal $s[k]$ einen Vektor $\vec{s}$ zu, dessen Zeilen durch die Werte $s(k)$ gebildet werden, so wird der zugehörige unendlichdimensionale Vektorraum durch die Einheitsimpulsfolge $\delta[k]$ aufgespannt:

$$s[k] = \sum_{i=0}^{\infty} s[i]\,\delta[n-i] \tag{2.23}$$

Die Folge $\delta[k]$ stellt eine vollständige, orthogonale Basis dar, der zugehörige Vektorraum wird auch *Hilbertraum* genannt.

Das Systemverhalten ist dann durch

$$\vec{y} = \mathcal{M}_s\,\vec{x} \tag{2.24}$$

charakterisiert, wobei $\mathcal{M}_s$ eine unendlichdimensionale, nichtdiagonale Matrix ist.

Alternative Beschreibungsformen für das System $\mathcal{S}$ lassen sich finden, wenn es gelingt, die Basis geeignet zu transformieren. Eine besonders angenehme Basis ist die sogenannte *Eigendarstellung*, in der $\mathcal{M}_s$ diagonal ist, d. h. nur noch Hauptachsenelemente

[26] Finite Impulse Response Filter
[27] Infinite Impulse Response Filter

hat, da dann das Systemverhalten vollständig über die Diagonalelemente (die Eigenwerte) charakterisiert ist. Der Vorgang wird *Diagnonalisierung* genannt und wird in der linearen Algebra durch Berechnung der Eigenvektoren und Eigenwerte von $\mathcal{M}_s$ durchgeführt. Die Eigenvektoren können dann als Basisvektoren benutzt und die Signale nach der neuen Basis entwickelt werden[28].

Diskrete digitale Systeme, die durch (2.22) beschrieben werden können, besitzen Eigensignalfolgen $e[k]$ der Form $e[k] = z^k$ mit $z \in \mathbb{C}$ [52]. Das bedeutet, dass das System $\mathcal{S}$ komplexe Schwingungen $z[k] = r \cdot e^{ik\varphi}$ nur mit einem konstanten komplexen Faktor $\lambda(z)$, dem zu der Eigenfolge gehörenden Eigenwert skaliert (Abb. 2.16).

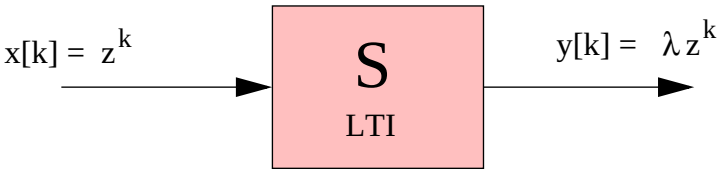


Abbildung 2.16.: Eigenfolge und Eigenwerte eines LTI-Systems. Das System lässt komplexe Schwingungen der Form $z[k] = r \cdot e^{ik\varphi}$ bis auf einen Skalierungsfaktor, den Eigenwert, passieren.

Entwickeln wir nun die Signalvektoren s nach der neuen Basis, so erhalten wir die neuen Signalvektoren S äquivalent zu (2.23):

$$S(z) = \sum_{k=0}^{\infty} s[k]\, z^k \tag{2.25}$$

Dies stellt den kausalen Fall dar (einseitige Folgen). Im allgemeinen Fall mit zweiseitigen Folgen muss die untere Summationsgrenze dabei $-\infty$ sein.

Bezeichnet man die diagonalisierte Matrix $\mathcal{M}_s$ mit $\mathcal{H}_s$, so wird das neue System durch

$$\vec{Y} = \mathcal{H}_s\, \vec{X} \tag{2.26}$$

beschrieben. Da $\mathcal{H}_s$ diagonal ist, kann man auch schreiben

$$Y(z) = \mathcal{H}_s(z)\, X(z) \tag{2.27}$$

Damit ist die Differenzengleichung (2.22) in eine algebraische Form gebracht worden.

Eine solche Transformation wird *z-Transformation* genannt.

Der Satz der Basisfunktionen z^k ist dabei nicht eindeutig, da z nicht weiter eingeschränkt ist. Gebräuchlich ist neben der allgemeinen z-Transformation auch die *diskrete Fouriertransformation*, die die Signale auf eine *Orthonormalbasis* projiziert, d. h. $|z^k| \equiv 1$ oder $z(k) = e^{ik\varphi}$.

Im Folgenden werden die Eigenschaften und Möglichkeiten dieser beiden Transformationen soweit sie für diese Arbeit wichtig sind beschrieben.

[28] Eine theoretisch fundierte Beschreibung der mathematischen Hintergründe (allerdings auf quantenmechanische Systeme bezogen) findet sich z. B. in [90].

2.9.3. Die z-Transformation

Äquivalent zu der Laplace-Transformation für kontinuierliche Differentialgleichungssysteme, kann das Gleichungssystem (2.22) wie oben beschrieben mit der z-Transformation in eine algebraische Form überführt werden, die Ausgangspunkt für weitere Untersuchungen ist. Sie ist auch die Voraussetzung für das Auffinden von Hardwarearchitekturen für die Realisierung von Filterstrukturen entsprechend Gl. (2.22), die nicht der naiven, direkt aus der Gleichung ersichtlichen Struktur entsprechen.

Analog zur Laplace-Transformation gelten auch für die z-Transformation eine Vielzahl von Rechen- und Transformationsregeln, die Hin- und Rücktransformationen stark vereinfachen und die hier nicht weiter vertieft werden sollen, da sie Bestandteil aller Standardwerke zur digitalen Signalverarbeitung sind (siehe z. B. [46]).

Die z-Transformation ordnet der diskreten Signalfolge $x[k]$ eine Funktion $X(z)$ der komplexen Variablen z zu:

$$X(z) \equiv \sum_{k=-\infty}^{\infty} x[k]\, z^{-k} \tag{2.28}$$

Sie konvergiert im Allgemeinen nicht für beliebige z, sondern nur in einem konzentrischen Kreisring der komplexen Zahlenebene, dem *Konvergenzgebiet*. Für rechtsseitige Signale ($x[k] \equiv 0$ für $k < 0$) ist das Konvergenzgebiet u.U. unendlich groß. Die z-Transformierte eines kausalen digitalen Filters nach Gl. (2.22) ergibt sich daher zu

$$\mathcal{H}_s(z) = \frac{\sum_{l=0}^{m} b_l\, z^{-l}}{\sum_{i=0}^{n} a_i\, z^{-i}} \overset{a_0 \equiv 1}{=} \frac{\sum_{l=0}^{m} b_l\, z^{-i}}{1 + \sum_{i=1}^{n} a_i\, z^{-i}} \quad . \tag{2.29}$$

Der Skalierungskoeffizient a_0 wurde dabei ohne Einschränkungen auf eins gesetzt.

Ein Vergleich der z-Transformierten mit der Differenzengleichung im Zeitbereich (2.22) liefert folgende Erkenntnisse:

1. Das Zählerpolynom repräsentiert den nichtrekursiven (FIR-) Anteil des Filters.
2. Das Nennerpolynom repräsentiert den rekursiven (IIR-) Anteil des Filters.
3. Die Faktoren z^{-i} entsprechen im Zeitbereich einer Verzögerung um i Zeitschritte.

2.9.4. Filterarchitekturen

Entscheidend für eine Hardwarerealisierung ist die Anordnung der Addierer und Multiplizierer um eine durch Gleichung (2.22) gegebene Filterstruktur zu realisieren. Eine direkte Implementierung ist in Abb. 2.17 gezeigt, diese Struktur wird *Direktform 1* genannt.

Zieht man in dem Bruch in Gl. (2.29) den rekursiven und nicht-rekursiven Anteil auseinander und führt für die beiden Terme die Abkürzungen $\mathcal{H}_a$ und $\mathcal{H}_b$ ein, sieht

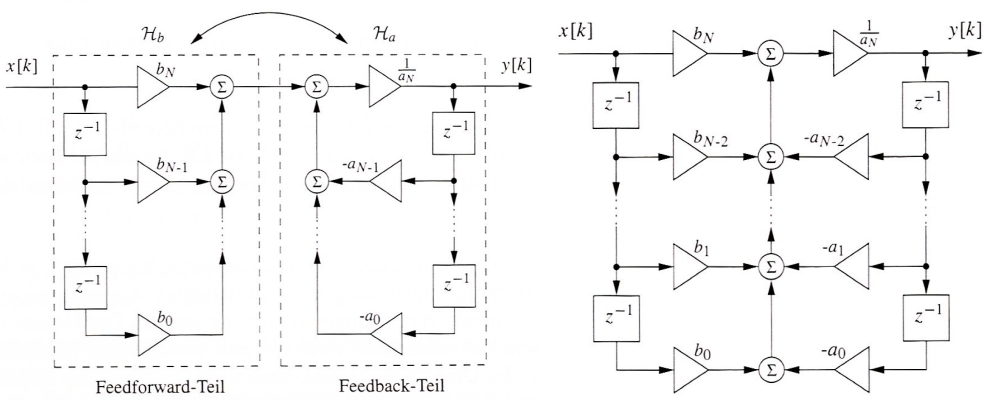


Abbildung 2.17.: Filter in Direktform 1. Diese Architektur wird direkt aus der Differenzengleichung (2.22) abgeleitet. Multiplizierer sind hier durch Dreiecke dargestellt, Addierer durch das Summenzeichen Σ. Die Kästen mit der Bezeichnung z^{-1} verzögern das Signal um einen Zeitschritt und werden als Register oder Speicher realisiert. In dieser Darstellung ist die Reihenfolge der Koeffizienten a_i und b_i gegenüber dem Text umgekehrt.
In der linken Darstellung ist der rekursive und nichtrekursive Anteil separat dargestellt, die Struktur auf der rechten Seite verwendet gemeinsame Addierer (aus [46]).

man, dass es egal ist, ob $\mathcal{H}_a$ oder $\mathcal{H}_b$ zuerst abgearbeitet wird (Kommutativität der Multiplikation):

$$\mathcal{H}_s(z) = \underbrace{\frac{1}{\sum_{i=0}^{n} a_i z^{-i}}}_{=:\mathcal{H}_a(z)} \cdot \underbrace{\frac{\sum_{l=0}^{m} b_l z^{-l}}{1}}_{=:\mathcal{H}_b(z)} = \mathcal{H}_a(z)\,\mathcal{H}_b(z) = \mathcal{H}_b(z)\,\mathcal{H}_a(z) \tag{2.30}$$

Dies ist im linken Teil von Abb. 2.17 durch den bidirektionalen Pfeil angedeutet. Man gelangt damit zur *Direktform 2* (Abb. 2.18), in der die Register z^{-1} von beiden Teilstrukturen genutzt werden, womit bei $m = n = N$ die Hälfte der Register $(N/2)$ eingespart werden können.

Ein Nachteil dieser Struktur sind die direkt verbundenen Addierer, die in jedem Zyklus zwei Additionskaskaden der Länge N durchführen müssen, was bei Filtern hoher Ordnung für einen sehr langen kritischen Pfad sorgt.

Diese Struktur lässt sich modifizieren, indem man im Nenner- und Zählerpolynom sequentiell den Verzögerungsterm z^{-1} ausklammert:

$$\sum_{i=0}^{p} c_i z^{-i} = c_0 + z^{-1}(c_1 + z^{-1}(c_2 + z^{-1}(\cdots z^{-1}(c_{p-1} + z^{-1}(c_p)\cdots)))) \tag{2.31}$$

Arbeitet man den Klammerausdruck von innen nach außen ab, so sieht man, dass in dieser Form jetzt nicht mehr das zeitlich um i Takte verzögerte Originalsignal gespeichert wird, sondern dieses vorher mit dem Koeffizienten c_i multipliziert und auf das

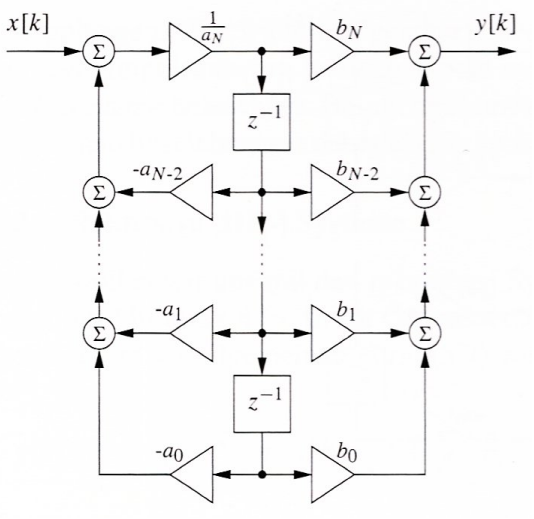


Abbildung 2.18.: Filter in Direktform 2. Diese Form wird aus der Direktform 1 durch triviale Faktorisierung der Systemfunktion $\mathcal{H}_s = \mathcal{H}_1 \cdot \mathcal{H}_2$ und Vertauschung der Reihenfolge gewonnen. Sie zeichnet sich durch die Einsparung der Hälfte der Zustandsregister z^{-1} aus. Der kritische Pfad durch die Addiererkaskade ist gegenüber der Direktform 1 nicht kürzer geworden (aus [46]).

Element der nächsten Kaskadenstufe addiert wird. Diese Form nennt man *Direktform 3* und ist in Abb. 2.19 gezeigt.

Man kann sie auch durch Transposition aus der Direktform 2 gewinnen, weshalb sie auch mit *transponierter Direktform 2* bezeichnet wird. Dazu müssen Ein- und Ausgang, alle Pfeilrichtungen sowie Multiplikationen und Additionen gegeneinander ausgetauscht werden.

Eine weitere Möglichkeit zur Architekturänderung besteht in der Faktorisierung der Systemfunktion $\mathcal{H}_s$ in Filter niedrigerer Ordnung. Das Filter wird dann durch die Hintereinanderausführung kleinerer Filter realisiert. Oft kommen Filter 2. Ordnung zum Einsatz:

$$\mathcal{H}_s(z) = \prod_{q=0}^{N/2-1} \frac{\sum_{l=0}^{2} b_{ql}\, z^{-l}}{\sum_{i=0}^{2} a_{qi}\, z^{-i}} = \mathcal{H}_0(z) \cdot \mathcal{H}_1(z) \cdots \mathcal{H}_{N/2-1}(z) \tag{2.32}$$

Dies ist insbesondere bei grob quantisierten IIR-Filtern von Vorteil, da die Koeffizienten größer werden und der Quantisierungsfehler damit stark abnimmt. Es ist oftmals die einzige Möglichkeit, IIR-Filter in Festkommaarithmetik zu realisieren, da sonst sehr große Wortlängen erforderlich werden.

Weiterhin ist es durch geschickten Entwurf möglich, die Koeffizienten c_{qi} für alle Kaskadenstufen q gleich zu halten, so dass die Filterkaskade durch ein einziges, fest parametrisiertes Filter realisiert werden kann, was zu sehr effizienten Hardwarestrukturen

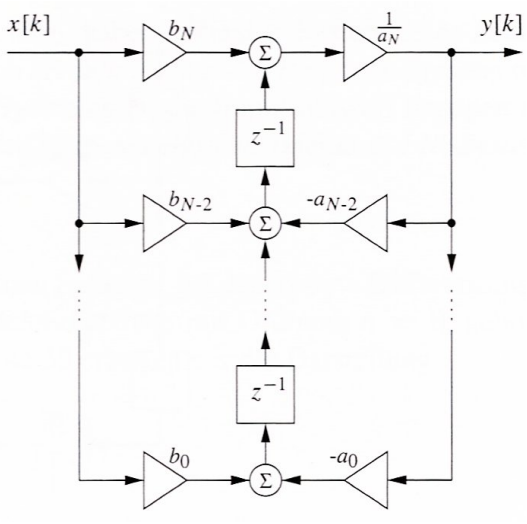


Abbildung 2.19.: Filter in Direktform 3 oder transponierter Direktform 2. Diese Form wird durch sequentielles Ausklammern des Verzögerungsglieds z^{-1} aus der Systemfunktion $\mathcal{H}_s$ gewonnen. Sie hat den gleichen Ressourcenbedarf wie die Direktform 2, jedoch sind die Addierer durch Register getrennt, wodurch der kritische Pfad nur die Länge einer einzigen Stufe hat. Nachteilig ist allerdings die hohe Belastung der Ein- bzw. Ausgänge (Fan-Out) durch die Multiplizierer (aus [46]).

führt[29].

Zu weiteren Architekturen gelangt man durch eine systemtheoretische Betrachtung: Ausgehend von dem Blockschaltbild in Abb. 2.17 betrachten wir den Filter als ein System, dessen innerer Zustand $\vec{z}$ durch die Werte in den Verzögerungsregistern z^{-1} bestimmt wird. Der Folgezustand $\vec{z}[k+1]$ ist eine Linearkombination aus dem jetzigen Zustand $\vec{z}[k]$ (rekursiver Anteil) und dem Eingangsvektor $\vec{x}[k]$ (nichtrekursiver Anteil):

$$\vec{z}[k+1] = \mathbf{A}\vec{z}[k] + \mathbf{B}\vec{x}[k] \tag{2.33}$$

Im z-Raum mit dem Zustandsvektor $\vec{Z}$ lautet die Systemgleichung entsprechend

$$z\vec{Z}(z) = \mathbf{A}\vec{Z}(z) + \mathbf{B}\vec{X}(z) \tag{2.34}$$

Die Gleichung ist identisch mit der Zustandsraumbeschreibung eines zeitkontinuierlichen Systems im Laplace-Raum. Man kann nun durch eine lineare Transformationsmatrix $\mathbf{T}$ den Zustandsvektor $\vec{Z}$ beliebig in einen neuen Zustandsvektor $\vec{Z}' = \mathbf{T}\vec{Z}$

[29] Der in Abschnitt 5.2 beispielhaft diskutierte Bandpassfilter für eine Gammatonfilterbank ist z. B. als vierfach-Kaskade eines fest parametrisierten IIR-Filters erster Ordnung realisiert.

umformen[30]:

$$\begin{aligned} \mathbf{T}z\vec{Z}(z) &= \underbrace{\mathbf{TAT}^{-1}}_{=:\hat{\mathbf{A}}}\underbrace{\mathbf{T}\vec{Z}(z)}_{=:\vec{Z}'(z)} + \underbrace{\mathbf{TB}}_{=:\hat{\mathbf{B}}}\vec{X}(z) \\ &= \hat{\mathbf{A}}\vec{Z}'(z) + \hat{\mathbf{B}}\vec{X}(z) \end{aligned} \tag{2.35}$$

Somit lassen sich eine unbegrenzte Anzahl von Architekturen finden, die sich bezüglich der Anzahl der Ressourcen (Multiplizierer, Addierer und Register) sowie Quantisierungseigenschaften bzw. Rauschverhalten unterscheiden.

2.9.5. Stabilität und Quantisierung

Man bezeichnet ein Filter als stabil, wenn er auf ein beschränktes (absolut summierbares) Eingangssignal mit einem beschränkten Ausgangssignal reagiert[31]. FIR-Filter sind per definitionem (s.o.) beschränkt. In der Transferfunktion (2.29) gibt es keinen rekursiven Anteil.

Bei IIR-Filtern (d. h. es existiert mindestens ein $a_i \neq 0$ für $i > 0$) treten u.U. mehrere Pole auf.

Die *Quantisierung* der Koeffizienten schränkt die Lage der Pole in der z-Ebene jedoch ein. Die Lagen hängen von der Filterarchitektur ab und haben einen erheblichen Einfluss auf das Rauschverhalten.

Als Beispiel soll hier ein IIR-Filter 2. Ordnung mit der Übertragungsfunktion

$$\mathcal{H}_s(z) = \frac{N(z)}{1 + b_1\, z^{-1} + b_2\, z^{-2}} \tag{2.36}$$

dienen. Wir betrachten hier nur den für die Polstellen relevanten rekursiven Anteil, daher steht im Zähler allgemein $N(z)$. Gold und Rader transformieren die Koeffizienten b_1 und b_2 in [56] in eine Winkeldarstellung $(b_1, b_2) \rightarrow (r, \phi)$ gemäß der Vorschrift

$$r = \sqrt{b_2}\,, \quad \cos\phi = \frac{b_1}{2\sqrt{b_2}}\,, \tag{2.37}$$

so dass sich Gl. 2.36 wie folgt schreiben lässt:

$$\mathcal{H}_s(z) = \frac{N(z)}{1 - 2r\cos(\phi)z^{-1} + r^2\, z^{-2}} \tag{2.38}$$

Somit ist r der Radius und ϕ der zugehörige Phasenwinkel des komplexen Polpaares[32]. Da jetzt diese Parameter anstelle von b_1 und b_2 quantisiert werden, erhält man auch eine andere Lage der Polstellen. In Abb. 2.20 sind sie für dieses Beispiel veranschaulicht.

[30] Es wird eine Linksmultiplikation der Gl. 2.34 mit $\mathbf{T}$ durchgeführt sowie die Einheitsmatrix $(\mathbf{T}^{-1})\mathbf{T}$ eingefügt. $\mathbf{T}^{-1}$ muss dazu existieren.

[31] Dies wird auch BIBO-Stabilität (Bounded-Input, Bounded-Output) genannt.

[32] Details finden sich in [56].

Durch die andere Architektur sind allerdings weitere Arithmetik-Einheiten notwendig: Statt zweier Addierer und Multiplizierer in der Direktform benötigt die sog. *Gold-Rader-Architektur* vier Multiplizierer und drei Addierer.

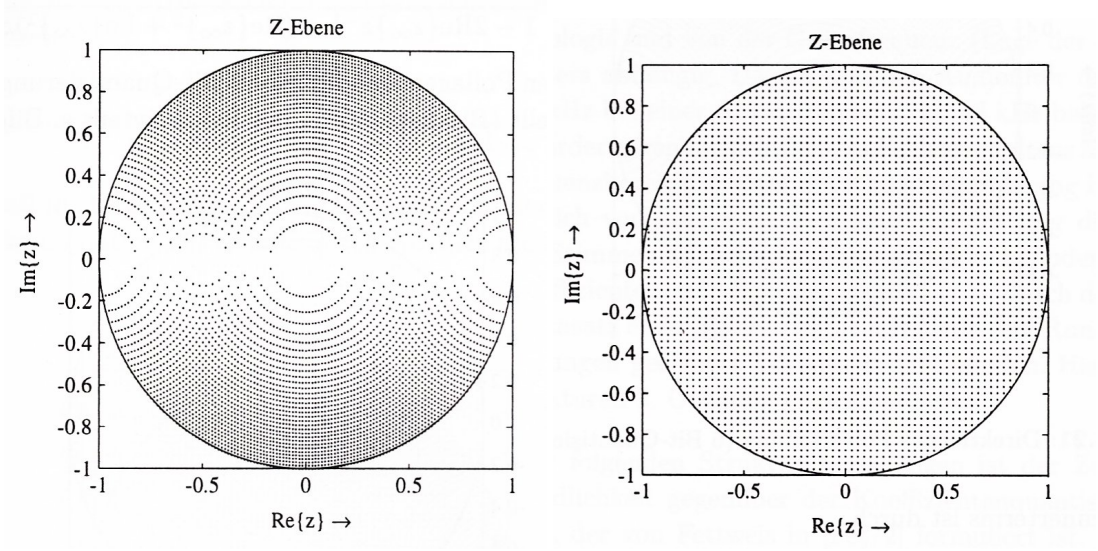


Abbildung 2.20.: Lage der Polstellen für zwei rekursive Filterarchitekturen. Die linke Abbildung zeigt die möglichen Polstellenlagen für die Direktform 1, die rechte Abbildung für die Architektur von Gold und Rader. Diese kodiert die Filterkoeffizienten in Polarkoordinaten, d. h. die möglichen Polstellen sind in der z-Ebene äquidistant. Die Quantisierung der Koeffizienten beträgt in diesem Beispiel 6 Bit, für höhere Quantisierungen liegen die Punkte entsprechend dichter (aus [184]).

Viele andere Architekturen sind denkbar, von denen einige energieverbrauchsrelevante in Abschnitt 4.6 vorgestellt werden.

Ebenfalls problematisch ist die durch die Quantisierung der Arithmetik bedingte Kumulierung von Rundungsfehlern im Rekursionspfad, die einen unquantisiert stabilen Filter destabilisieren können. Weitere Überlegungen und Untersuchungen zu Filterarchitekturen im Low-Power-Entwurf finden sich im Abschnitt 4.6.

2.9.6. Diskrete Fouriertransformation

Wie oben bereits angedeutet, stellt die diskrete Fouriertransformation (DFT) lediglich einen Spezialfall der z-Transformation dar. Sie ist eine Entwicklung des Eingangssignals nach Schwingungen in der komplexen Zahlenebene mit konstantem Betrag[33]. Aus $|z^k| = 1$ folgt, dass nur noch der Phasenwinkel φ von z frei ist: $z^k(\varphi) = e^{ik\varphi}$. Setzen wir $z = z(\varphi)$ in die z-Transformation (2.28) ein, so erhalten wir die *Fouriertransformierte* einer

[33] Die Schwingungen werden in der Literatur oft „ungedämpft" genannt, da der Dämpfungskoeffizient einer Schwingung auf dem Einheitskreis der komplexen Zahlenebene identisch eins ist. Ein Dämpfungskoeffizient > 1, der eine aufklingende Exponentialfolge bewirkt, gilt somit auch als „gedämpfte" Schwingung.

diskreten Folge $x[k]$ zu

$$X(\varphi) = \sum_{k=-\infty}^{\infty} x[k]\, e^{-ik\varphi} \tag{2.39}$$

Der Signalvektor $\vec{x}$ wird nach der *Orthonormalbasis* $e^{ik\varphi}$ entwickelt. Das Signal wird somit auf Schwingungen der Frequenz φ projiziert.

Da wir stets endliche und kausale Folgen der Länge N betrachten, stellt sich die Frage, welche Frequenzanteile in dem Signal vorkommen können. Die höchste Frequenz ist durch die Anzahl der Abtastpunkte, d. h. die Dimension von $\vec{x}$ gegeben, die wir zu N annehmen[34]: dim $\vec{x} = N$. Die Entwicklungskoeffizienten sind periodisch mit 2π[35], somit ist die höchste in einer Folge der Länge N auftretende Frequenz $\varphi_{\max} = 2\pi/N$. Da die Punkte k auf der diskreten Zeitachse äquidistant sind, können nur ganzzahlige Vielfache von $\varphi_{\max}$ in der Folge enthalten sein, somit nur die diskreten Frequenzen $\varphi[n] = n\,\varphi_{\max}$.

Eine kausale diskrete Folge der Länge N lässt sich somit durch die Projektion auf N verschiedene Frequenzen darstellen. Eine Verschiebung der Schwingungen gegenüber dem Ursprung (*Phase*) wird dabei durch die Projektion auf ein Kosinus-Sinus-Paar berücksichtigt, d. h. jeder Frequenzanteil ist eine Linearkombination aus dem Kosinus- und dem um 90 Grad verschobenen Sinuswert der Frequenz: $e^{i\varphi} = i\,\sin\varphi + \cos\varphi$.

Gleichung (2.39) vereinfacht sich damit zu

$$X[f] = \sum_{k=0}^{N-1} x[k]\, e^{-i\,\frac{2\pi fk}{N}} \tag{2.40}$$

Definiert man den sogenannten „Twiddle"-Faktor ω_N als höchsten Frequenzanteil, der bei einer Folge der Länge N vorkommen kann zu

$$\omega_N \equiv e^{-\frac{2\pi i}{N}} \tag{2.41}$$

so kann man (2.40) vereinfacht schreiben als

$$X[f] = \sum_{k=0}^{N-1} x[k]\, \omega_N^{fk} \tag{2.42}$$

Es ist nicht direkt ersichtlich, dass die aus der Analysis abgeleiteten Eigenschaften der Fouriertransformation auch für die diskrete Variante gelten. Es kann aber gezeigt werden, dass dies für die entscheidenden Sätze zutrifft, siehe [19].

Die Umkehrtransformation zu (2.40), die die Rückgewinnung eines Zeitsignals aus einem Frequenzspektrum ermöglicht, ist wie folgt definiert:

$$x[k] = \frac{1}{N} \sum_{f=0}^{N-1} X[f]\, \omega_N^{-fk} \tag{2.43}$$

[34] Siehe auch das hier nicht behandelte Abtasttheorem [135].

[35] Dies liegt an der Periodizität der komplexen e-Funktion: $e^{-ik\varphi} = e^{-ik(\varphi+2\pi)}$.

Bis auf die Skalierung und das Vorzeichen des Exponenten des Twiddle-Faktors hat sie die gleiche Struktur wie die Hintransformation.

Schnelle Fouriertransformation (FFT)

Wegen ihrer enormen technischen und wirtschaftlichen Bedeutung gehört die diskrete Fouriertransformation zu den am besten untersuchten Problemen überhaupt und ist immer noch Gegenstand der Forschung.

Die Motivation, schnelle Algorithmen zu entwickeln, liegt in der Tatsache, dass die DFT eine zu N^2 proportionale Anzahl von Operationen benötigen, der Aufwand für große Eingangslängen also stark ansteigt.

Historisch ist der Cooley-Tukey-Algorithmus [29] aus dem Jahr 1965 mit einer Zeitkomplexität von $N \log_2 N$ der bedeutendste Entwicklungschritt. Nach vielen Weiter- und Neuentwicklungen reduzierte die 1976 von Winograd [175] vorgestellte FFT[36] die theoretische Anzahl der Multiplikationen nochmals deutlich.

Trotz seiner theoretischen Überlegenheit wird der Winograd-Algorithmus heute sehr selten verwendet. Dies liegt hauptsächlich an seiner sehr komplexen Struktur, die zu leistungsmäßig enttäuschenden Implementierungen geführt hat [103]. Er kann Eingangsvektoren der Länge 2^n nicht direkt verarbeiten, da er auf einer Primzahlzerlegung basiert, die als kleinstes Element ein Produkt von zwei unterschiedlichen Primzahlen hat. Außerdem eignet er sich im Gegensatz zu den auf der Cooley-Tukey-Zerlegung basierenden Algorithmen nicht für eine „in-place"-Berechnung, d. h. er benötigt zusätzlichen Speicher zum Ablegen von Zwischenergebnissen. Dies lässt ihn insbesondere für eine Hardwareimplementierung ungeeignet erscheinen.

Glücklicherweise gelang 1984 mit der sogenannten Split-Radix-FFT ein neuer Durchbruch bei den auf der Cooley-Tukey-Zerlegung basierenden Algorithmen. Sie erreicht für Eingangsvektoren der Länge 2^n die bisher beste Berechnungseffizienz bei gleichzeitig relativ einfacher Struktur [40, 39, 38].

Aus Platzgründen soll hier nur auf die gebräuchlichsten Algorithmen eingegangen werden, die auch die Basis der Implementierung der DFT als Modul darstellen (siehe Abschnitt 4.6.3).

Die Cooley-Tukey-Zerlegung

Bei dieser Zerlegung wird der Eingangsvektor rekursiv in kleinere Vektoren zerlegt. Die *Radix-n*-Zerlegung teilt den Vektor jeweils in n Vektoren auf, so dass Eingangsvektoren der Länge 2^n möglich sind. Mit steigendem n steigt das Optimierungspotential, jedoch werden die Eingangsvektorgrößen immer grobkörniger, so dass meist eine Radix-2- oder Radix-4-Zerlegung zum Einsatz kommt. Die zugehörigen Grundstrukturen sind in Abb. 2.21 und Abb. 2.22 gezeigt. Die Split-Radix-Zerlegung basiert auf eine Kombination beider Elemente.

Das grundsätzliche Prinzip der *Divide and Conquer*-Strategie sei anhand der Radix-2-Zerlegung erläutert. In diesem Beispiel erfolgt die Zerlegung in geradzahlig und ungeradzahlig indizierte Elemente des Eingangsvektors:

[36] Sie wird in der Literatur oft mit „WFTA" = Winograd Fourier Transform Algorithm abgekürzt.

$$
\begin{aligned}
F_k &= \sum_{n=0}^{N-1} f_n\, \omega_N^{nk} \\
&= \sum_{n=0}^{\frac{N}{2}-1} f_{2n}\, \omega_N^{2nk} + \sum_{n=0}^{\frac{N}{2}-1} f_{2n+1}\, \omega_N^{(2n+1)k} \\
&= \underbrace{\sum_{n=0}^{\frac{N}{2}-1} f_{2n}\, \omega_{\frac{N}{2}}^{nk}}_{=:{}^eF_k} + \omega_N^k \underbrace{\sum_{n=0}^{\frac{N}{2}-1} f_{2n+1}\, \omega_{\frac{N}{2}}^{nk}}_{=:{}^oF_k} \\
&= {}^eF_k + \omega_N^k\, {}^oF_k
\end{aligned}
\tag{2.44}
$$

Es entstehen zwei DFTs, wobei die erste über die geradzahlig indizierten Elemente (${}^eF_k$) und die zweite über die ungeradzahlig indizierten Elemente (${}^oF_k$) geht. Die zweite DFT unterscheidet sich von der ersten durch einen zusätzlichen Twiddle-Faktor, der die Verschiebung der ungeradzahligen Elemente gegenüber den geradzahligen Elementen im Ortsraum reflektiert (Verschiebungssatz der Fouriertransformation[37]). Der Zerlegungsschritt (2.44) wird jetzt rekursiv durchgeführt, bis nur noch Vektoren der Länge eins übrig sind; das sind $\log_2 N$ Zerlegungen. Die DFT von Vektoren der Länge eins ist die Identität (2.40) und somit trivial ausführbar.

Eine weitere Vereinfachung ergibt die paarweise Betrachtung der Twiddle-Faktoren mit Indexabstand $N/2$. Mit $N' := \frac{N}{2}$ und $k' := k + N' \quad (k < N')$ ergibt sich:

$$
\omega_{N'}^{nk'} = \omega_{N'}^{n(k+N')} = \omega_{N'}^{nk}\, \omega_{N'}^{nN'} = \omega_{N'}^{nk} \underbrace{e^{2\pi i n}}_{=1} = \omega_{N'}^{nk} \tag{2.45}
$$

Das bedeutet, dass die Twiddlefaktoren für eine DFT für die erste und zweite Hälfte identisch sind.

$$
\omega_N^{k'} = \omega_N^{k+N'} = \omega_N^k\, \omega_N^{N'} = \omega_N^k \underbrace{e^{\pi i}}_{=-1} = -\omega_N^k \tag{2.46}
$$

Diese Gleichung sagt aus, dass der bei der Zerlegung auftretende Phasenfaktor sich für die erste und zweite Hälfte nur durch das Vorzeichen unterscheidet.

Zusammengefasst gilt also

[37] Der Verschiebungssatz der Fouriertransformation besagt, dass eine zeitliche Verschiebung im Ortsraum, im Frequenzraum einem Phasenfaktor entspricht: $\{f_{i+c}\} \circ\!\!-\!\!\bullet \{\omega^{ck} F_k\}$

$$\left.\begin{aligned} F_k &= {}^eF_k + \omega_N^k \, {}^oF_k \\ F_{k+N'} &= {}^eF_k - \omega_N^k \, {}^oF_k \end{aligned}\right\} \quad 0 \leqslant k < N' \tag{2.47}$$

Beim Zusammensetzen nach Gleichung (2.47) ergeben sich dann in jeder Rekursionsstufe die gleichen Berechnungsmuster: Das komplexwertige Produkt des Twiddle-Faktors mit dem ungeradzahlig indizierten Teil ($\omega_N^k \, {}^oF_k$) wird einmal zum geradzahlig indizierten Teil addiert, einmal subtrahiert. Dieses Gebilde wird aufgrund der Form des Signalflussgraphen auch Butterfly genannt (Abb. 2.21).

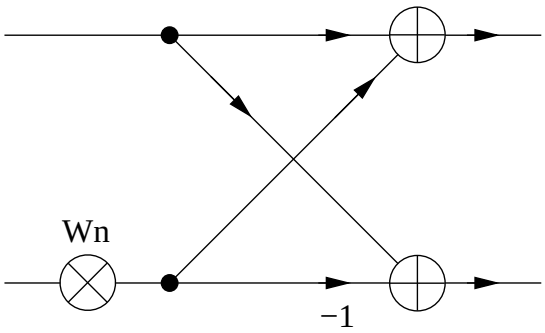


Abbildung 2.21.: Butterfly. Gezeigt ist das Basisberechnungselement der Radix-2-Zerlegung für die sog. Decimation-in-time (DIT)-Zerlegung. Er beinhaltet eine Multiplikation mit dem Twiddle-Faktor und zwei Additionen (aus [78]).

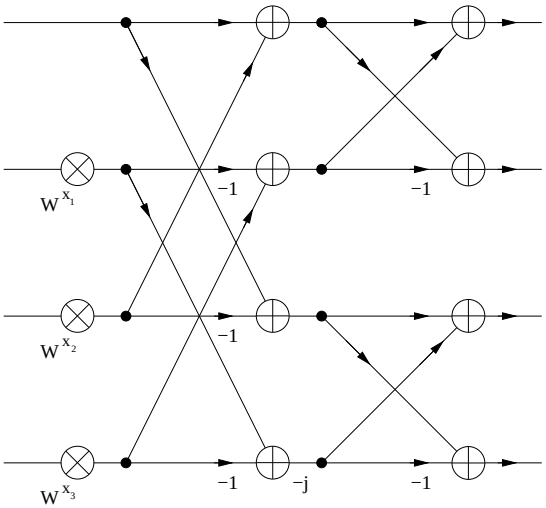


Abbildung 2.22.: Dragonfly. Gezeigt ist das Basisberechnungselement der Radix-4-Decimation-in-time (DIT)-Zerlegung. Er beinhaltet drei Multiplikation mit den Twiddle-Faktoren und acht Additionen (aus [78]).

Eine FFT beliebiger Länge kann durch Verschaltung dieser identischen Basiseinheiten aufgebaut werden. Einzelheiten dazu finden sich in Abschnitt 4.6.

3. Low-Power Entwicklungssysteme: Stand der Forschung

Im folgenden werden einige Systeme vorgestellt, die zumindest in Teilaspekten einen ähnlichen Ansatz haben wie das Low-Power Framework. Nach einem kurzen Überblick über die Entstehung und Historie des jeweiligen Werkzeugs wird der zugeordnete Design-Flow beschrieben. Darauf folgt eine Bewertung. Am Ende des Kapitels steht ein Resümee, welche die Haupteigenschaften der betrachteten Werkzeuge zusammenfasst.

3.1. Überblick

Aus der Vielfalt der Low-Power Literatur wurden hier Ansätze ausgewählt, die entweder

1. eine Methodik beschreiben oder implementieren, die der Entwicklerin eine ebenenübergreifende Betrachtung und Optimierung des System ermöglichen oder
2. von einer Systembeschreibung auf algorithmischer Ebene ausgehen und damit die Optimierungsmöglichkeiten gegenüber dem klassischen Entwurfsansatz auf RT-Ebene erheblich erweitern (siehe 4).

Zu ersteren gehört **POSE** (Abschnitt 3.2), das zwar auf einer RT-Eingabe aufsetzt, bei den Optimierungen jedoch tiefer liegende Ebenen berücksichtigt.

Das folgende System, **HyperLP** (Abschnitt 3.3), erfasst auch die algorithmische Ebene in seiner Methodik und zählt demnach zu (2.) in obiger Kategorisierung. Der automatisierte Design-Flow setzt jedoch erst auf RT-Ebene an.

Aus HyperLP ist das **SSHAFT**-Projekt (Abschnitt 3.4) entstanden, das die Automatisierung bis in den algorithmischen Bereich fortsetzt, dort jedoch gewissen Einschränkungen unterliegt[1].

In dieser Reihe folgt der **Xilinx System Generator for DSP** (Abschnitt 3.5), der einen Teil der SSHAFT-Methodik in ein kommerzielles Produkt umsetzt.

Die darauf folgenden Werkzeuge (ab Abschnitt 3.6) gehen von einer allgemeinen algorithmischen Darstellung ohne prinzipielle Einschränkungen des resultierenden Kontroll-Datenfluss-Graphen aus. **ForSyDe** betrachtet den zu implementierenden Algorithmus sehr formal und dient im wesentlichen der wissenschaftlichen Untersuchung eines Verfahrens zur Optimierung der Überführung einer Algorithmusbeschreibung in die RT-Form.

[1] Das Werkzeug beherrscht keine Kontrollpfad-Synthese und -Optimierungen, was die automatisch explorierbaren Architekturvarianten stark einschränkt.

ExTra (Abschnitt 3.7) versucht die aus dem Compilerbau bekannten Optimierungen auf die Hardwareentwicklung zu übertragen.

ORINOCO schließlich (Abschnitt 3.8) stellt einen vergleichbaren Ansatz dar, implementiert jedoch zusätzlich eine Binding-Optimierung und verfügt über einen ausgefeilteren Power-Schätzer.

3.2. POSE

POSE (*Power Optimization and Synthesis Environment*) [72] gehört zu den frühen Ansätzen, einzelne Low-Power Optimierungen in einem Framework mit einer Energiebewertung zusammenzufassen. Durch die implizite Methodik des Werkzeugs wird der Hardware-Entwicklerin ein Power-optimierender Hardware-Entwurf erleichtert.

Obwohl POSE auf RT-Ebene ansetzt und damit höhere Ebenen von Optimierungen ausgeschlossen sind, ermöglicht das System doch einen ebenenübergreifenden Entwurf, da als Maßzahl der Optimierung der Energiebedarf auf Gatterebene (also nach dem „Compile"-Vorgang) herangezogen wird. Die Ergebnisse des System sind daher im allgemeinen wesentlich besser, als durch Einzelmaßnahmen bewirkte Einsparungen.

3.2.1. Prinzip

POSE ermöglicht eine Power-optimierte Synthese von CMOS-Logik, d. h. Maßzahl der Optimierung ist das Produkt aus Schaltaktivität, Taktfrequenz und Umladeenergie eines CMOS-Transistorpaares (siehe Abschnitt 2.5). Dabei stehen verschiedene Lastmodelle zur Verfügung. Statische Verlustleistung wird nicht berücksichtigt.

Die Eingangsspezifikation für den Optimierungsprozess von POSE ist ein Boolesches Netzwerk (siehe Abb. 3.1). Für dieses Netzwerk wird zu jedem Knoten die Schaltaktivität berechnet, wobei als Basis der von der Designerin gelieferte Stimulus dient[2].

Die eigentliche Optimierung erfolgt nun durch Umstrukturieren des Booleschen Netzwerks. Dazu kommen algebraische Verfahren zum Einsatz, wie sie auch für die Flächenoptimierung eingesetzt werden, nur dass das Optimierungskriterium nicht Fläche, sondern Leistung ist. Zu diesen Verfahren gehören die Extraktion gemeinsamer Unterfunktionen und das Aufbrechen von Funktionen und Funktions-Faktorisierung. Diese Verfahren haben inzwischen Einzug in EDA-Werkzeuge erhalten. Die Optimierungsroutine versucht dabei, die Ausgangslast für Knoten mit hoher Schaltaktivität durch das Hinzufügen von Knoten mit geringerer Ausgangsaktivität zu vermindern. Das bedeutet im wesentlichen, dass die Fläche zu Gunsten der Leistung erhöht wird.

3.2.2. Bewertung

POSE stellt den ersten ebenenübergreifenden Ansatz in der Low-Power Schaltungsoptimierung dar. Es berücksichtigt zwar nur die RT- und Gatterebene, das System zeigt

[2] Um den Berechnungsaufwand gering zu halten, kommt ein besonders effizientes Näherungsverfahren zum Einsatz (Semi-lokale BDDs, siehe [72]). Dies ist nötig, weil die Repräsentation der Schaltung auf der Abstraktionsebene der Booleschen Netzwerke Graphen mit sehr vielen Knoten ergibt.

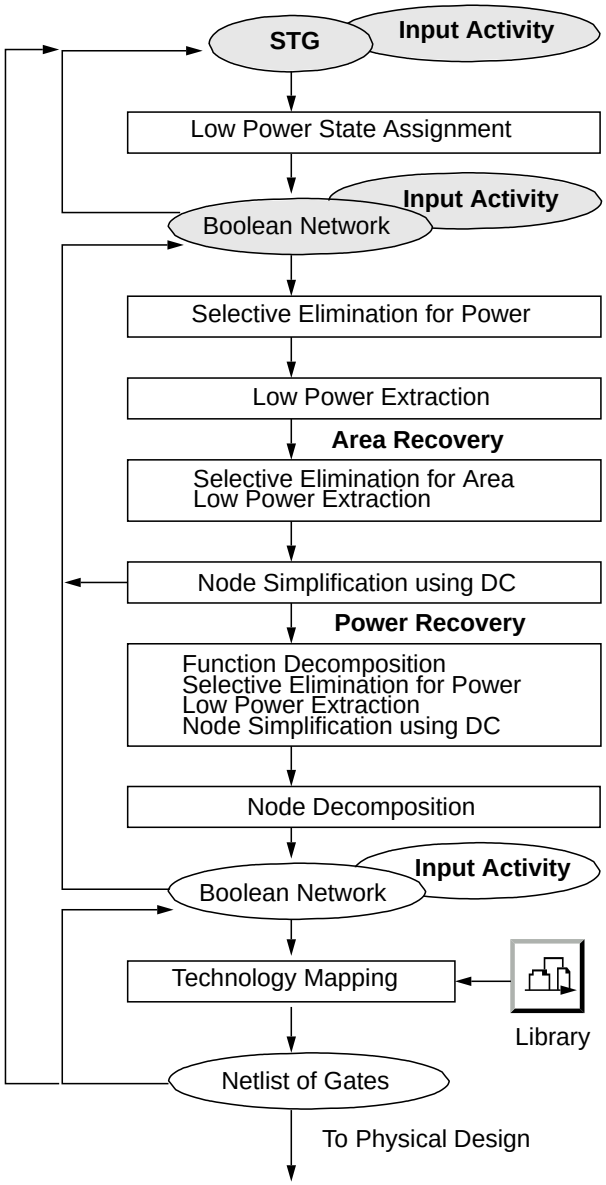


Abbildung 3.1.: Design-Flow bei POSE. Die ovalen Elemente stellen Benutzerdaten dar, die rechteckigen von POSE ausgeführte Schritte.
Die Entwicklerin gibt einen Zustandsübergangsgraphen (engl. *state transition graph*) und repräsentative Eingangsdaten vor. Der Graph wird in eine FSM überführt, wobei hier eine aus der Literatur bekannte, besonders energiesparende Kodierungsvariante gewählt wird. Das Ergebnis ist ein Boolesches Netzwerk, d. h. ein gerichteter Graph, dessen Knoten entweder Boolesche Funktionen oder Speicherelemente sind. Dann beginnt ein iterativer Optimierungsprozess, an dessen Ende die Abbildung der Netzliste auf eine Technologie-Bibliothek stattfindet. Diese Abbildung kann ebenfalls iterativ optimiert werden (z. B. durch Einfügen von Komplexgattern oder u.U. Vertauschen von Gatterpins). Das anschließende Platzieren und Verdrahten wird durch ein externes Werkzeug erledigt (aus [72]).

aber, dass selbst bei nur zwei betrachteten Ebenen die Koppelung an eine durch eine Framework implementierte Entwurfsmethodik den Entwurf durch eine Formalisierung erheblich vereinfacht und effektiver macht.

Die in POSE implementierten Optimierungen sind inzwischen zur Gänze in kommerzielle EDA-Werkzeuge eingeflossen.

Größter Nachteil des Systems ist die Einschränkung der Eingangsspezifikation auf RT-Ebene.

3.3. HyperLP

Das erste Framework, das zur energieoptimierten Schaltungssynthese entwickelt wurde, stellt Hyper-LP dar [26], dessen Nützlichkeit aus allgemeinen Überlegungen zur Entwicklung der Komplexität und des Energieverbrauchs vorhergesagt wurde [25]. Es wurde wie sein Nachfolger SSHAFT (siehe 3.4) an der Universität Berkeley unter der Leitung von Robert W. Brodersen entwickelt. Die Autoren sprechen zwar von einem High-Level System, die automatischen Funktionen für Hyper-LP setzen jedoch auf der RT-Ebene an. Das Framework umfasst aber auch Entwicklungs-Richtlinien für die Hardware-Designerin, die höhere Abstraktionsebenen umfassen.

3.3.1. Prinzip

Hyper-LP ist ein Framework zur Power-Optimierung von synchroner CMOS Schaltungstechnik, das eine Reihe von Transformationen zur Power-Reduktion auf RT-Ebene mit einer RT-Power-Schätzung zusammenfasst. Auf diese Weise war es möglich, die Auswirkungen der durchgeführten Transformationen ohne zeitaufwendige Synthese und Zieltechnologie-Mapping abzuschätzen. Die Transformationen lassen sich in folgende Kategorien einteilen:

1. Reduktion des kritischen Pfads.

   Hier kommen die Techniken Pipelining und das Ausrollen von RT-Schleifen zum Einsatz, d. h. eine Parallelisierung, die bei gleichem Datendurchsatz eine Reduktion der Taktfrequenz ermöglichen. Weiterhin werden Operatorvarianten mit kurzem kritischen Pfad bei der Synthese bevorzugt, z. B. Addierer mit CSA-Struktur gegenüber Addierern mit Ripple-Struktur oder der Einsatz von Shift-Addern bei Konstantenmultiplikation.

2. Reduktion der Anzahl der Operationen.

   Dieser Punkt umfasst Optimierungen des Kontroll-Datenfluss-Graphs durch Eliminierung gemeinsamer Unterfunktionen und Zusammenfassung von Schleifen.

3. Reduktion der Interconnect-Kapazität.

   Die Verdrahtung wird in diesem System indirekt beeinflusst: Die Autoren verstehen unter diesem Punkt eine effiziente Ausnutzung der vorhandenen Hardware-Module, d. h. ein möglichst gutes Scheduling, um ohne Durchsatzreduktion oder Takterhöhung Ressourcen sparen zu können.

Die unter Punkt 1 genannten Transformationen werden zum Teil durch eine Modul-Bibliothek implementiert , der andere Teil umfasst Designrichtlinien für die Entwicklerin [24].

Neben den Richtlinien und der Modul-Bibliothek ist ein integriertes Power Schätzwerkzeug Bestandteil von Hyper-LP. Hier wird eine relativ grobe Abschätzung auf RT-Ebene durchgeführt, die auf Kapazitätsmodellen für Operatoren (Addierer und Multiplizierer), Register, FSM und Verbindungsleitungen basiert. Die Gesamtkapazität wird additiv aus den Einzelkapazitäten berechnet:

$$C_{tot} = C_{op} + C_{reg} + C_{fsm} + C_{interconnect} \tag{3.1}$$

Der Optimierungsparameter ist die daraus resultierende Leitung mit

$$P = C_{tot} \cdot V_{dd}^2 \cdot f. \tag{3.2}$$

3.3.2. Bewertung

Hyper-LP ist das erste Framework für die Low-Power Schaltungssynthese. Die Autoren haben schon 1990 die Notwendigkeit eines solchen Werkzeugs erkannt und ein System erdacht, das als Grundlage für alle zukünftigen Systeme angesehen werden kann. Die damals implementierten Transformationen sind heute Bestandteil kommerzieller Synthesewerkzeuge, ebenso wie eine integrierte Power-Schätzung (siehe Abschnitt 4.2.2). Ebenfalls zur Sprache kommt bei diesem System die Notwendigkeit einer speziellen Entwurfsmethodik, die die Autoren in dem SSHAFT-System (siehe 3.4) weiterentwickelt haben.

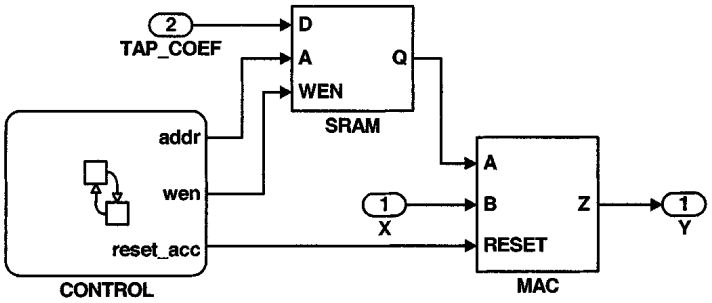
3.4. SSHAFT

Das SSHAFT-System [36, 35] ist am Berkeley Wireless Research Center in der Zeit von 1998 bis 2002 entstanden. Es ist ein High-Level Entwurfssystem, das Kommunikations-ASICs als Zieltechnologie avisiert. Die Entwicklung des Systems fußt auf der Beobachtung, dass eine maximale Energieeinsparung nur mit spezifischer Schaltungstechnik möglich ist, der Weg zu einem „optimalen IC“ im Gegensatz zu einer Softwareimplementierung aber sehr steinig ist.

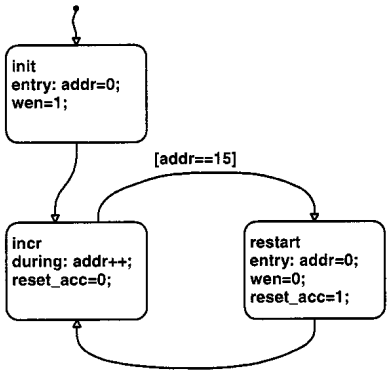
3.4.1. Prinzip

Um eine maximale Exploration von Design-immanenter Parallelität zu ermöglichen, ist die Systemspezifikation bei SSHAFT ein Datenflussgraph und keine prozedurale Beschreibung. Als Eingabesprache wird hier Simulink verwendet[3]. Da Kontrollfluss in Simulink nur schlecht modelliert werden kann, wurde hier der Weg über eine Skript-Beschreibung gegangen. Die Kontrollflussbeschreibung kann sowohl für die Simulink-Simulation als auch für eine Hardware-Synthese verwendet werden, das es sich um die Beschreibung eines einfachen Zustandsautomaten handelt (siehe Abb. 3.2).

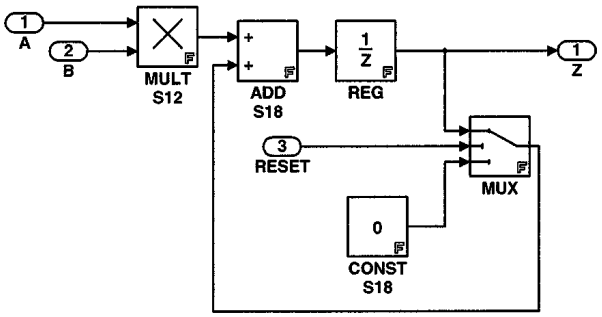
[3] siehe auch Abschnitt 3.5.



(a) SSHAFT Datenflussgraph



(b) SSHAFT Detail Kontrollfluss



(c) SSHAFT Detail MAC

Abbildung 3.2.: SSHAFT Entwurfsüberblick. Die Systemspezifikation erfolgt als Datenflussgraph in Simulink, in (a) ist ein gemultiplextes FIR-Filter dargestellt: Ein Eingangsdatenstrom wird in den Eingang B der MAC-Einheit eingespeist, Koeffizienten kommen über den Eingang A aus einem SRAM. Gesteuert wird das System von einem separaten Block, der in (b) im Detail dargestellt ist. Dieser enthält eine Beschreibung des Kontrollflusses auf RT-Ebene (typischerweise eine FSM). Der MAC-Block in (a) beinhaltet wiederum eine Substruktur, die in (c) gezeigt wird. Wird die Hierarchie von (a) aufgelöst, bleiben nur noch Blöcke übrig, für die eine synthetisierbare RT-Beschreibung existiert (Abbildungen aus [36]).

Jeder Simulink-Block kann weitere Blöcke beinhalten, so dass ein hierarchischer Systemaufbau möglich ist. Die Primitivblöcke, d. h. diejenigen Blöcke, die keine weiteren Unterblöcke haben, müssen in einer simulations- und synthesefähigen Beschreibung vorliegen, so dass das gesamte System mit herkömmlichen RT-Synthesewerkzeugen weiterverarbeitet werden kann. Dies sind im allgemeinen eine Matlab-Beschreibung für die Simulation und VHDL- oder Verilog-Code für die Hardwaresynthese.

3.4.2. Bewertung

Das System ähnelt verblüffend dem Xilinx System Generator (Abschnitt 3.5). Es ermöglicht den graphischen Systementwurf mit vordefinierten Verarbeitungsblöcken und implementiert im Gegensatz zum System Generator einen ASIC-Flow. Es ist jedoch kein echtes High-Level Entwurfswerkzeug, da sich hinter jedem Block eine RT-Beschreibung verbirgt.

Ein weiterer Nachteil ist, dass für jeden Block sowohl eine Simulationsbeschreibung, als auch eine synthesefähige Beschreibung vorliegen muss, was mit einem hohen Verifikationsaufwand verbunden ist und eine Simulation auf Gatterebene erfordert.

3.5. Xilinx System Generator for DSP

Bei dem „System Generator for DSP“ [181] handelt es sich nicht um ein spezielles Low-Power Entwicklungssystem. Durch seine guten Explorationseigenschaften und die hohe Abstraktionsebene eignet es sich jedoch gut für einen Low-Power Designflow. Zudem ist es das industriell bedeutsamste High-Level Entwurfssystem, da sich Simulink als abstrakte Systembeschreibung für die digitale Signalverarbeitung etabliert hat.

Das System beschränkt sich jedoch auf FPGAs als Zielsysteme und auf ein Datenfluss-dominiertes System-Design. Es ist daher gut für DSP-Anwendungen geeignet, worauf auch der Name hinweist. Die Idee des Systems wird in [61] beschrieben.

3.5.1. Prinzip

Die System-Spezifikation erfolgt mit dem Werkzeug Simulink der Firma Mathworks [124, 95]. In Abb. 3.3 ist das Beispiel eines einfachen FIR-Filters zu sehen. Der System-Generator ist als Toolbox für Simulink ausgeführt, die ein umfangreiches Sortiment an Funktionsblöcken für Simulink zur Verfügung stellt.

Jeder dieser Funktionsblöcke ist auf HDL-Ebene mit einem IP-Modul von Xilinx identisch, wie es durch das Werkzeug „Coregen“ zur Verfügung gestellt wird. Die Simulink-Toolbox stellt für jedes IP-Modul ein bit- und taktgenaues Simulationsmodell zur Verfügung, ähnlich wie das Entwicklungsframework Simulationsmodelle bereitstellt (siehe Kap. 4).

Die Xilinx-Blöcke sind durch Ein- und Ausgangsschnittstellen von den anderen Simulink-Blöcken getrennt (in der Abbildung gelb markiert). Diese Schnittstellen führen

die Konvertierung von dem Matlab Fließkomma-Datentyp in einen speziellen Xilinx Festkommadatentyp durch. Dieser Festkommadatentyp ist nicht mit dem Simulink Festkommatyp kompatibel, wie er durch die Festkomma-Toolbox bereitgestellt wird. Nur die blau dargestellten Xilinx-Blöcke sind synthesefähig.

Aus dieser Tatsache folgt die wichtigste Einschränkung des Entwurfs: Hardware kann nur aus den blauen Xilinx-Blöcken generiert werden, andere Simulink-Blöcke müssen immer über Schnittstellenblöcke angebunden werden. Ein einfacher Kontrollfluss ist jedoch möglich: Über eine spezielle Funktion, die der Spezifikation eines endlichen Automaten auf RT-Ebene entsprechen, können Sprungbedingungen in den synthesefähigen Abschnitt eingeführt werden. Für komplexere Steuerungsaufgaben ist es möglich, einen RISC-Kern als synthesefähigen Block einzubinden.

3.5.2. Bewertung

Der „System Generator for DSP“ ist ein kommerziell verfügbares High-Level Hardwaresynthese Werkzeug[4]. Der wesentliche Vorteil für einen Low-Power Entwurfsfluss ist die Spezifikation und Simulation auf System-Ebene, wodurch sich prinzipiell ein hohes Einsparpotenzial ergibt. Es eignet sich beispielsweise hervorragend zur Exploration und Optimierung der Berechnungsgenauigkeit bei Festkommaarithmetik, da alle Ergebnisse und Zwischenergebnisse[5] mit Simulink-Werkzeugen visualisiert und interpretiert werden können.

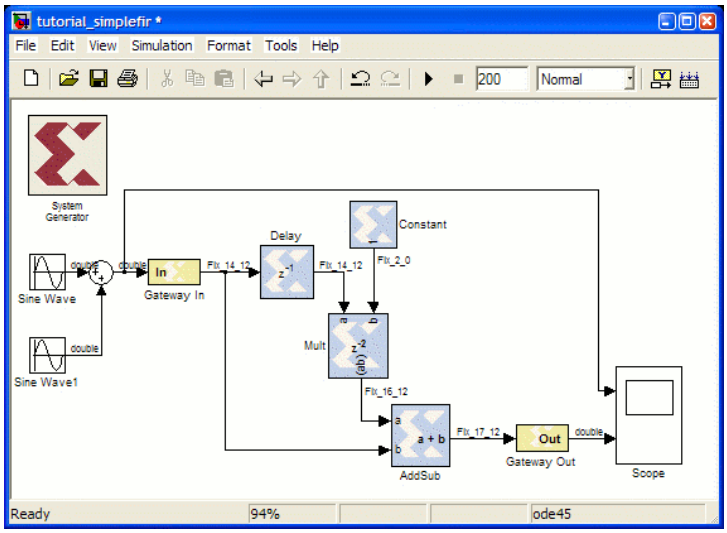
Die Hauptnachteile sind:

1. Beschränkung auf FPGAs als Zielhardware.
2. Der Kontrollfluss bleibt auf RT-Ebene beschränkt, komplexere Kontrollstrukturen sind über einen eingebetteten CPU-Kern möglich aber vom Energieverbrauch nicht akzeptabel.
3. Nur Festkommaarithmetik möglich.
4. Keine Unterstützung für Entwurfsstrategien mit variabler Wiederverwendung von Komponenten.
5. Keine direkte Energiebewertung in Simulink möglich, da die einzelnen Xilinx-Blöcke nicht Power-charakterisiert sind. Die Energiebewertung ist erst auf der HDL-Ebene[6] möglich.

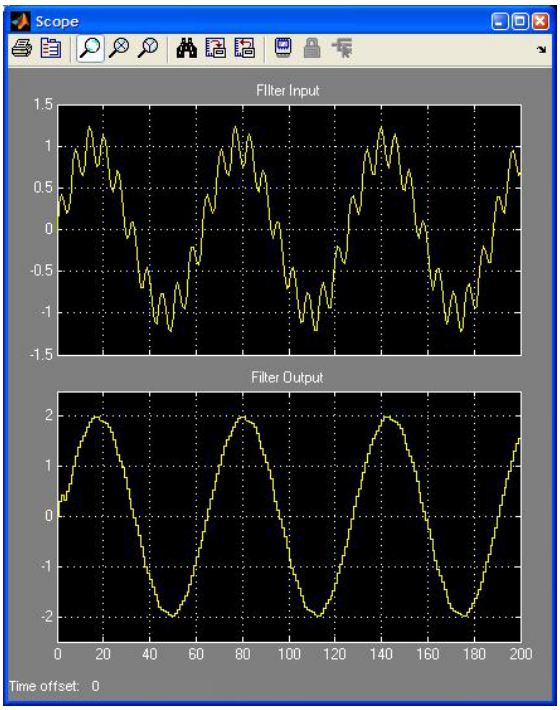
[4] Der „System Generator for DSP“ war das erste kommerzielle Werkzeug dieser Art. Inzwischen (Stand 9/2008) gibt es auch von dem Xilinx-Konkurrenten Altera ein vergleichbares Werkzeug mit dem Namen „DSP Builder“ [5].

[5] Es können nur Zwischenergebnisse auf Blockebene dargestellt werden, da der interne Aufbau der Blöcke nicht zugänglich ist.

[6] Mit dem Werkzeug XPower sind Powerschätzungen in unterschiedlichen Genauigkeitsabstufungen für unplatzierte, platzierte sowie platzierte und verdrahtete FPGA-Design möglich. Dabei kann der Energieverbrauch sowohl auf Netz-Ebene als auch für den gesamten Chip bestimmt werden.



(a) Simulink Flussgraph



(b) Simulink Scope

Abbildung 3.3.: System Generator for DSP. Auf dem linken Screenshot ist der Aufbau eines FIR-Filters aus Simulink-Blöcken zu sehen. Hierbei kommen die in der Xilinx-Bibliothek bereitgestellten synthesefähigen Blöcke zum Einsatz (blau), die das Verhalten der FPGA-Implementierung bit-genau simulieren. Zur Stimulusaufschaltung und Analyse können Standard-Simulink Werkzeuge benutzt werden (weiß). Die Verbindung stellen spezielle Schnittstellenblöcke her (gelb). Das rechte Bild ist die Darstellung des Simulink-Oszilloskops im linken Bild. Oben ist der Filtereingang zu sehen, unten das gefilterte Ausgangssignal (aus [181]).

3.6. ForSyDe

ForSyDe (*Formal System Design*) ist das Ergebnis eines 6-jährigen Forschungsprojekts, das bis Anfang 2006 am KTH[7] in Stockholm durchgeführt wurde. Im Vordergrund stand die Entwicklung einer formalen Entwicklungsmethodik für die Spezifikation von eingebetteten System und SOCs. Dabei spielte die Energieoptimierung eigentlich keine Rolle. Da die im folgenden vorgestellt Methodik jedoch prinzipiell eine weitreichende architektonische Exploration des Entwurfsraums mit verschiedenen Optimierungskriterien zulässt, finden sich einige Aspekte auch in der dem Low-Power-Framework (siehe Kap. 4) zugrundeliegenden formalen Methodik wieder.

3.6.1. Prinzip

Ausgangspunkt für die Entwicklung von ForSyDe war die durch ein effektives Design komplexer Systeme bedingte Forderung nach einer Entwurfsspezifikation auf einer hohen Abstraktionsebene ([130, 129]). ForSyDe hat dabei zwei zentrale Aspekte:

1. Die Konzentration auf ein *mathematisches Modell* als zentralen Aspekt einer funktionalen Systemspezifikation und
2. die Überführung der funktionalen Spezifikation in eine konkrete Implementierung mit einer speziellen Methodik, die durch ein (noch zu erstellendes) EDA-Werkzeug umgesetzt wird (siehe Abb. 3.4).

Das System hat als Grundlage einen *synchronen Ansatz*, wie er z. B. von Berry beschrieben wird [17], der ihn in dem grafischen Programmiersystem *Esterel* umgesetzt hat[8]. Die grundlegende Annahme ist hier, dass Eingänge und Ausgänge des Systems in dem Sinne synchronisiert sind, dass die Ausgänge auf die Eingänge ohne messbare Verzögerung reagieren, wodurch sich die Kommunikation verschiedener Systembausteine stark vereinfacht. Weiterhin lassen sie solche *reaktiven Systeme* mathematisch vollständig beschreiben, was die Entwicklung von automatischen Verifikationsmechanismen ermöglicht. Die Annahme perfekter Synchronität ist für genügend schnelle, getaktete Systeme richtig, solange alle Operationen innerhalb eines einzigen Taktzyklus' ablaufen.

In ForSyDe werden Systeme in der *funktionalen Programmiersprache* Haskell beschrieben, die sich im Gegensatz zu der Mehrzahl der funktionalen Sprachen auch gut für die Beschreibung von kontrollflussdominierten Systemen eignet. Die Beschreibung des Systems erfolgt dabei als hierarchische Gruppierung von einzelnen synchronen Prozessen, die gleichzeitig ablaufen.

Diese Beschreibung des Systems kann auf hoher Abstraktionsebene simuliert werden. Um das System auf realen Hardware zu übertragen, können die einzelnen synchronen Prozesse auf zwei Arten über eine Transformationsbibliothek verarbeitet werden:

[7] Kungliga Tekniska högskolan (Royal Institute of Technology)

[8] Es gibt noch zahlreiche weitere synchrone Programmiersprachen, von den *Signal* und *Lustre* die bekanntesten sind. Einen kompakten Überblick bietet [60].

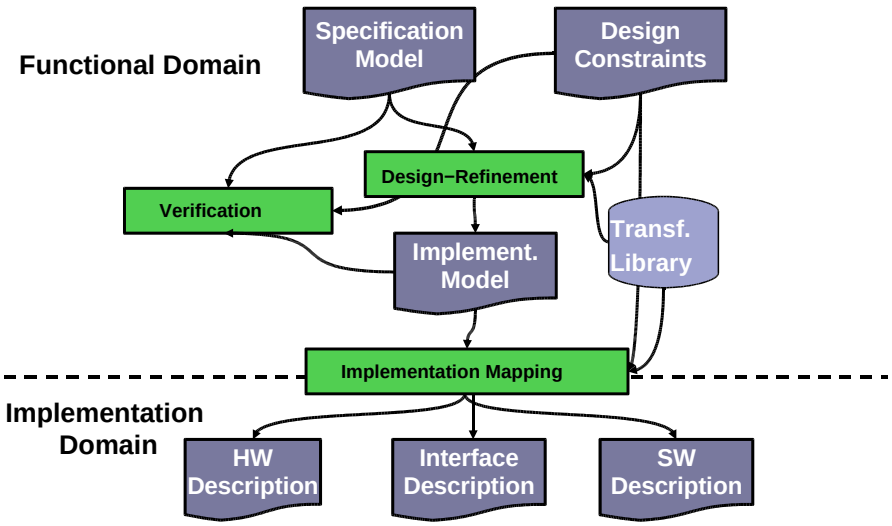


Abbildung 3.4.: ForSyDe Design-Flow (aus [123]).

1. Ein oder mehrere Prozesse (ein *Prozessnetzwerk*) können in ein anderes Prozessnetzwerk überführt werden, das das gleiche Ein-/Ausgangsverhalten aufweist (dieselbe *charakteristische Funktion* hat).
2. Ein einzelner Prozess kann auf eine konkrete Implementierung (z. B. eine synthesefähige Hardwarebeschreibung) abgebildet werden.

Da es nicht für jeden Prozess eine gültige Implementierung gibt, muss der erste Punkt solange iteriert werden, bis ein gültiges Mapping für das Gesamtsystem gefunden werden kann. Dabei wird sowohl 1. als auch 2. über Optimierungsparameter gesteuert, so dass z. B. auch auf Energiebedarf optimiert werden kann, wenn die Prozessnetzwerke entsprechend charakterisierbar sind.

Die eindeutige mathematische Beschreibung des Systems macht die Simulation auf der Implementierungsebene (für Hardware also z. B. die Gatterebene) bei einer korrekt implementierten Abbildung auf Prozesse und Architekturen theoretisch überflüssig.

3.6.2. Bewertung

Mit dem vorgestellten System ist durch die Verwendung einer funktionalen synchronen Programmiersprache eine mathematisch eindeutige abstrakte Systembeschreibung möglich, was auf der einen Seite eine formale Verifikation, d. h. den Vergleich der Implementierung mit der Spezifikation, möglich macht, auf der anderen Seite ein hohes Optimierungspotential birgt, da außer der Synchronität keine Annahmen über die Implementierung in der Systembeschreibung stecken.

Der umrissene Formalismus ermöglicht die Abbildung auf eine Beschreibung, die für die automatische Hardwaresynthese geeignet ist.

Ein kritischer Punkt von ForSyDe ist, dass für gute Ergebnisse eine hinreichend große Anzahl von Transformationen in das System eingebaut werden muss. Durch den allgemeinen Sprachansatz ist nur eine Teilmenge der beschreibbaren Systeme mit einer

endlichen Anzahl von Transformationen auf konkrete Hardware abbildbar. Im gegenwärtigen Zustand kann ForSyDe nur sehr kleine Systeme implementieren[9]. Weiterhin ist der Energiebedarf als Optimierungskriterium in das System bisher nicht integriert worden.

Eine große Einschränkung des Systems ist das Problem der Abbildung synchroner Blöcke auf reale Hardwarestrukturen, die eine endliche Laufzeit aufweisen; d. h. die Annahme perfekter Synchronität ist nur unter gewissen Rahmenbedingungen erfüllt, die von der Zieltechnologie abhängen[10]. Eine effiziente Exploration z. B. von parallelen Architekturen ist deshalb nur unter genauer Kenntnis des zieltechnologieabhängigen Timings möglich.

3.7. ExTra

Das ExTra-System (*Design Space Exploration Using Transformations*) ist über den Zeitraum von 1996 bis 2002 im Rahmen des DFG-Schwerpunktprogramms „Rapid Prototyping Integrierter Steuerungssysteme mit harten Zeitbedingungen“ an der Universität Tübingen unter der Leitung von J. Gerlach und W. Rosenstiel entstanden [51]. Es zielt auf eine durch einen High-Level-Ansatz optimierte Schaltungssynthese. Neben den klassischen Optimierungskriterien Fläche und Geschwindigkeit kann auch der Energieverbrauch in die Optimierung einbezogen werden.

3.7.1. Prinzip

ExTra ist ein Werkzeug zur automatischen Optimierung einer algorithmischen Beschreibung. Es besteht im wesentlichen aus

1. einem Transformations-Kontroll-Algorithmus,
2. einer Transformations-Bibliothek und
3. einem High-Level Gütemaß, dass die Qualität der durchgeführten Transformation berechnen kann, ohne die Design-Hierarchie nach unten zu konkretisieren

Das Werkzeug verspricht also neben einer Optimierung auf hoher Abstraktionsebene auch einen erheblichen Zeitgewinn gegenüber Werkzeugen, deren Gütemaß auf tieferen Abstraktionsschichten aufsetzt (siehe Abb. 3.5).

Die Eingabe für ExTra ist eine algorithmische Beschreibung in der Programmiersprache *C*. Für die Optimierung kommen aus dem Compilerbau bekannte Datenfluss- und Kontrollfluss-Transformationen zum Einsatz, dazu zählen z. B.

- Optimierung von Feldzugriffen
- Skalarisierung und Kontraktion von Feldern
- Propagierung von Konstanten

[9] Ziel des Projekts ist, die prinzipielle Machbarkeit eines solchen Systems zu zeigen.

[10] Die Ausgänge eines synchronen Blocks müssen innerhalb eines Taktzyklus' stabil sein. Signallaufzeiten sind u. a. zieltechnologieabhängig.

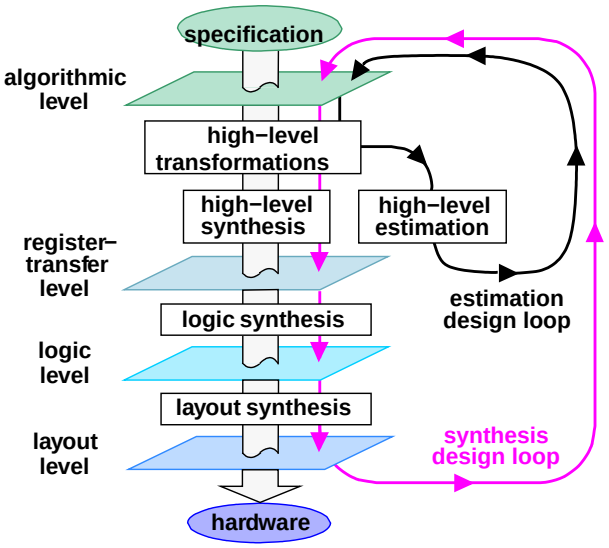


Abbildung 3.5.: Design-Flow bei ExTra. Das Werkzeug implementiert die im oberen Teil der Abbildung gezeigte Optimierungsschleife. Es startet mit einer Algorithmus-Beschreibung, führt dann eine Reihe von Transformationen aus und bewertet diese. Diese Schritte werden iterativ ausgeführt, bis das Ergebnis des Gütemaßes konvergiert (aus [51]).

für den Datenfluss und

- Partitionierung/Verschmelzen/Verschieben von Grundblöcken
- Entrollen/Partitionierung von Schleifen
- Pipelining von Schleifen

für den Kontrollfluss. Insgesamt beinhaltet ExTra 34 Transformationen, eine ausführliche Auflistung und Erklärung findet sich in [50].

Da der Suchraum trotz Beschränkung auf die algorithmische Ebene sehr groß ist, kann nur ein Teil der Transformationen durchgeführt werden, dazu kommen Varianten der klassischen Optimierungsstrategien *Hill-Climbing* und *Simulated Annealing* zum Einsatz. Der Beitrag von ExTra besteht in der Bereitstellung einer auf eine anschließende Hardware-Synthese angepassten Transformationssteuerung und -bewertung.

Die Transformationssteuerung besteht aus einer variablen Kostenfunktion, die die Gesamtkosten des transformierten Algorithmus' bezüglich Fläche, Geschwindigkeit und Energie berechnet. Die Kosten werden dabei wie folgt ermittelt:

1. Scheduling und Ressourcen-Allokation: Der Algorithmus wird in eine prozedurale Beschreibung überführt und alle Operationen werden einer begrenzten Anzahl von Operatoren zugeordnet (siehe Abschnitt 4.5.1). Hierfür wird ein gewöhnlicher C-Compiler verwendet.
2. Verwendung einer Bibliothek mit vorcharakterisierten RT-Komponenten: In einer Tabelle werden für die verschiedenen Operatoren (z. B. Additionen und Multiplikationen) Kostenwerte hinterlegt, die auf RT-Ebene exemplarisch ermittelt wurden.

Weiterhin existiert eine Heuristik zur Abschätzung der Verdrahtungskosten (weitere Details zur Kostenfunktion finden sich in [49]).

3.7.2. Bewertung

ExTra stellt einen interessanten Ansatz zur effizienten Optimierung auf algorithmischer Ebene dar. Im Gegensatz zu ORINOCO (siehe Abschnitt 3.8) macht es sich die aus dem Compilerbau bekannten Optimierungstransformationen zu nutze und passt sie über eine entsprechende Kostenfunktion an Schaltungstechnik als Zielarchitektur an.

Größter Schwachpunkt des Systems ist eine ungenügende Anbindung an Synthesewerkzeuge, was bedeutet, dass nach Abschluss der Optimierungen eine händische Umsetzung in eine Hardwarebeschreibungssprache erfolgen muss. Erst dann kann eine implementierungsnahe Abschätzung des Energieverbrauchs erfolgen.

3.8. ORINOCO

Das Werkzeug ORINOCO (OFFIS Research INstitute pOwer Characterizer, estimator and Optimizer) ist seit 1997 an der Universität Oldenburg in der Forschungsgruppe EHS unter Leitung von W. Nebel entstanden und wird immer noch weiterentwickelt. Es dient der Verlustleistungsanalyse und -optimierung auf algorithmischer Ebene.

3.8.1. Prinzip

Die Eingangsbeschreibung für ORINOCO ist eine algorithmische Beschreibung in VHDL oder C++ (siehe Abb. 3.6). Im Laufe der Entwicklung wurde VHDL durch SystemC ersetzt[11]. Das Werkzeug führt hiermit ein Scheduling, eine Ressourcen-Allokation und ein Binding durch, d. h. jeder Operation wird ein Operator zugewiesen[12]. Für die Leistungsschätzung sind die Operatoren (u.U. in verschiedenen Architekturen) vorcharakterisiert [76]. Weiterhin erfolgt eine Ermittlung der statistischen Eingangsaktivität aller Operatoren durch Simulation, so dass die Verlustleistungsschätzung die Datenstatistik berücksichtigt[13]. Außerdem existieren in ORINOCO Speichermodelle, Interconnect-Modelle und FSM-Modelle.

Die Verlustleistungsoptimierung besteht in der Auswahl eines verlustleistungsarmen Bindings (siehe Abb. 3.7). Das Werkzeug ermittelt dazu die Verlustleistung für verschiedene Allokations- und Binding-Varianten des Algorithmus'. Da der Entwurfsraum sehr groß ist, kommt hier ein stochastisches Verfahren zum Einsatz, das ein obere und untere Schranke für die Verlustleistung jeder einzelnen Komponente berechnet [85]. Die

[11] VHDL wird als Eingangssprache seit 2003 nicht mehr unterstützt. Dies hatte keine technischen Gründe, sondern lag an den hohen Lizenzgebühren für den von Synopsys vermarkteten VHDL- und Verilog-Compiler LEDA, der als Frontend für den VHDL-Code eingesetzt wurde. Stattdessen wurde dann SystemC unterstützt.

[12] Die Begriffe werden in Abschnitt 4.5.1 genauer erklärt

[13] Die Ermittlung der Gesamtverlustleistung erfolgt mittels vorcharakerisierter Makros. Details zur Makro-Charaktersierung können [31] entnommen werden.

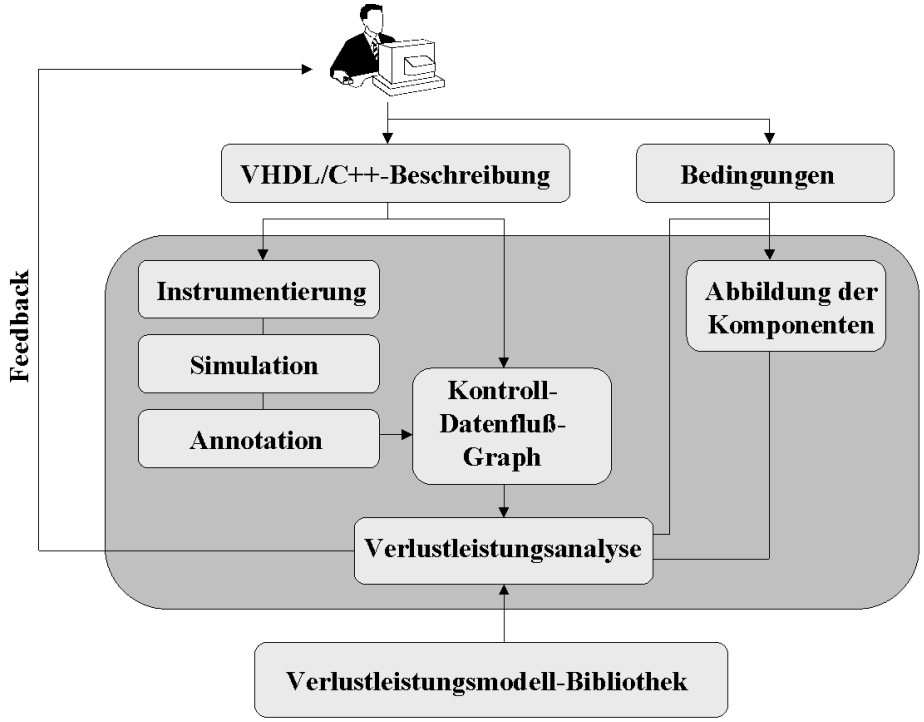


Abbildung 3.6.: ORINOCO Design-Flow. (aus [144]).

gefundenen Grenzen werden dann in einem zweiten Schritt verwendet, um eine synthesefähige Gesamtlösung möglichst nach am Optimum zu finden.

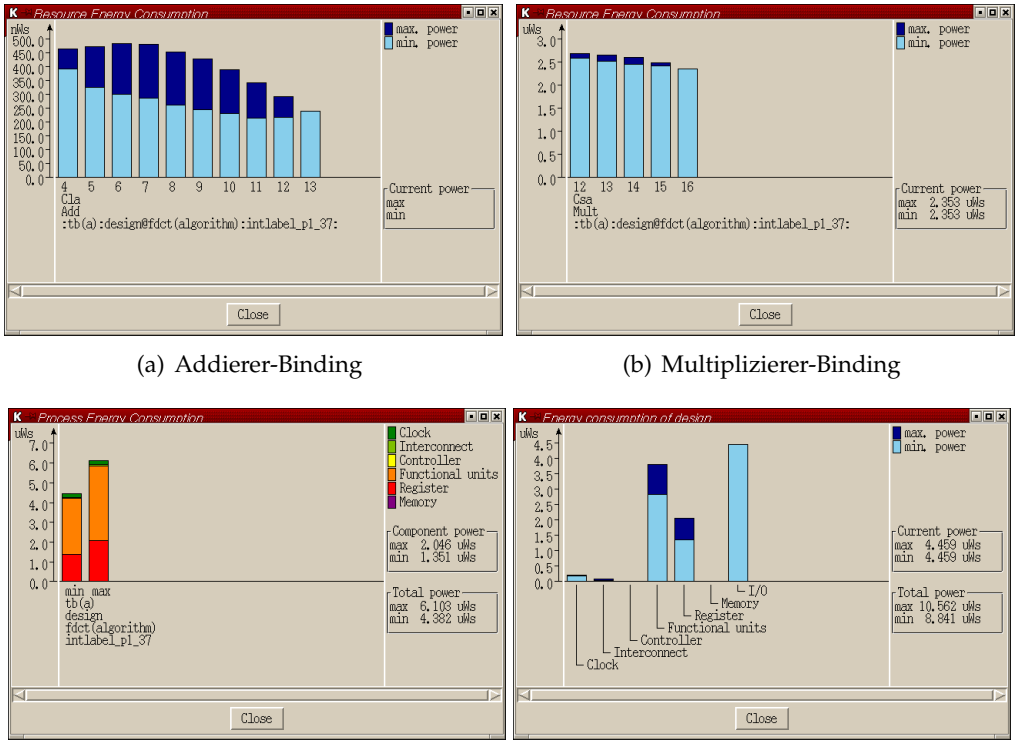
PowerOpt

Im April 2008 wurde das Analysewerkzeug ORINOCO um ein Synthesewerkzeug namens „PowerOpt“ ergänzt [7]. Das Werkzeug ORINOCO ist nicht mehr separat erhältlich. PowerOpt kann eine SystemC- oder ANSI-C-Spezifikation auf Verhaltensebene in Verilog-RTL-Code überführen. Es ersetzt damit den Behavioral Compiler von Synopsys, der vormals für die Synthese erforderlich war. Der generierte RT-Code kann z. B. mit dem DesignCompiler von Synopsys weiterverarbeitet werden.

PowerOpt konnte aufgrund seines späten Erscheinens im Rahmen dieser Arbeit leider nicht mehr eingehend untersucht und bewertet werden.

3.8.2. Bewertung

Mit dem Werkzeug ORINOCO ist ausgehend von einer algorithmischen Beschreibung eine Energiebewertung möglich, ohne eine vollständige Synthese durchzuführen. Das ermöglicht es der Entwicklerin, in relativ kurzer Zeit verschiedene Algorithmen zu vergleichen. Weiterhin ist eine Binding-Optimierung möglich. In der Praxis ist eine Ressourcen-Wiederverwendung jedoch nur dann günstig in Bezug auf die Verlustleis-



(a) Addierer-Binding (b) Multiplizierer-Binding

(c) Prozess-Energie (d) Gesamt-Energie

Abbildung 3.7.: ORINOCO iDCT Beispiel. Gezeigt ist eine beispielhafte Optimierung für eine inverse DCT. In dem Algorithmus kommen 13 Additionen und 16 Multiplikationen vor. Je nach Ressourcen kann für jede Operation ein eigener Operator vorgesehen werden (in (a) und (b) ganz rechts) oder die Operationen können auf weniger Einheiten aufgeteilt werden (Allokationsschritt). Idealerweise gibt es ein Energieminimum bei einer Variante mit Ressourcen-Wiederverwertung; in (a) ist das bei 11 Addierern der Fall, in (b) bringt Ressourcen-Wiederverwendung in jedem Fall eine Verschlechterung des Energieverhaltens. In (c) ist der kumulierte Energiebedarf des jeweils schlechtesten und besten Bindings aller funktionalen Einheiten gezeigt, in (d) werden die Funktionsblöcke separat dargestellt. Die Screenshots wurden im Rahmen dieser Arbeit mit einer Test-Lizenz von ORINOCO für Linux angefertigt.

tung, wenn die Komponenten nicht zu 100% der Zeit benutzt werden, ansonsten ist eine vollständig parallele Variante meist besser (siehe Kap. 5).

Erfolgt die Spezifikation in Behavioral-VHDL (siehe Abschn. 4.5.1), kann ORINOCO eine Spezifikations- (Constraint-) Datei für den Synopsys Behavioral Compiler erzeugen, das den Compiler zur Umsetzung des von ORINOCO ermittelten Bindings zwingt. Für diesen Weg existiert demnach ein vollständig automatisierter Entwurfsfluss bis hinab zur Gatter-Ebene.

Mittels PowerOpt ist ein vergleichbarer Entwurfsfluss auch für ANSI-C-und SystemC-Spezifikationen verfügbar, der hier evaluierte VHDL-Entwurfsfluß ist seit 2003 nicht mehr verfügbar (s.o.).

3.9. Resümee

Vom Abstraktionsniveau der Eingabebeschreibung fallen die vorgestellten Systeme grob in zwei Klassen: POSE, HyperLP, SSHAFT und der Xilinx System Generator erfordern eine Beschreibung, in der das Taktverhalten bereits feststeht. Es handelt sich somit nicht um echte High-Level Entwurfssysteme, auch wenn HyperLP zumindest eine Methodik vorschlägt, die über RT-Niveau hinausgeht. SSHAFT und der System Generator bieten durch ihre graphische Eingabe einen Zusatznutzen, da sich hier relativ leicht verschiedene Architekturen realisieren lassen. Allerdings ist zur Architekturexploration Handarbeit erforderlich, die Fehleranfälligkeit ist durch die graphische Eingabe gegenüber einer textuellen Eingabe aber reduziert. Diese Kategorie von Werkzeugen hat mit dem Xilinx System Generator inzwischen kommerzielle Reife erlangt.

Auf der anderen Seite stehen die Systeme ForSyDe, ExTra und ORINOCO, die eine ungetaktete Eingangsbeschreibung verarbeiten. ForSyDe gehört dabei eigentlich auch zur ersteren Kategorie, da sich durch die synchrone Programmiersprache und die Aufteilung in synchrone Blöcke eine implizite Aufteilung in Taktzyklen ergibt. Diese kann jedoch innerhalb des Systems automatisch durch Neugruppierung variiert werden.

ExTra und ORINOCO dagegen bilden die ungetaktete Eingangsbeschreibung durch ein explizites Scheduling (siehe Abschnitt 4.5.1) und eine Ressourcenallokation auf eine RT-Beschreibung ab, es kommen aber vollkommen unterschiedliche Optimierungsstrategien zum Einsatz. Nur ORINOCO verfügt über eine Anbindung an ein High-Level-Synthesetool. Von kommerzieller Reife sind alle Werkzeuge in dieser Klasse noch weit entfernt[14]. Das liegt zum einen an den vielen Freiheitsgraden, die eine ungetaktete Beschreibung für die Hardwareumsetzung bietet. Zum anderen liegt ein entscheidendes Problem darin, unter den gültigen, d. h. theoretisch auf Hardware realisierbaren Umsetzungen eine „optimale“ bezüglich der verwendeten Zieltechnologie zu finden und diese dann auch zu synthetisieren.

ExTra verfügt noch über keine automatische Syntheselösung, ORINOCO hat für die Eingabe in Behavioral VHDL den Synopsys Behavioral Compiler zur Synthese vorgesehen, inzwischen steht mit PowerOpt eine eigene Behavioral-zu-RT-Synthese zur

[14]Das seit April 2008 verfügbare PowerOpt wird als Paket mit ORINOCO von der Firma ChipVision vertrieben [7]. Es hat von den hier vorgestellten Systemdesign-Werkzeugen den höchsten Reifegrad, konnte aber leider nicht mehr im Detail untersucht werden.

Verfügung. Das Problem der Zieltechnologieabhängigkeit versuchen beide Werkzeuge über den Weg einer Vorcharakterisierung von Modulen bezüglich der Optimierungsparameter zu lösen. Bei den Modulen handelt es sich um arithmetische Einheiten wie Addierer und Multiplizierer.

Zusammenfassend lässt sich sagen, dass für die erste Kategorie schon gute Lösungen bezüglich Energieoptimierung und Entwurfsmethodik existieren, auf die bei Bedarf zugegriffen werden kann (siehe auch die Beschreibung des PowerCompilers von Synopsys, ab Abschnitt 4.2.2). Da die Schaltungstechnik jedoch immer komplexer wird und Zieltechnologien innerhalb eines Produktlebenszyklus' immer weiter optimiert werden, erscheint der Ansatz, die Optimierung auf einer getakteten Beschreibung zu beginnen nicht mehr zukunftssicher. Der Schritt zu einer ungetakteten Beschreibung ist jedoch so groß, dass noch keine anerkannte Strategie für eine korrekte und optimierte Umsetzung einer solchen Beschreibung auf Hardware existiert, was sich auch am Entwicklungsstand und der Inhomogenität der oben beschriebenen Ansätze von ForSyDe, ExTra und ORINOCO erkennen lässt.

Das mit dem im folgenden Kapitel beschriebenen System verfolgte Ziel ist deshalb, durch die Einschränkung des Entwurfsraums und die Verwendung möglichst vieler bereits bestehender kommerzieller oder nichtkommerzieller Werkzeuge ein funktionsfähiges Entwurfssystem zu erstellen, das die Möglichkeiten eines ungetakteten Entwurfs nutzt, ohne auf bereits etablierte Optimierungsverfahren für getaktete Entwürfe zu verzichten.

4. Das Entwicklungsframework

Die Nichtverfügbarkeit von Hardwarebeschreibungssprachen bzw. Synthesewerkzeugen, die einen konsistenten Entwurf auf Systemebene ermöglichen (s. Abschnitt 2.4) legt die Idee nahe, durch eine Kombination von Software- und Hardwareentwicklungstools einen Ersatz zu finden. Dieser ist zwar funktional eingeschränkt, ermöglicht jedoch für spezifische Entwurfs- und Optimierungskriterien eine Hardwaresynthese aus einer abstrakteren Beschreibung als der Register-Transfer-Ebene. Die im vorigen Kapitel 3 beschriebenen Ansätze stellen z. T. solche Systeme dar, jedoch ist bei keinem dieser Systeme ein vollständiger Entwurfsfluss von der Verhaltensebene bis zur Gatterebene implementiert. Weiterhin beschreiben die Systeme nicht klar, ob und wie externe Werkzeuge zur Schaltungssynthese oder -optimierung einbindbar sind.

In diesem Kapitel wird beschrieben, wie durch die Kombination bestehender kommerzieller Werkzeuge mit einer selbst entworfenen und implementierten, auf Software-Prototypen basierenden Modulbibliothek und die Zusammenfassung in ein Software-Rahmenwerk (Framework) ein effizientes, praxistaugliches, erweiterbares und zieltechnologieunabhängiges Entwurfssystem für verlustleistungsoptimierte integrierte Schaltkreise entworfen werden kann. Das Framework sorgt dabei für eine konsistente und für den Benutzer transparente Aneinanderkoppelung der verwendeten Werkzeuge, die eine ebenenübergreifende Optimierung ermöglicht.

Nach einem groben Überblick wird auf die einzelnen Komponenten des Frameworks eingegangen. Anwendungsbeispiele in Form von Fallstudien mit Ergebnissen finden sich im Kapitel 5. Eine knapper Überblick über das Framework findet sich in [43] und [169].

4.1. Motivation und Überblick

Die grundlegenden Ideen des Entwurfsframeworks sind:

1. Die Ausnutzung eines großen Energieoptimierungspotentials durch einen Ansatz auf einer hohen Abstraktionsebene (Abb. 4.1).
2. Die Ermöglichung der Wiederverwendung durch eine Modulbibliothek, deren Module auf einer geeigneten Abstraktionsebene entworfen sind (Abb. 4.2).
3. Eine entwurfsebenenübergreifende Optimierung und Durchgängigkeit des Entwurfsflusses, d. h. im Optimalfall eine Spezifikation auf hoher Abstraktionsebene und ein automatisierter Synthese-Fluss bis zum platzierten und verdrahteten Chip.
4. Die Verwendung und einfache Einbindung bestehender, auch kommerzieller EDA-Werkzeuge und anderer Hilfsmittel.

Dabei ergibt sich der dritte Punkt aus den ersten beiden, denn eine Wiederverwendung von Komponenten kann nur dann gewährleistet werden, wenn die Entwurfsschnittstelle sauber definiert ist und manuelle Anpassungen auf tieferen Ebenen auf ein absolutes Minimum beschränkt werden.

Da es aufgrund der Entwurfslücke (Abschnitt 1.1.2) für Abstraktionsebenen höher als Register-Transfer wenige Werkzeuge mit der erforderlichen Domänenunabhängigkeit[1] gibt (siehe auch Kapitel 3) wurde für die Eingangsspezifikation die Programmiersprache C++ gewählt (siehe Abschnitt 4.1.2).

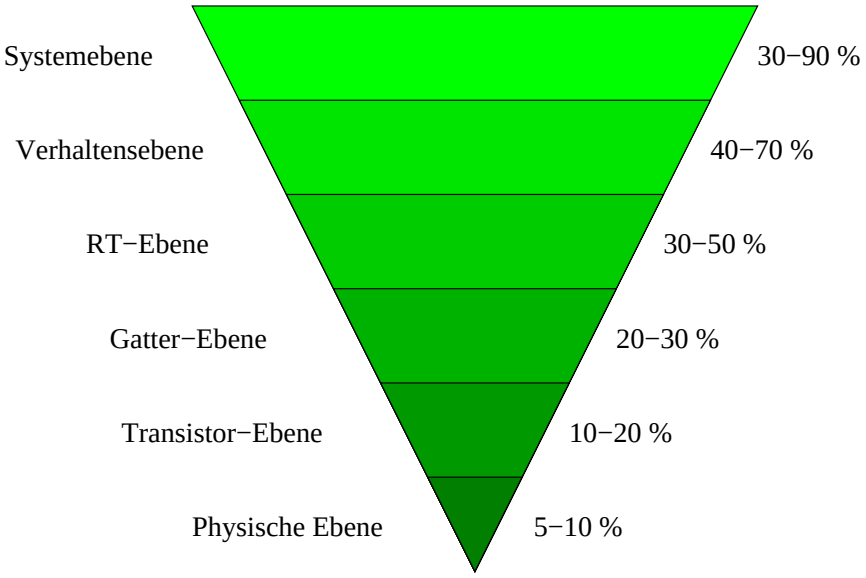


Abbildung 4.1.: Optimierungspotential nach Entwurfsebenen. Dargestellt ist die aus einer Studie ermittelte potentielle Energieeinsparung für eine Optimierung des Entwurfs auf einer bestimmten Abstraktionsebene in Prozent (aus [143]).

Nach einer Motivation für die Einbindung kommerzieller Werkzeuge im folgenden Abschnitt 4.1.1 wird in Abschnitt 4.1.2 auf die Eingabesprache für das Framework eingegangen. Daraufhin wird in 4.1.3 ein Überblick über alle verwendeten Komponenten gegeben. Im letzten Abschnitt dieses Kapitels 4.1.4 wird schließlich der Entwurfsfluss skizziert.

4.1.1. Einbindung kommerzieller Werkzeuge

RT-Synthese

Seit Erscheinen des ersten kommerziellen Compilers für die RT-Synthese 1988[2] (siehe auch Abschnitt 2.4) sind diese Werkzeuge in den vergangenen Jahren immer besser

[1] Für spezielle Probleme z. B. in der HF-Technik oder für den Entwurf digitaler Audiofilter gibt es durchaus brauchbare Lösungen. Diese haben aber alle eine proprietäre Eingabesprache und lösen nur die Probleme der jeweiligen Domäne.

[2] Es handelte sich um die erste Version des DesignCompilers von Synopsys, der seinerzeit nur Verilog als Eingabesprache unterstützte.

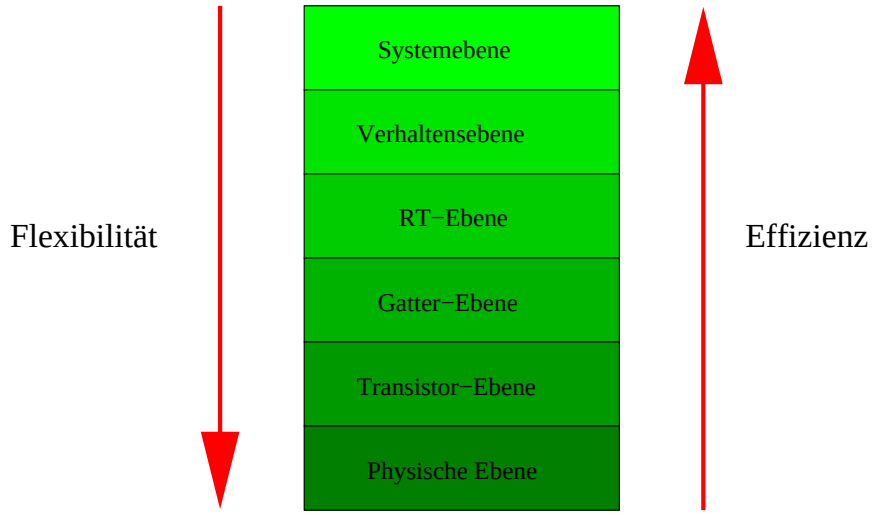


Abbildung 4.2.: Wiederverwendbarkeit von Modulen nach Entwurfsebenen. Mit höherer Abstraktionsebene für den Komponentenentwurf steigt nach Abb. 4.1 das Optimierungspotential, mithin die Energieeffizienz des daraus entstehenden Moduls. Nachteil ist, dass die Wiederverwendbarkeit des Moduls durch die starke Spezialisierung eingeschränkt wird.

geworden. Da diese Werkzeuge durch die lange Entwicklungszeit und den breiten industriellen Einsatz eine hohe Güte erreicht haben und die Entwicklung und Pflege eines solchen Produktes wegen der starken Abhängigkeiten von der Zieltechnologie und den verwendeten Bibliotheken extrem aufwendig ist, erscheint die Benutzung nicht nur gerechtfertigt, sondern im Sinne eines möglichst guten Gesamtoptimierungsergebnisses unausweichlich.

Behavioral-Synthese

Synthesewerkzeuge für höhere Abstraktionsebenen als RT werden seit mehreren Jahrzehnten entwickelt (siehe auch Abschnitt 2.7.3). Da eine höhere Abstraktionsebene den Entwurfsraum prinzipiell vergrößert, sind sehr viele dieser Produkte auf ganz spezielle Anwendungs-Domänen beschränkt. Eines der wenigen Produkte ohne solche Beschränkung ist der „Behavioral Compiler“ der Firma Synopsys, den es seit Anfang 1996 gibt. Dieses Werkzeug wird im Framework nicht nur wegen seines Allzweck-Designs, sondern auch wegen der sehr guten Integration in die auf die Behavioral-Synthese folgende RT-Synthese eingesetzt. Eine Funktionsbeschreibung und die Art der Integration des Produkts in das Framework wird in Abschnitt 4.5.1 beschrieben.

Synopsys hat die Weiterentwicklung des Behavioral Compilers 2004 eingestellt. Der Grund war die Kundenunzufriedenheit mit der Qualität der generierten RT-Hardware-Beschreibungen sowie der hohe Support-Aufwand bei einem nur geringen Umsatzanteil des Behavioral Compilers [55]. Gleichzeitig wurden die SystemC-Synthese aus dem Lieferprogramm gestrichen. Damit scheint sich Synopsys eher auf die Synthese ab RT-Ebene zu konzentrieren und das Feld der Behavioral-Synthese anderen, meist kleineren Herstellern zu überlassen. Beispiele sind CoWare, Summit, Forte, Celoxica, AccelChip

und Verisity, die inzwischen ähnliche Produkte im Sortiment haben, allerdings nicht mit dem Anspruch an Generizität, den Synopsys mit dem Behavioral Compiler verfolgt hat.

IP-Blöcke

Um einen effizienten Entwurfsablauf zu gewährleisten, wird von kommerziellen vorgefertigten Modulen für bestimmte Hardware-Verarbeitungsblöcke Gebrauch gemacht. Es handelt sich zum einen um Operatorimplementierungen aus der DesignWare-Bibliothek[3] von Synopsys. Diese sind zieltechnologieunabhängig, das Prinzip wird in Abschnitt 4.6.2 näher beschrieben. Zum anderen kommen für das Rapid-Prototyping System IP-Module der Firma Xilinx zum Einsatz. Es handelt sich dabei um Module für arithmetische Basisoperationen und verschiedene Signalverarbeitungsblöcke, die speziell für die FPGA-Architektur optimiert wurden.

HDL-Simulation

HDL-Simulation ist auf der einen Seite zur funktionalen Verifikation der Eingangsbeschreibung und aller folgenden Synthesestufen notwendig. Auf der anderen Seite dient eine HDL-Simulation auch zur Ermittlung der Schaltaktivität auf den Netzen der Schaltung, die zur Abschätzung der Verlustleistung auf den tieferen Entwurfsebenen (ab RT) benutzt wird. Aus Performanzgründen kommt im Framework der VHDL- und Verilog-Simulator „ncsim“ der Firma Cadence zum Einsatz. Er verarbeitet HDL-Texte wesentlich schneller als vergleichbare Produkte der Firma Synopsys, die ebenfalls im Rahmen dieser Arbeit evaluiert wurden.

Verlustleistungsschätzung

Die Verlustleistungsschätzung auf RT-Ebene und tiefer ist inzwischen ein zentrales Element des Chipdesigns. Von den großen EDA-Herstellern gibt es entsprechende Produkte seit etwa zehn Jahren, in denen diese Werkzeuge kontinuierlich verbessert wurden (siehe Abschnitt 4.2). Im Framework kommen hier die Werkzeuge „PowerCompiler“ bzw. „DesignPower“ sowie „Primepower“ von Synopsys zum Einsatz.

Für die Verlustleistungsschätzung auf Verhaltensebene wurde das Werkzeug ORINOCO unseres Projektpartners EHS evaluiert. Details zu den genannten Tools finden sich in Abschnitt 4.2.

[3] DesignWare ist eine sehr umfangreiche IP-Bibliothek von Synopsys. Sie enthält zum einen Standardimplementierungen für die arithmetischen Grundoperationen, z. B. Ripple-, CSA- und CLA-Addierer. Diese können automatisch inferiert werden, d. h. während der Synthese wird automatisch eine geeignete Implementierung synthetisiert. Zum anderen sind auch IP-Blöcke für komplexere Funktionen wie z. B. PCI-Interfaces oder UARTs vorhanden, die aber explizit im HDL-Quelltext instantiiert werden müssen.

4.1.2. Eingabesprache für das Framework

Die Abstraktionsebene für die Eingabebeschreibung ist beim Framework auf der Verhaltensebene angesetzt. Für diese Ebene existieren schon kommerzielle Synthesewerkzeuge, hier wird der Behavioral Compiler (BC) von Synopsys eingesetzt (siehe Kap. 4.5.1), der Behavioral VHDL (BC-VHDL) als Eingabesprache hat. Um zügig verschiedene Implementierungen eines Algorithmus' auf Verhaltensebene evaluieren zu können, ist eine BC-VHDL-Implementierung jedoch nicht geeignet, da

1. BC-VHDL nicht objektorientiert ist und die Verwendung anderer Typen für Variablen erhebliche Änderungen am Code erfordert und
2. die Simulation von BC-VHDL zwar erheblich schneller als RT-VHDL ist, jedoch immer noch zu langsam, um in vertretbarer Zeit (einige Sekunden) die Verarbeitung eines Audiotestsamples von mehreren Sekunden durchzuführen.

Es wurde daher der Weg gewählt, den Algorithmus zunächst in C++ zu implementieren, um eine sehr schnelle Exploration und Evaluation der für den Algorithmus in Frage kommenden Architekturen vornehmen zu können. Diese Beschreibung muss dann händisch in BC-VHDL umgesetzt werden; ausgehend von der BC-VHDL-Beschreibung sind keine Anpassungen mehr notwendig. Der erforderliche Umsetzungsschritt in BC-VHDL stellt jedoch keine Einschränkung des Frameworks dar; inzwischen gibt es Werkzeuge für die Behavioral-Synthese, die eine C++-Beschreibung als Eingabesprache haben[4].

4.1.3. Komponenten

Wie oben erwähnt, dient das Framework der Verknüpfung der (in der Regel kommerziellen) Synthese- und Simulationssoftware mit den selbstentwickelten Komponenten, um dem Anwender einen konsistenten automatisierten Entwurfsfluss zu bieten. Eine grobe Übersicht über die Komponenten bietet Abb. 4.3.

Die einzelnen Komponenten leisten Folgendes:

Modulbibliothek Die im Rahmen dieser Arbeit entwickelte Modulbibliothek besteht aus in Schaltungstechnik umsetzbaren funktionalen Einheiten. Sie werden vom Entwickler zur Realisierung in der Signalverarbeitung häufig benutzter Funktionen eingesetzt. Eine detaillierte Beschreibung folgt in Abschnitt 4.6.

Rapid-Prototyping Hierbei handelt es sich um eine Komponente, die den funktionalen Test der entworfenen Schaltungstechnik auf einem FPGA ermöglicht. Durch generische Schnittstellen ist es möglich, Teile eines umzusetzenden Algorithmus in Software zu simulieren, während ein anderer Teil auf dem FPGA läuft. Die Schnittstelle ist für den Entwickler transparent und wird über zwei auf der PCI-FPGA-Karte[5] vorhandene RAM-Bänke realisiert. Eine detaillierte Beschreibung folgt in Abschnitt 4.7.

Audioqualitätsbewertung Diese Komponente wird während der Quantisierungsphase benötigt und dient der Abschätzung des Rauschverhaltens der resultierenden

[4] Ein Beispiel ist das Werkzeug *Catapult SL* der Firma Mentor Graphics, vorgestellt am 12.6.2006 [3].
[5] Die FPGA-Karte wird in Abschnitt 4.7 beschrieben.

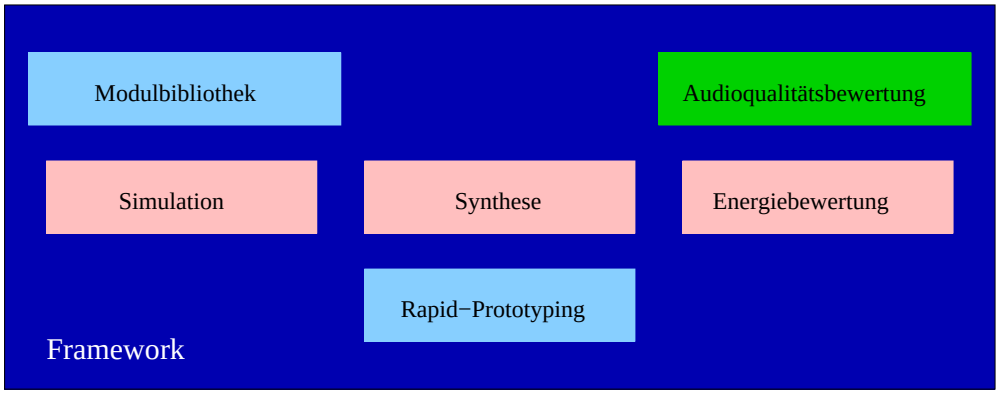


Abbildung 4.3.: Framework-Komponenten. Das Framework fasst kommerzielle (rot) und eigene Komponenten (blau und grün) zusammen und stellt dem Benutzer einen durchgehenden Low-Power-Entwurfsfluss von der Spezifikation bis zur Gatternetzliste zur Verfügung. Die Audioqualitätsbewertung ist im Rahmen des PRO-DASP-Projektes bei der AG MEDI entstanden.
Die Zusammenfassung der Komponenten im Rahmenwerk erfolgt über ein hierarchisches Skript-System, das den Implementierungsvorgang steuert.

schaltungstechnischen Umsetzung des Signalverarbeitungsalgorithmus'. Durch die integrierte Audioqualitätsbewertung ist weiterhin eine vereinfachte Verifikation des Systems über die Implementierungsebenen hinweg möglich.

Implementierung des Rahmenwerks als Skriptsystem

Die Verknüpfung aller Komponenten erfolgt automatisiert über ein hierarchisches *Skriptsystem*, das den eigentlichen Rahmen um die Komponenten bildet.

Die Skripte der höchsten Ebene rufen dabei sukzessive die einzelnen Komponenten in der richtigen Reihenfolge auf. Diese werden ihrerseits über Skripte gesteuert. Dabei wird sichergestellt, dass die Eingabedaten für das jeweilige Modul im richtigen Format zur Verfügung stehen, nötigenfalls wird eine Konvertierung durchgeführt.

Aufgrund der Heterogenität der Komponenten, die jeweils eigene Software-Schnittstellen (APIs[6]) haben, konnte nicht eine einzige Programmiersprache verwendet werden. Die für das Skipt-System verwendeten Programmiersprachen sind Python, Bash, TCL und Perl.

4.1.4. Entwurfsfluss

Die Design-Ziele des Frameworks **Durchgängigkeit**, **hohes Optimierungspotential**, **Modularität** und **Wiederverwendbarkeit** spiegeln sich direkt im Entwurfsfluss wieder.

[6] *Application Programming Interface*, eine Programmierschnittstelle, die es ermöglicht, über ein Programm mit dem Modul, das die API bereitstellt, zu kommunizieren. Die API ist in der Regel von der verwendeten Programmiersprache für das Modul abhängig.

Durchgängigkeit Die Spezifikation des zu implementierenden Systems erfolgt auf der Verhaltensebene. Die durch die Entwicklerin erforderliche händische Neukodierung des Algorithmus auf tieferen Abstraktionsebenen entfällt.

Die Eingabesprache des Frameworks ist C++, deshalb muss eine Matlab-Implementierung momentan noch händisch in C++ kodiert werden, allerdings bleibt die Abstraktionsebene erhalten, weshalb für die Portierung keine Architekturentscheidungen getroffen werden müssen. Mittels des *RT-Workshops*[7] wäre auch eine automatische Codegenerierung denkbar, dies wurde jedoch im Rahmen dieser Arbeit nicht ausprobiert.

Die Analyse und Optimierung des Algorithmus erfolgt mittels der C++-Beschreibung, jedoch ist für die Anbindung an die Synthese-Toolchain eine Umsetzung in Behavioral-VHDL erforderlich. Die Abstraktionsebene bleibt wiederum erhalten, es erfolgt hiermit noch keine Festlegung der RT-Architektur[8].

Hohes Optimierungspotential Da die Systemspezifikation auf Verhaltensebene erfolgt, wird die getaktete RT-Struktur noch nicht festgelegt und kann automatisch exploriert und optimiert werden (siehe Abschnitt 4.5.1). Weiterhin sind für die Audiosignalverarbeitung optimierte Module in das Framework integriert. Mittels der integrierten Audioqualitätsmaße (Abschnitt 4.4) ist eine Neuquantisierung des Algorithmus' möglich, ohne die gehörte Audioqualität zu beeinflussen. Eine integrierte Abschätzung der durch eine Implementierungsvariante benötigten Energie ermöglicht den Vergleich verschiedener Architekturen (siehe Abschnitt 4.2).

Modularität und Wiederverwendbarkeit Der Algorithmus kann unter Zuhilfenahme einer Modulbibliothek spezifiziert werden, die hardwareunabhängige, wiederverwendbare Module enthält (siehe Abb. 4.5 und Abschnitt 4.6).

Der Analyse-, Partitionierungs- und Quantisierungsablauf, der für jeden zu implementierenden Algorithmus zunächst auf Basis eines C++-Modells erstellt wird, ist in Abb. 4.4 dargestellt.

Dabei wird aus der Algorithmus-Spezifikation, die typischerweise in Matlab erfolgt, ein C++-Modell hergeleitet, dessen Validität mittels der Gütemaße und repräsentativer Audiosamples bestätigt werden kann. Diese C++-Referenz kann entweder schon aus bestehende Framework-Modulen aufgebaut werden, oder es erfolgt in einem weiteren Schritt die Partitionierung des Algorithmus' auf die vorhandenen Module.

Jetzt kann für jeden Operator eine optimale Quantisierung gefunden werden, indem Qualitätsmessungen für verschiedene von der Entwicklerin festgelegte Zahlensysteme und Genauigkeiten erfolgen[9], zudem können verschiedene Modulvarianten (z. B. für

[7] Der RT-Workshop ist ein Zusatzprogramm für Matlab/Simulink, das es ermöglicht, C- oder C++-Code aus einer Matlab- oder Simulink-Beschreibung zu generieren. Es ergeben sich aber teils sehr subtile Probleme; so werden z. B. die Datentypen der Festkomma-Toolbox auf ANSI-C-Typen umgesetzt, was sowohl eine spätere Requantisierung als auch das Ersetzen des Zahlensystems sehr erschwert, da hierfür abstrakte, gekapselte Datentypen notwendig sind.

[8] Inzwischen sind kommerzielle Werkzeuge vorhanden, die diese Umsetzung vollständig automatisieren könnten, s.o..

[9] Diese Exploration der Zahlendarstellung ist sehr aufwendig, da theoretisch jedes einzelne Operator optimiert werden kann. Deswegen stellt das Framework Skripte bereit, die eine automatische Exploration

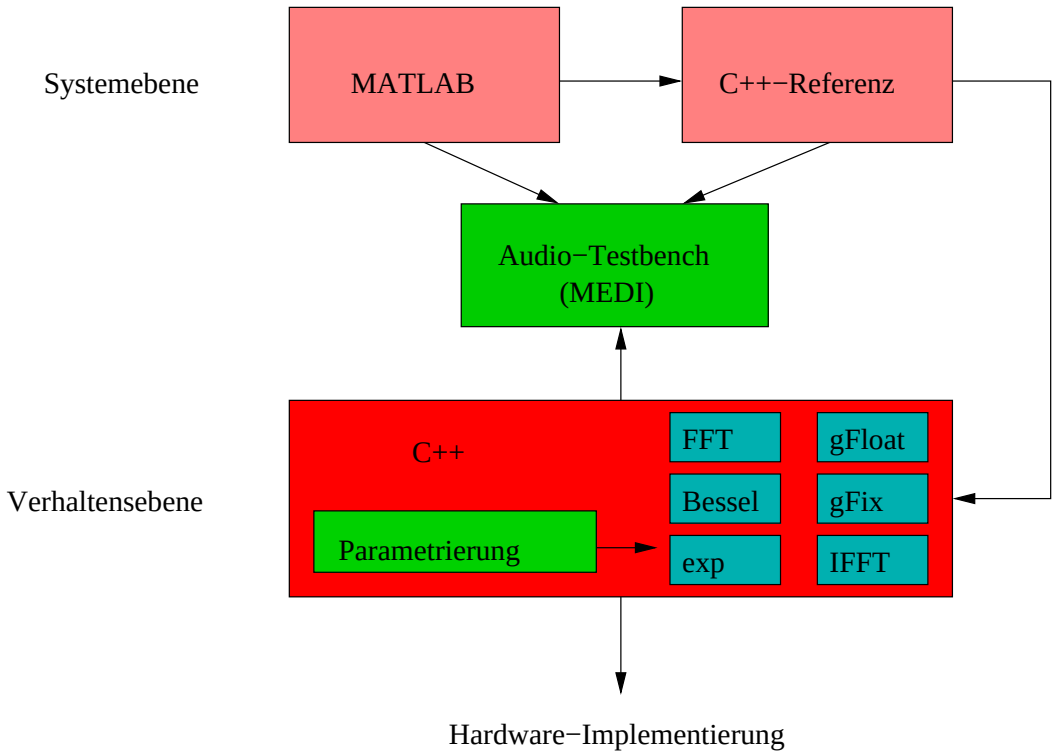


Abbildung 4.4.: Optimierung des Software-Prototypen. Auf Systemebene sind die zu optimierenden Algorithmen oft als Matlab-Quelltext vorhanden und müssen dann in eine C++-Referenz-Implementierung umgesetzt werden. Die Validität der Umsetzung kann mit der von der AG MEDI gelieferten Audiotestbench, die ein Gütemaß darstellt, überprüft werden (siehe Abschnitt 4.4.2).
Innerhalb des Frameworks werden auf Verhaltensebene Schritt für Schritt Framework-Module eingesetzt, für die bit-genaue Simulationsprototypen existieren. Die Module werden dann parametriert (z. B. durch Festlegung der Zahlendarstellung oder Quantisierung) und können sofort mithilfe des Gütemaßes überprüft werden, außerdem kann eine erste relative Energieschätzung vorgenommen werden.
Auf diese Weise wird der Algorithmus iterativ optimiert, bis sowohl eine befriedigende Verarbeitungsgüte als auch Energieeffizienz erzielt ist.
Mit der gewonnenen Darstellung kann ein Behavioral-VHDL-Prototyp erstellt werden, der die Grundlage für den folgenden Hardware-Synthese-Flow ist.

FIR-Filter) exploriert werden (Abb. 4.5). Weiterhin kann eine (grobe) relative Energieschätzung erfolgen[10].

Für den nicht durch die Module abgedeckten Teil des Algorithmus' ist eine händische Kodierung in Behavioral-VHDL nötig. Die Module verfügen bereits über eine solche Beschreibung, die sich durch einen einfachen Funktionsaufruf in die Top-Level Beschreibung integrieren lässt[11].

Liegt eine solche Top-Level Beschreibung in VHDL vor, ist jederzeit eine automatische Abbildung auf Hardware möglich, die einen genauen Vergleich der durch die geschätzten Varianten benötigten Energie erlaubt.

In Abb. 4.6 ist der gesamte Entwurfs-Fluss überblicksartig dargestellt.

ohne manuellen Eingreifen ermöglichen.

[10] Die Module können bezüglich des Energiebedarfs vorcharakterisiert werden (siehe Abschnitt 4.6). Die Schätzung ist umso genauer, je größer der durch die Module abgedeckte Teil des Gesamtalgorithmus' ist. Für die Modul-externen Teile des Algorithmus wird momentan noch keine Power-Schätzung auf dieser Ebene vorgenommen; denkbar ist hier aber der Einsatz z. B. von ORINOCO (siehe Abschnitt 4.6).

[11] Der Integrationsmechanismus ist in Abschnitt 4.6.2 dargestellt.

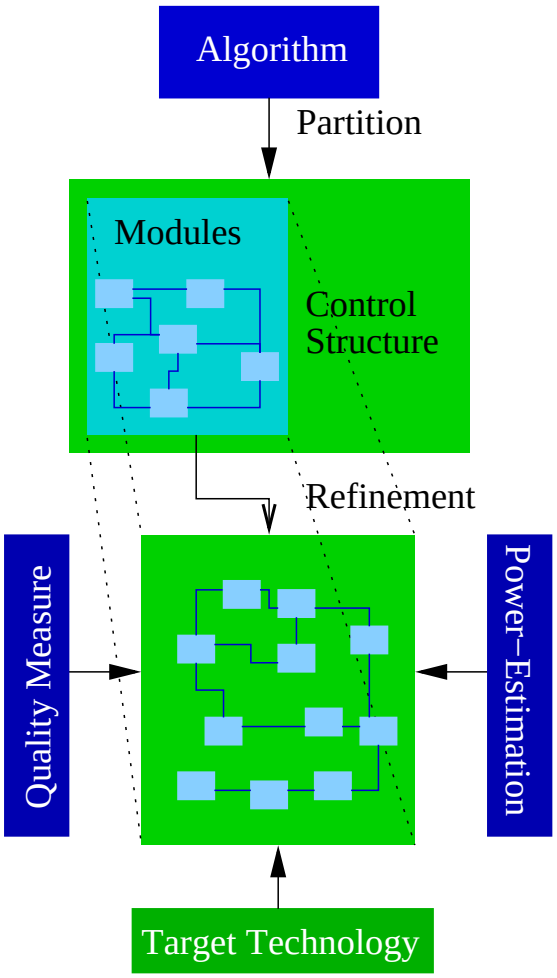


Abbildung 4.5.: Modul-Verfeinerung. Ist der Algorithmus auf die Module abgebildet worden, kann eine schrittweise Optimierung der Darstellung erfolgen. Auf der einen Seite können verschiedene Module, die die gleiche Funktion implementieren, evaluiert werden. Auf der anderen Seite kann die Zahlendarstellung festgelegt werden, worunter die Auswahl des Zahlensystem und eine operatorspezifische Festlegung der Quantisierung zu verstehen ist.
Liegt für das Top-Level Design, das die Module integriert, eine Behavioral VDHL Darstellung vor, so kann ein automatisiertes Mapping auf eine Hardwarestruktur erfolgen. Dabei sind neben einer Architekturexploration viele Standardmaßnahmen verfügbar, um den Energiebedarf der Schaltung zu senken (Abschnitt 4.3). Die Schätzung des Energiebedarfs erfolgt mit der in Abschnitt 4.2 dargestellten Methodik, die über Skripte in das Framework integriert wurde.

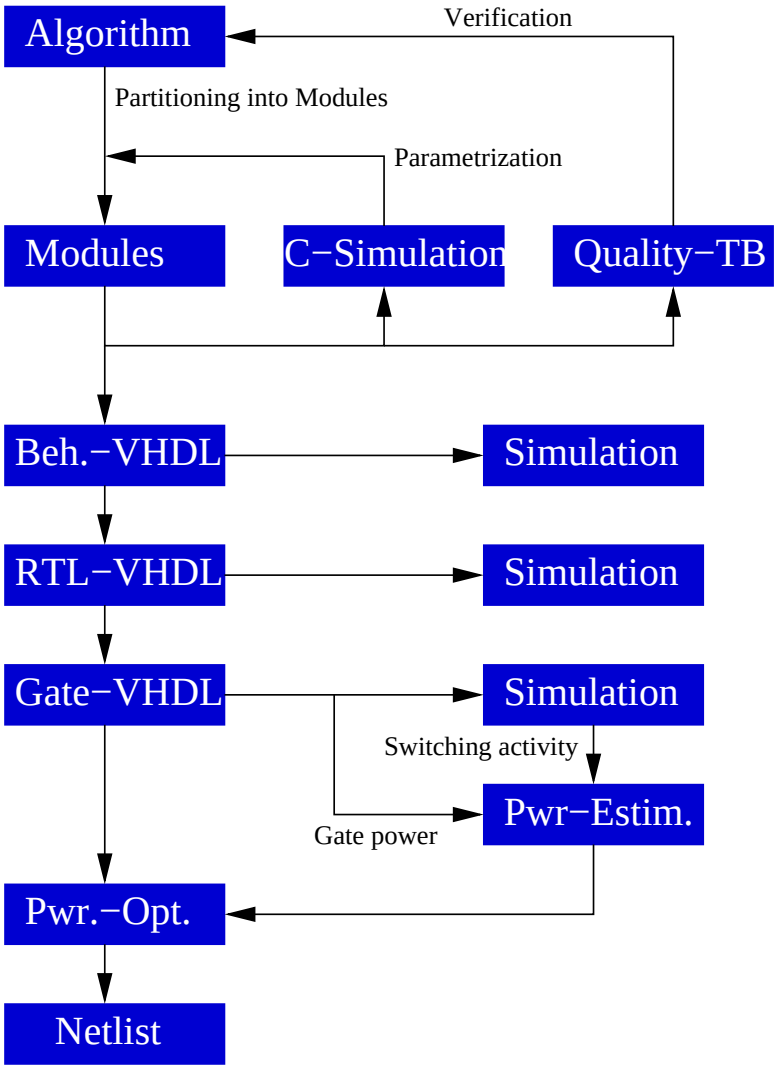


Abbildung 4.6.: Vollständiger Entwurfsfluss des Frameworks. Zunächst erfolgt eine Partitionierung des Algorithmus' in Module. Danach erfolgt die Optimierung auf Verhaltensebene durch Simulation des Algorithmus in C++ und Ermittlung der Audioqualität mittels der Gütemaße.
Dann erfolgt die Umsetzung auf eine Zieltechnologie durch eine Toolchain zur Behavioral-Synthese. Der Energieverbrauch kann durch Ermittlung der Schaltaktivität durch eine Simulation auf jeder Abstraktionsebene geschätzt werden, jedoch steigt die Genauigkeit mit abnehmender Abstraktionsebene. Die Simulationsergebnisse werden ferner für eine automatische, qualitätsmaßbasierte Verifikation genutzt (in der Abb. nicht dargestellt).

4.2. Verlustleistungsschätzung

Voraussetzung für valide Optimierungsstrategien ist eine hinreichend genaue Verlustleistungsschätzung, die zumindest Aussagen über den relativen Verlustleistungsunterschied zwischen Designvarianten ermöglicht.

4.2.1. Grundbegriffe

Alle Werkzeuge zur Verlustleistungsschätzung teilen die Verlustleistung in statische und dynamische Verlustleistung ein[12]:

Statische Verlustleistung

Die statische Verlustleistung beschreibt die Verlustleistung zu der Zeit, in der sich die Schaltung im Gleichgewicht befindet, d. h. zwischen den Taktflanken. Diese ist für jedes Gatter abhängig vom Zustand der Eingänge, weshalb eine gut charakterisierte Bibliothek für jedes Gatter eine Tabelle analog zur Wahrheitstabelle des Gatters hinterlegt hat, die die statische Verlustleistung für jeden Logikzustand bei einer Norm-Temperatur wiedergibt (siehe Tab. 4.1).

a	b	P
0	0	P_0
0	1	P_1
1	0	P_2
1	1	P_3

Tabelle 4.1.: Tabelle für statische Verlustleistung eines 2-Eingangs-Gatters

Neben festen Werten für P kann die Tabelle auch eine Polynomfunktion enthalten, die die Verlustleistung in Abhängigkeit der Temperatur und Versorgungsspannung modelliert. Dies bezeichnet die Firma Synopsys mit SPPM (*Scalable Polynomial Power Model*) oder SPLM (*Scalable Polynomial Leakage Model*) [152]. Dies ist wichtig, wenn der Schaltkreis nicht mit der spezifizierten Versorgungsspannung betrieben wird oder unter großen Abweichungen von der Normtemperatur.

Da schnell getaktete Schaltkreise im Betrieb hohe Sperrschichttemperaturen erreichen können (> 80 °C), nimmt die Eigenleitung des Halbleitermaterials stark zu, womit große Leckströme verbunden sind. Bei kleinen Strukturgrößen mit niedrigem V_t und einer großen Chipfläche ist die statische Verlustleistung dann gegenüber der dynamischen Verlustleistung dominierend. Bei den untersuchten Algorithmen und einer 0,18 μm 1,8 V Technologie konnte diese Dominanz jedoch noch nicht beobachtet werden (siehe Kap. 5). Bei modernen Mikroprozessoren ist dies jedoch schon länger der Fall (siehe Abschnitt 2.5).

[12] Eine genaue Beschreibung der Arten und Entstehung von Verlustleistung findet sich in Kap. 2.5.

Dynamische Verlustleistung

Die dynamische Verlustleistung entsteht durch Transitionen auf Netzen. Sie wird weiter unterteilt in Schaltleistung und Zell-interne Leistung.

Die Schaltleistung wird beschrieben durch die Energie, die notwendig ist, um die Kapazität an einem Ausgang einer Standardzelle umzuladen. Diese ist abhängig von der Zeitkonstante der Verbindungsleitungen (Interconnect) und den Zeitkonstanten der an den Ausgang angeschlossenen Zell-Eingänge (*Fan-Out*).

Die Zell-interne Verlustleistung wird durch die bei einer Eingangstransition innerhalb der Standardzelle benötigte Energie beschrieben. Diese ist im wesentlichen durch interne Kapazitäten und Kurzschlussströme in CMOS-Paaren bedingt (siehe Kap. 2.5).

Die Berechnung der dynamischen Verlustleistung erfolgt unterschiedlich, je nachdem ob eine ereignisbasierte Berechnung oder eine auf einer Aktivitätsstatistik basierende Berechnung erfolgt.

Bei der **ereignisbasierten** Berechnung wird für jedes in der Simulation erkanntes Ereignis wie folgt vorgegangen:

1. Die Eingangsrampensteilheit wird aus der Zeitkonstante für die vom Netz betroffenen Eingänge und der Zeitkonstante des Netzes berechnet
2. Die Zell-interne Verlustleistung wird aus den Eingangsrampensteilheiten aller betroffenen Eingänge ermittelt. Dies passiert über eine Tabelle, die jeder Rampenzeit eine Energie zuordnet, wobei für steigende und fallende Flanken verschiedene Tabellen zum Einsatz kommen können. Weiterhin können für Zellen mit mehreren Eingängen für jeden Eingang eigene Tabellen hinterlegt sein. Die Tabelle kann statt eines festen Energiewertes auch ein Polynom enthalten, das die Energie in Abhängigkeit der Versorgungsspannung und Temperatur wiedergibt analog zu der Berechnung der statischen Verlustleistung (s.o.).
3. Die Schaltleistung wird nach Ermittlung der Zeitkonstante des Ausgangsnetzes ebenfalls über einen Tabelleneintrag ermittelt, der verschiedenen Zeitkonstanten eine Transitionsenergie zuweist.

Zusätzlich kann eine zustandsabhängige Leistungsaufnahme modelliert werden, wobei der Zustand einer Zelle als zusätzlicher Tabellenparameter auftritt. Dies ist z. B. bei RAM-Zellen wichtig, da deren Leistungsaufnahme vom aktuellen Inhalt abhängt. Der Zustand der Zelle muss vom Simulator extra annotiert werden.

Je nach Güte der Power-Charakterisierung weisen die Tabellen unterschiedlich viele Einträge auf.

Bei der **Berechnung auf Basis einer Aktivitätsstatistik** wird jeder Zelleingang mit einer Schaltwahrscheinlichkeit pro Taktzyklus annotiert. Daraus kann das Schätzwerkzeug einen Mittelwert für die Leistungsaufnahme jeder Standardzelle berechnen.

4.2.2. Synopsys PowerCompiler und DesignPower

Im Sommer 1994 wurde von Synopsys ein „High-Level“ Verlustleistungsanalyse-Werkzeug namens DesignPower [30] vorgestellt. Hiermit war es mit einem kommerziellen

Werkzeug erstmals möglich, für ein Design ab der RT-Ebene eine automatische Schätzung der Verlustleistung durchzuführen. Die frühe Schätzung der Verlustleistung soll es dem Entwickler ermöglichen, bereits hier Optimierungen durchzuführen, die entsprechend große Auswirkungen auf tiefere Schichten im Designflow haben (Abb. 4.7).

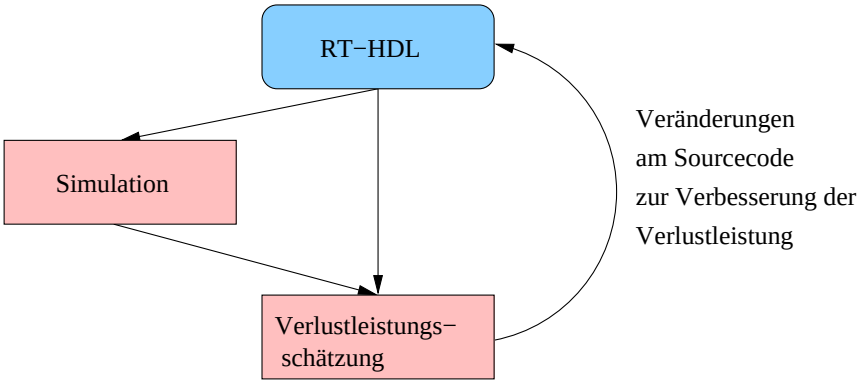


Abbildung 4.7.: Verlustleistungsschätzung auf RT-Ebene mit DesignPower bzw. PowerCompiler. DesignPower kann eine Verlustleistungsschätzung auf RT-Ebene vornehmen. Die Schaltwahrscheinlichkeiten für die syntheseinvarianten Elemente können dabei aus einer RT-Simulation kommen. Die kurzen Laufzeiten für die Simulation und Verlustleistungsschätzung und der hohe Automatisierungsgrad ermöglichen viele Optimierungszyklen. Die durch die Synthese erzeugte Gatternetzliste kann auf analoge Weise mit DesignPower bewertet werden.

Synopsys DesignPower

Während ältere Ansätze lediglich die Schaltvorgänge im Design zählten und auf dieser Grundlagen eine Schätzung durchführten, hatte DesignPower [152] einen probabilistischen Ansatz, der durch Propagierung einer annotierten Schaltwahrscheinlichkeit[13] der Eingänge eine „post-layout"-Schätzung auf Basis der RT-Beschreibung ermöglichte. Die Schaltwahrscheinlichkeit der syntheseinvarianten Eingänge wird entweder durch eine RT-Simulation ermittelt oder direkt vom Entwickler spezifiziert. Ähnlich wie bei der Ermittlung des Timings eines Designs, bildet DesignPower für die Verlustleistungsschätzung die synthetischen Operatoren auf DesignWare-Module, die für die entsprechende Zieltechnologie optimiert werden, ab und propagiert die annotierten Schaltwahrscheinlichkeiten in diese Module. Zur Berechnung der absoluten Verlustleistung wird dann lediglich eine power-charakterisierte Zielbibliothek benötigt (was zum damaligen Zeitpunkt allerdings eine Seltenheit war). Für genauere Schätzungen kann auch eine Gatternetzliste dienen, wobei neben den syntheseinvarianten Netzen nun auch die Synthesevarianten über eine Simulation mit einer realen Schaltwahrscheinlichkeit annotiert werden können, was die Genauigkeit weiter erhöht.

[13] Die Grundlage für die Verlustleistungsschätzung ist die Wahrscheinlichkeit, dass eine logische Eins anliegt sowie die Transitionsrate, d. h. die Anzahl der 0→1- und 1→0-Übergänge pro Zeiteinheit.

Synopsys PowerCompiler

Im April 1996 benannte Synopsys DesignPower in PowerCompiler [152] um, da neben einer leicht verbesserten Schätzungs-Methode jetzt die Möglichkeit bestand, die Verlustleistung (neben Fläche und Geschwindigkeit) als Optimierungskriterium bei der Synthese zu verwenden. Es handelt sich hierbei faktisch um die Erweiterung des DesignCompilers um Verlustleistungs-Constraints.

Weiterhin kann ein automatisiertes Clock-Gating durchgeführt werden und eventuelle Einsparungen durch eine zusätzliche „Operandenisolierung“ erzielt werden.

Weiterhin wurde mit dem PowerCompiler die Möglichkeit geschaffen, die durch Leckströme verursachte Verlustleistung zu schätzen und (falls eine Multi-V_t-Zieltechnologie zur Verfügung steht) zu optimieren. Ein kurzer Überblick ist Abbildung 4.8 zu entnehmen.

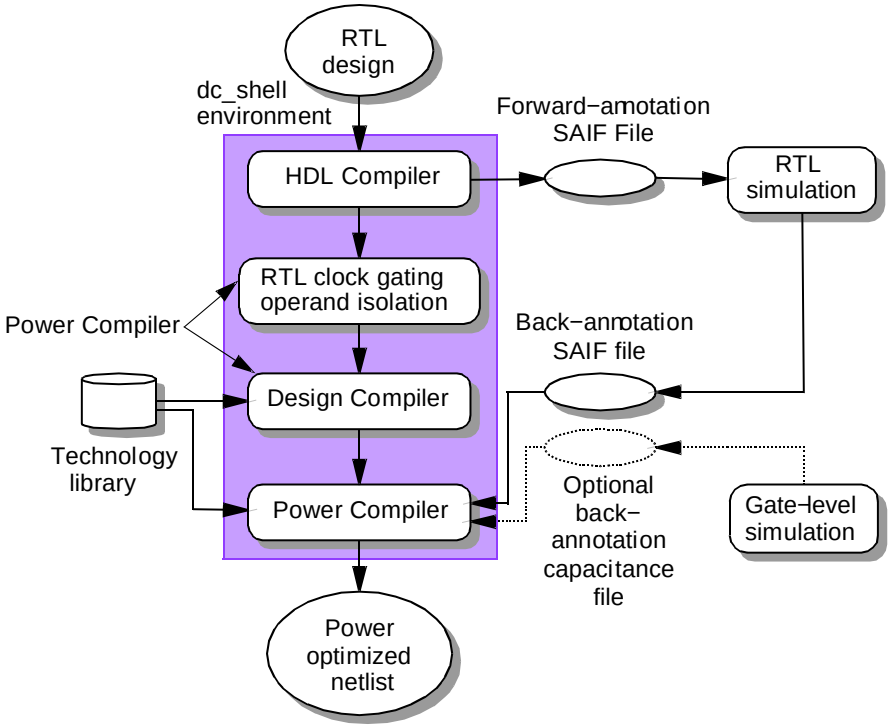


Abbildung 4.8.: Vollständiger PowerCompiler-Flow. Die Darstellung zeigt den vollständigen Leistungsschätzungs- und -optimierungs-Fluss für den PowerCompiler. Für die Verlustleistungsschätzung auf RT-Ebene muss zunächst die Schaltaktivität der syntheseinvarianten Knoten ermittelt werden. Die Knotenliste wird vom PowerCompiler ermittelt und in eine Datei geschrieben, die sogenannte vorwärts-kommentierte (forward-annotated) Liste. Sie hat das Format eines standardisierten SAIF (Switching Acitivity Information File), die jedoch keine gültigen Einträge für die Schaltaktivitäten hat. Aus dieser Liste extrahiert der Simulator (je nach Simulator mit mehr oder weniger Hilfe des Entwicklers) die während der Simulation zu beobachtenden Knoten. Der Simulator ergänzt nach erfolgter Simulation die (statistische) Schaltaktivität jedes Knotens. Diese Liste wird jetzt als rückwärtskommentierte (backward-annotated) Liste bezeichnet und kann vom PowerCompiler für die Verlustleistungsschätzung verwendet werden (aus [152]).

Genauigkeit

Synopsys schweigt sich offiziell über die Genauigkeit der PowerCompilers aus, da sie sehr stark von der Genauigkeit der Verlustleistungscharakterisierung der Bibliotheken abhängt. Bei diese Stufe der Schätzung geht es eher darum, relative Aussagen zwischen verschiedenen (sich nicht allzu stark unterscheidenden) Design-Varianten zu ermöglichen, um die Auswirkungen bestimmter Änderungen bereits auf RT-Ebene zu ermitteln.

Die Verlustleistungsschätzung auf Gatter-Ebene, d. h. wenn die komplette Architektur feststeht, kann schon wesentlich genauer sein, obgleich Synopsys auch hierzu keine konkrete Angabe macht. Technologiehersteller wie z. B. LSI sprechen jedoch von einer Abweichung von ca. ±25 % für eine 0.25μm-Prozess gegenüber einer Simulation auf Polygon-Ebene etwa mit SPICE [91].

Die inzwischen verfügbaren Low-Power Standardzellen-Bibliotheken wie die für die Power-Abschätzungen in dieser Arbeit verwendete UMC-Bibliothek weist eine sehr detaillierte Power-Charakterisierung auf, so dass der Hersteller von einer Genauigkeit von ±5 % spricht [164]. Diese gilt allerdings nur für die Standardzellen, nicht für die Verbindungsleitungen (*Interconnect*).

Unterhalb von 0.18μm wird die absolute Genauigkeit aufgrund der höheren Prozessvariabilität wieder schlechter [128].

Die Verdrahtung kann bei großen Chips (> 10 Mio. Transistoren) erheblich zur dynamischen Verlustleistung beitragen.

4.2.3. Synopsys PrimePower

Im September 2000 hat Synopsys das Werkzeug PrimePower vorgestellt, das letztlich DesignPower als Standalone-Verlustleistungsschätzer ablösen sollte[14]. Vorteile gegenüber dem alten Werkzeug sind neben einer verbesserten Skalierbarkeit vor allem die Möglichkeit einer ereignisbasierten Powerschätzung und damit eine sehr hohe Vorhersagegüte. Dieses Werkzeug kam daher als Ergänzung zum PowerCompiler insbesondere für die Modul-Charakterisierung zum Einsatz. Der PrimePower-Flow ist in Abbildung 4.9 dargestellt, nähere Erläuterungen stehen im nächsten Abschnitt.

PrimePower benötigt Power-charakterisierte Standardzellen-Bibliotheken. Die Powercharakterisierung kann z. B. mit dem Synopsys-Produkt PowerArc erstellt werden und bestimmt maßgeblich die Güte der Vorhersage.

Ereignisbasierte Verlustleistungsschätzung

Bei der ereignisbasierten Verlustleistungsschätzung wird die statische und dynamische Verlustleistung für jede Signaltransition jeder Zelle aufsummiert. Das bedeutet konkret, das der Simulator dem Analysator die zeitlichen Verläufe für alle Signale (*Waveforms*) zur Verfügung stellen muss.

[14] DesignPower war zu diesem Zeitpunkt schon nicht mehr erhältlich, der Schätzalgorithmus ist in den PowerCompiler als Optimierungswerkzeug eingeflossen

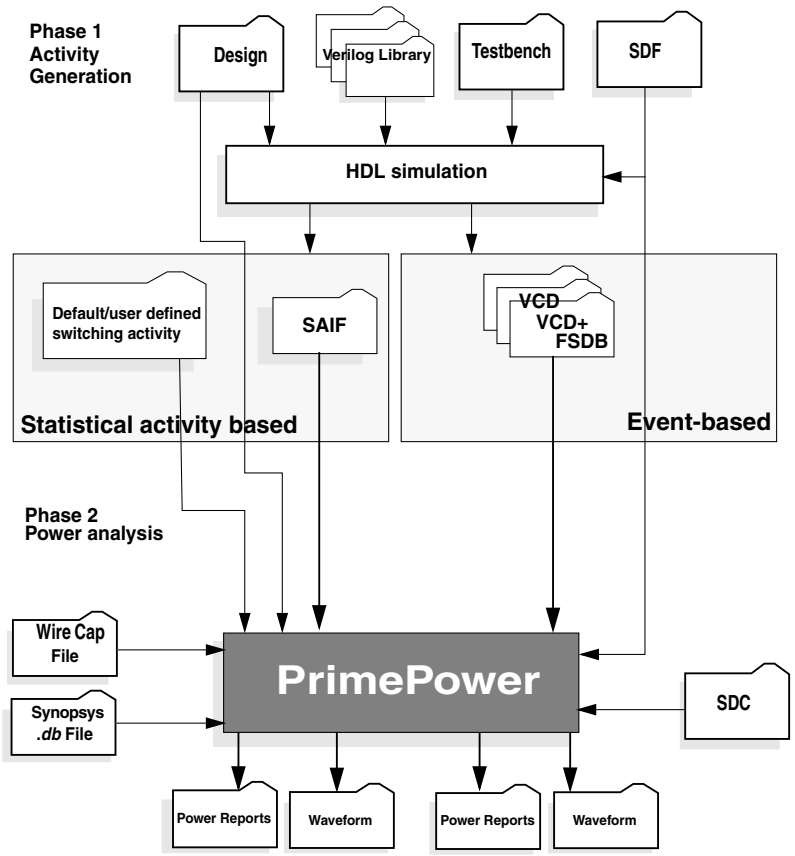


Abbildung 4.9.: PrimePower-Flow. Das Werkzeug bietet sowohl die sehr genaue Ereignis-basierte Schätzung als auch eine statistische Schätzung an. Für die Charakterisierung der Framework-Module wurde der Ereignis-basierte Schätzer verwendet. Neben der Netzliste im Verilog-Format und der Testbench kann noch eine SDF-Datei (*Structured Delay File*) spezifiziert werden, die die RC-Werte der Verbindungsleitungen der platzierten und verdrahteten Netzliste enthält. Dadurch ist eine Schätzung auf Floorplan-Ebene möglich (aus [153]).

Um genaue Ergebnisse zu erhalten, ist es außerdem wichtig, den Kapazitäts- und Widerstandsbelag der Interconnect-Leitungen zu berücksichtigen, die allerdings nur nach dem Platzieren und Verdrahten der synthetisierten Netzliste verfügbar sind. Im Framework wurde deshalb der PhysicalCompiler von Synopsys genutzt, um eine vollautomatische Platzierung und Verdrahtung auf einem annähend quadratischen Die zu erzeugen. Daraus konnte dann eine SDF (*Structured Delay File*) Beschreibung extrahiert werden, die für jede Leitung einen *RC*-Wert vorgibt.

4.2.4. ORINOCO

Im Gegensatz zu den vorgenannten Werkzeugen ermöglicht ORINOCO eine Verlustleistungsschätzung schon auf der Verhaltensebene. Da es sich bei ORINOCO um einen eigenständigen Forschungsansatz zur Low-Power Optimierung handelt und „Work in Progress“ darstellt, wurde es in Abschnitt 3.8 bereits betrachtet.

Aufgrund der häufigen Umstrukturierungen ist es bisher leider noch nicht gelungen, ORINOCO in das Entwicklungsframework zu integrieren.

4.2.5. Bewertung

Die genaueste Abschätzung der Verlustleistung liefert eine ereignisbasierte Schätzung durch PrimePower. Die Heuristiken der statistischen Abschätzung sind jedoch so gut, dass für die untersuchten Beispiele keine großen Abweichungen registriert wurden (siehe Kapitel 5).

Der Nachteil ist eine sehr lange Laufzeit der Schätzung, da für eine genaue Schätzung eine Simulation auf Floorplan-Ebene durchgeführt werden muss.

Für die Framework-Module wurde dennoch dieser Weg gegangen, auch um die Ergebnisse besser vergleichen zu können.

Eine Schätzung auf Verhaltensebene wie durch ORINOCO möglich, erscheint als sehr attraktiver Weg, schon bei der Erstellung eines Algorithmus eine Vergleichsmöglichkeit bezüglich verschiedener Architekturen zu haben und das Design so sehr früh optimieren zu können. Knackpunkt ist hier jedoch die Tatsache, dass für eine realistische Schätzung schon die Detail-Architektur feststehen muss, d. h. es muss eine Allokation und ein Scheduling durchgeführt werden und es muss detaillierte Technologieinformation vorliegen. Aufgrund der vielen Architekturoptionen, die eine Spezifikation auf Verhaltensebene bietet, ist dies ein sehr schwieriges Unterfangen. Nicht zuletzt scheiterte die Integration ORINOCOs daran, dass für die verwendete UMC-Low-Power-Bibliothek keine ORINOCO-Charakterisierung vorhanden war und die Ergebnisse so nicht vergleichbar waren.

Die gleichen Probleme treten prinzipiell auch bei der im Framework implementierten Variante der Power-Charakterisierung der Framework-Module auf (siehe Abschnitt 4.6.2), jedoch soll hier nur ein relativer Vergleich zwischen Designvarianten ermöglicht werden und der Gesamtalgorithmus ist idealerweise zu einem hohen Grad aus

bekannten Modulen aufgebaut. Auch sind alle Synthese- und Charakterisierungsvorgänge für das Framework automatisiert (skriptgesteuert), so dass Anpassungen an neue Bibliotheken mit einem Minimum an Aufwand zu realisieren sind.

Abschließend lässt sich sagen, dass jede Verlustleistungsschätzung nur so gut wie die Power-Charakterisierung der zugrunde liegenden Zieltechnologiebibliothek sein kann. Gerade Bibliotheken, die nicht explizit für Low-Power-Prozesse erstellt wurden, sind oft sehr mangelhaft charakterisiert (siehe Abschnitt 4.6.2).

4.3. Verlustleistungsoptimierung

Die Verlustleistungsoptimierung des Frameworks beginnt beim Entwurf des Gesamtsystems. Durch die Bereitstellung einer Modulbibliothek mit optimierten Funktionen kann die Entwicklerin schon beim Entwurf das System aus geeigneten Standardkomponenten zusammensetzen. Diese sind zieltechnologieunabhängig entworfen, haben aber durch die relativ abstrakte Spezifikation ein hohes Optimierungspotenzial.

Im nächsten Schritt wird die Quantisierung der Module durch die Entwicklerin aufgrund von Simulationsläufen innerhalb des Frameworks festgelegt. Dazu gibt es eine Reihe von Analysemöglichkeiten, um unter Einbeziehung des integrierten Audiogütemaßes (Abschnitt 4.4) die für den jeweiligen Einsatzzweck beste Quantisierung für jedes einzelne Modul zu finden (siehe Abschnitt 4.4).

Die automatische Verlustleistungs-Optimierung erfolgt dann mit kommerziellen Werkzeugen ab der Verhaltensebene.

Zunächst erfolgt eine Beschreibung der Optimierungsstrategien auf Verhaltensebene, in Abschnitt 4.3.2 sind die verwendeten Optimierungen ab der RT-Ebene zusammengefasst.

4.3.1. Optimierung auf Verhaltensebene

Die Low-Power-Optimierung auf Verhaltensebene erfolgt skriptgesteuert durch den Behavioral Compiler. Das Scheduling des BC ist zwar durch das Syntheseskript beeinflussbar, ein spezielles Low-Power Scheduling wie etwa in [182] beschrieben kann damit aber nicht umgesetzt werden[15].

Die im Framework umgesetzten Strategien sind nachfolgend beschrieben.

Parallelisierung

Um ein besonders großes Einsparpotenzial zu erzielen, ist es notwendig, den Systemtakt zu senken und den Durchsatz zu steigern[16]. Dies ist in erster Linie über eine Parallelisierung von Operationen möglich. Als Nebeneffekt benötigt ein paralleles Design mehr Chip-Fläche und hat damit eine höhere statische Verlustleistung. Bei den untersuchten Algorithmen war dieser Effekt im Vergleich zur eingesparten Verlustleistung jedoch immer klein (siehe Abschnitt 5).

Der Behavioral Compiler ermöglicht die automatische Exploration paralleler Architekturen durch die Angabe entsprechender Synthese-Parameter. Erkennt der BC während des sog. Elaborierens[17], dass eine parallele Abarbeitung von Operationen möglich ist, können so automatisch die entsprechenden Ressourcen alloziert werden.

[15] Ebenfalls interessant erscheint die Berücksichtigung mehrerer Spannungs-Domänen beim Scheduling [183], auch dies ist aufgrund der Limitierungen des BC nicht möglich.

[16] Dies ermöglicht das Absenken der Versorgungsspannung, siehe Abschnitt 2.5.1.

[17] Elaborieren (*elaborate*) bedeutet bei der Synopsys-Synthese das Abbilden von Operatoren auf synthetische Module, siehe Abschnitt 4.5.1

Pipelining

Neben der Parallelisierung ist auch das Pipelining von Komponenten sinnvoll, da es den kritischen Pfad verkürzt und so ein weiteres Absenken der Versorgungsspannung ermöglicht.

Unter Pipelining versteht man das Aufteilen einer einzigen Operation, die in einem Takt ablaufen würde, auf mehrere Teiloperationen, die jeweils einen ganzen Takt benötigen. Die Einzeloperationen werden dabei jeweils über Register von ihrer Nachfolgeoperation getrennt. Bei einem kontinuierlichen Eingangsdatenstrom sinkt der Durchsatz nicht, es dauert lediglich einige Takte (die Anzahl der Pipeline-Stufen) bis das zum aktuellen Eingangsdatum gehörende Ausgangsdatum am Ausgang anliegt. Dies bezeichnet man als Latenzzeit.

Bei einem nichtkontinuierlichen Datenstrom ist Pipelining ungünstig, weil die Pipeline immer wieder neu gefüllt werden muss und der Durchsatz dadurch stark sinkt. In der Signalverarbeitung dominieren jedoch Datenpfad-intensive Verarbeitungsstrukturen, so dass sich ein Pipelining in der Regel positiv auswirkt.

Scheduling und Allokation für Zieltechnologie-Bibliothek

Der Behavioral Compiler erstellt sein Scheduling aufgrund der ermittelten Timing-Information. Dazu werden alle synthetischen Komponenten für die Zieltechnologie synthetisiert (siehe Abschnitt 4.5.1). Dieser Vorgang ist zwar sehr zeitaufwendig, stellt aber sicher, dass eine optimale Anpassung an die Zielbibliothek erreicht wird.

So ist es ohne manuelle Eingriffe möglich, zur funktionalen Verifikation einen FPGA-Prototypen zu generieren, der eine völlig andere Architektur haben kann als der ASIC-Prototyp desselben Algorithmus'.

4.3.2. Optimierung ab der RT-Ebene

Die Optimierung ab der RT-Ebene wird durch den in das Framework eingebundenen PowerCompiler der Firma Synopsys übernommen. Dieser kann aus der Literatur bekannte Optimierungen vornehmen, die im Abschnitt 2.6 beschrieben sind. Dies sind im einzelnen

1. Propagierung von Konstanten
2. Operanden-Isolation
3. Clock Gating
4. Glitch-Optimierung

Aufgrund der Komplexität einer V_{dd}-Skalierung (siehe Abschnitt 2.6.5) kann diese nicht automatisch erfolgen.

4.4. Gütemaß

Ein wichtiger Bestandteil des Frameworks ist die Möglichkeit, die resultierende Audioqualität der Hardware-Implementierung objektiv zu überprüfen. Dies ist sowohl auf der oberen als auch auf den tieferen Entwurfsebenen möglich, da die C++-Prototypen des Frameworks eine bitgenaue Simulation ermöglichen. Es lässt sich damit daher sehr schnell überprüfen, ob eine von der Entwicklerin vorgesehene Quantisierung oder Transformation negative Einflüsse auf die resultierende Signalqualität hat.

Zur Qualitätsbewertung kommen dabei zwei Verfahren zum Einsatz: zum einen das bewährte Maß des Signal-Rausch-Verhältnis' (Abschnitt 4.4.1), das sehr schnell zu berechnen ist und daher für eine erste Abschätzung sehr praktisch ist. Zum anderen ist das von einem Oldenburger Projektpartner entwickelte psychoakustisch valide Maß PSM (Abschnitt 4.4.2) integriert, das genaue Vorhersagen über die von Testhörern empfundenen Qualitätseinbußen gegenüber dem unverzerrten Signal ermöglicht.

4.4.1. Das Signal/Rausch-Verhältnis

Das Signal/Rausch-Verhältnis (engl. SNR = *Signal-Noise-Ratio*) beschreibt das Verhältnis der Leistung $p_r = \sigma_r^2$ des rauschfreien Nutz- oder Referenzsignals r_t mit der Amplitude σ_r von der Leistung p_e des Rauschsignals $e_t = x_t - r_t$ (x_t ist das verzerrte Eingangssignal) auf einer logarithmischen Skala:

$$\mathrm{SNR}(x_t) = 10\ \log_{10}\left(\frac{\sigma_r^2}{\sigma_e^2}\right) = 20\ \log_{10}\left(\frac{\sigma_r}{\sigma_e}\right) \tag{4.1}$$

Zum Vergleich mit einem Referenzsample r wird das Fehler- oder Rauschsignal durch die Differenz mit dem durch die verlustbehaftete Verarbeitung verzerrten Sample x gebildet. Die Maßzahl zur Bewertung der Verzerrung ist dann der Mittelwert über alle Samples (T ist die Gesamtzahl der Samples):

$$\mathrm{SNR} = \frac{20}{T}\sum_{t=0}^{T}\log_{10}\left(\frac{\sigma_r}{\sigma_{(x_t-r_t)}}\right) \tag{4.2}$$

Wenn der Fehler spektral und zeitlich gleichmäßig über das Testsample verteilt ist, stellt das Signal/Rausch-Verhältnis ein valides Maß zur Beurteilung der Signalqualität dar [184]. Dies trifft z. B. auf ein gleichmäßiges Quantisierungsrauschen zu; Berechnungsartefakte wie Überläufe oder Abscheiden können jedoch nicht hinreichend bewertet werden.

4.4.2. Das Gütemaß PSM

Überblick

Das für das Framework verwendete Gütemaß wurde unter Leitung von B. Kollmeier an der Universität Oldenburg entwickelt [68, 69]. Es stellt eine Erweiterung des Sprach-

qualitätsmaßes q_c [63, 64] dar. Dieses basiert auf einem quantitativen psychoakustischen Verarbeitungsmodell des peripheren auditorischen Systems, das von T. Dau entwickelt wurde [33, 34].

Um den subjektiv empfundenen Qualitätsverlust eines verzerrten Signals quantifizieren zu können, wird mit Hilfe des Verarbeitungsmodells eine „interne Repräsentation" sowohl des verzerrten als auch unverzerrten Signals berechnet (Abb. 4.10). Der lineare Kreuzkorrelationskoeffizient zwischen den internen Repräsentationen ist dann das Maß für den subjektiv empfundenen Qualitätsverlust. Das Maß wurde durch ITU- und MPEG-konforme subjektive Hörversuche validiert.

Das Perzeptionsmodell

Kern des Gütemaßes ist das Perzeptionsmodell, das aus einem Audioeingangssignal x_t eine interne Repräsentation X berechnet (Abb. 4.10). Als Eingangssignale kommen dabei zum einen das unverzerrte Referenzsignal zum Einsatz, zum anderen das z. B. durch Berechnung mit endlicher Wortlänge erzeugte, zu bewertende Signal.

Die Verarbeitung beginnt mit einer Aufspaltung des Signals in 33 Frequenzbänder. Als Bandpassfilter kommt dabei eine Gammatonfilterbank mit äquidistanten Mittenfrequenzen auf der ERB-Skala[18] von 235 Hz bis 14500 Hz zum Einsatz, die dem der Charakteristik der Basilarmembran besonders nahe kommt [116][19].

Jedes der 33 Bänder wird jetzt einzeln weiterverarbeitet. Die nächste Stufe modelliert die inneren Haarzellen unter der Basilarmembran, was ungefähr einer Halbwellengleichrichtung (die Haarzellenauslenkung wird nur auf einem Weg in ein Nervensignal umgewandelt) mit nachgeschaltetem 1 kHz-Tiefpass entspricht[20]. Die Grenzfrequenz des Tiefpasses entspricht dabei ungefähr der Frequenz, bis zu der die Haarzellen der Erregung synchron folgen können; danach wird nur noch die Hüllkurve übertragen.

Jetzt erfolgt eine Schwellwertbildung, die auf der einen Seite eine absolute Hörschwelle in das Modell bringt, die nachfolgenden fünf hintereinander geschalteten nichtlinearen Schleifen aber auch gegenüber steilen Eingangssignalen unempfindlich macht. Diese Schleifen, die aus jeweils einer Division und einem Tiefpass bestehen, modellieren zeitliche Maskierungs- Adaptions- und Kontrastierungseffekte entlang der Hörfasern. Die Adaptionsschleifen übertragen schnelle Signaländerungen fast linear, stationäre Signale werden jedoch stark abgeschwächt[21].

Die letzte Verarbeitungsstufe, die Modulationsfilterbank, ist das wesentliche Unterscheidungsmerkmal dieses Perzeptionsmodells von seinen Vorgängern (die an dieser Stelle einen einfachen 8 Hz Tiefpass hatten). Hier kommen pro Kanal acht Modulationsfilter mit Mittenfrequenzen bis zu 129 Hz zum Einsatz.

[18] Die ERB (*Effective Rectangular Bandwidth*) eines Bandpass-Filters ist die Bandbreite eines Rechteck-Filters mit gleicher Maximalverstärkung und Energie der Impulsantwort.

[19] Eine genauere Erklärung der Gammatonfilterbank und eine schematische Darstellung des Innenohrs ist in Kap. 5.2 zu finden.

[20] Die äußeren Haarzellen, die aktive Verstärkungsprozesse bewirken, werden hier nicht berücksichtigt.

[21] Die Ausgangssignal entspricht ungefähr der 32. Wurzel des Eingangssignals; es wird so eine logarithmische Dämpfung approximiert.

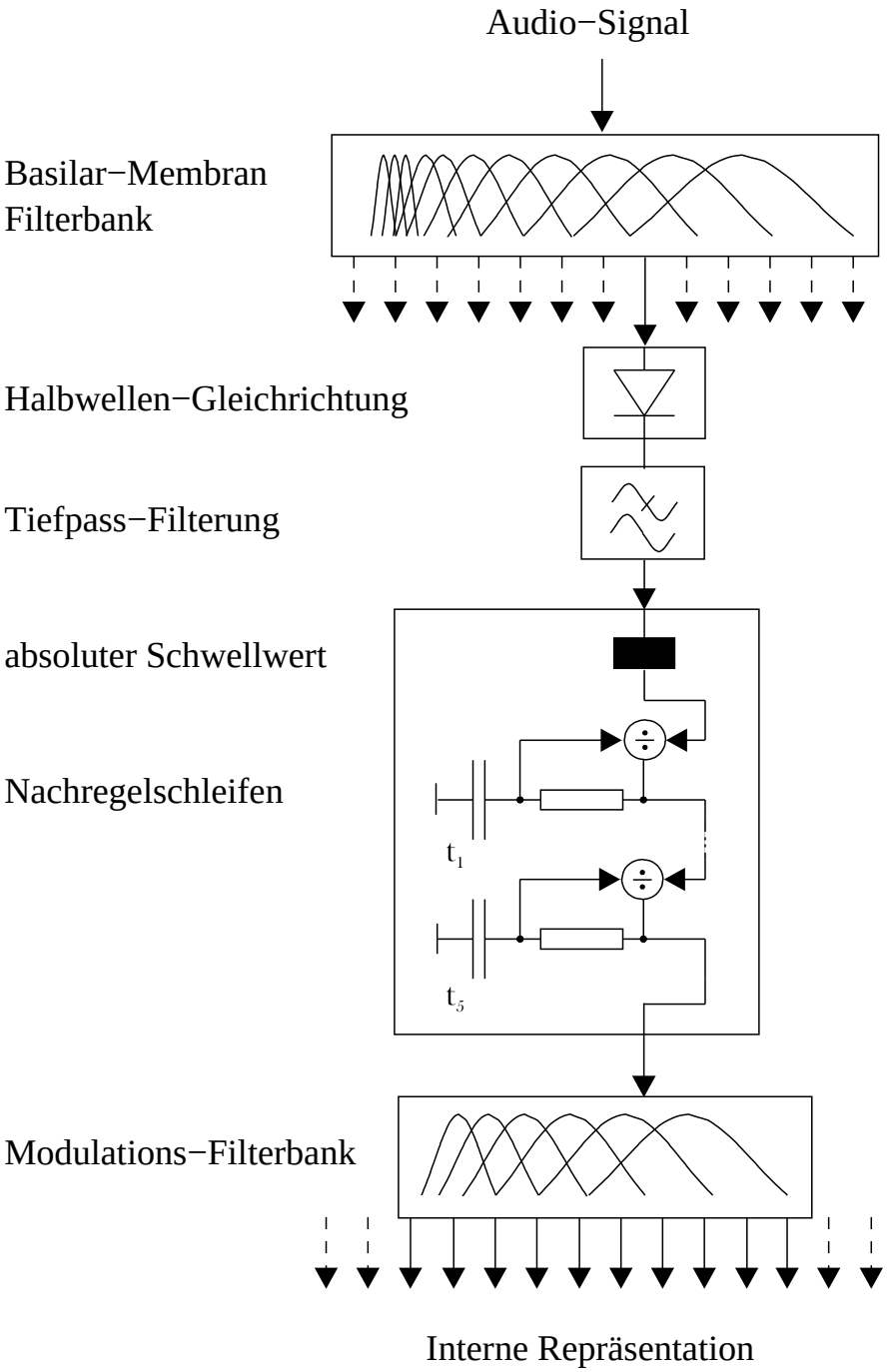


Abbildung 4.10.: Verarbeitung des Audiosignals. Das Audiosignal wird durch eine Gammatonfilterbank in 33 kritische Kanäle aufgespalten. Dies entspricht dem Bandpass-Verhalten der Basilarmembran. Dann folgt eine Halbwellen-Gleichrichtung und ein 1 kHz-Tiefpass, der das Verhalten der inneren Haarzellen modelliert. Der folgende absolute Schwellwert bildet eine Hörschwelle (das leiseste hörbare Signal) ab. Die folgenden Nachregelschleifen (bestehend aus fünf hintereinander geschalteten Dividieren und Tiefpässen) simulieren zeitliche Maskierung und Adaption. Jeder Ausgang durchläuft jetzt noch eine Modulations-Filterbank mit acht Filtern pro Kanal. Das entstehende Signal aus $8 \cdot 33$ Kanälen ist die interne Repräsentation des Perzeptionsmodells (aus [67]).

Die resultierende „interne Repräsentation“ des Perzeptionsmodells ist eine dreidimensionale Matrix $X = (X_{tfm})$, die jedem Zeitpunkt t im Audiostrom x_t einen Punkt im Modulationsfrequenzraum zuordnet, der durch die Modulationsfrequenz m und die Frequenz f des Gammatonfilterbankkanals bestimmt ist.

Um Speicherplatz und Rechenzeit zu sparen, erfolgt eine Unterabtastung der resultierenden Modulationsfilterkanäle mit etwa der sechsfachen Mittenfrequenz des entsprechenden Modulationsfilters.

Ermittlung der Audioqualität mit dem Perzeptionsmodell

Um die empfundene Ungleichheit zweier Audiosignale zu ermitteln, wird die „interne Repräsentation“ Y_{tfm} des verzerrten Signals y_t der internen Repräsentation X_{tfm} des unverzerrten Signals x_t angepasst: Elemente aus Y, die kleiner sind als die korrespondierenden Elemente aus X, werden durch den Mittelwert aus beiden Werten ersetzt:

$$Y_{tfm} = \begin{cases} \big(Y_{tfm} + X_{tfm}\big)/2, & |Y_{tfm}| < |X_{tfm}| \\ Y_{tfm} & |Y_{tfm}| \geqslant |X_{tfm}| \end{cases} \tag{4.3}$$

Dadurch wird der Beobachtung Rechnung getragen, dass fehlende Komponenten in einem verzerrten Signal als weniger störend empfunden werden als additive Komponenten.

Dann wird eine lineare Kreuz-Korrelation durchgeführt, wobei jeder Modulationskanal m einzeln betrachtet wird, so dass bei M Modulationsfilterfrequenzen M zweidimensionale Matrizen $(X_{tf}|_{m=const})$ und $(Y_{tf}|_{m=const})$ übrig bleiben. Diese werden jeweils über eine lineare Kreuzkorrelation miteinander verglichen:

$$r_{m=const} = \frac{\sum_{t,f=1}^{T,F}(X_{tf} - \bar{X})(Y_{tf} - \bar{Y})}{\sqrt{\sum_{t,f=1}^{T,F}(X_{tf} - \bar{X})^2 \sum_{t,f=1}^{T,F}(Y_{tf} - \bar{Y})^2}} \tag{4.4}$$

Dabei ist T die Anzahl der Samples im Audiostrom und F die Anzahl der Gammatonfrequenzkanäle, $\bar{X}$ und $\bar{Y}$ sind die Mittelwerte über den Summenindex.

Um das Gütemaß PSM (*Perceptual Similarity Measure*) zu erhalten, werden jetzt noch die einzelnen Korrelationskoeffizienten r_m mit dem normierten quadratischen Mittelwert des Modulationsfilterbankkanals gewichtet. Damit erhält man das Gütemaß zu

$$\mathrm{PSM} = \sum_{m}^{M} w_m\, r_m \qquad \text{mit} \quad w_m = \frac{\sum_{t,f=1}^{T,F} Y_{tfm}^2}{\sum_{t,f,m'=1}^{T,F,M} Y_{tfm'}^2} \tag{4.5}$$

Das PSM liefert Werte im Intervall $[-1, 1]$, wobei ein Wert 1 der Identität beider Audiosamples entspricht und Werte < 1 Abweichungen vom Referenzsignal beschreiben. -1 wird dabei für komplett unkorrelierte Signale geliefert.

Klassifikation des Gütemaßes: ODG

Die gewonnenen Werte für PSM lassen sich über Hörversuche mit Testpersonen auf eine von der ITU[22] definierte Bewertungsskala abbilden. Die Störungen werden dabei in fünf Stufen klassifiziert (unhörbar; hörbar, aber nicht störend; leicht störend; störend; sehr störend). Die Fit-Skala wurde hier von -4 bis 0 gewählt. Dabei entspricht ein Wert von 0 der Einstufung „unhörbar“ und ein Wert von -4 der Einstufung „sehr störend“. Die Fit-Funktion ergibt sich dann zu

$$\mathrm{ODG}(\mathrm{PSM}) = \begin{cases} \max(-4, \quad a/(\mathrm{PSM} - b) - c) & : \mathrm{PSM} < 0,864 \\ d\,\mathrm{PSM} - d & : \mathrm{PSM} \geqslant 0,864 \end{cases} \tag{4.6}$$

Die sich ergebende Zahl wird ODG (*Objective Differential Grade*) genannt. Die aus den subjektiven Hörversuchen ermittelten Fit-Parameter ergeben sich zu $a = -0,22; \quad b = 0,98; \quad c = 4,13; \quad d = 16,4$.

Die für einen Teil der Versuche verwendete Audio-Testbench in der Version 1 verwendet ODG als Bewertungsskala.

Klassifikation des Gütemaßes: 3-AFC

Ein weiterer Hörversuch, der zur Klassifikation verwendet wurde, ist der 3-AFC-Test (*3 Alternatives Forced Choice*). Die Testperson hört hintereinander drei Audiosamples, von denen zwei identisch sind. Die Testperson muss sich nun entscheiden, welches der drei Samples das verrauschte Sample ist. Der 3-AFC-Wert gibt die Wahrscheinlichkeit an, mit der die Testperson das verrauschte Signal korrekt identifiziert. Eine Wahrscheinlichkeit von 33% markiert dabei die Ratewahrscheinlichkeit und entspricht einem ODG von 0, also unhörbaren Störungen.

Da die Testpersonen keine quantitative Beurteilung des Signals vornehmen, lässt sich der Rest der ODG-Skala nicht ohne weiteres auf den 3-AFC-Test übertragen. Dieser Test eignet sich deshalb besonders, wenn bei einer Hardwareumsetzung keine Einschränkung der subjektiv empfundenen Audioqualität hinnehmbar ist, d. h. die Detektionsschwelle einer verarbeitungsbedingten Verzerrung ermittelt werden soll. Da die 3-AFC-Werte jedoch ebenfalls über den durch das Perzeptionsmodell ermittelten Korrelationskoeffizienten bestimmt werden (durch eine zu Gl. (4.6) analoges Fit-Verfahren), lassen sich die Werte umrechnen. Dieses Klassifikationsmodell wird auch als *PEMO_Q* bezeichnet.

Die für die meisten Versuche verwendete Audiotestbench in der Version 2 verwendet das 3-AFC-Maß als Bewertungsskala.

[22] Die *International Telecommunication Union* ist eine Unterorganisation der UN und beschäftigt sich u. a. mit der weltweiten Normierung im Fernmeldewesen.

4.4.3. Bewertung

Für eine erste Abschätzung der Signalqualität auf der höchsten Entwurfsebene ist das SNR das bevorzugte Maß, da es sehr schnell berechnet werden kann. Hiermit lässt sich sehr schnell überprüfen, ob die gewählte Architektur ein näherungsweise korrektes Ausgangssignal liefert. Insbesondere ist das Maß bei der Festlegung der Berechnungsgenauigkeit oder Quantisierung der einzelnen Funktionsblöcke von Vorteil, da der Entwurfsraum hier sehr groß ist und ein langsames Gütemaß nicht akzeptabel wäre.

Für eine genauere Betrachtung der subjektiv empfundenen Audioqualität kann in einem zweiten Schritt das PSM eingesetzt werden. Dies ist für eine verlässliche Bewertung unumgänglich, da nur dieses Maß eine Aussage darüber ermöglicht, ob und wie stark eine Verzerrung für einen menschlichen Hörer wahrnehmbar ist.

Bei beiden Maßen ist die Auswahl geeigneter („repräsentativer“) Audiosamples für die Bewertung notwendig. Das PSM ist auf einen von der ITU genormten Samplesatz abgestimmt, der auch zur Bewertung von Audio-CODECs wie MPEG Layer 3 zum Einsatz kommt. Dieser Satz wurde auch für die Bestimmung des SNR verwendet.

Die Berechnungszeit für das PSM liegt bei der im Projekt verwendeten Matlab-Implementierung auf einem Pentium-4 mit 2 GHz für ein 30 s langes Sample bei einigen Minuten, während das SNR in einigen Millisekunden berechnet ist. Damit ist das PSM selbst für die skriptgesteuerte Entwurfsraumexploration zu langsam.

4.5. Hardwaresynthese

In diesem Kapitel wird der Hardware-Synthese-Flow des Frameworks beschrieben. Dies ist der Vorgang des Überführens der Verhaltensbeschreibung des Algorithmus in den platzierten und verdrahteten Chip. Der grundsätzliche Ablauf ist in Abb. 4.11 skizziert.

Der Ausgangspunkt ist die BC-VHDL-Beschreibung. Diese wird von der Entwicklerin per Hand aus dem innerhalb des Frameworks optimierten und in C++ spezifizierten Algorithmus erstellt. Die Einbindung von Modulen aus der Framework-eigenen Modulbibliothek erfolgt dabei durch einen einfachen Funktionsaufruf (siehe Abschnitt 4.6.2), so dass im Idealfall (der Algorithmus besteht nur aus einer Verknüpfung von Framework-Modulen) lediglich die externe Schnittstelle des zu implementierenden Chips spezifiziert werden muss.

Das Entwicklungsframework weist gegenüber herkömmlichen Design-Flows folgende Besonderheiten auf:

RT-Synthese mit dem Synopsys Behavioral Compiler (BC) Die etablierten Designflows für die Synthese beginnen auf RT-Ebene. Um vollständig zieltechnologieunabhängig zu sein und eine höheres Optimierungspotenzial nutzen zu können, werden Algorithmen und Module in dem Framework auf Verhaltensebene spezifiziert. Für die automatische Generierung einer RT-Beschreibung wird ein Werkzeug der Firma Synopsys genutzt, das im folgenden Abschnitt 4.5.1 genauer beschrieben wird.

Automatische Einbindung der Framework-Modulbibliothek Die in das Framework integrierte Module können durch eine Integration der Bibliothek in den Synthese-Prozess vom BC automatisch inferiert werden. Dies ermöglicht ein hohes Optimierungspotenzial, ohne dass die Entwicklerin den Quellcode der Module verändern muss. Die Art der Einbindung ist in Abschnitt 4.6.2 beschrieben.

Automatische Power-Optimierung auf RT-Ebene Die Synthese-Skripte binden bei Bedarf den PowerCompiler der Firma Synopsys in die Optimierungszyklen ein. Dies bedeutet, dass zusätzlich zu den klassischen Optimierungszielen Platzbedarf und Geschwindigkeit auch der Energiebedarf des resultierenden Chips optimiert wird. Die Optimierungsfunktionen des PowerCompilers sind in Abschnitt 4.3 näher beschrieben. Da die restliche Synthese ab RT-Ebene dem gängigen Industriestandard entspricht, erfolgt an dieser Stelle keine detaillierte Beschreibung mehr.

4.5.1. Der Synopsys Behavioral Compiler

Um aus einer Verhaltensbeschreibung eine RT-Beschreibung zu generieren, macht der Behavioral Compiler (BC) folgendes (siehe auch Abb. 4.12):

1. Alle inferierbaren Operatoren aus *Synthetic Libraries* (s.u.) werden für die Zielbibliothek synthetisiert, um das Timing zu ermitteln.
2. Ein gültiges *Scheduling* (s. u.) wird für jede Operation erzeugt.

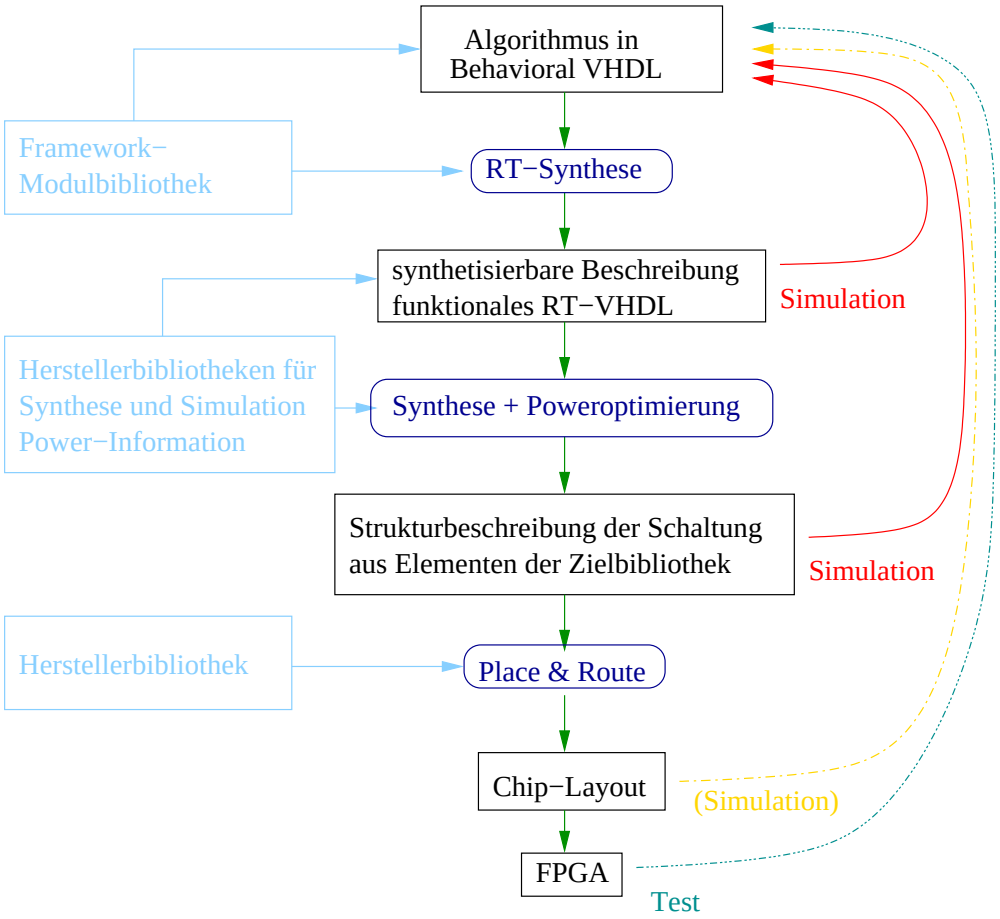


Abbildung 4.11.: Ablauf der Hardwaresynthese. Der Aufbau des Algorithmus in BC-VHDL erfolgt unter Nutzung der Framework-Modulbibliothek. Die automatische Generierung der RT-Beschreibung übernimmt der Behavioral Compiler (BC) von Synopsys. Er liefert eine synthesefähige VHDL-Beschreibung, die in einer Simulation gegen die Verhaltensbeschreibung geprüft wird (roter Pfeil). Bei dem nachfolgenden Erzeugen der Strukturbeschreibung durch den DesignCompiler erfolgt eine automatische Energieoptimierung durch den PowerCompiler. Das resultierende strukturelle VHDL kann wiederum durch eine Simulation verifiziert werden. Beim dann folgenden Platzieren und Verdrahten durch den PhysicalCompiler wird ein gültiger Floorplan erstellt. Optional erfolgt eine Implementierung für ein FPGA, das einen Echtzeit-Test ermöglicht.

3. Dazu werden die benötigen Ressourcen (*Synthetic Modules* und Register) alloziert.
4. Es wird ein Datenpfad mit den allozierten Ressourcen durch Einfügen von Multiplexern und Verbindungsleitungen erstellt.
5. Es wird eine endlicher Automat (FSM, *Finite State Machine*) zur Steuerung des Datenpfades erzeugt.

Eine *Synthetic Library* enthält synthetisierbare Operatoren wie Addition, Multiplikation oder – wie für Module der Framework-Modulbibliothek – Filter- und andere komplexe Signalverarbeitungsoperationen. Weiterhin kann eine *Synthetic Library* auch RAM oder ROM enthalten, das im BC-Quelltext wie ein Array verwaltet wird. Zusätzlich ist zu jedem Operator eine abstrakte Schnittstellendefinition und ein Syntheseskript hinterlegt.

Abbildung 4.13 zeigt die Ein- und Ausgaben des BC im Überblick. Neben dem VHDL-Quelltext benötigt der BC noch die Technologiebibliothek, um das Zeitverhalten der Operatoren auf der Zieltechnologie ermitteln zu können.

Gültige Implementierungen von Operatoren werden aus *Synthetic Libraries* erzeugt. Für das Framework wichtig sind die *Synthetic Libraries* der DesignWare-Bibliothek von Synopsys und die in Abb. 4.13 rot markierten Zusatzmodule. Dies sind die Framework-Module, die auf diese Art automatisch in den Synthese-Prozess eingebunden werden.

Allokation und Scheduling

Einen einfaches Beispiel für Scheduling und Allokation bietet Abb. 4.14. Der BC lässt sich im Synthese-Skript bezüglich des Ergebnisses beeinflussen. Dies wird im Framework genutzt, um eine möglichst starke Parallelisierung des Algorithmus' zu ermöglichen (siehe Abschnitt 5.2).

Eine reales Beispiel ist in Abb. 4.15 wiedergegeben. Hier sollte der BC ein RT-Design für einen digitalen Filter der Form

$$y(n) := b_0x(n) + b_1x(n-1) + b_2x(n-2) + a_1y(n-1) + a_2y(n-2) \tag{4.7}$$

erzeugen. Der Ressourcenplan für komplexe Designs wird sehr groß und unübersichtlich, so dass der graphische Plan bei der Beurteilung nur von eingeschränktem Nutzen ist.

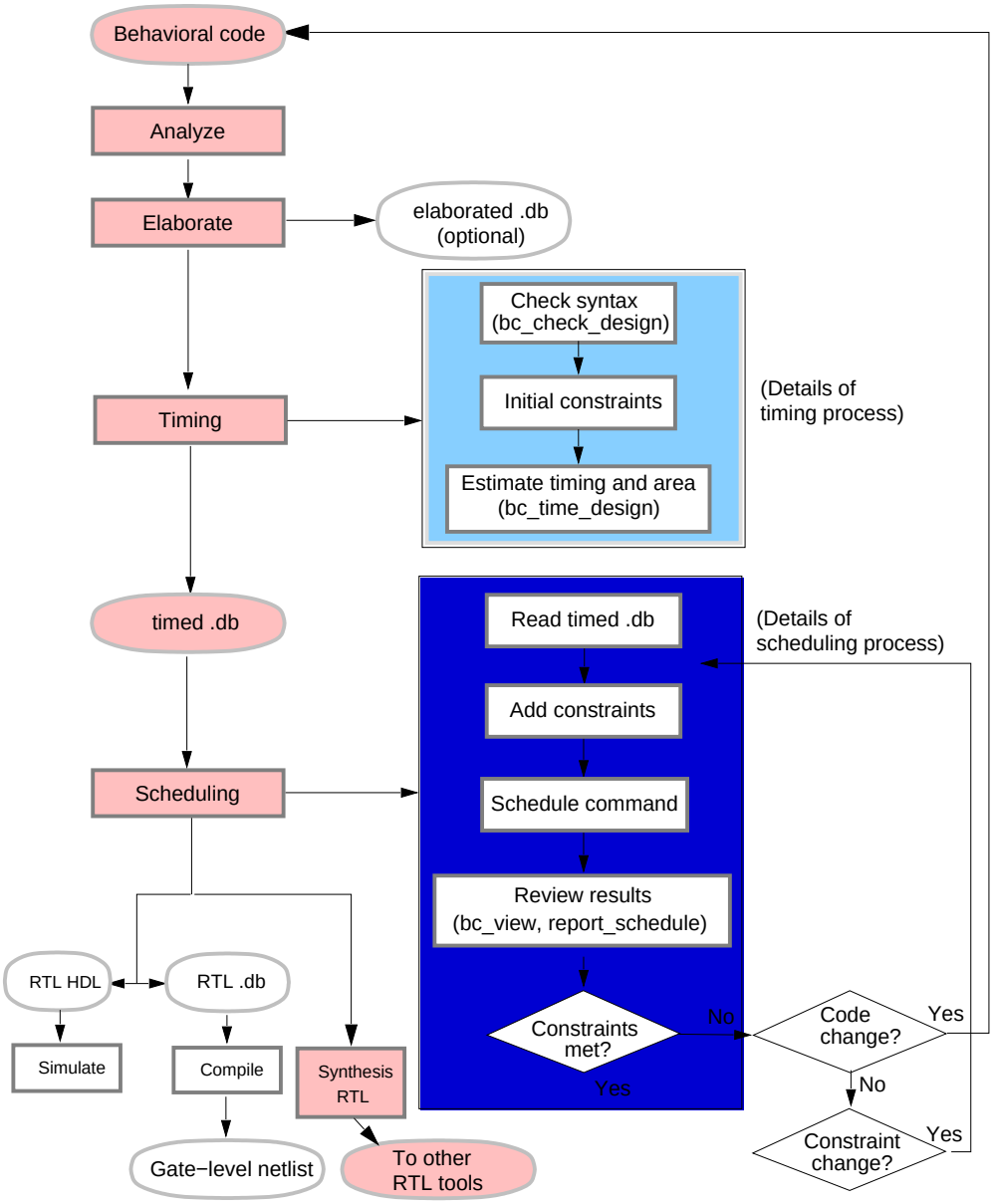


Abbildung 4.12.: Synthese-Ablauf des Behavioral Compilers. Der für das Framework verwendete Pfad ist rot markiert. Der BC-VHDL-Code wird zunächst geprüft („analyze“), danach werden Operatoren auf *Synthetic Modules* abgebildet („elaboration“). Beim „timing“ werden alle Operatoren für die Zielbibliothek synthetisiert, d. h. das Syntheseskript jedes einzelnen benötigten *Synthetic Module* wird ausgeführt. Enthält ein *Synthetic Module* mehrere Implementierungen, werden alle synthetisiert und das mit dem besten Timing wird ausgewählt.
Dann erfolgt die Allokation von Operatoren. Je nach den Vorgaben der Entwicklerin werden ein oder mehrere identische Operatoren erzeugt, an dieser Stelle wird also über eine mögliche Parallelisierung des Designs entschieden. Dann wird ein *Scheduling* erstellt, dies ist ein Plan, welche Ressource wann durch welche Operation oder welches Datum belegt ist (in Anlehnung an [150]).

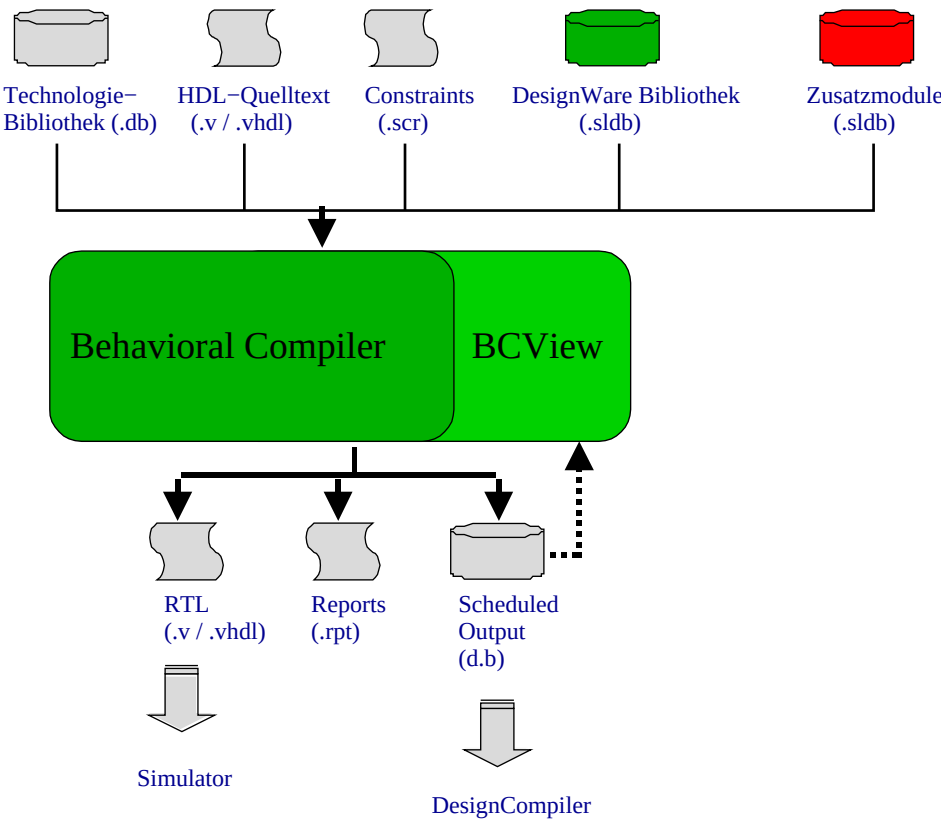


Abbildung 4.13.: Behavioral Compiler Überblick. Der zu implementierende Algorithmus wird als BC-VHDL-Quelltext eingelesen. Alle inferierbaren Operatoren werden aus *Synthetic Libraries* erzeugt, dies sind die DesignWare-Bibliothek für alle Standardoperatoren und die rot markierten Zusatzmodule. Dabei handelt es sich die Framework-eigene Modulbibliothek (in Anlehnung an [150]).

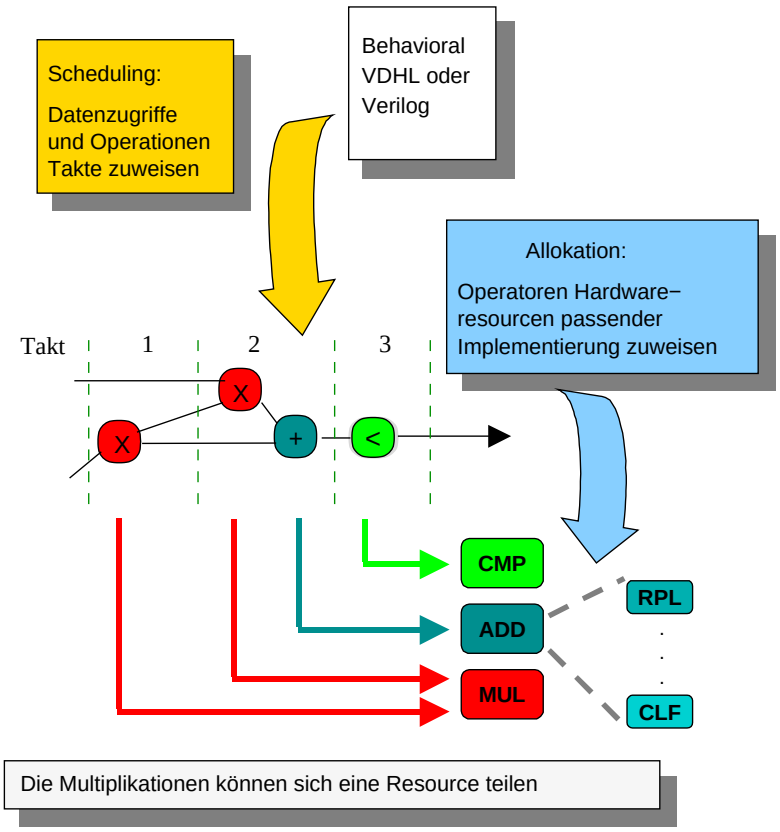


Abbildung 4.14.: Allokation und Scheduling. Dargestellt ist das Scheduling und die Allokation für den Ausdruck $a \cdot b + a \cdot b \cdot c < 0$. Dieses Scheduling benötigt 3 Takte zur Auswertung des Ausdrucks. Dabei gibt es nur einen physikalischen Multiplizierer für die beiden notwendigen Multiplikationen. Unter *Binding* versteht man die Zuordnung von Operation auf Operator (in Anlehnung an [150]).

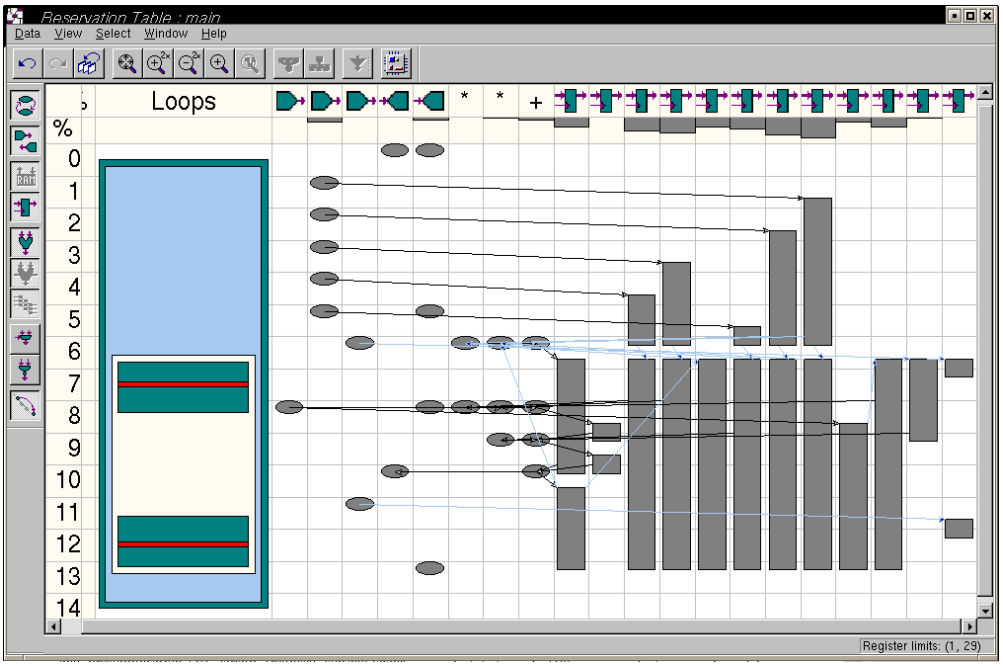


Abbildung 4.15.: Ressourcenplan für ein IIR-Filter zweiter Ordnung nach Gl. (4.7). In der ersten Zeile sind die allozierten Ressourcen aufgelistet; sie beginnt mit Ports, dann folgen die arithmetischen Operatoren und schließlich die Register.
Jede Zeile entspricht einem Takt. Ein Balken in der entsprechenden Spalte bedeutet, dass die Ressource belegt ist. Die ersten 5 Takte nach dem Reset werden benötigt, um die Koeffizienten zu laden (diese sind als Variablen und nicht als Konstanten spezifiziert). Im 6. Takt erfolgen zwei Multiplikationen und eine Addition. Der 7. Takt ist eine Warteschleife (für das Start-Handshake-Signal). Die eigentliche Arbeitsschleife besteht aus den Takten 8-11. Takt 12 ist wiederum eine Handshake-Schleife, die dem Filter das Auslesen des Ergebniswortes signalisiert.

4.6. Modulbibliothek

Die Modulbibliothek gehört zu den Kernbestandteilen des Entwicklungsframeworks [43]. Sie ermöglicht der Entwicklerin, ihr Signalverarbeitungssystem aus möglichst vielen Standardkomponenten aufzubauen, die bezüglich Verlustleistung, Geschwindigkeit und Platzbedarf dem Framework bekannt (vorcharakterisiert) sind. Auf diese Weise ist es möglich, verschiedene Architekturen sehr schnell zu bewerten.

Im Gegensatz zu anderen modulbasierten Ansätzen sind in dieser Modulbibliothek alle Module auf Verhaltensebene spezifiziert und dadurch im Gegensatz zu einer Spezifikation auf RT-Ebene zieltechnologieunabhängig (siehe Abb. 4.16)[23].

Zur Demonstration der Verwendungsmöglichkeiten der Modulbibliothek innerhalb des Entwicklungsframeworks sind in Abschnitt 4.6.4 einige Ergebnisse für einzelne Modulimplementierungen beschrieben. Es handelt sich hierbei nicht um komplette Signalverarbeitungssysteme, sondern um einzelne Filter. Komplette Fallbeispiele finden sich in Kapitel 5.

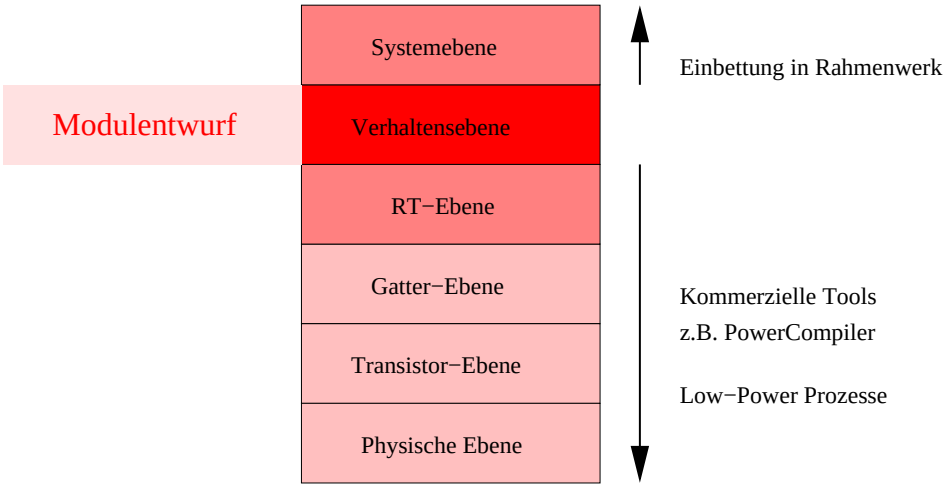


Abbildung 4.16.: Entwurfsebene für die Module. Die Spezifikation auf Verhaltensebene ermöglicht eine maximale Zieltechnologieunabhängigkeit und automatische Architekturoptimierungen und Anwendungsanpassungen auf tieferen Abstraktionsebenen. Jedes Modul kann taktlos in ein System, das mit dem Framework erstellt wird, eingebunden werden. Für die Hardwaresynthese erfolgt die Einbindung über eine generische Handshake-Steuerung, die die Gültigkeit der anliegenden Daten signalisiert und die Verarbeitung anstößt. Die Optimierung auf tieferen Ebenen erfolgt weitgehend automatisch durch kommerziell verfügbare Werkzeuge.

[23] Es können aber beliebige IP-Module eingebunden werden. Dies wurde exemplarische für einige Module aus einer Xilinx-Bibliothek für FPGAs gemacht, siehe Abschnitt 4.7.2

4.6.1. Konzeption und Überblick

Hierarchischer Aufbau

Grundsätzlich ist die Bibliothek hierarchisch aufgebaut, d. h. Module einer höheren Ebene können Module einer tieferen Hierarchieebene frei einbinden (siehe Abb. 4.17). Theoretisch sind viele Hierarchiestufen denkbar, jedoch wird der Entwurfsraum schnell sehr groß und eine schnelle Schätzung erschwert, weil sehr viele Kombinationen überprüft werden müssen[24].

Die für das Framework erstellten Module wurden zwei Hierarchieebenen zugeteilt und werden Highlevel- und Lowlevel-Module genannt. Lowlevel-Module stellen hierbei die zugrunde liegende Arithmetik dar, z. B. Fließkomma- oder Festkomma-Arithmetik, sie implementieren also einen arithmetischen Operator wie z. B. den Additionsoperator „+“. Auf diese Weise ist die Verknüpfung der Module zu realisieren, indem die Operatoren ausgetauscht werden (siehe Kap. 4.6.2).

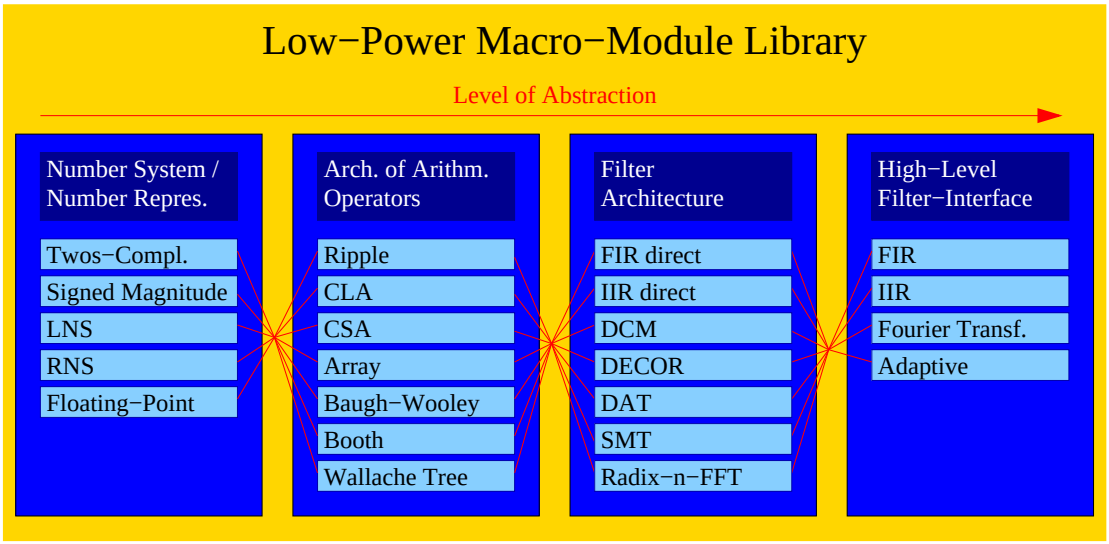


Abbildung 4.17.: Hierarchischer Aufbau der Modulbibliothek. Hier sind vier Abstraktionsebene gezeigt, wobei gegenwärtig nur zwei Stufen realisiert sind: Die oberen beiden Stufen der Abbildung („Filter Architecture“ und „High-Level Filter-Interface“) sind zu High-Level Modules zusammengefasst. Von den unteren beiden Stufen („Number System“ sowie „Architecture of Arithmetic Operators“) ist in der Modulbibliothek nur die unterste implementiert. Die Architektur der Operatoren wird ausschließlich über die Standardimplementierungen der Synopsys DesignWare-Bibliothek implementiert.
Der größte Entwurfsraum und damit das größte Optimierungspotenzial ergibt sich, wenn der Entwickler ein Modul der höchsten Abstraktionsebene auswählt (in der Abbildung rechts), wie z. B. einen allgemeinen FIR-Filter. Der Entwurfsraum spannt sich damit über verschiedene FIR-Architekturen (z. B. Direktform 1-3, siehe Kap. 2.9), die ihrerseits verschiedene Arithmetiken nutzen können, die wiederum auf unterschiedlichen Zahlensystemen aufgebaut sein können.

[24] Aufgrund der gewählten Syntheseeinbindung über *Synthetic Libraries* steigt der Zeitaufwand für die Synthese schnell sehr stark an, siehe auch Abschnitt 4.6.2.

Entwurfsebene

Um die Module möglichst hardwareunabhängig zu halten, wurden sie auf Verhaltensebene, d. h. ohne Festlegung einer Taktung spezifiziert. Die Module sind daher auch zielanwendungsunabhängig, da sie nicht für einen bestimmten Platzbedarf, Durchsatz oder Energiebedarf optimiert wurden, sondern die Optimierung erst während der Simulation bzw. Synthese des Zielsystems stattfindet. Erst hier wird der Grad der Parallelisierung und des Pipelinings sowie der endgültige Takt festgelegt.

Für die Hardwaresynthese wurden die Module in Behavioral-VHDL kodiert. Die Software-Simulation wurde aus Geschindigkeitsgründen mit funktional identischen C++-Modellen durchgeführt. Das Behavioral-VHDL wurde mittels des Behavioral Compilers (BC) (Kap. 4.5.1) in eine RT-Beschreibung umgesetzt[25].

Aufbau eines Moduls

Alle Module bestehen aus zwei Teilen (Abb. 4.18):

Ein bitgenauer Simulationsprototyp, der in C++ programmiert ist, ermöglicht eine schnelle Simulation und Verifikation des zu implementierenden Systems. Anhand einer Charakterisierung des Moduls bezüglich des Energiebedarfs kann schon hier eine vergleichende Energiebedarfsaussage gemacht werden.

Ein synthesefähiger Behavioral-VHDL Prototyp bindet jedes Modul bei Bedarf in den Synthese-Fluss des Frameworks ein. Jeder Prototyp verfügt über eine generische taktlose Schnittstelle, die die Gültigkeit der Daten über Handshake-Signale festlegt.

Experimente und eine Evaluation mit Modulen, die ihren BC-VHDL-Quelltext selbst generieren, wurden im Rahmen der Diplomarbeit von M. Scholz [132] getätigt. Der dort entstandene „DesignTransformer“ war in der Lage, aus einer von Matlab generierten Filterbeschreibung verschiedene Filterarchitekturen als VHDL-Quelltext zu generieren. Obgleich mit dem „DesignTransformer“ gute Ergebnisse erzielt wurden (siehe [168]), ist der Ansatz des Code-Generators im Rahmen der vorliegenden Arbeit nicht mehr weiterverfolgt worden. Der Grund ist die aufwändige Wartung des Code-Generators. Viele Architekturvarianten werden durch die Syntheseskripte exploriert, die ohnehin für jedes Modul erstellt werden mussten. Die zunächst wartungsintensiver erscheinende Variante mit zwei Quellbeschreibungen erwies sich somit als wartungsärmer und von der Struktur klarer.

4.6.2. Integration in das Framework

Die Modulbibliothek ist auf eine Weise in das Entwicklungsframework eingebunden, die sowohl eine bitakkurate Simulation des Gesamtalgorithmus' in C++ und eine Bewertung mit den Audiogütemaßen (Abschnitt 4.4) als auch eine vergleichende energetische Bewertung ermöglicht. Die Energiebewertung wird durch eine Charakterisierung der Module ermöglicht, die im folgenden Kapitel beschrieben wird.

[25] Da der BC z. T. eine sehr ungünstige RT-Struktur erzeugt, wurden einige Lowlevel-Module ausnahmsweise auch in RT-VHDL umgesetzt.

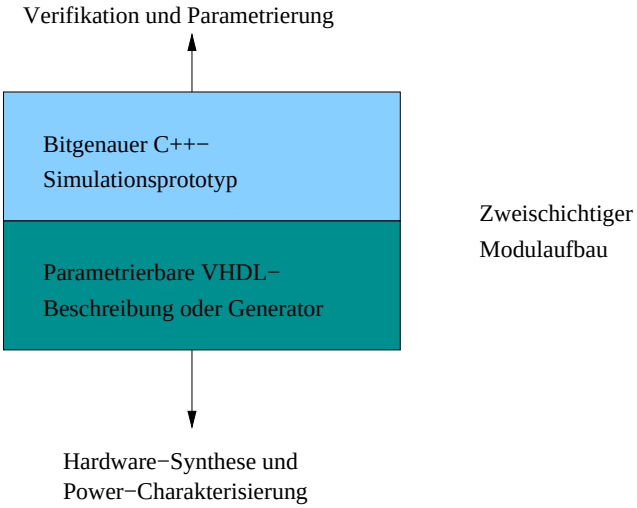


Abbildung 4.18.: Zweischichtiger Aufbau eines Moduls. Jedes Modul hat eine Doppelspezifikation. Auf der einen Seite ist das Modulverhalten bitgenau in einer C++-Beschreibung festgehalten, die der Quantisierung und Verifikation dient. Zusätzlich gibt es eine Behavioral-VHDL Spezifikation für die Hardware-Synthese. Der im Anfangsstadium der Arbeit vorgesehene VHDL-Generator, der als Prototyp eine automatische Generierung von fertig parametriertem VHDL-Code ermöglichte (siehe [168]), wurde konzeptionell nicht weiter verfolgt.

Der Abschnitt danach erläutert die Integration in den Hardware-Synthese-Fluss. Dieser wurde so gestaltet, dass an dem zu jedem Modul gehörenden Behavioral-VHDL-Prototypen möglichst wenig verändert werden muss, um ein optimales Ergebnis zu erhalten.

Exemplarisch verwendete ASIC-Prozesse

Für diese Arbeit wurden zwei über EUROPRACTICE (`http://www.europractice.com`) für angeschlossene Hochschulen verfügbare ASIC-Prozesse verwendet. Beide Prozesse wurden über vom Hersteller erstellte und charakterisierte Standardzellen-Bibliotheken verwendet, d. h. das Design konnte nicht auf Polygon-Ebene optimiert werden, sondern es wurde auf eine optimale Herstellerimplementierung vertraut.

Kurzbeschreibung der Prozesse:

1. Alcatel-Mietec 0.25 μm Prozess [9]. Dieser Prozess ist für eine Versorgungsspannung von 3,3 V spezifiziert. Es ist ein relativ moderner Standardprozess, der aber nicht auf eine niedrige Stromaufnahme optimiert wurde. Gleichwohl haben alle Zellen vom Hersteller eine Energiecharakterisierung, so dass eine entsprechende Optimierung möglich ist. Eine Angabe zur Genauigkeit der Energiecharakterisierung findet sich im Datenblatt nicht.
2. UMC 0.18 μm Prozess [164]. Dieser Prozess ist für eine Versorgungsspannung von 1,8 V ausgelegt. Er ist vom Hersteller als Low-Power-Prozess spezifiziert, verfügt also über entsprechend optimierte Standardzellen. Die Energiecharakterisierung

wurde von UMC besonders sorgfältig durchgeführt, so dass sich laut Datenblatt eine maximale relative Abweichung von 5% für die Zellen-interne Dissipation einhalten lässt, wenn der ASIC innerhalb der spezifizierten Parameter betrieben wird.

Die Schätzung für die durch den Kapazitäts- und Widerstandsbelag der Verbindungsleitungen (*Interconnect*) fußt nicht auf einer Herstellercharakterisierung, weshalb hier eine gewisse Unsicherheit besteht.

Charakterisierung der Module

Alle Module können bezüglich Geschwindigkeit und Platzbedarf sowie bezüglich der Leistung charakterisiert werden. Darunter versteht man zunächst eine Bewertung der Syntheseergebnisse für beispielhafte Parametrierungen der Module, aus denen dann eine möglichst einfache Funktion abgeleitet wird, die eine vergleichende Aussage zwischen verschiedenen Parametrierungen eines Moduls ermöglicht.

Da für diese Arbeit hauptsächlich der Energieverbrauch interessant ist, wurden die anderen Optimierungsziele hier nicht berücksichtigt. Der auf Basis der empirisch ermittelten „charakteristischen" Funktion berechnete Wert ist lediglich ein grober Richtwert, der den mit dem Hardwareentwurf eher unerfahrenen Algorithmen-Entwickler einen Anhaltspunkt geben soll. Um aus einer Verhaltensbeschreibung einen validen absoluten Energiebedarf zu ermitteln, ist ein weitaus größerer Aufwand notwendig. Es gibt viele Werkzeuge, die genau dieses Ziel haben, ein Beispiel ist ORINOCO (siehe Abschnitt 3.8).

Die Charakterisierung innerhalb des Frameworks sei hier noch an einem einfachen Beispiel demonstriert: In Abb. 4.19 ist die Ableitung einer Kostenfunktion für eine Fließkomma-MAC[26]-Einheit dargestellt.

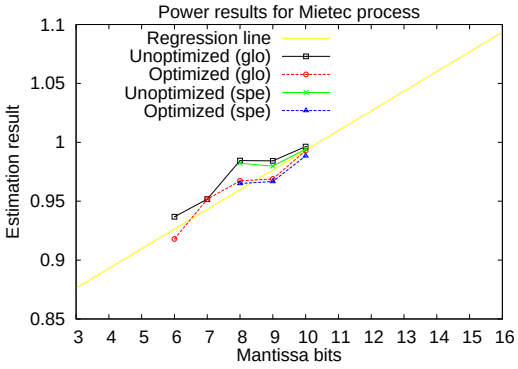
Integration in den Synthesefluss

Alle Module verfügen über eine synthesefähige Behavioral VHDL Beschreibung für die Integration in den Synthesefluss. Da alle Module der Bibliothek Datenfluss-orientiert sind und VHDL keine komfortable abstrakte Schnittstellenbeschreibung bietet, die eine transparente Einbindung der Module in ein Gesamtsystem ermöglichen würde, sind die Module hier in eine *Synthetic Library* eingepflegt worden.

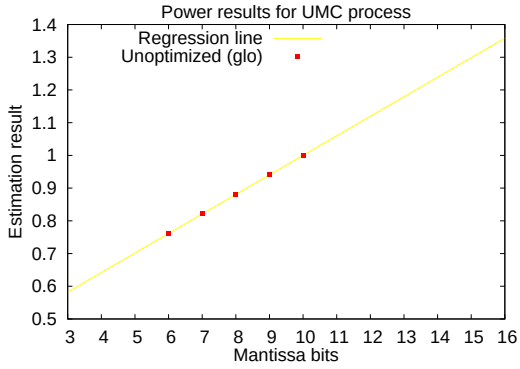
Diese ergänzt die HDL-Beschreibung der Module um eine abstraktere Schnittstellenbeschreibung, in der Handshake-Signale und Operandenbreiten festgelegt werden. Jedes Modul wird als Operator aufgefasst, der auf einem oder mehreren Operanden arbeitet und ein Ergebniswort zurück liefert, dessen Breite von den Eingangsoperanden abhängt. Erkennt der Compiler während des Compile-Vorgangs ein solche Operation, kann das Modul automatisch *inferiert* werden, d. h. der zugehörige *Synthetic Operator* wird automatisch mit den nötigen Operandenbreiten synthetisiert.

Jeder *Synthetic Operator* ist zunächst nur in seiner Funktion, der Operation und seiner Schnittstelle definiert, bezüglich seinem Taktverhalten aber abstrakt. Hinter jedem

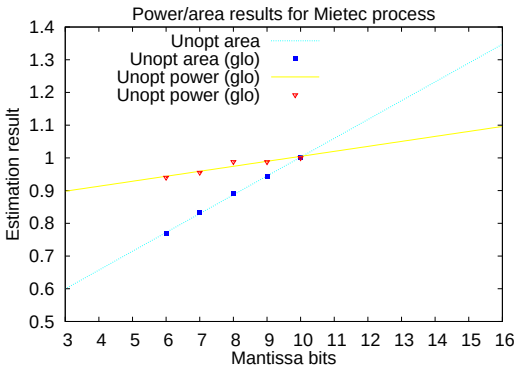
[26] Multiply and Accumulate



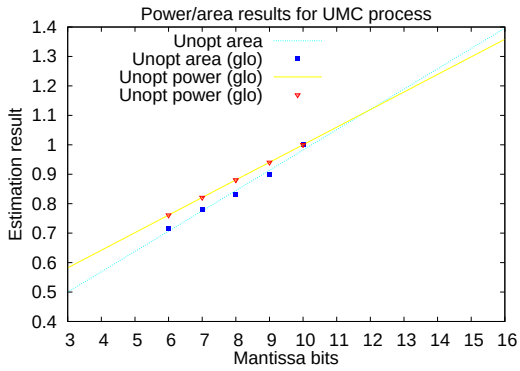
(a) Leistungsbedarf Mietec-Prozess



(b) Leistungsbedarf UMC-Prozess



(c) Leistungs-/Flächenbedarf Mietec-Prozess



(d) Leistungs-/Flächenbedarf UMC-Prozess

Abbildung 4.19.: Charakterisierung des Fließkomma-MAC-Moduls. Dargestellt ist die relative Leistung und der Flächenbedarf für zwei exemplarische Standardzellprozesse (Mietec: 0,35 μm, UMC: 0,18 μm). Der Flächenbedarf wurde mit dem Synopsys PhysicalCompiler für die platzierte und verdrahtete Netzliste ermittelt, der Energiebedarf mit Synopsys PrimePower für die gleiche Netzliste unter Verwendung verschiedener Audiosamples der Audiotestbench als Eingangsdaten.
Variiert wurde hier die Anzahl der Bits der Mantisse (Abszisse). Es lässt sich ein linearer Zusammenhang erkennen, der durch eine Fit-Funktion approximiert wird (in diesem Falle eine Regressionsgerade). Es ist klar erkennbar, dass die beiden Prozesse schon bei dieser einfachen MAC-Funktion sehr stark variieren. Dies kann neben der unterschiedlichen Physik der Prozesse verschiedenste Ursachen haben: Bibliotheken mit einer großen Anzahl von Zellen bergen ein höheres Optimierungspotential, ebenso Bibliotheken, die eine sehr genaue Leistungscharakterisierung haben. Bei letzteren sind zusätzlich die Vorhersagen präziser. Die UMC-Bibliothek ist eine Low-Power-Bibliothek, die sehr genau charakterisiert ist.

Synthetic Operator können sich daher verschiedene Hardware-Architekturen verbergen (z. B. verschiedene FIR-Architekturen) hinter denen wieder verschiedenen Implementierungen, z. B. der Addierer stecken können (siehe auch Abb. 4.20). Der Behavioral-Compiler synthetisiert zur Compile-Zeit alle möglichen Varianten und wählt dann diejenige aus, deren Eigenschaften den gesetzten Synthese-Bedingungen am nächsten kommt[27].

Den oben beschriebenen Mechanismus hat Synopsys geschaffen, um zieltechnologieunabhängige Module für Basisoperationen wie Addition und Multiplikation benutzen zu können. Er ist aber so allgemein gehalten, dass er sich für beliebige Operationen einsetzen lässt. In Abbildung 4.20 ist der Mechanismus, der auch hinter der Synopsys DesignWare-Bibliothek steckt, am Beispiel einer Multiplikationsoperation erläutert.

4.6.3. Beschreibung der implementierten Module

Hier folgt eine knapp gehaltene Auflistung der implementierten Module. Im ersten Teil sind die High-Level Module beschrieben, das sind solche Module, die auf Low-Level Modulen aufbauen können.

Im zweiten Teil erfolgt eine Beschreibung der implementierten Low-Level Module.

Eine endgültige Bewertung und Ergebnisse zu dem Energieeinsparpotentialen der Module finden sich in Abschnitt 4.6.4.

Digitale Filter

Von besonderer Bedeutung für die digitale Signalverarbeitung sind Filter, die zu jedem Eingangsdatum $x[n]$ ein Ausgangsdatum $y[n]$ erzeugen [179]. Die zugrunde liegende Theorie ist in Abschnitt 2.9 beschrieben.

Für das Framework wurde neben den aus der Grundlagenliteratur bekannten Filterarchitekturen *Direktform 1+2* und *Transponierte Direktform* auch besonders auf Energiesparpotenzial optimierte Architekturen implementiert. Diese versuchen, durch *Dekorrelation* (s.u.) der Filterkoeffizienten die Filterstruktur zu vereinfachen. Aufgrund der großen Bedeutung digitaler Filter für die Signalverarbeitung ist die Literatur sehr umfangreich. Im Zuge dieser Arbeit wurden viele Verfahren evaluiert und auf ihre Eignung für das Framework geprüft. Im folgenden werden die Module beschrieben, die aus den zur Verfügung stehenden ausgewählt wurden[28].

Das grundsätzliche Optimierungsziel bei der Architekturauswahl ist, neben der strukturellen Vereinfachung der Filter, die zu einer Reduktion des arithmetischen Aufwands führt, die *Verkürzung des kritischen Pfades* bei *gleichzeitiger Erhöhung des Filterdurchsatzes*

[27] Bei komplexen Modulen (die auch wieder synthetische Operatoren enthalten können) wird der Entwurfsraum sehr groß, daher kann der BC Tage damit beschäftigt sein, alle möglichen Architekturen zu synthetisieren. Mann kann daher über Zusatzbedingungen festlegen, welchen Weg der BC durch den Suchraum geht.

[28] Eine vollständige Beschreibung würde den Rahmen dieser Arbeit sprengen. Zu den untersuchten Verfahren gehören u. a. auch [10, 136]

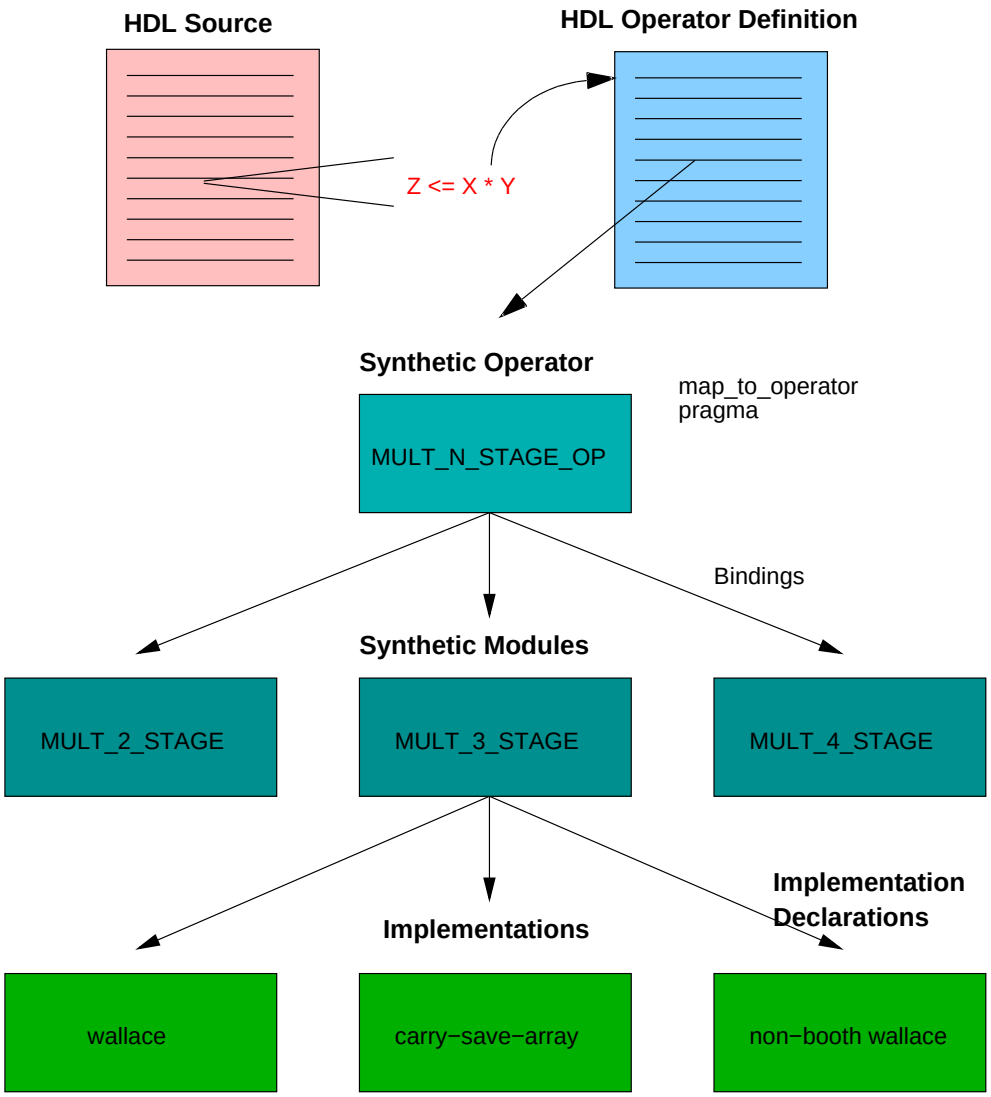


Abbildung 4.20.: Integration des Moduls in das Framework am Beispiel einer Multiplikation. Der Ablauf ist folgender: Der Compiler erkennt eine Operation, für die es eine Deklaration in einer *Synthetic Library*, einen *Synthetic Operator* gibt. Dieser kann Bindungen auf ein oder mehrere Module haben, die das Operatorverhalten auf verschiedene Art und Weise implementieren. Diese Module haben jeweils eigene Syntheseskripte. In diesem Beispiel handelt es sich um Pipeline-Multiplizierer verschiedener Pipelinetiefe. Diese Module können wiederum in verschiedenen Implementierungen vorliegen, in diesem Fall einige aus der Literatur bekannten Multiplizierer-Architekturen.

pro Takt, d. h. pro Taktzyklus ist eine minimale Anzahl von Gattern hintereinander geschaltet. Das ermöglicht die Reduktion der Taktfrequenz und damit u.U. das Absenken der Versorgungsspannung (siehe Abschnitt 2.5).

Direktform und Transponierte Direktform

Diese Architekturen und ihre theoretischen Auswirkungen auf den kritischen Pfad sind im Kapitel 2.9 beschrieben, insbesondere in den Abbildungen 2.17, 2.18 und 2.31.

Symmetrische (Transponierte) Direktform

Da aufgrund ihrer besseren Quantisierungseigenschaften und linearer Phase oft FIR-Filter zum Einsatz kommen, die üblicherweise symmetrische Koeffizienten b_i haben, kann dieser Umstand ausgenutzt werden, um die Anzahl der Multiplizierer zu verringern.

Haben wir also einen FIR-Filter mit N symmetrischen Koeffizienten, also $b_{N-i} \equiv b_i$ so können wir die Filtergleichung

$$y[n] = \sum_{i=0}^{N} b_i x[n-i] \tag{4.8}$$

auch schreiben als

$$\begin{aligned} y[n] &= \sum_{i=0}^{N/2-1} b_i \left(x[n-i] \pm x[n-N+i]\right), N \text{ ungerade} \\ y[n] &= b_{\frac{N}{2}} x\left[n - \frac{N}{2}\right] \\ &\quad + \sum_{i=0}^{N/2-1} b_i (x[n-i] \pm x[n-N+i]), N \text{ gerade} \end{aligned} \tag{4.9}$$

Dieser Filter hat jetzt nur noch halb so viele Multiplikationen, da zwei gleiche Koeffizienten $b_{i+N/2} \equiv b_i$ nur noch einen Multiplizierer benötigen. Diese Filterarchitektur ist im Vergleich zu anderen Direktformfiltern in Abb. 4.21 dargestellt.

Energiespartransformationen für digitale Filter: Dekorrelation

Um eine maximale Einsparung zu erzielen, ist es zunächst immer notwendig, den Filter auszurollen, d. h. alle Einzelprodukte parallel zu berechnen. Dies geht zwar zu Lasten des Flächenbedarfs, ermöglicht jedoch eine vergleichsweise geringe Taktfrequenz. Da die Filterkoeffizienten oft konstant sind[29], lassen sich die Multiplizierer weiterhin in

[29] Dies ist bei *dynamischen* oder *adaptiven* Filtern nicht mehr der Fall. Ein Beispiel dafür ist das im folgenden noch angesprochene DAT-Modul.

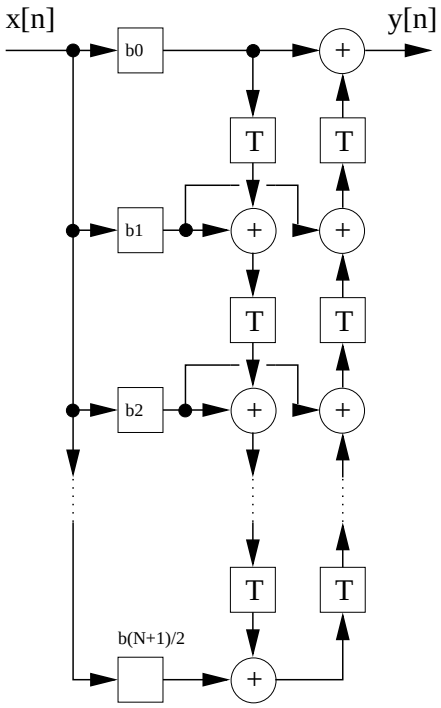


Abbildung 4.21.: FIR-Filter in transponierter Direktform 2 mit symmetrischen Koeffizienten. Hier wurde der Umstand ausgenutzt, dass FIR-Filter symmetrische Koeffizienten haben können, deshalb ist für zwei gleiche Koeffizienten nur ein Multiplizierer notwendig.

einfache Addiererstrukturen überführen, d. h. eine Multiplikation mit einer Konstante mit Hamming-Gewicht g kann durch g Additionen und Verschiebeoperationen der Faktoren ersetzt werden[30].

In [35] konnten die Autoren nur durch diese Maßnahme für zwei exemplarische Filter eine (theoretische) Energieeinsparung um 99% erzielen[31].

Die im folgenden vorgestellten Transformationen basieren auf einer Reduktion des oben erwähnten Hamming-Gewichts g der konstanten Koeffizienten, was die Multiplizierer weiter vereinfacht. Diese Technik wird in der Literatur als *Dekorrelation* bezeichnet.

DCM: Differential Coefficients Method

Sankarayya et al. haben 1997 die DCM vorgestellt [131], welches die Multiplikation mit einem Koeffizienten in eine Multiplikation mit einem Differenzen-Koeffizienten und eine Korrektur-Addition überführt. Die Koeffizienten

$$b_k^n \equiv b_k^{n-1} - b_{k-1}^{n-1} \tag{4.10}$$

[30] Das Hamming-Gewicht ist der Hamming-Abstand vom Nullvektor, also die Anzahl der von Null verschiedenen Bits.

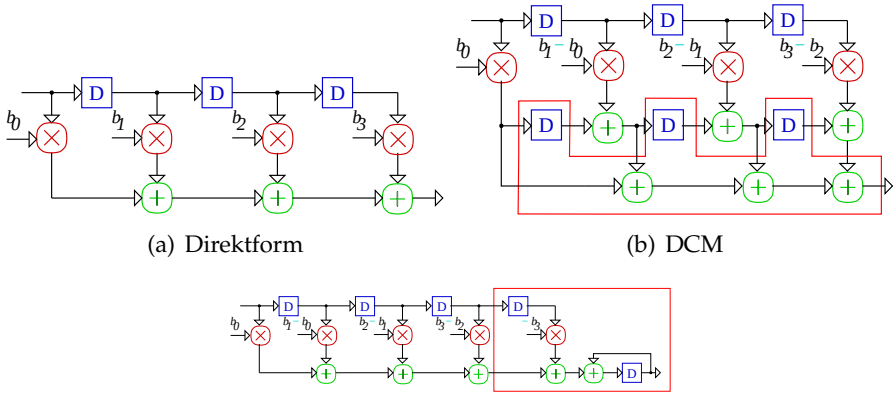
[31] Basis solcher Energieeinsparpotenziale, die in der Literatur sehr häufig zu finden sind, ist immer eine durch Parallelisierung und Verkürzung des kritischen Pfades begründete extrapolierte Absenkung der Versorgungsspannung. In der Praxis sind solche Werte für die beschriebenen Architekturen in der Regel nicht vollständig nachvollziehbar (siehe auch Abschnitt 4.6.4).

ergeben dann folgende Filter-Formel für die Dekorrelation erster Ordnung:

$$\begin{aligned} y[n] &= \sum_{i=0}^{N} b_i x[n-i] \\ &= b_0 x[n] + \sum_{i=1}^{N} (b_i - b_{i-1})\, x[n-i] + b_{i-1} x[n-i] \end{aligned} \tag{4.11}$$

Größter Nachteil sind die $N-1$ zusätzlichen Addierer und Register, die den Vorteil in der Praxis meist aufheben (siehe Abschnitt 4.6.4).

Die resultierende Filterstruktur ist in 4.22(b) dargestellt.



(a) Direktform (b) DCM

(c) DECOR, $\alpha = -1$, $\beta = 1$, $m = 1$

Abbildung 4.22.: Verschiedene Implementierungen eines FIR-Filters 4. Ordnung. Zusätzliche arithmetische Einheiten und Register gegenüber der Direktform sind durch den Kasten markiert.

DECOR

Ein allgemeinerer Ansatz wurde 1999 von Ramprasad et al. [122] vorgestellt. Dieser DECOR genannte Ansatz transformiert die Koeffizienten in eine günstigere Form als DCM und kommt mit weniger Zusatzaufwand aus.

Die Idee ist, die z-Übertragungsfunktion des Filter mit eins zu multiplizieren:

$$Y(z) \equiv H(z) \frac{(1+\alpha z^{-\beta})^m}{(1+\alpha z^{-\beta})^m} X(z) \tag{4.12}$$

α, β und m hängen vom Filtertyp ab. In [122] finden sich Empfehlungen für gängige Filter wie Hochpass, Tiefpass usw. Die Struktur des Filters ist in 4.22(c) dargestellt. Die Multiplikation im Frequenzraum erzeugt eine zusätzliche rekursive Stufe, d. h. FIR-Filter werden immer in IIR-Filter umgewandelt. Dadurch kann es bei Berechnungen mit reduzierter Genauigkeit leicht zu einem kumulierten Fehler kommen, der zu zusätzlichem Rauschen, Instabilität und Oszillation des Filters führen kann. Die Autoren empfehlen als Gegenmaßnahme die Verwendung von Sättigungs-Addierern, die das Problem teilweise entschärfen können.

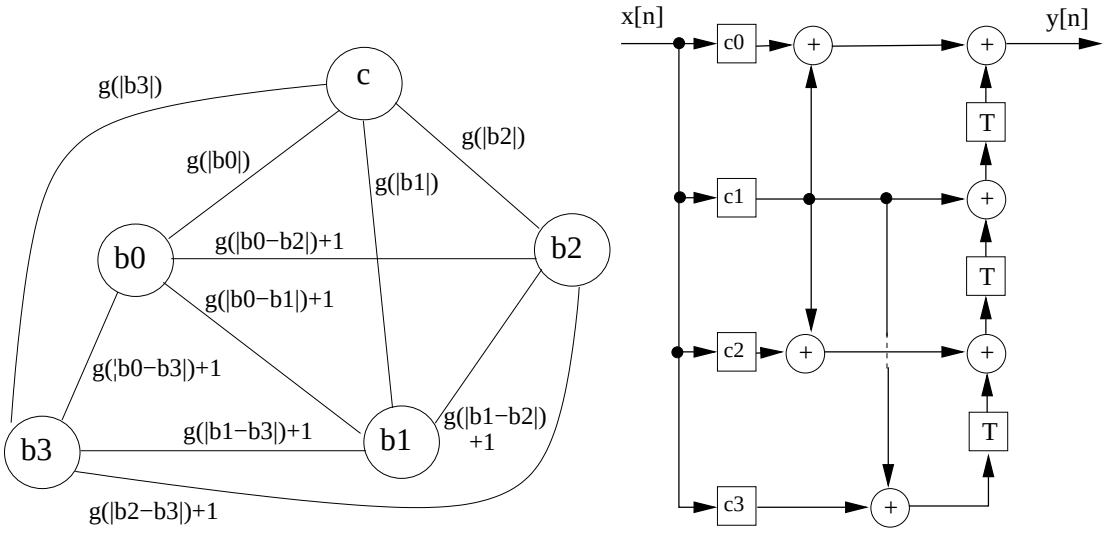
Der Zusatzaufwand der Transformation ist $2\beta m$ für Verzögerung und zusätzliche Addierer sowie βm für zusätzliche Multiplizierer. Wenn $|\alpha| = 1$ gewählt wird, werden um β entfernte Koeffizienten entweder addiert oder subtrahiert.

HODCMI: 01-Hybrid Optimal Differential Coefficients for Multiplier-less Implementation

DCM und DECOR werden gleichförmig auf das gesamte Filter angewendet. Dies erzeugt eine regelmäßige Struktur, führt jedoch mit einer sehr geringen Wahrscheinlichkeit zu einer optimalen Dekorrelation.

In [106] wird von K. Muhammad et al. aus der Arbeitsgruppe von K. Roy, die sich schon sehr lange mit der Low-Power-Optimierung von FIR-Filtern befasst [105, 104], eine Methode zur maximalen Dekorrelation von FIR-Filter-Koeffizienten vorgestellt, die in [107] und [27] weiter verfeinert wurde.

Abbildung 4.23 zeigt das Prinzip: Für jede mögliche Kombination von Koeffizienten und Differenzenkoeffizienten werden die Kosten ermittelt. Mit einem graphentheoretischen Ansatz wird dann ein aufspannender Baum minimaler Kosten erzeugt.



(a) Graph für HODCMI-Koeffizienten (b) 01-hybrid ODCMI Struktur

Abbildung 4.23.: Überblick über die ODCMI-Methode. Die Kanten in (a) werden mit den Kosten der Multiplikation mit allen Koeffizienten und Differenzenkoeffizienten inklusive der Korrektur-Addition annotiert. Dann wird ein aufspannender Baum mit minimalen Kosten extrahiert und in eine Filterstruktur analog zu (b) umgewandelt.

Dies kann mathematisch wie folgt beschrieben werden:

$$\begin{aligned} y[n] &= \sum_{i=0}^{N} b_i x[n-i] = \sum_{i=0}^{N} T_i\{b_i x[n]\} \\ &= \sum_{i=0}^{\tilde{N}-1} T_{p(i)}\{b_i x[n]\} + \sum_{i=\tilde{N}}^{N} T_{p(i)}\left\{b_{p(i)} x[n]\right\} \\ &= \sum_{i=0}^{\tilde{N}-1} T_{p(i)}\{b_i x[n]\} \\ &\quad + \sum_{i=\tilde{N}}^{N} T_{p(i)}\left\{\left(b_{p(i)} - b_{q(i)}\right) x[n] + \underbrace{b_{q(i)} x[n]}_{\text{Korrekturaddition}}\right\} \end{aligned} \tag{4.13}$$

$T_i\{x\}$ ist eine Verzögerung des Signals x über i Zeit-Einheiten, $p(i)$ und $q(i)$ sind Permutationen. Die Anzahl der Register und Multiplizierer erhöht sich nicht, es sind aber zusätzliche Addierer nötig.

Add-Overlap FFT

Anstelle einer Filterung im Zeitbereich ist es u.U. günstiger, die Filterung im Frequenzbereich durchzuführen. Der resultierende Energieunterschied in Hardware hängt von zahlreichen Faktoren wie Filterordnung, Quantisierungsgüte und Fenstergröße ab und soll hier nicht näher betrachtet werden. Eine ausführliche Erläuterung findet sich z. B. in [165, 82].

Da bei der Fouriertransformation (siehe Abschnitt 2.9.6) immer Blöcke einer bestimmten Länge verarbeitet werden müssen, ist das Zeitverhalten gänzlich anders als bei der Verarbeitung im Zeitbereich.

Das bedeutet konkret, dass der Signalverarbeitungsalgorithmus üblicherweise stark überarbeitet werden muss, um akzeptable Ergebnisse zu erzielen. Das Add-Overlap-FFT Modul ist daher kein einfacher Ersatz für die Filterung im Zeitbereich, sondern stellt eine Ergänzung der Modulbibliothek dar, um auch auf der FFT basierende Systeme mit dem Framework umsetzbar zu machen.

Eine beispielhafte Implementierung wurde von Felix Klöckner im Rahmen seiner Diplomarbeit angefertigt und wird in Abschnitt 5.3 kurz beschrieben.

Weitere High-Level-Module

Der Vollständigkeit halber folgt eine Auflistung der weiteren implementierten Module.

Für alle Module wurde ein C++-Prototyp erstellt und getestet. Anhand der Ergebnisse wurde entschieden, ob die Entwicklung einer Referenzhardware für dieses Modul lohnenswert erscheint. Bei den folgenden High-Level-Modulen wurde von der Erstellung

einer Referenzhardware abgesehen, die einzige Ausnahme ist die Integer-Split-Radix-FFT. Die Begründung findet sich in der folgenden Aufstellung.

Dynamic Algorithm Transformations (DAT) Hierbei handelt es sich um einen dynamischen FIR-Filter, der seine Koeffizienten mittels eines zweiten, einfacheren Referenz-Filters (mit kürzerem kritischen Pfad) zur Laufzeit anpasst und dabei so viele Filterstufen wie möglich abschaltet. Das Prinzip ist in [53, 54] beschrieben. Das Modul wurde in C++ implementiert und ist im Framework verwendbar. Von einer VHDL-Implementierung wurde aufgrund des hohen Aufwands für den SMA-Block[32] abgesehen. Wegen der dynamischen Koeffizienten sind bei einem ausgerollten Filter zudem sehr viele Multiplizierer erforderlich, was den Platz- und Energiebedarf gegenüber einer statischen Implementierung stark erhöht. Das Prinzip lohnt sich daher nur bei Filtern sehr hoher Ordnung, die eingerollt, u.U. in Fließkomma-Arithmetik, implementiert werden.

Cache-Multiplier Dieser spezielle Multiplizierer nutzt die Korrelation von Audiodaten, speziell die Tatsache, dass die höherwertigen Bits eine wesentlich geringere Schaltwahrscheinlichkeit haben als die niederwertigen Bits (siehe auch Abb. 4.26). Die Multiplikation wird dabei in einen oberen und unteren Teil zerlegt, wobei die Ergebnisse des oberen Teils zwischengespeichert werden und nur bei Bedarf (Cache-Miss) eine Neuberechnung erfolgt. Das Prinzip ist (allerdings für Videodaten) in [62] beschrieben, die vorausgesagte Energieeinsparung mit 10% aber eher gering. Aufgrund des hohen Aufwands für die Cache-Verwaltung (einfach assoziativ) wurde von einer VHDL-Implementierung abgesehen.

ANT-Filter *Algorithmic Noise-Tolerance*-Filter versuchen die (groben) Berechnungs-Fehler eines FIR-Filters aufgrund einer mit der V_{dd}-Absenkung einhergehenden Laufzeitverlängerung durch ein einfaches Fehlermaß zu korrigieren. Die in [172] ermittelte theoretische Energieeinsparung liegt bei ca. 40%. Aufgrund der in Abschnitt 2.6.5 erwähnten Probleme bei der quantitativen Abschätzung einer Energieeinsparung durch eine V_{dd}-Skalierung wurde von einer VHDL-Implementierung abgesehen.

Integer-Split-Radix-FFT Bei diesem Modul, für das sowohl eine C++- als auch eine VHDL-Implementierung existiert, handelt es sich um eine auf Ganzzahlen basierende Frequenztransformation, die eine Split-Radix-FFT approximiert. Die Implementierung orientiert sich an [112]. Die *Twiddle*-Faktoren (siehe Abschnitt 2.9.6) werden dabei zu Ganzzahlen konvertiert, ohne dass ihre Inversionseigenschaften verlorengehen. Die Komplexität der Multiplikation eines komplexwertigen Twiddle-Faktors kann ebenfalls reduziert werden[33]. Um die Auflösung einer 16-bit FFT zu erzielen, wird eine interne Genauigkeit von 27-bit für die Integer-FFT benötigt. Allerdings fällt bei der Integer-FFT dann *kein* Quantisierungsfehler mehr an. Dies wirkt sich bei einer Länge von 256 Punkten aber noch nicht ent-

[32] Der *Signal-Monitoring-Block* prüft das durch den *Signal-Processing-Algorithm*-(SPA)-Block gefilterte Signal auf die geforderten Eigenschaften (SNR und Energieverbrauch) und nimmt ggf. Anpassungen an Koeffizienten vor.

[33] Normalerweise werden für eine komplexe Multiplikation vier reelle Multiplikationen benötigt, durch die Transformation nur noch drei, die vierte Multiplikation wird in eine Addition überführt, so dass dann statt zwei Additionen drei nötig sind.

scheidend auf den Rauschabstand aus, da der kumulierte Quantisierungsfehler auch bei der Festkomma-FFT noch gering ist.

Bei einem Vergleich mit einer für die FPGA-Prototyping-Hardware optimierten Radix-4 FFT konnte in einer Simulation der Schaltaktivität der FPGA-Zielprimitiven eine Verschlechterung um ca. 50% festgestellt werden, das Modul wird damit seinen Erwartungen leider nicht gerecht[34]. Möglicherweise wirkt sich das Prinzip bei längeren FFTs positiv aus.

PrimeSort Bei dieser in [109] vorgestellten Filter-Transformation werden die Filterkoeffizienten eines FIR-Filters so gewählt, dass sie Produkte der ersten sieben Primzahlen sind. Auf diese Weise lassen sich Zwischenergebnisse weiterverwenden, was zu einer Einsparung von ca. 40% der arithmetischen Operationen führen kann. Ein Problem ist allerdings, dass die Zwischenergebnisse abgespeichert werden müssen, was zu zusätzlichem Aufwand führt. Die Filterkoeffizienten müssen so angepasst werden, dass sie die obige Bedingung erfüllen. Weiterhin müssen die faktorisierten Koeffizienten so sortiert werden, dass möglichst viele die gleiche Anordnung von Faktoren haben, um die Zwischenergebnisse für möglichst viele Operationen nutzen zu können.

Diese Probleme, die bei der Evaluation des C++-Prototypen zu Tage getreten sind, haben dazu geführt, dass für diesen FIR-Filter kein VHDL-Prototyp erstellt wurde.

Low-Level Module

Bei den Low-Level Modulen handelt es sich um Implementierungen von Zahlensystemen. Die Theorie ist in Abschnitt 2.8 beschrieben. Für die in der Hardwareentwicklung dominierende Festkommadarstellung wurden zwei Module implementiert:

Zweier-Komplement Dies ist die gebräuchliche Darstellung einer binären Zahl, in der für Additionen das Vorzeichen unbeachtet bleiben kann [142]. Sie hat für Multiplikationen den Nachteil, dass jedes mal eine Betragsbildung erfolgen muss, was die Schaltaktivität stark erhöhen kann. Weiterhin ist die Bitaktivität bei Werten um die Null sehr groß. Das Zweier-Komplement ist der standardmäßig verwendete Zahlentyp, der vom DesignCompiler direkt unterstützt wird. Es wurde ein C++-Modul zur Bewertung der Bit-Aktivität erstellt (siehe Abschnitt 4.6.4).

Vorzeichendarstellung Bei dieser Darstellung wird der Betrag der Zahl und zusätzlich das Vorzeichen gespeichert. Additionen verkomplizieren sich dadurch, Multiplikationen werden einfacher. Bei Audiosignalen ist dies oft die energetisch günstigere Darstellungsform (siehe auch Abschnitt 4.6.4). Für die Vorzeichendarstellung wurde ein C++-Modul zur Bewertung der Bit-Aktivität erstellt (siehe Abschnitt 4.6.4). Weiterhin sind BC-VHDL-Module für diese Zahlenrepräsentation erstellt worden.

An zusätzlichen Zahlensystem wurden die folgenden implementiert (Theorie in Abschnitt 2.8):

[34] Die Radix-4-FFT ist ein IP-Block von Xilinx, ein Vergleich auf Basis einer Standardzellenbibliothek konnte daher nicht durchgeführt werden.

LNS Logarithmisches Zahlensystem. Implementierung als C++-Prototyp und als synthesefähige BC-VHDL-Beschreibung. Aufgrund der aufwändigen Addition ist ein genereller Einsatz nicht zu empfehlen.

RNS Residuen-basiertes Zahlensystem. Implementierung als C++-Prototyp. Auf eine Implementierung in BC-VHDL wurde aufgrund der komplexen Konverter verzichtet. Ein Einsatz als direkter Austausch für andere Zahlensystem erscheint nicht lohnenswert, da aufgrund der Arithmetik der Algorithmus speziell auf das Residuen-System angepasst werden muss.

Fließkomma Fließkomma-Zahlensystem. Dieses Zahlensystem ist in der Computertechnik sehr verbreitet. Es sind C++- und BC-VHDL-Module erstellt worden, die sich am IEEE 754-Standard [145] orientieren[35]. Die Module sind parametrierbar bezüglich der Bitbreite des Exponenten und der Mantisse.

Im Zuge der Umsetzung der Beispielalgorithmen wurden noch zwei weitere Low-Level Module erstellt und in das Framework integriert:

Dividierer Obwohl auch in der Synopsys' DesignWare mehrere Makromodule für Festkomma-Dividierer vorhanden sind, wurde ein Dividierer-Modul in Behavioral VHDL erstellt. Dies hat gegenüber den DesignWare-Dividieren den Vorteil, durch Synthese-Anweisungen beliebige Pipeline-Tiefen zu unterstützen. Weiterhin gibt es direkte Unterstützung für die Vorzeichen-Darstellung, die den Dividierer effizienter macht, da die Vorzeichen-Behandlung sich vereinfacht[36].

Weiterhin kann das Modul als Basis für einen noch zu implementierenden Gleitkomma-Dividierer dienen.

Logarithmische Lookup-Tabelle Bei der Umsetzung eines Algorithmus' auf Hardware ist es oftmals sinnvoll, Berechnungen im voraus auszuführen und in einer Tabelle abzulegen. Zwischenwerte werden dann oft linear interpoliert[37].

Wenn der abzudeckende Wertebereich groß ist und große Argumente sehr viel seltener vorkommen als kleine, ist die Verwendung einer Tabelle mit logarithmischen Tabellenabständen lohnenswert, da der Speicherbedarf stark reduziert wird, ohne die Audioqualität messbar zu beeinflussen.

Im PRO-DASP-Projekt wurde eine solche Tabelle im Zuge der Umsetzung des Ephraim-Malah-Algorithmus' für die Ersetzung der Kombination zweier Bessel-Funktionen mit der Exponential- und Wurzelfunktion verwendet [82]. Die Häufigkeitsverteilung dieser Funktion ist in Abb. 4.24 dargestellt.

[35] Auf die Behandlung einiger Sonderfälle wurde aus Komplexitätsgründen verzichtet. Die implementierte Low-Power-Multiplikation orientiert sich an [133, 171, 177].

[36] Die DesignWare-Dividierer wandeln jede negative Zweier-Komplement Zahl zunächst in eine positive Zahl um. Sie können keine Zahlen in Vorzeichendarstellung (s.o.) verarbeiten.

[37] Dies hat in Hardware einen recht großen Zusatzaufwand zur Folge, da für jeden Funktionswert zwei Tabellenwerte nachgeschlagen werden müssen. Wenn dies in einem Taktzyklus passieren soll, ist der Einsatz von Dual-Ported-RAM notwendig.

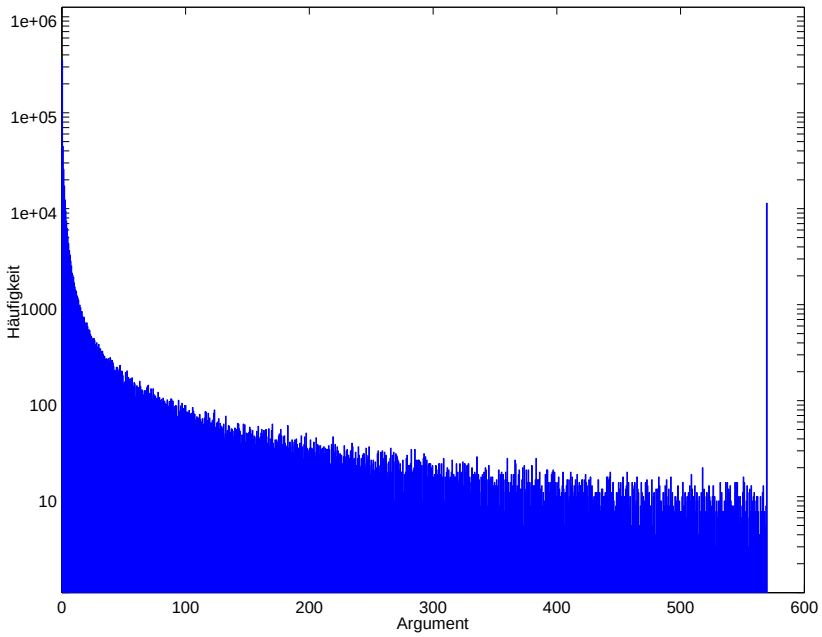


Abbildung 4.24.: Histogramm und Wertebereich des Arguments der modifizierten Besselfunktion des Ephraim-Malah-Algorithmus'. Die Spitze bei 570 ist dadurch zu erklären, dass das Funktionsargument nach oben hin auf diesen Wert begrenzt und gegebenenfalls abgeschnitten wird. Zu beachten ist die logarithmische Skalierung der Häufigkeit. Die Abbildung stammt aus der Diplomarbeit von F. Klöckner [82].

4.6.4. Messergebnisse

Filtertransformationen

In Abb. 4.25 sind Ergebnisse für die Synthese eines ausgerollten FIR Tiefpass-Filters 58. Ordnung dargestellt. Die Filterkoeffizienten wurden mit dem Filter-Entwurfswerkzeug von Matlab (*FDA-Toolbox*) berechnet. Die erste Simulation erfolgte in Simulink.

Dann wurde der Simulations- und Synthesefluss des Frameworks verwendet (siehe Abschnitt 4.1.4): Die Übereinstimmung des C++-Prototypen mit der Matlab-Referenz wurde zunächst mittels der Audiogütemaße überprüft.

Dann wurden die dargestellten Architekturen für zwei verschiedene Standardzell-Prozesse synthetisiert. Dabei sollte eine Echtzeitverarbeitung eines 44,1 kHz-Audiosignals erzielt werden. Die Optimierung auf RT- und Gatter-Ebene erfolgte jeweils durch den PowerCompiler von Synopsys.

Augenfällig ist zunächst der Unterschied zwischen den Technologien: Während der Mietec-Prozess durch den PowerCompiler nicht optimiert werden kann[38] („Power" im Vergleich zu „Power opt"), erzielt der PowerCompiler beim UMC-Prozess eine enorme Einsparung (bis zu 70% Einsparung gegenüber der unoptimierten Variante). Dies ist darauf zurückzuführen, dass die UMC-Bibliothek eine Low-Power-Bibliothek ist, die auf der einen Seite wesentlich mehr Standardzellen enthält, zum anderen auch wesentlich besser Power-charakterisiert ist.

Weiterhin ist erkennbar, dass die Ausnutzung der Symmetrie bei FIR-Filtern mit keinerlei Einsparungen verbunden ist. Dies liegt daran, dass der DesignCompiler für Konstantenmultiplizierer ohnehin Addierer-Strukturen verwendet, sodass die Strukturänderungen bei der symmetrischen Form eher mit Nachteilen verbunden ist, da der kritische Pfad gegenüber der Direktform länger ist. Die Anzahl der Zellen reduziert sich dagegen für einen ausgerollten Filter nicht messbar (siehe Abb. 4.25).

Eine maximale Dekorrelation der Koeffizienten mit dem HODCMI-Verfahren kann gegenüber der Direktform eine Einsparung von 40% ergeben, sie ist jedoch nicht wesentlich größer als die Einsparung bei der transponierten Direktform. Nach der Anwendung des PowerCompiler beim UMC-Prozess ist die transponierte Direktform sogar am energieeffizientesten, dabei ist die symmetrische transponierte Form in etwa gleichauf.

Fazit für die Filtertransformationen

Die aus der Literatur extrahierten Low-Power-Transformationen für Filter halten in der Praxis nicht, was sie in der Theorie versprechen.

Die relative strukturelle Irrelgularität der transformierten Formen gegenüber den Direktformen erschwert der Hardwaresynthese die Arbeit und führt zu vergleichsweise schlechten Ergebnissen, die weit unter dem durch die theoretische Reduktion der Schaltaktivität vorgegebenem Maximum liegen.

Weiterhin ist bemerkenswert, wie stark sich die Ergebnisse in den beiden verglichenen ASIC-Standardzellprozessen unterscheiden. Dies lässt den Schluss zu, dass die

[38] Bei der Variante mit symmetrischen Koeffizienten ist das Ergebnis sogar schlechter.

Wahl einer optimalen Hardware-Architektur sehr stark von der Zieltechnologie abhängt, weshalb die Module zwingend möglichst realitätsnah (in diesem Fall mit dem PhysicalCompiler platziert und verdrahtet) energiecharakterisiert sein müssen.

Weiterhin sind generell Transformationen zu bevorzugen, die strukturell möglichst regulär sind und somit vom Synthesewerkzeug effizient in eine reguläre Standardzell-Topologie überführt werden können. Als Nebeneffekt bleibt der kritische Pfad damit kurz, so dass Potential für eine V_{dd}-Absenkung geschaffen wird.

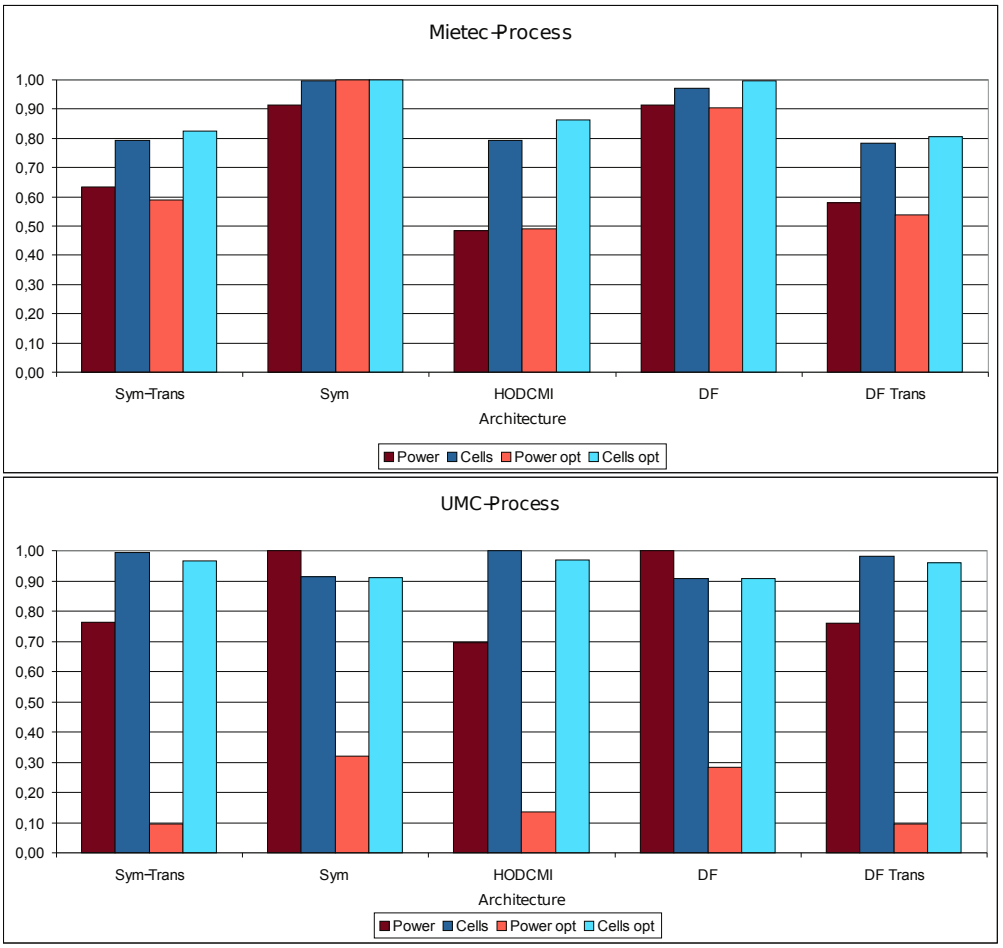


Abbildung 4.25.: Ergebnisse für einen ausgerollten FIR-Tiefpass 58. Ordnung. Die Standardzellprozesse sind Mietec (0,35 μm 3,3 V) und UMC (0,18 μm 1,8 V). Dargestellt ist der relative Leistungsbedarf und die relative Standardzellenanzahl für eine Echtzeitverarbeitung mit 44,1 kHz, jeweils vor und nach („opt“) der Anwendung des PowerCompilers.

Die Abkürzungen für die Filtertransformationen sind: Sym-Trans - transponierte Direktform unter Ausnutzung der symmetrischen Koeffizienten; Sym - Direktform unter Ausnutzung der symmetrischen Koeffizienten; DF - Direktform; DF-Trans - transponierte Direktform

Die theoretisch maximale Dekorrelation durch die HODCMI-Transformation ergibt für den platzierten und verdrahteten Chip keine deutlich besseren Ergebnisse als die transponierte Direktform, weder beim Flächen- noch beim Leistungsbedarf.
Für die UMC-Standardzell-Bibliothek (Bild unten) konnte der PowerCompiler erhebliche Optimierungen vornehmen. Aufgrund der geringen Anzahl und schlechten Power-Charakterisierung der Mietec-Bibliothek bringt dort die Optimierung keine weitere Verbesserung. Weitere Erläuterungen siehe Text.

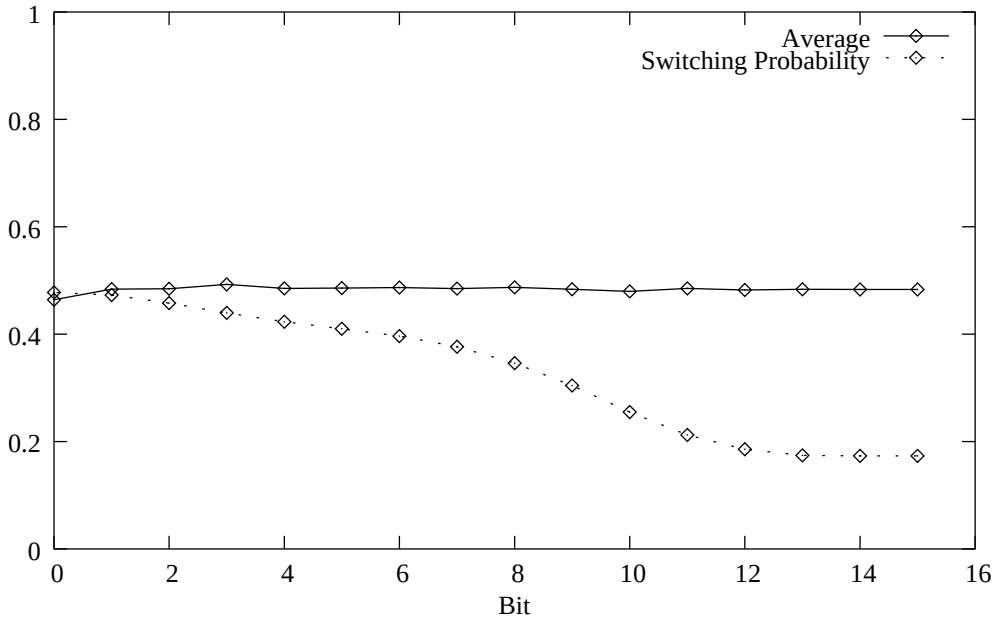
Zahlendarstellungen

Das Standardsystem in der Software- wie in der Hardwaretechnik ist die Zweier-Komplementdarstellung. Für die Audiosignalverarbeitung kann es jedoch ein großer energetischer Vorteil sein, die Zahlen in der Vorzeichendarstellung zu speichern [114]. Um diesen Vorteil abschätzen zu können, wurde ein Statistikmodul erstellt, dass die Bitaktivität innerhalb eines Moduls erfasst und graphisch veranschaulicht.

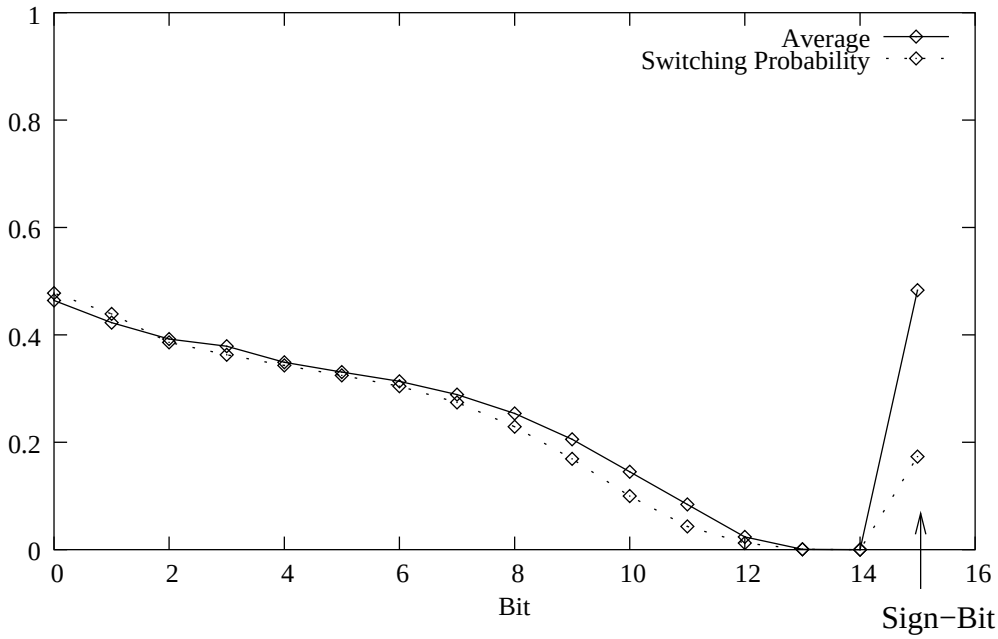
Festkommazahlen Eine beispielhafte Aktivitätsstatistik für Festkommazahlen ist in Abb. 4.26 dargestellt. Es handelt sich hierbei um einen FIR-Tiefpass 58. Ordnung, der ein repräsentatives Audiosignal filtert. Es ist klar erkennbar, dass die Vorzeichendarstellung hier Vorteile bietet, da die Schaltwahrscheinlichkeit bereits ab Bit 9 unter das Minimum der Zweier-Komplement-Darstellung fällt.

Weiterhin ist in Abb. 4.26 erkennbar, dass die charakteristische Korrelation von Audiosignalen (die Verteilung um den Nullpunkt) in der Zweier-Komplementdarstellung völlig verlorengeht; der Durchschnittswert ist für alle Bits bei 0,5. In der Vorzeichendarstellung sieht man dagegen, dass der Durchschnittswert zu den höherwertigen Bits abfällt. Die Wahl der Zahlendarstellung trägt damit erheblich zu einer Energie-optimalen Implementierung bei.

Die Quantisierung, d. h. Festlegung der Bitanzahl für die Festkommaarithmetik erfolgt mittels der Gütemaße des Frameworks. Ein ausführliches Beispiel ist in Abschnitt 5.2 beschrieben, weshalb an dieser Stelle auf weitere Messergebnisse verzichtet wird.



(a) Statistik der Zweier-Komplement Darstellung



(b) Statistik der Vorzeichendarstellung

Abbildung 4.26.: Festkomma Bit-Statistik für einen FIR-Filter. Die Statistik wurde für die Registervariable eines nicht ausgerollten FIR-Filters 58. Ordnung aufgenommen. „Average“ ist der durchschnittliche Wert eines Bits.
Die Schaltwahrscheinlichkeit als Anhalt für den Leistungsbedarf ist gestrichelt wiedergegeben.

Gleitkommazahlen Bei einer hohen Dynamik des zu verarbeitenden Datenstrom ist die Verwendung einer Gleitkommadarstellung u. U. günstiger als die Festkommadarstellung. Um den Aufwand abschätzen zu können, wird zunächst eine Quantisierungsuntersuchung mit Hilfe des C++-Framework-Moduls durchgeführt. Als Beispiel soll hier ein mit Matlab entworfener FIR-Filter 58. Ordnung dienen. Die Ergebnisse der C++-Simulation sind in Abb. 4.27 dargestellt.

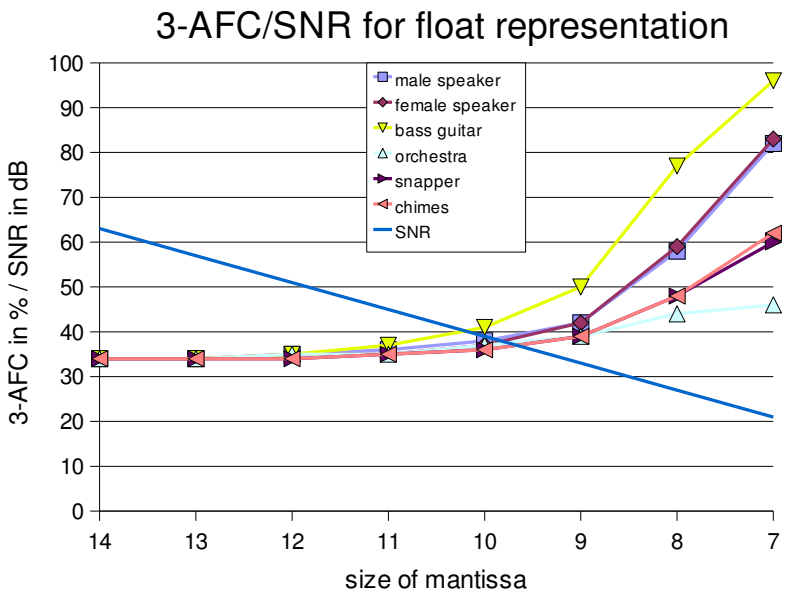


Abbildung 4.27.: Quantisierung für Gleitkommazahlen. Dargestellt ist das Qualitätsmaß 3-AFC und das Signal-Rausch-Verhältnis SNR, die mittels des C++-Software-Prototypen berechnet wurden. Die Berechnung wurde für einen FIR-Filter 58. Ordnung durchgeführt. Der 3-AFC-Wert ist für die in der Testbench verwendeten repräsentativen Audiosamples getrennt dargestellt. Üblicherweise wird ein Mittelwert zur Bewertung herangezogen, es sei denn, das Audiosystem beschränkt sich auf eine bestimmte Anwendung, wie z. B. eine Spracherkennung, für die nur das Sprach-Sample von Interesse wäre [156, 155]. Der für normale Anwendungen gesetzte 3-AFC Wert von 70% wird bei einer Mantissengröße von etwa 9 Bit sicher erreicht. Die Ratewahrscheinlichkeit liegt bei 33%, d. h. ab dieser Schwelle können verrauschtes und originales Sample nicht mehr unterschieden werden. Dies ist ab etwa 12 Bit der Fall.
Bei dem SNR-Wert handelt es sich um eine Regressionsgerade über alle Samples.

Transfer auf Hardware Für den Transfer auf Hardware wird zunächst eine Abschätzung der benötigten Dynamik durch eine Aktivitätsstatistik der Exponentenbits (Abb. 4.28) durchgeführt.

Diese erfolgt mit einer hohen Mantissengenauigkeit und soll sicherstellen, dass im Exponenten keine Überläufe auftreten können, was starke Auswirkungen auf die Audioqualität hätte. In diesem Beispiel (Abb. 4.28) sind mindestens fünf Bit für den Exponenten vorzusehen.

Die benötigte Bitbreite für die Mantisse wird dann mittels der Framework-Gütemaße abgeschätzt (Abb. 4.29).

Die in Abb. 4.28 und 4.29 dargestellten Kurven wurden in diesem Beispiel durch eine

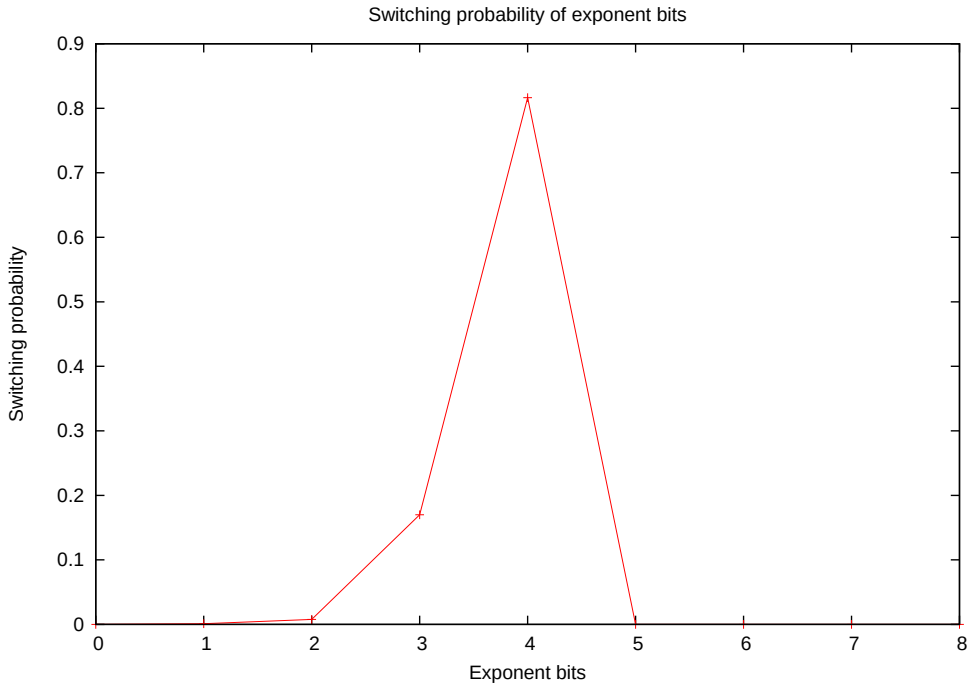
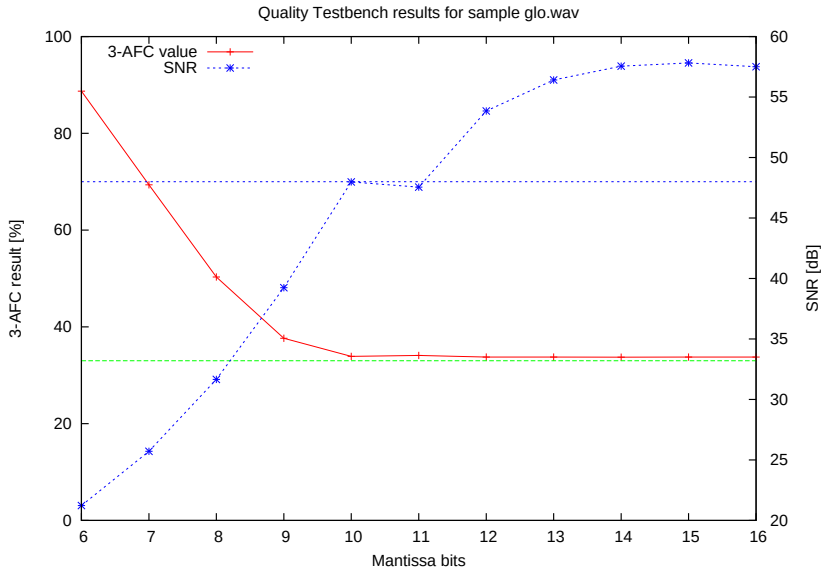
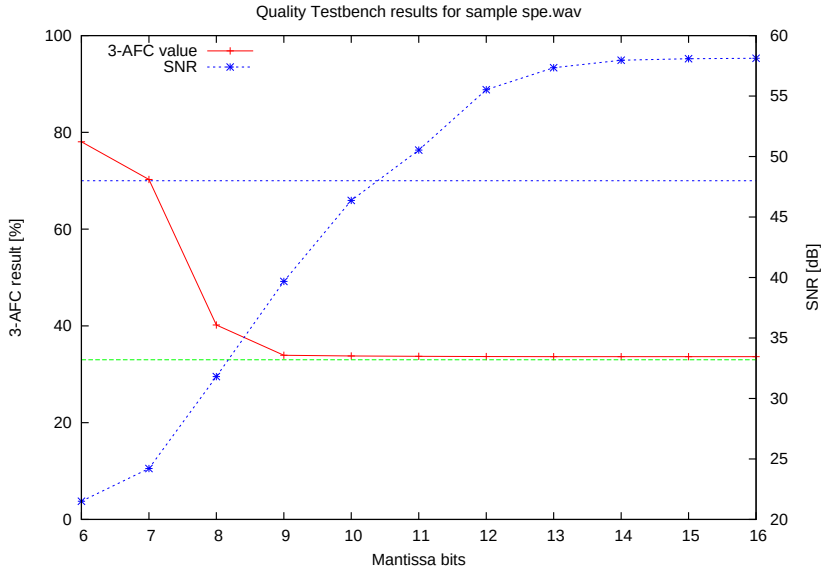


Abbildung 4.28.: Gleitkomma-HW-Ergebnisse für einen FIR-Filter 1. Gezeigt ist die Aktivitätsstatistik für die Exponentenbits eines FIR-Filters 58. Ordnung.
Auffällig ist die geringe Aktivität in den unteren Bits, die leider nicht für eine Optimierung genutzt werden kann.



(a) Audioqualität in Abhängigkeit der Anzahl der Mantissenbits für Glockspielsample



(b) Audioqualität in Abhängigkeit der Anzahl der Mantissenbits für Sprachsample

Abbildung 4.29.: Gleitkomma-HW-Ergebnisse für einen FIR-Filter 2. Die beiden Diagramme zeigen die beiden im Framework verwendeten Audiogütemaße für einen Filter 58. Ordnung in Abhängigkeit der Mantissenlänge für zwei verschiedene Audiosamples aus dem repräsentativen Ensemble.
Dabei stellt die grüne Line (33 %) die Rateschwelle für den 3-AFC-Test und die blaue Linie (70 %) die vereinbarte Akzeptanzschwelle, ab der die Qualität akzeptabel ist, dar.
Auffällig ist der nicht monotone Verlauf des Signal-Rausch-Verhältnisses, in das einzelne, z. B. durch Überläufe bedingte Verzerrungen stark einfließen, da hier um die Rechenzeit zu begrenzen kurze Ausschnitte aus den Audiosamples verwendet wurden.
Das 3-AFC-Maß ignoriert diese Ausreißer, sofern sie durch das zugrundeliegende Hörmodell als „nicht hörbar“ eingestuft werden.

Hardware-Simulation der platzierten und verdrahteten Netzliste gewonnen. Zu beachten ist hier, dass nicht Samples voller Länge genutzt wurden, um den Aufwand für die Simulation auf einige Stunden zu begrenzen. Mit den bei der Simulation gewonnen zeitlichen Signalverläufen für die einzelnen Netze wurde auch die Energiecharakterisierung durchgeführt (siehe Abschnitt 4.6.2).

Der Vergleich der Ergebnisse der C++-Implementierungen (Abb. 4.27) mit den Ergebnissen aus der Hardware-Simulation (Abb. 4.29) dient gleichzeitig der Verifikation des Framework-Moduls für die Fließkomma-Arithmetik: Da der Synthese-Vorgang „verlustfrei“ sein sollte, müssen die Ergebnisse für die Hardware- und Software-Simulation identisch sein. In den erwähnten Abbildungen können leichte Differenzen erkannt werden, da hier unterschiedlich lange Testdaten herangezogen wurden. Für identische Testdaten sind die Ergebnisse jedoch identisch.

Eine generelle Bewertung der verwendeten Zahlensysteme steht in Abschnitt 2.8.5.

4.7. Rapid-Prototyping

Um die Hardwareumsetzung der Signalverarbeitungsalgorithmen validieren zu können, verfügt das Framework über eine Anbindung an ein Rapid-Prototyping System[39].

Das System basiert auf einem Xilinx Virtex2-FPGA, das eine Echtzeitverarbeitung vieler Algorithmen ermöglicht.

Durch die generischen Modulschnittstellen (siehe Abschnitt 4.6) ist es möglich, einzelne Module auf dem FPGA auszuführen und den Rest des Systems in Software auf dem Host ablaufen zu lassen.

4.7.1. Systembeschreibung

Abbildung 4.30 zeigt den Aufbau des Systems. Bei dem Rapid-Prototyping-Board handelt es sich um das Ballynuey-Board der Firma Nallatech [108], das über eine PCI-Schnittstelle an den Host-PC angebunden ist.

Um eine transparente Integration eines Framework-Moduls auf dem FPGA zu ermöglichen, wurde ein generisches IO-Master-Design entwickelt, welches die Hardware-(FPGA-)seitige Anbindung an das Framework darstellt. Auf dem Host-PC läuft ein IO-Backend, das das auf der Hardware laufende Modul mit den restlichen Modulen der Simulationsumgebung verbindet.

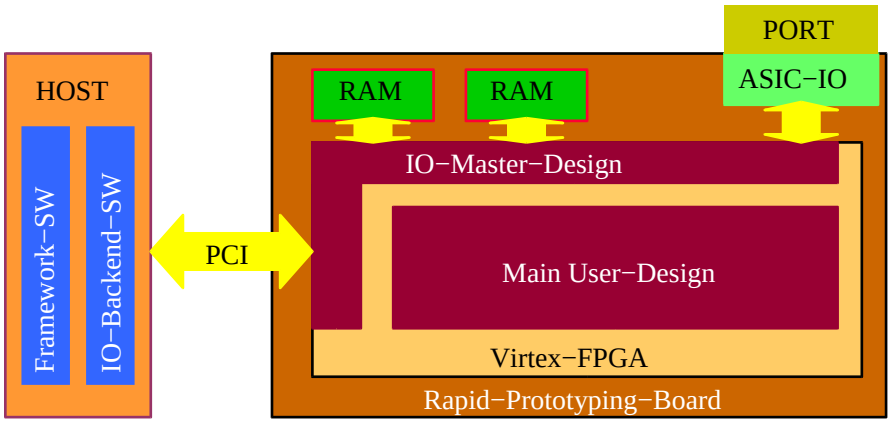


Abbildung 4.30.: Rapid-Prototyping Umgebung für das Framework. Der rechte Block stellt das FPGA-System dar. Das Hauptdesign (ein oder mehrere Module) wird über ein IO-Master-Design an das System-RAM und den Host-PC via PCI angebunden. Das System verfügt weiterhin über frei benutzbare Ports (ASIC-IO), die es ermöglichen, z. B. ein externes ASIC einzubinden.

4.7.2. Einbindung in das Framework

Um zu Validierungs- und Demonstrationszwecken neben einzelnen Modulen auch komplexe Signalverarbeitungsalgorithmen auf dem Prototyping-System zu implemen-

[39] Das System wird in [166] knapp beschrieben. Es handelt sich um eine Weiterentwicklung des in [167] verwendeten Systems.

tieren, war es notwendig, speziell an das FPGA angepasste Module einzubinden.

Diese Module stammen aus einer Bibliothek der Firma Xilinx und werden von einem Werkzeug mit dem Namen „Coregen“ [180] für die Verwendung auf dem FPGA konfiguriert. Es handelt sich um eine Modulsammlung für den Signalverarbeitungsbereich, die diverse Filter, Transformationen und andere arithmetische Module enthält. Diese Module liegen im Gegensatz zu den Frameworkmodulen nicht als synthesefähige Verhaltensbeschreibung vor, sondern als Netzliste, die z. T. vorverdrahtet ist, um auf dem FPGA eine besonders hohe Performanz zu erzielen. Für diese Module gibt es Simulationsmodelle in VHDL aber nicht in C++, so dass für die Einbindung dieser Module C++-Simulationen entwickelt werden mussten.

Um diese Coregen-Module in den Synthese-Flow des Frameworks einzubinden, wurde für jedes einzubindende Modul ein VHDL-Wrapper mit der generischen Modul-Schnittstelle erstellt und das BC-Syntheseskript des zugehörigen synthetischen Moduls so gestaltet, dass dem BC für die Synthese das RT-Verhalten des Moduls bekannt ist.

Eine Energiebewertung ist allerdings nur mit dem Werkzeug „XPower“ des FPGA-Herstellers möglich, da die Primitiv-Zellen des FPGA nicht wie Standardzellen bezüglich der elektrischen Leistung charakterisiert sind. Da FPGAs aufgrund ihrer generischen Struktur eine sehr hohen Verdrahtungsanteil haben, müssen Strategien zur Energieoptimierung hierauf besondere Rücksicht nehmen. Da das Prototyping-System nur zur schnellen Verifikation der Schnittstellen und Algorithmen genutzt wird, soll auf solche Strategien hier nicht eingegangen werden. Weitere Informationen hierzu finden sich z. B. in [18].

4.8. Zusammenfassung

In diesem Kapitel wurde das **Entwicklungsrahmenwerk** beschrieben, das den Kern der vorliegenden Arbeit darstellt.

Beginnend mit einem Überblick über die **Komponenten** und den **Entwurfsfluss** innerhalb des Frameworks wurden in den folgenden Abschnitten die Einzelaspekte **Verlustleistungsschätzung** und **Verlustleistungsoptimierung** innerhalb des Frameworks beschrieben.

Dann wurde dargestellt, welche **Gütemaße** für die Bewertung der Audioqualität eines Framework-Algorithmus herangezogen werden. Diese Gütemaße sind zentraler Bestandteil des Frameworks und ermöglichen sowohl eine **Verifikation** der umgesetzten Algorithmen als auch eine verlustleistungsoptimale **Quantisierung** aller Operatoren.

Im nächsten Abschnitt wurden die für die eigentliche **Hardware-Synthese** verwendeten Werkzeuge beschrieben. Es handelt sich zwar um **kommerzielle Standard-Werkzeuge**, diese wurden aber so in das Framework integriert, dass eine **ebenenübergreifende Optimierung** bei gleichzeitigem **minimalen Eingriff** in die VHDL-Hardware-Beschreibung möglich ist.

Das wichtigste Mittel zu diesem Ziel ist die Framework-eigene **Modulbibliothek**, die in dem folgenden Abschnitt 4.6 beschrieben wurden. Dazu gehört eine Beschreibung der implementierten Module, wobei die Wirksamkeit durch **Messergebnisse** für einzelne Module belegt wurde.

Abschließend erfolgte eine Beschreibung der in das Framework integrierten FPGA-basierten **Rapid-Prototyping** Plattform.

5. Exemplarische Implementierungen

In diesem Kapitel folgen einige beispielhafte Implementierungen, die die Anwendung und Nützlichkeit des Frameworks demonstrieren sollen. Die Algorithmen stammen allesamt von unseren Projektpartnern aus der medizinischen Physik in Oldenburg (siehe Abschnitt 1.2).

5.1. Überblick

Die Beispielalgorithmen dienen der gehörgerechten Manipulation von Audiodaten, wie sie z. B. in einem Hörgerät oder in der Vorverarbeitungsstufe eines Spracherkenners stattfinden. Gemeinsames Merkmal ist die frequenz- und zeitselektive Veränderung der Daten, die eine Aufspaltung des Audio-Signals in Frequenzbänder oder die Fourier-Transformation in den Frequenzraum erforderlich macht. Sofern diese z. B. in einem Hörgerät benötigt werden, müssen die manipulierten Daten in einem weiteren Schritt wieder in ein Audio-Signal zurückgeführt werden[1].

Für diesen Vor- bzw. Nachverarbeitungsschritt, der üblicherweise einen Großteil der Zeitkomplexität des Gesamtalgorithmus' darstellt, wurden in Oldenburg zwei Verfahren entwickelt bzw. spezifiziert:

1. Eine auf einem physiologischen Hörmodell[2] basierende Bandpassfilterbank (Abschnitt 5.2) und
2. eine konventionelle Kurzzeit-Frequenztransformation mit Overlap-Add-Struktur (Abschnitt 5.3).

Diese beiden Verfahren dienen als „Hülle" oder Rahmen für die eigentlichen Audioalgorithmen, die zwischen der Analyse und Resynthese der Daten angeordnet ist. Wegen der stark unterschiedlichen Strukturen der Analysefilter sind diese jedoch nicht austauschbar, sondern müssen speziell angepasst werden. Ein Fallbeispiel für die Kurzzeit-FFT, die Ephraim-Malah Störgeräuschunterdrückung, ist in Abschnitt 5.4 dargestellt.

Alle implementierten Beispiel haben die in Abschnitt 4.6 erwähnten Eigenschaften eines Moduls, sind also wiederum als Module in das Framework integriert und können so Bestandteil anderer Audiosignalverarbeitungsalgorithmen werden.

In allen Fällen diente eine Matlab-Implementierung des Signalverarbeitungsalgorithmus, der uns im Rahmen des PRO-DASP-Projektes von der AG MEDI der Universität

[1] Für andere Audio-Algorithmen wie Spracherkenner ist eine solche Rücktransformation nicht erforderlich.

[2] Es handelt sich hierbei um das Hörmodell, das auch Grundlage des physiologisch korrekten Gütemaßes ist (Abschnitt 4.4.2).

Oldenburg zur Verfügung gestellt wurde, als Basis für eine schaltungstechnische Umsetzung.

5.2. Die Gammatonfilterbank und Resynthese

Die Gammatonfilterbank ist Teil eines in Oldenburg entwickelten Perzeptionsmodells für das menschliche Gehör. Es ist in den beiden Veröffentlichungen [33] und [34] beschrieben. Da im PRO-DASP-Projekt eine Low-Power-Implementierung der Gammatonresynthese festgeschrieben ist, soll hier in Hinblick auf die Energiebewertung nur auf diese eingegangen werden. Die Analysefilterbank liegt als synthesefähige Behavioral-VHDL-Beschreibung ebenfalls vor, eine detaillierte Untersuchung und Optimierung ist jedoch nur für die Resynthese erfolgt.

Das Prinzip des Gammaton-Frameworks wird im folgenden beschrieben.

5.2.1. Prinzip

Zentrale Funktionseinheit im Innenohr ist die Cochlea, in der akustische Druckwellen in elektrische Nervenimpulse umgewandelt werden. Die Frequenz-Orts-Umsetzung wird dabei durch die Basilarmembran ausgeführt, deren Elastizität ortsabhängig ist (siehe Abb. 5.1). Eine gute Näherung des frequenzselektiven Verhaltens der Basilarmembran stellt das Gammatonfilter dar, dessen Impulsantwort folgendermaßen definiert ist:

$$g_r(t) := \gamma(n,\lambda)\,\theta(t)t^{n-1}\,\exp(-\gamma t)\cos(2\pi f_0 t) \qquad n \geq 1, \lambda > 0. \tag{5.1}$$

Dabei ist $\theta(t)$ die Heavyside-Funktion, n die Filterordnung, $\gamma > 0$ der Dämpfungsfaktor, f_0 die Mittenfrequenz des Filters und $\gamma(n,\lambda)$ eine Normierungskonstante. Details zur gehörgerechten Parametrisierung der hier verwendeten Filterbank finden sich in [33].

Abbildung 5.2 zeigt die Struktur des verwendeten Analyse-Resynthese-Rahmens. Das Eingangssignal wird von der Analysefilterbank in 30 Bandpass-Kanäle aufgespalten. Dann erfolgt eine Modifikation der Daten. Im letzten Schritt werden die 30 Filterkanäle wieder aufsummiert und stehen als digitales Audiosignal am Ausgang zur Verfügung.

5.2.2. Analyse und Partitionierung

Analysefilterbank

Für dieses Beispiel kommt eine lineare Gammatonfilterbank mit 30 Kanälen zum Einsatz, wie sie bereits von einer Projektgruppe an der Universität Oldenburg im Rahmen einer Pegelanpassung für ein Altera-FPGA implementiert wurde [48].

Eine ursprünglich von Patterson in [116] vorgeschlagene Implementierung wurde in Oldenburg modifiziert [66][3]. Die Mittenfrequenzen der 30 Kanäle liegen zwischen 73 Hz und 6 681 Hz, sie sind äquidistant auf der sog. ERB-Skala angeordnet.

[3] Auch die in [138, 139, 140] beschriebenen Implementierungen basieren auf dem Modell von Patterson.

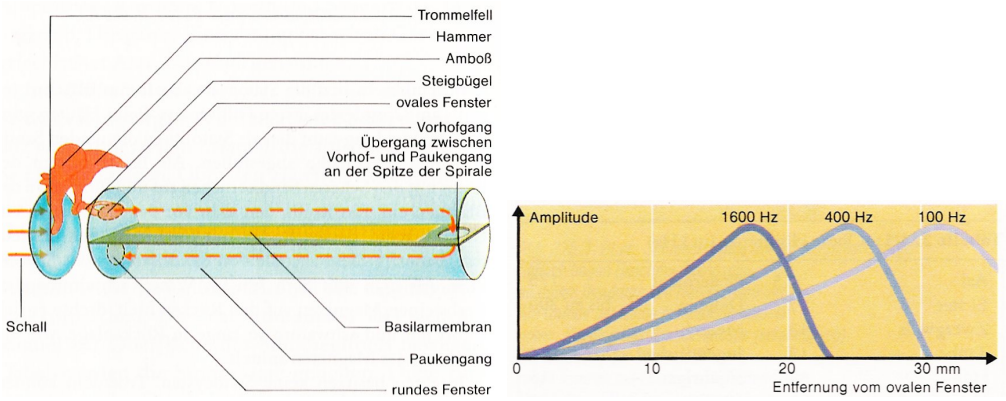


Abbildung 5.1.: Modell des Innenohrs und Eigenschaften der Basilarmembran. Das linke Bild ist eine schematische Darstellung des Säugetierohres. Hammer, Amboss und Steigbügel übertragen die Schwingungen des Trommelfells auf das ovale Fenster der Hörschnecke (Cochlea). Sie führen dabei im wesentlichen eine Impedanzanpassung durch. Die Cochlea ist hier ausgerollt dargestellt. Die Schwingungen laufen durch den flüssigkeitsgefüllten Vorhofgang zur Spitze der Hörschnecke und von dort den Paukengang entlang zum runden Fenster. Entlang des Paukengangs sitzt die mit Sinneszellen bedeckte Basilarmembran, die zur Schneckenspitze immer breiter wird. Dadurch hat die Basilarmembran örtlich unterschiedliche Resonanzfrequenzen, so dass eine Frequenz-Ortsumsetzung stattfindet, die an den Hörnerv übermittelt wird. Das rechte Bild zeigt die Lage der Schwingungsmaxima der Basilarmembran für sinusförmige Schwingungen verschiedener Frequenz. Diese Eigenschaft der Basilarmembran wird durch die Gammatonfilterbank modelliert (aus [83]).

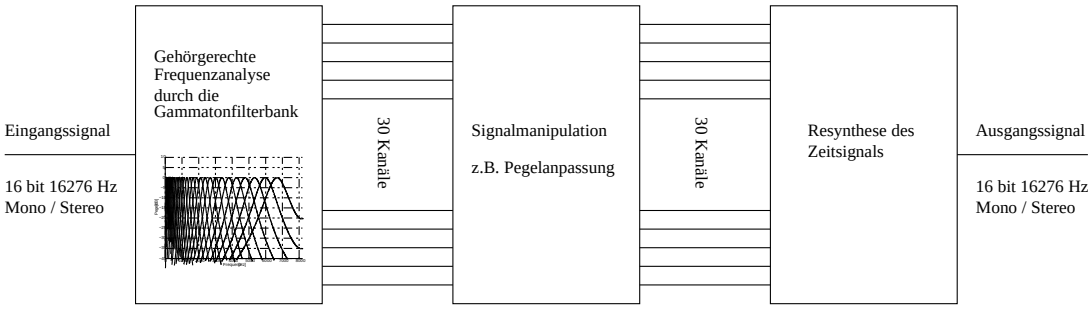


Abbildung 5.2.: Die Gammatonfilterbank mit entsprechender Resynthese. Für die schaltungstechnische Realisierung wurde ein auf 30 Kanäle parametrisierte Variante verwendet.

Die 30 Bandpassfilter sind jeweils als IIR-Filter vierter Ordnung realisiert:

$$y[n] := bx[n] + \sum_{i=1}^{4} a_i\, y[n-i] \qquad b, x \in \mathbb{R} \quad a_i, y \in \mathbb{C} \tag{5.2}$$

Die Hardwarerealisierung erfolgte im Silicon-Ear Projekt[4] als vierfache Kaskade eines komplexwertigen IIR-Filters erster Ordnung [134]:

$$y[n] := ay[n-1] + bx[n] \qquad b, x \in \mathbb{R} \quad a, y \in \mathbb{C} \tag{5.3}$$

Dabei ist $b := 1 - |a|$ reellwertig. Die kaskadierte Version wurde aus dem ursprünglichen IIR-Filter vierter Ordnung durch Faktorisierung der Transferfunktion gewonnen (vgl. Abschnitt 2.9.4).

Die Realisierung erfolgte in transponierter Direktform II, so dass die vier Additionen und komplexen Multiplikationen parallel durchgeführt werden können. Diese Implementierung ist aus energetischer Sicht bereits optimal: Der Aufwand wäre mit einem FIR-Filter wesentlich höher, da gegenüber einem IIR-Filter sehr viel mehr Stufen erforderlich wären, was die Anzahl der benötigten Operationen stark vergrößern würde. Durch die Kaskadierung ist es möglich, mit einem einzigen, fest parametrierten Filter erster Ordnung auszukommen, wodurch eine Gleitkommaimplementierung relativ einfach möglich wurde, da die Anzahl der arithmetischen Einheiten vergleichsweise gering ist. Die Notwendigkeit einer Gleitkommafilterung hat sich bereits aus Voruntersuchungen ergeben [134].[5]

Die ressourcenoptimierte Implementierung ist in diesem Fall mit einer energieoptimimalen Architektur identisch, so dass sie hier nicht weiter modifiziert wurde und mit der Quantisierung übernommen wurde.

Die Quantisierung des Filter erfolgte ebenfalls im Silicon-Ear Projekt, damals noch mit einem konventionellen Fehlermaß (siehe Abb. 5.3). In Absprache mit der AG MEDI wurde ein Maximalwert von 10^{-3} vereinbart, daraus resultiert eine Berechnungsgenauigkeit von 24 Bit.

Pro Eingangssample und Bandpasskanal sind jeweils vier komplexwertige Additionen und Multiplikationen mit 24 Bit sowie eine reellwertige Additionen und Multiplikation mit 16 Bit nötig. Bei 30 Frequenzbändern und einem Stereosignal sind dies $4 * 60 = 240$ komplexwertige 24 Bit Additionen und Multiplikationen und weiter 60 reellwertige Additionen und Multiplikationen mit 16 Bit.

Der Speicherbedarf beträgt vier Worte pro Bandpasskanal, insgesamt also 240 Worte.

[4] Das Silicon-Ear Projekt [119] war ein ebenfalls von der DFG gefördertes Vorgängerprojekt von PRO-DASP. In PRO-DASP 1 und 2 wurden die Modelle und Algorithmen aus Silicon Ear weiterentwickelt und Energie als Optimierungskriterium hinzugefügt.

[5] Eine detaillierte Beschreibung der Filterstrukturen findet sich in Abschnitt 2.9.

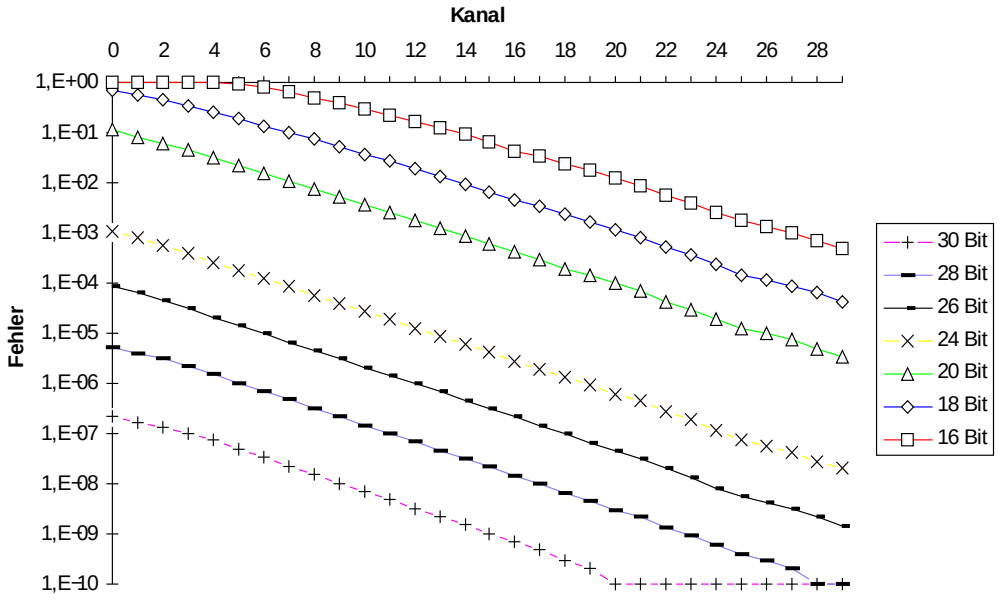


Abbildung 5.3.: Fehlerplot der Gammaton-Analysefilterbank. Gezeigt ist der mittlere quadratische relative Fehler gegenüber der Matlab-Referenzimplementierung, die auf 80 Bit Gleitkommazahlen basiert (aus [84]).

Resynthese

Die sich an den Verarbeitungsschritt anschließende Resynthese summiert die einzelnen Filterbänder wiederum zu dem Gesamtsignal auf. Problematisch ist in diesem Schritt die Tatsache, dass die einzelnen Bandpassfilter unterschiedliche Gruppenlaufzeiten aufweisen, wodurch es bei der Überlagerung zu destruktiver Interferenz kommen kann und die Signalqualität stark leidet.

Daher müssen die höheren Frequenzen, die eine geringere Laufzeit haben, verzögert werden, so dass die Einhüllendenmaxima der Impulsantworten der Bandpässe annähernd übereinander liegen. Wie in [48] beschrieben, wird für akzeptable Ergebnisse eine maximale Verzögerung von mindestens 14 ms benötigt. Dies ist jedoch kein akzeptabler Weg, da bei einer Verwendung als Hörgerät höhere Latenzen als ca. 4 ms zu Irritationen der Patienten führt, da die akustischen Reize nicht mehr mit dem Lippenbild beim Sprechen übereinstimmen.

Es wird daher zusätzlich zur zeitlichen Verzögerung eine Multiplikation der Bandpassantwort mit einer komplexen Konstanten durchgeführt, die analog zum Verschiebungssatz der Z-Transformation eine zeitliche Verschiebung des Signals bewirkt. Diese Verschiebung betrifft allerdings nur die reelle Feinstruktur innerhalb der festen Einhüllenden, weshalb auch dieses Verfahren begrenzt ist[6]. Es ist so aber möglich, die zeitliche Auflösung der Verschiebung zu verbessern, da eine reine Verzögerung nur um ganzzahlige Vielfache des Kehrwertes der Samplingfrequenz möglich ist.

Zum Einsatz kommt deshalb eine Kombination aus zeitlicher Verzögerung sowie Verschiebung durch komplexe Multiplikation. Die dafür notwendigen Multiplikations-

[6] Eine ausführliche Darstellung zu Wavelets und Filterbänken findet sich z. B. in [163].

konstanten wurden aus [48] übernommen. Das Ausgangssignal z wird damit wie folgt berechnet:

$$z := \sum_{i=1}^{30} D_i \left(C_i \, k_i\right) \qquad C_i, k_i \in \mathbb{C} \tag{5.4}$$

Dabei ist D der Verzögerungsoperator und D_i beschreibt die Verzögerung des i-ten Kanals, C_i ist die komplexe Multiplikationskonstante des i-ten Kanals.

Für die vorliegende Resynthese sind die zeitlichen Verzögerungen für die höchsten elf Kanäle identisch Null, so dass gilt:

$$z = \sum_{i=1}^{11} C_i \, k_i + \sum_{i=12}^{30} D_i \left(C_i \, k_i\right) \tag{5.5}$$

Für die höchsten elf Kanäle wird nur eine komplexe Multiplikation durchgeführt, die tieferen 19 Kanäle werden zusätzlich zeitlich verzögert. Da die Verzögerung frequenzbandabhängig ist, müssen bei diesem Ansatz sehr viele Werte zwischengespeichert werden, der Speicherbedarf in Worten S wäre

$$S = \sum_{i} |D_i| = 644 \text{ Worte für } \max|D_i| \equiv 56 \tag{5.6}$$

Dies entspricht den aufsummierten Verzögerungszeiten der einzelnen Kanäle in Takten, da jeder Kanal getrennt gespeichert und anschließend die verzögerten Werte aufsummiert werden. Analog zu der Implementierung in [48] werden die Werte in einem Ringpuffer direkt aufaddiert (siehe Abschnitt 5.2.3). Damit ist der Speicherbedarf $\max|D_i| = 56$ Worte.

Die Verzögerung beträgt dabei 56 Samples, das entspricht bei der gewählten Abtastrate von 16276 Hz einer Latenz von 3,4 ms. Pro Kanal wird demnach eine komplexe Multiplikation und eine Addition durchgeführt. Damit ist die Resynthese von der Arithmetik weniger komplex als die Analysefilterbank, die vier weitere Addierer sowie einen zusätzlichen reellwertigen Multiplizierer pro Kanal hat.

Da für den linken und den rechten Stereokanal jeweils ein Ringpuffer für die letzten 56 Samples verwaltet werden muss und zudem 60 Worte für die C_i benötigt werden, hat die Resynthese einen Speicherbedarf von 172 Worten und benötigt damit etwa ein Drittel weniger Speicher als die Analysefilterbank.

Abgeleitet vom vorigen Abschnitt muss der Resyntheseblock die folgenden Schritte für jedes Sample durchführen:

1. Komplexe Multiplikation des Kanalwertes mit dem komplexen Zeitkorrekturfaktor
2. Verzögerung der tiefen Frequenzen um maximal 4 ms
3. Aufsummation der so gewonnen Ergebnisse

Aufgrund der einfachen Struktur ist keine Anwendung von speziellen Low-Power-Transformationen möglich. Es wurde deshalb neben Untersuchungen zur Zahlendarstellung nur eine Optimierung mittels des Behavioral Compilers vorgenommen.

Quantisierungsuntersuchungen

Ziel war es, eine geeignete Zahlenrepräsentation für die Verarbeitung zu ermitteln. Die Untersuchung basiert auf der C-Implementation des Algorithmus', die so modifiziert wurde, das die C++-Klassen für die Festkomma- und Gleitkommamodule (siehe Abschnitt 4.6) als Berechnungsdatentypen verwendet werden können. Für die Qualitätsberechnung wird die Verarbeitungskette

Audiosignal → Gammaton-Analysefilterung → Gammatonresynthese → Audiosignal

ausgenutzt. Dabei erfolgte die Analyse und Resynthese mit parametrierbarem Datentyp und Auflösung. Da für die Analysefilterbank bereits im Vorgängerprojekt Quantisierungsuntersuchungen durchgeführt wurden, ist die Verarbeitungsgenauigkeit auf 24 Bit Festkomma festgelegt worden (siehe Abb. 5.3).

Für die Resynthese wurde sowohl Zahlendarstellung als auch Auflösung variiert. Die Ergebnisse der Berechnungen mit reduzierter Auflösung wurden dann mit Hilfe der MEDI-Testbench mit den Original-Datensätzen verglichen. Da diese Quantisierungsuntersuchung zu Beginn der Laufzeit des PRO-DASP-Projektes durchgeführt wurden, wird hier die Version 1 der Testbench benutzt, die anstelle des 3-AFC-Maßes das ODG-Maß zur Klassifikation verwendet[7].

Die Testbench liefert als Ergebnis das von MEDI entwickelte objektive Gütemaß ODG (Objective Difference Grade), das durch einen Skalar zwischen 0 (Qualitätsverlust nicht wahrnehmbar) und -4 (Qualitätsverlust wirkt sehr störend) repräsentiert wird. Für eine akzeptable Qualität streben wir ein ODG zwischen 0 und -1 (Qualitätsverlust wahrnehmbar, aber nicht störend) an.

Eine akzeptable Qualität (ODG > -1) liefert eine Gleitkommafilterung mit einer Fraktionsbreite von 6 Bit und einer Exponentenbreite von 6 Bit (Abb. 5.4 bzw. 5.5, ODG ≈ -0.75). Es wurde zudem festgestellt, dass der Dynamikbedarf stark von dem untersuchten Datensatz abhängt. Das Glockenspiel hat mit Abstand den höchsten Dynamikbedarf (Abb. 5.6), was vermutlich auf die hohen Obertonanteile dieses Musikinstruments zurückzuführen ist.

Die Ergebnisse der Festkommaresynthese sind in Abb. 5.7 dargestellt. Es wurde so skaliert, dass keine Überläufe stattfinden. Eine Berechnungsgenauigkeit von 14 Bit ist hier ausreichend, um den ODG für alle Samples größer -1 zu halten.

Problematisch war bei der Festkommaimplementierung jedoch der Fall bestimmter künstlich generierter voll ausgesteuerter Signale, die zu Überläufen führen, so z. B. eine einfache Sinusschwingung. Da dies sofort zu einem starken Einbruch der Qualität führen würde, wurde die interne Berechnungsgenauigkeit der Addition auf 25 Bit angehoben. Die Probleme traten jedoch nur bei künstlichen Signalen auf, für die Samples der MEDI-Testbench konnten keine Überläufe festgestellt werden. Um eine direkte

[7] Näheres zur Testbench findet sich in Kap. 4.4.

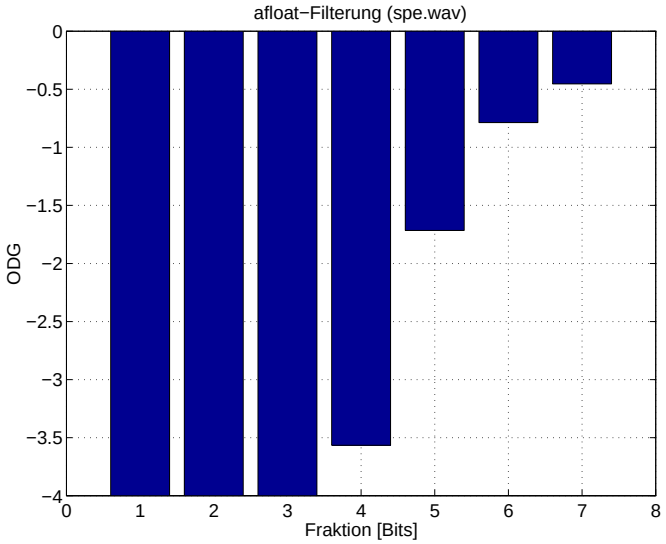


Abbildung 5.4.: Ergebnisse der Sprachfilterung mit variabler Fraktion. Der Exponent wurde hier so hoch gewählt, dass keine Überläufe auftraten (s.u.). Für eine akzeptable Qualität (ODG > -1) sind sechs Fraktionsbits notwendig.

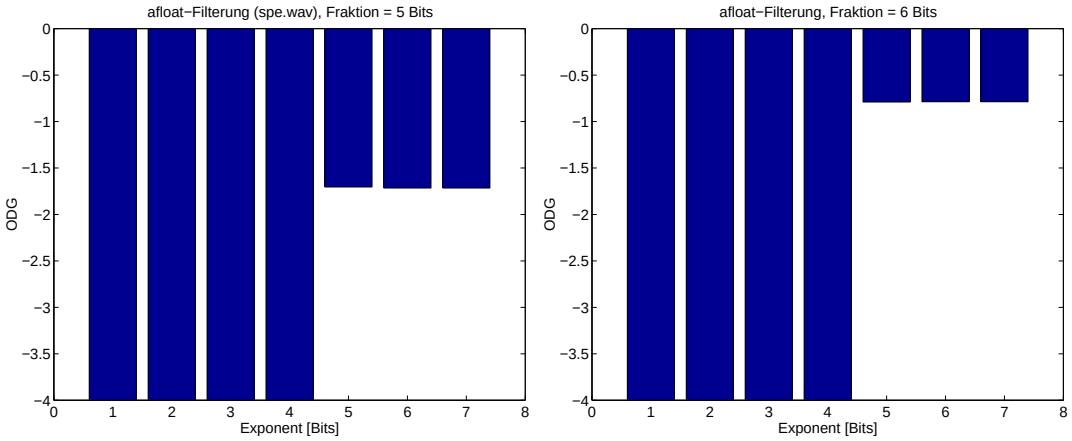


Abbildung 5.5.: Ergebnisse der Sprachfilterung mit variablem Exponenten. Die Fraktion wurde dabei im linken Diagramm auf 5 Bit und im rechten auf 6 Bit festgelegt. Bis zu einer Exponentenbreite von 4 Bit treten Überläufe auf, die Audioqualität bewegt sich im inakzeptablen Bereich. Mehr als fünf Exponentenbits bringen keine Qualitätsverbesserung mehr.

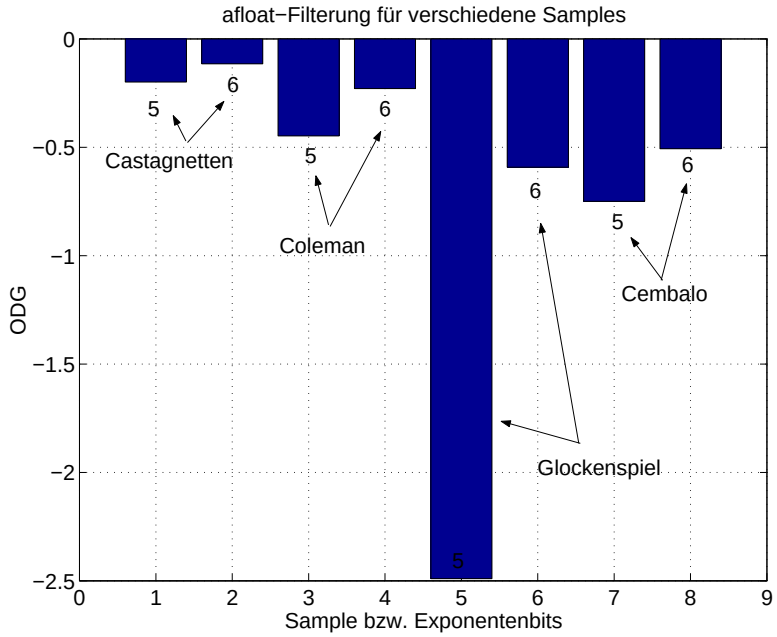


Abbildung 5.6.: Ergebnisse der Gleitkomma-Resynthese für unterschiedliche Samples. Dargestellt sind die ODG-Klassifikationen für vier verschiedene Audiosamples aus dem repräsentativen Ensemble mit jeweils fünf und sechs Bits für den Exponenten, für die Fraktion wurden hier sechs Bits verwendet.
Beim Glockenspiel treten bei der Verwendung von nur fünf Exponentenbits noch verstärkt Überläufe auf, worunter die Qualität stark leidet. Bei allen anderen Samples wird die Qualität nur unwesentlich schlechter, was auf eine geringe Zunahme der Überläufe beim Wechsel von fünf zu sechs Bits für die Exponentenrepräsentation schließen lässt.

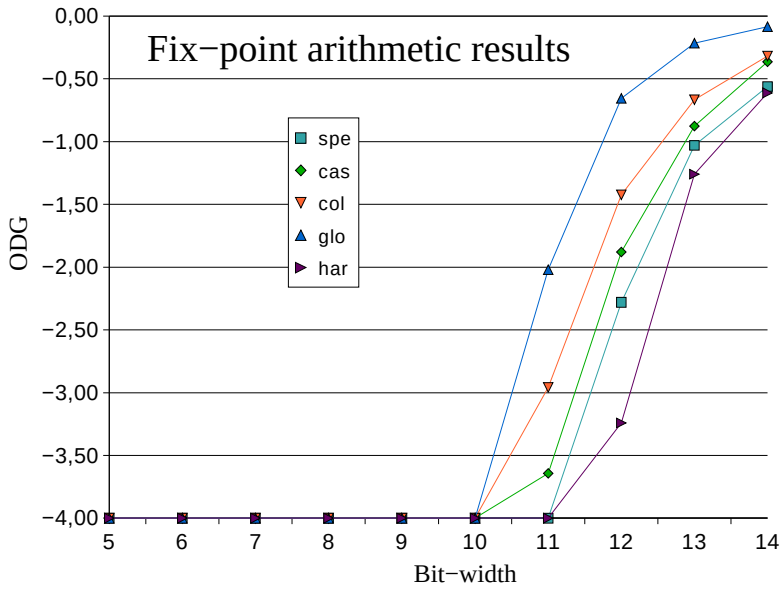


Abbildung 5.7.: Ergebnisse der Resynthese für Festkommazahlen. Hier sind alle Audio-Samples des repräsentativen Testsatzes der Testbench I eingetragen (siehe auch Abschnitt 4.4.2). Ab 14 Bit wird für alle Samples eine gute Qualität erreicht.

Vergleichbarkeit der Ergebnisse mit der Projektgruppenimplementierung aufrechtzuerhalten, wurde die Quantisierung trotzdem übernommen.

Zusammenfassung

In Anbetracht des erheblichen Aufwands einer Fließkommaimplementierung (siehe Kap. 4.6) erscheint hier ein Festkommavariante günstiger, insbesondere da dieses Design sehr gut parallelisierbar ist und viele Gleitkommakomponenten sehr viel Platz benötigen würden. Zudem ist die benötigte Festkommagenauigkeit mit nur 16 Bit sehr gering.

Obgleich ein interne Berechnungsgenauigkeit von 14 Bit befriedigende Ergebnisse mit der Testbench I lieferte, wurde die von der Projektgruppe aufgrund von Problemen mit künstlichen Signalen gewählte Quantisierung beibehalten, um eine direkte Vergleichbarkeit zu gewährleisten. Eine Reduktion auf 14 Bit wäre durch einfache Anpassungen aber leicht möglich, hätte aufgrund der Dominanz der Multiplizierer jedoch keine dramatischen Auswirkungen auf den Leistungsbedarf oder die Größe des resultierenden Chips (s.u.).

5.2.3. Implementierung

Analysefilterbank

Für die Analysefilterbank erfolgte eine Neuimplementierung in Behavioral VHDL, so dass verschiedene Parallelisierungsvarianten exploriert werden konnten. Durch eine

geschickte Implementierung können durch den DesignCompiler automatisch Konstantenmultiplizierer inferiert werden, dazu ist allerdings ein relativ hoher Parallelisierungsgrad notwendig, da jeder der 30 Bandpassfilter einen anderen Koeffizienten hat.

Bei dem minimalen Parallelisierungsgrad, bei dem die Inferenz von Konstantenmultiplizierern möglich ist, müssen von einem Multiplizierer die vier Filtermultiplikationen hintereinander ausgeführt werden, zudem ist ein reellwertiger Konstantenmultiplizierer notwendig (s.o.).

Der Ressourcenbedarf liegt in diesem Falle bei 30 komplexwertigen Addierern und Konstantenmultiplizierern mit 24 Bit und 30 reellwertigen Addierern und Multiplizierern mit 16 Bit. Der Takt liegt bei einer Abtastfrequenz von 16.276 Hz für das Audiosignal bei 130 kHz, da die komplexwertigen MAC-Einheiten pro Stereo-Sample acht Operationen durchführen müssen.

Steht der Platz für diese Ressourcen auf dem Zielbaustein nicht zur Verfügung, können keine Konstantenmultiplizierer mehr inferiert werden, da jeder Multiplizierer nun mehr als einen Kanal bearbeiten muss, womit die Multiplizierer je nach Zielbaustein sehr viel größer werden. Weiterhin wird eine Multiplexerstruktur zum Laden der Koeffizienten benötigt, die zusätzlichen Platz- und Energiebedarf hat. Unter Energiegesichtspunkten ist eine solche Architektur deshalb wesentlich ungünstiger.

Resynthesefilterbank

Ausgehend von den Analyseergebnissen wurde eine Neuimplementierung der Gammatonresynthesefilterbank vorgenommen. Die verwendete Quantisierung entspricht derjenigen der Projektgruppe [84], was den weiteren Vorteil hat, dass die Ergebnisse direkt vergleichbar sind.

Um die Resynthese in das im Silicon-Ear Projekt entstandene System integrieren zu können, sollte die Schnittstelle beibehalten werden. Um parallele Architekturen zu ermöglichen, war es deshalb nötig, vor die Resynthese einen entsprechenden Seriell-Parallel-Konverter zu platzieren (Abb. 5.8).

Die variable Parallelisierung wird dabei über eine ausgerollte Schleife realisiert, die in eine eingerollten Schleife eingebettet ist. Dabei muss die Anzahl der Iterationen der inneren Schleife mal der Anzahl der Iterationen der äußeren Schleife der Gesamtzahl der Frequenzkanäle entsprechen:

$$N_F \equiv N_i \cdot N_a \qquad N_F, N_i, N_a \in \mathbb{N} \tag{5.7}$$

wobei N_F die Anzahl der Frequenzkanäle und N_i bzw. N_a die Iterationen der inneren bzw. äußeren Schleife ist.

Für $N_F = 30$ ergeben sich somit acht verschiedene parallele Architekturen zwischen einem und 30 Durchläufen der äußeren Schleife, wobei bei nur einem Durchlauf der maximale Parallelisierungsgrad erreicht ist. Das entsprechende VHDL-Code-Fragment der Schleifenkonstruktion ist im folgenden wiedergegeben:

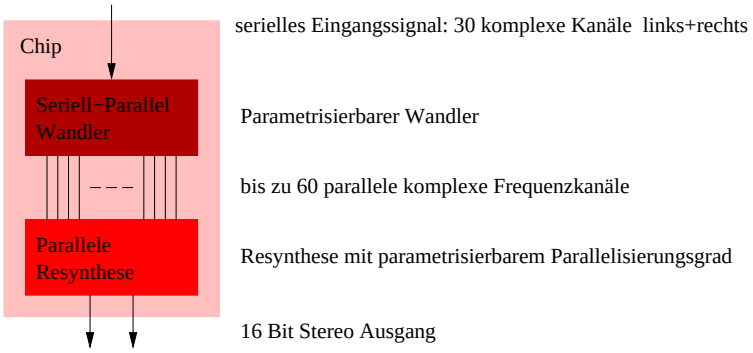


Abbildung 5.8.: Resynthese mit parametrierbarem Seriell-Parallel-Wandler. Die Parallelisierungsstufe des Gesamtchips kann über einen Parameter gesteuert werden, der sowohl die Resynthese als auch den vorgeschalteten Seriell-Parallel-Konverter entsprechend modifiziert. Die Außenanschlüsse des Gesamtchips sind immer gleich und stellen ein serielles Eingangsinterface zur Verfügung. Dies ist zum einen sinnvoll, um den Chip in die im Silicon-Ear Projekt erarbeitete Umgebung einzubauen; Zum anderen vereinfacht dies den Test, da immer die gleiche Testbench zum Einsatz kommt. Eine Prototypen-Produktion des Chips ist zudem nur mit dem seriellen Interface möglich, da eine parallele Variante ca. 2000 Pins benötigen würde.

```
type loop_cnt is
  array(0 to 7) of integer range 0 to FREQCHANNELS;

constant outer_loop_cnt :
  loop_cnt := (30, 15, 10,  6,  5,  3,  2,  1);

ch_loop:
for rolled_ch in 0 to outer_loop_cnt(rolled_index)-1 loop

  ch_un_loop:
  for unrolled_ch in 1 to outer_loop_cnt(7-rolled_index) loop

    freqchannel :=
     rolled_ch*outer_loop_cnt(7-rolled_index)+unrolled_ch-1;
```

Die Auswahl der gewünschten Variante erfolgt über die Variable `rolled_index`, der aktuelle Frequenzkanal wird in der letzten Zeile berechnet. In der ausgerollten Schleife `ch_un_loop` erfolgt die Berechnung beider Stereokanäle (komplexe Multiplikation und Verzögerung).

Je nach Synthesevorschrift für den Behavioral Compiler kann die innere Schleife in einem oder mehreren Taktzyklen mit entsprechend geänderten Ressourcen realisiert werden. Dies entspricht einer parallelen bzw. einer sequentiellen Verarbeitung beider Stereokanäle. Es hat sich jedoch als energetisch günstiger erwiesen, die innere Schleife parallel zu realisieren, da durch den symmetrischen Aufbau des linken und rechten Stereokanals auf diese Weise Multiplexer eingespart werden.

Der Verzögerungsspeicher wird über einen Ringpuffer realisiert, der über eine Speicherarray realisiert ist (Abb. 5.9). Der Behavioral Compiler kann das Array über eine

entsprechende synthetische Bibliothek sowohl auf Hardware-RAM-Speicher abbilden, als auch ein Registerfile oder individuelle Register inferieren (siehe Kapitel 2.4). Da aus einem Dual-Ported RAM nur auf zwei Werte pro Zyklus zugegriffen werden kann, ist eine Realisierung über RAM für eine Parallelisierung nur dann geeignet, wenn nicht mehr als zwei Frequenzkanäle gleichzeitig verarbeitet werden.

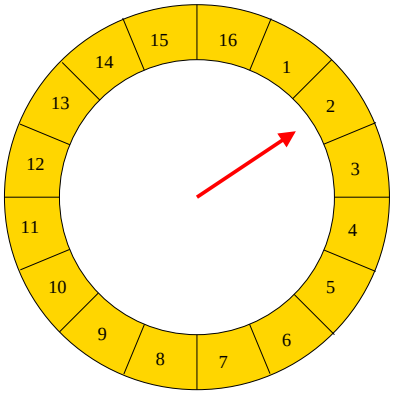


Abbildung 5.9.: Prinzip des Ringpuffers für die Verzögerung von Samples. In diesem Beispiel hat der Puffer 16 Speicherplätze entsprechend einer maximalen Verzögerung von 16 Takten. Die reale Implementierung hat 56 Speicherplätze. Bei einer seriellen Verarbeitung der Frequenzkanäle wird der Zeiger (rot) um die relative Verzögerung vom n. zum $n+1$. Frequenzkanal inkrementiert und der Wert des Kanals auf die entsprechende Zelle aufaddiert. Nach Abarbeitung aller Kanäle wird der Zeiger nochmals um eins inkrementiert und steht jetzt auf der Zelle, dessen Wert auf die unverzögerten Kanäle aufaddiert werden muss. Der Startpunkt des Zeigers verschiebt sich somit mit jedem Takt um eine Stelle im Uhrzeigersinn.

Wird RAM für den Ringpuffer eingesetzt, kann die innere Schleife in minimal drei Taktzyklen abgearbeitet werden, da das Handshaking für das Lesen und Schreiben des RAMs diese Zeit benötigt. Für eine Registerrealisierung des Ringpuffers kann die innere Schleife in nur einem Taktzyklus abgearbeitet werden. Diese Variante hat sich als die Effizienteste herausgestellt (s.u.).

Da der Suchraum des Behavioral Compilers mit zunehmender Parallelisierung stark zunimmt und die Speichernutzung für User-Prozesse auf dem verwendeten System auf 3 GB begrenzt war, konnten mit dem oben dargestellten Verfahren der eingebetteten Schleife keine vollständig parallele Variante erzeugt werden. Aufgrund des unten beschrieben Verhaltens des Behavioral Compilers ist dies allerdings auch nicht sinnvoll.

Um dennoch ein vollständig parallele Architektur zu generieren, wurde der VHDL-Quelltext modifiziert, so dass er im wesentlichen nur noch aus einer vollständig ausgerollten Schleife besteht.

5.2.4. Messwerte

Experimentelles Setup

Die folgenden Messwerte sind für den UMC ASIC Prozess mit einer Strukturbreite von 0,18 μm und einer Betriebsspannung von 1,8 V ermittelt worden. Aus Gründen des

begrenzten Hauptspeichers sind nicht alle acht Parallelisierungsgrade implementiert worden, sondern nur die vollständig serielle Version („p-0“), sowie die Versionen mit zwei („p-2“), drei („p-3“) und fünf („p-5“) gleichzeitig berechneten Frequenzkanälen. Die ebenfalls möglichen Varianten mit sechs, zehn und fünfzehn gleichzeitig berechneten Frequenzkanälen waren auf der vorhandenen Hardware nicht synthetisierbar, da der Hauptspeicher dafür nicht ausreichte[8].

Weiterhin ist die vollständig ausgerollte und im Quelltext optimierte Version („p-30“) synthetisiert worden. Für die Parallelisierungsvarianten p-0 bis p-15 wurde der gleiche VHDL-Quelltext mit unterschiedlichem `generic`-Parameter für den Ausrollgrad verwendet, die Variante p-30 ist eine handoptimierte, vollständig ausgerollte Version des gleichen Quelltextes.

Es wurde exakt der in Abschnitt 4.1.4 beschriebene Entwurfsfluss verwendet.

Takt

Da das Design aus der Resynthese mit dem vorgeschalteten Seriell-Parallel-Konverter besteht, gibt es zwei Taktdomänen. Der Takt des Konverters lag konstant bei 3 MHz. Der Takt der Resynthese wurde jeweils so gewählt, dass der geforderte Durchsatz von 16.276 Samples/s erzielt wurde, d. h. für die parallelen Varianten wurde der Takt entsprechend reduziert (Tabelle 5.1), wobei stets ein „Sicherheitsabstand“ eingeplant wurde, um keine Probleme mit der Synchronisation der beiden Designblöcke zu bekommen. Der Takt des Konverters wurde mit 3 MHz deutlich höher gewählt als die eigentlich erforderlichen 1 MHz (60 Kanäle · 16.275 Hz ≈ 1 MHz). Dies liegt darin begründet, dass für eine einfache Synchronisation der Blöcke die Taktperiode des Konverters ein ganzzahliger Teiler der Taktperiode der Resynthese sein muss.

Der Takt der Resynthese wird in einem Skript automatisch aus dem Parallelisierungsparameter nach folgender Formel berechnet:

$$T_r \equiv T_k \cdot (N_p + 1) \qquad T_r, T_k, N_p \in \mathbb{N} \tag{5.8}$$

Dabei ist T_r die Taktperiode der Resynthese, T_k die (feste) Taktperiode des Konverters und N_p die Anzahl der Iteration der äußeren (eingerollten) Schleife. Der resultierende Takt T_r ist als Optimierungsvorgabe für den Synthesevorgang verwendet worden, um möglichst gut angepasste Ergebnisse zu erzielen.

Variante	p-0	p-2	p-3	p-5	p-6	p-10	p-15	p-30
Takt [kHz]	1000	606	433	275	233	144	98	50

Tabelle 5.1.: Taktfrequenzen für die verschiedenen Parallelisierungsvarianten der Resynthese. Diese Taktraten sind Vorgaben für den Syntheseprozess, um den geforderten Durchsatz von 16.276 Samples/s zu erzielen. Sie wurden mittels Gl. 5.8 ermittelt. Die Varianten p-6, p-10 und p-15 konnten nicht synthetisiert werden (s.o.), sind aber der Vollständigkeit halber mit aufgeführt.

[8] Für die Parallelisierung muss immer ein ganzzahliger Teiler der insgesamt 30 Frequenzkanäle verwendet werden, um durch Replikation der Einheiten genau 30 Kanäle abzudecken.

Energiebewertung

Aufgrund der Ungenauigkeit der PowerCompiler-Ergebnisse sind hier nur die mit einer dynamischen PrimePower-Analyse ermittelten Werte wiedergegeben. Dazu wurden jeweils 500 ms des Sprachsamples mit der PEMO_Q-Testbench auf Floorplan-Ebene[9] mittels des `ncsim`-Simulators von Cadence verarbeitet.

Dabei wurden die von UMC für die Verilog-Simulation mitgelieferten Power-Modelle benutzt, die aufgrund der zustands- und pfadabhängigen Beschreibung für diese Low-Power Standardzellenbibliothek sehr genaue Werte liefern[10].

Ergebnisse

Das Design wurde in Varianten p-0 – p-5 und p-30 synthetisiert, entsprechend einer parallelen Bearbeitung von einem, zwei, drei, fünf und dreißig Frequenzkanälen. Die innere Schleife wurde in einem Takt geschedult, d. h. die Bearbeitung beider Stereokanäle erfolgt parallel.

Weiterhin wurde eine Version ohne Clock-Gating erzeugt sowie zwei Versionen mit aktiviertem Clock-Gating: Bei der zweiten wurde der DesignCompiler angewiesen, die Hierarchie aus dem Design zu entfernen, um ggf. hierarchieübergreifende Optimierungen durchführen zu können.

Weiterhin wurde von dem PhysicalCompiler eine SDF[11]-Datei erzeugt, die aus den RC-Werten der Interconnect-Leitungen zwischen den Standardzellen gewonnene Verzögerungszeiten enthält. Wenn diese mitsimuliert werden, verändert dies das Timing und kann u. U. zu anderen Energieverbrauchswerten führen. Um die Signifikanz zu bestimmen, ist auch jeweils eine Simulation unter Zuhilfenahme der SDF-Datei erfolgt. Wie aus den Abbildungen 5.10–5.12 folgt, lohnt sich der zusätzliche Zeitaufwand von ca. 20% jedoch nicht, da keine signifikante Änderung des Ergebnisses folgt.

V_{dd}-Reduktion durch positiven Zeitoffset bei der Synthese

Durch die Reduktion der Taktfrequenzen für die parallelen Varianten vereinfacht sich für den Compiler die Suche nach einem validen Design, da aufgrund der langen Taktzeit viele Logikkomponenten hintereinandergeschaltet werden können. Dies reduziert auf der einen Seite die Ausführungszeit des Compilers, auf der anderen Seite erübrigt sich daher das Aufbrechen von Operationen und das Einfügen von Zwischenregistern (z. B. beim Pipelining), was sich wiederum positiv auf den Energiebedarf des Chips auswirkt.

Da das geforderte Timing oft mit einer einfachen Architektur problemlos eingehalten werden kann, würde die Schaltung auch noch bei einem höheren Takt funktionieren. Die Zeit bis zum Ende der Taktperiode, die die Schaltung nichts tut, ist im Timing-Report des Compilers als „positive slack“ verzeichnet. Dies lässt sich ausnutzen, um die Betriebsspannung zu reduzieren und damit den Energiebedarf der Schaltung nochmals zu minimieren. Tabelle 5.2 gibt den „positive slack“ für die Resynthesevarianten wieder.

[9] In diesem Fall mit dem PhysicalCompiler platzierte und verdrahtete Standardzellen.
[10] Nähere Informationen zu den Verlustleistungsanalysewerkzeugen finden sich im Abschnitt 4.2.
[11] Structured Delay Format, siehe Abschnitt 4.2.

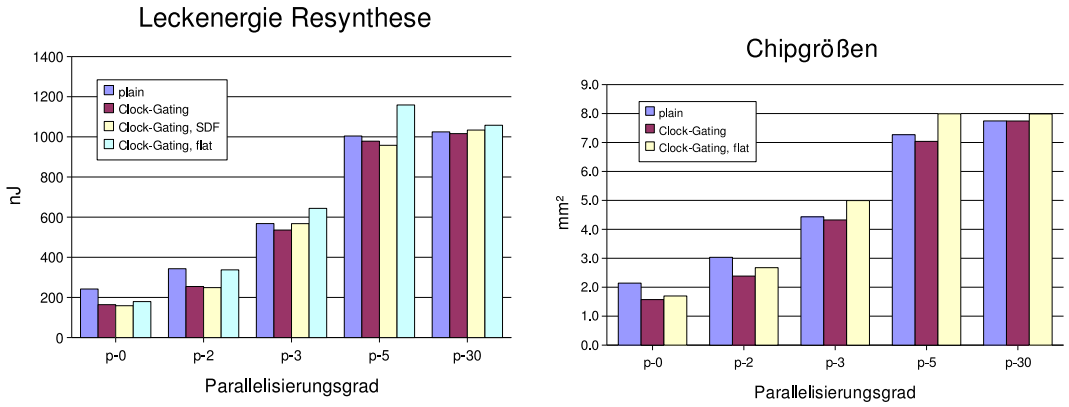


Abbildung 5.10.: Leckenergie und Chipgrößen für die Resynthese. Das linke Diagramm zeigt die Leckenergie des Resyntheseteils des Chips. Der Parallelisierungsgrad steigt von links nach rechts von einer vollständig seriellen Implementierung bis zu einer vollständig parallelen Implementierung (eine genaue Definition findet sich im Text). Die „plain"-Variante bezeichnet jeweils die Version ohne eine Poweroptimierung auf RT-Ebene, die anderen Balken sind mit aktiviertem Clock-Gating gemessen. Bei der „flat"-Variante wurde während des Compilings die Design-Hierarchie entfernt, was hierarchieübergreifende Optimierungen ermöglicht. „SDF" berücksichtigt in der Energieberechnung zusätzlich den Effekt der Interconnect-Kapazitäten. Es lässt sich gut die annähernd lineare Beziehung zwischen Chipgröße und Leckenergie erkennen.

Eine eventuell mögliche Reduktion der Betriebsspannung ist bei der Ermittlung der folgenden Ergebnisse nicht berücksichtigt worden, da es keine systematische Unterstützung durch die Designwerkzeuge für eine korrekte Verlustleistungsanalyse gibt.

Arithmetische Ressourcen

Der Bedarf an arithmetischen Ressourcen beträgt pro Frequenz- und Stereokanal zwei 16 Bit Multiplizierer und einen 32 Bit Addierer für die komplexe Multiplikation, da nur der Realteil benötigt wird. Weiterhin ist ein 24 Bit Addierer nötig, um das Multiplikationsergebnis aufzuaddieren. Pro Stereokanal ist ein weiterer 24 Bit Addierer notwendig, um aus dem aktuellen Summenwert und dem verzögerten Summenwert das Endergebnis zu berechnen. Für die Verwaltung der Ringpuffer sind weiterhin mehrere kleine Addierer und Komparatoren nötig, die hier vernachlässigt werden. Die Zusammenfassung der arithmetischen Ressourcen gibt Tabelle 5.3 wieder.

Interessanterweise vergrößert sich der Chip beim Sprung von p-5 auf die optimierte Version p-30 nur noch minimal (Abb. 5.10), obwohl sich der Bedarf an arithmetischen Ressourcen versechsfacht. Dies ist zum einen auf das vereinfachte Interface zum S/P-Konverter zurückzuführen, da der Behavioral Compiler hier keine Sampling-Register mehr generieren muss, um die beiden Designteile zu entkoppeln. Auf der anderen Seite

	p-0	p-2	p-3	p-5	p-30
Taktperiode [ns]	990	1650	2310	3630	20000
Slack [ns]	900	1270	1970	3510	19600

Tabelle 5.2.: Positiver Zeitoffset (slack) der Resynthese

	p-0	p-2	p-3	p-5	p-30
16 Bit Multiplizierer	4	8	12	20	120
24 Bit Addierer	4	6	8	12	62
32 Bit Addierer	2	4	6	10	60

Tabelle 5.3.: Arithmetische Ressourcen der Resynthese

ist der Behavioral Compiler aufgrund des großen Suchraums nicht mehr in der Lage, ein ressourcensparendes Design zu generieren[12].

Chipgröße

Der Chip hat 68 I/0-Ports. Die zu dem 0,18 μm-Prozess gehörende I/O-Bibliothek hat Pad-Zellen mit einer minimalen Breite von 60 μm. Mit dem notwendigen Sicherheitsabstand sowie den Pad-Zellen für die Versorgungsspannung und Masse ist eine quadratische Chipfläche von mindestens ca. 3,5 mm^2 nötig, um alle Padzellen anzuordnen. Diese Chipgröße wird erst ab der Variante p-3 erreicht (Abb. 5.10, dargestellt ist die Gesamtfläche inklusive Padzellen).

Einfluss des Clock-Gatings

Die Gesamtenergie der Resynthese für die verschiedenen Varianten ist in Abb. 5.11 dargestellt. Auffällig ist der große Einfluss des Clock-Gatings auf den Energieverbrauch. Dies ist dadurch begründet, dass die Ringpuffer als Register realisiert sind, die vom DesignCompiler auf Flipflops abgebildet werden. Pro Stereo-Kanal werden $56 \cdot 25$ Bit = 1400 Bit = 175 Byte benötigt, insgesamt also 2800 Flipflops. Ohne Clock-Gating werden sämtliche 2800 Flipflops mit jeder Taktflanke aktiviert. Es werden jedoch nur in der voll parallelen Variante (p-30) auch alle Register beschrieben, die seriellen Version beschreiben nur eine kleinen Teil (p-0 z. B. nur 48 Bit). Da die seriellen Versionen mit einem wesentlich höheren Takt laufen (Tab. 5.1), wirkt sich das eklatant auf den Energiebedarf aus. Der Energiebedarf der „plain"-Variante steigt deshalb proportional zum Takt und der Anzahl unbenutzter Register pro Takt.

Werden diese Register durch Clock-Gating vom Taktnetz getrennt, reduziert sich der Energieverbrauch, da bei den seriellen Varianten die meisten Register nur die Leckenergie benötigen. Ab der Variante p-3 ist der Energiebedarf der Register jedoch nicht mehr signifikant, so dass hier ein Clock-Gating keinen messbaren Vorteil mehr bringt.

Skalierung der parallelen Varianten der Resynthese

Auf Abb. 5.11 ist weiterhin zu erkennen, dass die Gesamtenergie der Resynthese von der Variante p-3 zu p-5 wieder leicht ansteigt. Die Begründung ist hier die gleiche wie

[12] Der Behavioral Compiler geht bei der Auflösung von Schleifen von innen nach außen vor. Es wird also zunächst versucht, für die innere (ausgerollte) Schleife ein optimales Scheduling zu erstellen. Da aber die Schleifenparameter nicht als Konstanten erkannt werden, obwohl sie aus einem konstanten `generic`-Parameter berechnet werden, können diese nicht eliminiert werden und es wird (eigentlich unnötige) zusätzliche Hardware generiert. Je mehr Schleifendurchläufe der Behavioral Compiler ausrollen muss, desto mehr dieser Hardware wird generiert und desto mehr Möglichkeiten für Scheduling und Allokation hat der Compiler.

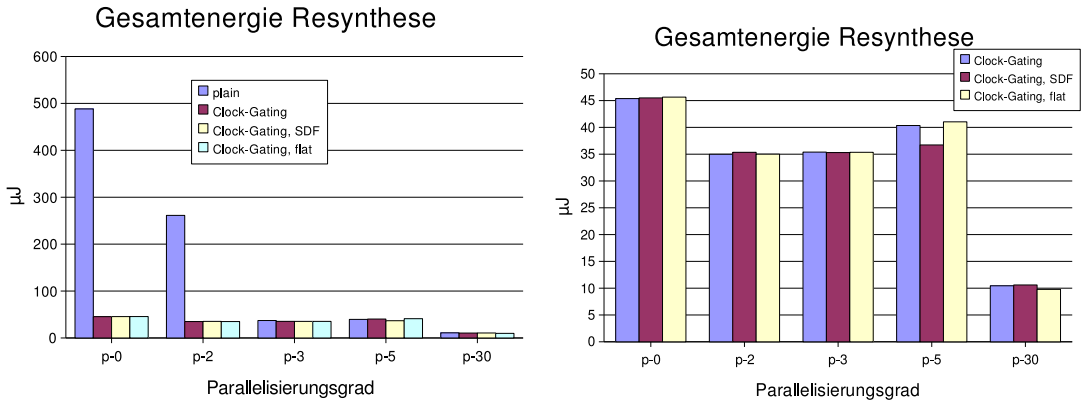


Abbildung 5.11.: Gesamtenergie der Resynthese. Gezeigt ist die Gesamtenergie der Resynthese ohne den Seriell-Parallel-Konverter und die Pad-Zellen. Die „plain"-Variante bezeichnet wieder die Version ohne eine Poweroptimierung auf RT-Ebene, die anderen Balken sind mit aktiviertem Clock-Gating gemessen. Das rechte Diagramm zeigt eine Vergrößerung des linken ohne die „plain"-Variante, um die Unterschiede klarer herauszuarbeiten. Auffällig ist der erhebliche Einfluss eines Clockgatings (siehe Abschnitt 4.3) insbesondere bei den seriellen Varianten. Dies ist auf den erheblich höheren Takt zurückzuführen, der notwendig ist, um bei einem seriellen Design den geforderten Durchsatz zu erzielen (siehe Text). Das rechte Diagramm zeigt, dass ab „p-5" wieder eine Verschlechterung des Energieverbrauchs beobachtet wird. Gleichzeitig steigen der Zeitbedarf für das Scheduling extrem an (nicht dargestellt). Der Suchraum für den Behavioral Compiler ist jetzt so groß, dass die gefundene Variante suboptimal ist. Eine manuelle Einschränkung des Suchraums in der Variante „p-30" bringt dann wieder die erwartete Verbesserung (siehe Text).

bei der Erklärung der nur minimal gestiegenen Chipfläche (s.o.): Der Behavioral Compiler ist ohne manuelles Eingreifen nicht mehr in der Lage, ein optimales Scheduling zu erzeugen. Hilft der Designer nach und entfernt die äußere (eingerollte) Schleife, kann der Energieverbrauch nochmals um ca. 70% reduziert werden. Der Energiebedarf liegt dann bei ca. 10 μJ für ein 500 ms langes Sprachsample. Zum Vergleich wurde das Projektgruppendesign mit der gleichen Methodik energiebewertet, es benötigt für das gleiche Sample 2070 μJ, also ca. das 200-fache[13].

Gesamtenergie des Chips

In Abb. 5.12 ist der Gesamtenergiebedarf des resultierenden Chips dargestellt, der außer der reinen Resynthese noch den Seriell-Parallel-Konverter und die Padzellen beinhaltet. Die Energie ist mit rund 280 μJ in der Variante p-30 etwa 28-mal höher als der Energiebedarf der Resynthese (Abb. 5.11).

Dies ist auf den hohen Energiebedarf des Seriell-Parallel-Wandlers zurückzuführen, der ebenfalls in Abb. 5.12 dargestellt ist. Dies ist im wesentlichen durch dessen hohe Betriebsfrequenz von 3 MHz zu erklären, die ca. das 60-fache der Resynthesefrequenz in der Variante p-30 und immer noch das dreifache der der Resynthesefrequenz in der

[13] Der Verbrauch ließe sich durch Clock-Gating relativ einfach reduzieren. Allerdings liegt man auch damit noch nicht auf dem Niveau der parallelen Implementierung. Der Quellcode ist so strukturiert, dass der Behavioral Compiler keine Parallelisierung der Arbeitsschleife vornehmen kann. Daher ist ein höherer Takt nötig, um den geforderten Durchsatz zu erreichen.

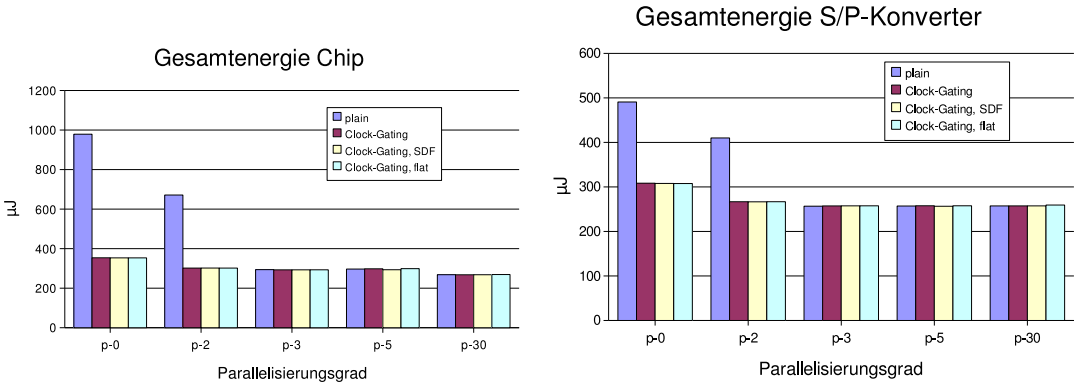


Abbildung 5.12.: Gesamtenergie des Chips und Energie des S/P-Konverters. Das linke Diagramm zeigt den Energiebedarf der platzierten und verdrahteten Standardzellen inklusive der Pad-Zellen. Der Minimalbedarf liegt für die Variante „p-30“ mit rund 280 μJ nur unwesentlich geringer als die Varianten „p-2“-„p-5“. Lediglich die vollständig serielle Variante („p-0“) liegt mit rund 350 μJ etwas höher. Wie in Abb. 5.11 sticht die „plain“-Variante für die stark seriellen Architekturen vor (siehe Text). Die geringen Unterschiede sind durch den hohen Energiebedarf des Konverters zu erklären, der mit einem verhältnismäßig hohen Takt von 3 MHz läuft und in den parallelen Varianten rund 250 μJ oder ca. 80% des Gesamtenergiebedarfs für den Chip ausmacht.

Variante p-0 beträgt.

Das Clock-Gating hat hier erwartungsgemäß einen viel geringeren Einfluss als bei der Resynthese, da die großen und schnell getakteten Pad-Zellen davon nicht betroffen sind. Das dennoch für die Varianten p-0 und p-2 ein signifikanter Unterschied zu verzeichnen ist, beruht auf einer Rückwirkung der angeschlossenen Resynthese: Durch das hier erfolgte abkoppeln der nicht benutzten Register reduziert sich die Treiberlast der als Register ausgeführten Zwischenspeicher des Seriell-Parallel-Konverters.

5.2.5. Bewertung der Resyntheseuntersuchungsergebnisse

Der Energiebedarf der reinen Resynthese konnte gegenüber der Projektgruppenimplementierung von 2070 μJ auf 10 μJ auf ca. 0,5% reduziert werden. Die Einsparung wurde im wesentlichen durch eine vollständige Parallelisierung erreicht, der Flächenbedarf stieg deshalb auf das Fünffache von ca. 1,5 mm^2 auf 7,5 mm^2. Mit einer moderaten Parallelisierung nur der arithmetischen Struktur für einen Frequenzkanal ist bereits eine Reduktion des Energiebedarfs auf ca. 45 μJ möglich, ohne die Chipfläche signifikant zu erhöhen, das entspricht 2,2% des Projektgruppendesigns. Dazu ist allerdings zusätzlich ein Clock-Gating notwendig, das moderne kommerzielle Synthesewerkzeuge aber automatisch ermöglichen.

Für den verwendeten 0,18μm Prozess von UMC ist der Einfluss der Leckenergie, der ungefähr proportional zur Chipfläche ist, nicht mehr zu vernachlässigen; In der voll parallelen Version beträgt sein Anteil an der Gesamtenergie mit ca. 1 μJ immerhin 10%. Für Prozesse mit geringeren Betriebsspannung ist eine weitere Zunahme des Einflusses der Leckenergie zu erwarten.

Bei den parallelen Varianten ist eine weitere drastische Reduktion der Verlustleistung durch Absenken der Betriebsspannung möglich (vgl. Tab. 5.2). In Ermangelung entsprechender Dokumentation für das Timing-Verhalten der Standardzellen bei einem Betrieb unterhalb der zulässigen Minimalspannung sind hier aber keine Berechnungen unternommen worden.

Durch das im Silicon-Ear Projekt festgelegte serielle Eingangsdatenprotokoll war es nötig, die Resynthese an einen Seriell-Parallel-Konverter zu koppeln. Da dieser mit einem sehr hohen Takt läuft, dominiert dieser mit ca. 80% den Gesamtenergiebedarf des Chips. Der Konverter kann weggelassen werden, wenn der Parallelisierungsgrad der Resynthese mit der Parallelität der Eingangsdaten übereinstimmt. Für einen Demonstrator-Chip sollte deshalb die gesamte Verarbeitung mit dem gleichen Parallelisierungsgrad implementiert werden, um Protokollkonvertierungen zu vermeiden.

5.3. Die Kurzzeit-Fouriertransformation

Die Kurzzeit-Fouriertransformation mit anschließender inverser Transformation und überlappender Addition (*Overlap-Add*) ist neben der oben beschriebenen Gammatonfilterbank das zweite Gerüst für den Aufbau von Audioalgorithmen. Im Gegensatz zu letzterem hat die Kurzzeit-Fouriertransformation keinen psychoakustischen Hintergrund, sondern wurde aus der Nachrichtentechnik übernommen. Zudem ist die schaltungstechnische Realisierung ressourcenaufwändiger, weshalb das Gammaton-Framework nach Möglichkeit vorzuziehen ist. Jedoch sind noch nicht alle Algorithmen unseres Projektpartners MEDI angepasst, so dass an dieser Stelle auf die Add-Overlap-FFT eingegangen werden soll.

5.3.1. Prinzip

Das Prinzip der Kurzzeit-FFT beruht auf einer Aufteilung des Eingangssignals in zeitlich überlappende Blöcke und anschließende Transformation in den Frequenzraum. Dann werden die Daten durch einen Audioalgorithmus bearbeitet, um anschließend nach der Rücktransformation überlappend aufaddiert zu werden (Abb. 5.13). Die Grundlagen zu der FFT finden sich in Abschnitt 2.9.

Die gewählte Blockgröße stellt einen Kompromiss zwischen Auflösung im Zeit- und Frequenzraum dar, zudem wächst der algorithmische Aufwand mit der Blockgröße n proportional zu der Zeitkomplexität der schnellen Fouriertransformation von $n \log n$.

Die zu transformierenden Eingangsblöcke werden beim *Overlap-Add*-Verfahren vor der Transformation mit einer Fensterfunktion aus dem Eingangssignal ausgeschnitten. Die Multiplikation des Eingangssignals mit einer Fensterfunktion im Zeitbereich entspricht einer Faltung im Frequenzbereich mit der fouriertransformierten Fensterfunktion, dadurch kommt es zu einer Verzerrung des Signalspektrums, dem sog. *Leakage*-Effekt.

Da der Signalausschnitt für eine DFT implizit periodisch fortgesetzt wird, kann es an den Rändern zu Unstetigkeiten kommen, die Alias-Frequenzen in das Spektrum einführen. Man verwendet daher gern Fensterfunktionen, die zu den Rändern stetig bis auf Null abfallen und füllt den Signalausschnitt dann mit Nullen auf, womit das Spektrum zwar auch verschmiert und außerdem gedämpft wird, dafür aber keine Aliase auftreten.

Die Anzahl der Füll-Nullen muss der Länge der Faltungsfunktion entsprechen, in diesem Falle also mit der gleichen Anzahl Nullen wie die Eingangsvektorlänge. Für ein 128-Punkte FFT nimmt man daher Blöcke der Größe 64, fenstert diese und füllt sie mit 64 Nullen auf. Wegen der Linearität der DFT kann man die so bewirkte Dämpfung durch eine Überlappung der Fenster (in unserem Fall über das halbe Fenster) kompensieren. Die Ergebnisse der Rücktransformation können dann einfach aufaddiert werden, daher der Name *Overlap-Add*-Verfahren.

Die für das Framework-Modul verwendete Fensterfunktion ist das sogenannte *Hanning*-Fenster der Länge T:

$$\mathbf{w}(t) = 1 - \cos\left(\frac{\pi \cdot t}{T}\right), \qquad 0 \leq t < T \tag{5.9}$$

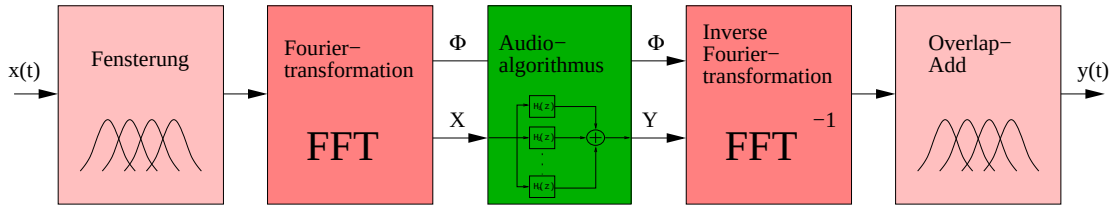


Abbildung 5.13.: Struktur des Kurzzeit-FFT Frameworks. Zunächst wird eine Fensterung des Eingangssignals $x(t)$ durchgeführt, so dass eine blockweise Weiterverarbeitung erfolgen kann. Die einzelnen Blöcke (Frames) werden dann fouriertransformiert. Der Betrag der Transformierten $X(f)$ wird in der folgenden Audioverarbeitung manipuliert (z. B. durch eine Störgeräuschunterdrückung), die Phase Φ bleibt meist unverändert. Dann folgt eine Rücktransformation der Blöcke vom Frequenzraum in den Zeitraum und eine anschließende überlappende Addition der Frames.

5.3.2. Analyse und Partitionierung

Das Kurzzeit-Fouriertransformationsframework zerfällt in vier Arbeitsschritte (Abb. 5.13):

Fensterung der Eingangsdaten Die Eingangssamples werden in zeitlich überlappende Verarbeitungsblöcke eingeteilt. Gleichzeitig werden die Blockdaten mit aus der Fensterfunktion bestimmten Gewichtungsfaktoren multipliziert und mit Nullen aufgefüllt (s.o.).

Transformation in den Frequenzraum Die gefensterten Blöcke werden mittels einer FFT in den Frequenzraum transformiert.

Rücktransformation in den Zeitraum Nach der Manipulation der Daten durch einen Audioalgorithmus werden die Frequenzdaten durch eine inverse FFT in den Zeitbereich zurücktransformiert.

Overlap-Add Die rücktransformierten Blöcke werden überlappend superpositioniert.

Struktur

Da die Eingangsdaten blockweise verarbeitet werden und keine Aufspaltung in mehrere Kanäle erfolgt, ist eine strukturelle Parallelisierung wie bei der Gammatonfilterbank hier nicht möglich. Aufgrund des verhältnismäßig hohen Ressourcenbedarfs für eine schnelle Fouriertransformation, die aus einem oder mehreren identischen Blöcken (meistens sog. Butterflys oder Dragonflys, siehe Abschnitt 2.9) besteht, wäre dies ohnehin nur sehr begrenzt sinnvoll.

Da zur Steuerung des oben geschilderten Ablaufs weiterhin nur ein sehr einfacher Kontrollfluss nötig ist, das allenfalls ein Multiplexing und/oder ein Pipelining realisieren muss, erscheint eine verhaltensbasierte Codierung in dieser Stufe der Designhierarchie

nicht sinnvoll, da die Vorteile des mit dem Behavioral Compiler einhergehenden variablen Scheduling nur sehr begrenzt nutzbar wären und die vom Compiler generierten Kontrollstrukturen das Design unnötig komplex machen würden.

Für eine effiziente Implementierung bietet sich daher in diesem Fall im Unterschied zu der Gammatonfilterbank eine Codierung auf RT-Ebene an, in der die benötigten Ressourcen instantiiert werden und die einen Zustandsautomaten für die Steuerung zur Verfügung stellt, der eventuell sinnvolles Pipelining (s.u.) realisiert.

Dies bedeutet nicht, dass die in das Verarbeitungsframework einzusetzende Audioverarbeitung (der grüne Block in Abb. 5.13) auch als RT-Komponente realisiert sein muss. Für die in Abschnitt 5.4 diskutierte Rauschunterdrückung, die auf diesem Framework beruht, wurde eine Behavioralcodierung gewählt (s.u.). Deren Taktverhalten muss der Steuerung des Frameworks allerdings bekannt sein.

Weiterhin ist die schnelle Fouriertransformation ein gut untersuchtes Problem (siehe Abschnitt 2.9), für das es bereits eine Reihe für die jeweilige Zieltechnologie optimierter IP-Cores gibt. Es sollte daher möglich sein, diese Komponenten einfach einsetzten und austauschen zu können, d. h. die Fensterung und das Overlap-Add sollten nicht mit der Fouriertransformation verschmolzen werden.

5.3.3. Implementierung

Die obigen Überlegungen lassen zwei Architekturen für das Kurzzeit-FFT Framework sinnvoll erscheinen:

1. Eine Variante für Audioalgorithmen mit komplexen Kontrollpfad[14] (z. B. die unten diskutierte Störgeräuschunterdrückung), die mit nur einer Modulinstanz für die schnelle Fouriertransformation auskommt. Diese Instanz wird dann sowohl für die Hin- als auch für Rücktransformation genutzt.
2. Für Algorithmen mit einem verhältnismäßig einfachem Kontrollfluss ist eine Variante mit zwei parallel arbeitenden FFT-Instanzen vorgesehen. Damit ist es möglich, Hin- und Rücktransformation aufeinanderfolgender Blöcke gleichzeitig zu erledigen.

Effiziente Algorithmen für die FFT sind z. B. die Radix-2, Radix-4 oder Split-Radix Zerlegungen (siehe Abschnitt 2.9). Es handelt sich hierbei um sog. lokale Algorithmen, d. h. sie benötigen nur den Speicher für den Eingangsdatenblock als Arbeitsspeicher.

[14] Ein komplexer Kontrollpfad lässt sich im Allgemeinen schlecht parallelisieren, so dass die Algorithmus-Implementierung mit einem relativ hohen Takt arbeiten muss. Es steht deshalb so viel Zeit zur Verfügung, dass die Fouriertransformation Leerlaufzeiten hat.

5.4. Fallbeispiel: Störgeräuschunterdrückung

Die Ephraim-Malah Störgeräuschunterdrückung wurde im Rahmen der Diplomarbeit von F. Klöckner [82] schaltungstechnisch umgesetzt. Die Quantisierung und der modulare Aufbau erfolgten dabei unter Anwendung des Entwurfsframeworks.

5.4.1. Störgeräuschunterdrückung nach Ephraim-Malah

Die Rauschunterdrückungsregel nach Ephraim-Malah wurde ursprünglich in [44] entwickelt und von der AG MEDI in Oldenburg modifiziert und in Matlab implementiert [94].

Die Störgeräuschunterdrückung basiert auf einer frequenz- und zeitabhängigen Schätzung der Rauschleistung und der Leistung des Nutzsignals. Die Schätzung wird unter der Annahme durchgeführt, dass es sich sowohl beim Rauschen als auch beim Nutzsignal um statistisch unabhängige Zufallsvariablen handelt[15]. Das angenommene Grundrauschen wird dabei in jeder Sprachpause neu justiert [162]. Damit wird über ein Verfahren zur Minimierung des mittleren quadratischen Fehlers das angenommene Rauschen $v(\omega_k, p)$ bei der Frequenz ω_k zum Zeitpunkt p ermittelt.

Eine einfache Subtraktion des geschätzten frequenz- und zeitabhängigen Anteils des Rauschens vom Eingangssignal im Frequenzraum vermindert zwar das Rauschen, führt aber aufgrund der Bandbegrenzung der DFT zu einem Aliasing-Effekt, der neue, vorher nicht vorhandene sinusförmige Störungen in das in den Zeitbereich zurücktransformierte Signal einbringt.

Die Störgeräuschunterdrückung nach Ephraim-Malah gewichtet diesen Wert deshalb mit einer komplexen Regel, die zu folgendem frequenz- und zeitabhängigen Gewichtungsfaktor G führt:

$$\mathrm{G} = \frac{\sqrt{\pi}}{2} \sqrt{\left(\frac{1}{1+R_{\mathrm{post}}}\right)\left(\frac{R_{\mathrm{prio}}}{1+R_{\mathrm{prio}}}\right)} \cdot \mathrm{M}\left[(1+R_{\mathrm{post}})\left(\frac{R_{\mathrm{prio}}}{1+R_{\mathrm{prio}}}\right)\right] \tag{5.10}$$

mit

$$\mathrm{M}(\theta) = \exp\left(-\frac{\theta}{2}\right) \cdot \left[(1+\theta) I_0\left(\frac{\theta}{2}\right) + \theta I_1\left(\frac{\theta}{2}\right)\right] \tag{5.11}$$

I_0 und I_1 sind dabei die modifizierten Bessel-Funktionen erster und zweiter Ordnung.

Dabei werden die A-Posteriori und A-Priori Signal-Rausch-Verhältnisse R_{post} und R_{prio} aus der Spektralkomponente $X(p, \omega_k)$[16] und der geschätzten Rauschleistung $v(\omega_k, p)$ wie folgt bestimmt:

[15] Andere Störgeräuschunterdrückungsregeln machen bestimmte Annahmen über die Korrelation des Nutzsignals und können so (falls diese Annahmen zutreffen) bessere Ergebnisse liefern.

[16] Das ist die für den Zeitschritt p mit der Kurzzeit-FFT in den Frequenzraum transformierte Eingangssignalfolge x_p.

$$R_{\text{post}}(p,\omega_k) = \left.\frac{\left|X(p,\omega_k)\right|^2}{v(\omega_k)} - 1\right|_{\geq 1} \tag{5.12}$$

$$R_{\text{prio}}(p,\omega_k) = (1-\alpha)R_{\text{post}}(p,\omega_k) + \alpha\frac{\left|G(p-1,\omega_k)X(p-1,\omega_k)\right|^2}{v(\omega_k)} \tag{5.13}$$

5.4.2. Implementierung

Die Details der Implementierung stehen in [82] und sollen hier nicht wiederholt werden.

Zusammengefasst wurde folgende Struktur gewählt:

1. Die Transformation in den Frequenzbereich und die Rücktransformation hat als Basis die Add-Overlap-FFT aus Abschnitt 5.3.
2. Für die Implementierung wurde eine Festkommaarithmetik mit Skalierung nach jedem Rechenschritt (*Block-Float*) gewählt. Dabei wurde die benötigte Quantisierungsgenauigkeit für jeden Operator mittels der Framework-Gütemaße bewertet und auf dieser Grundlage festgelegt.
3. Die Berechnung des Gain-Faktors (Gl. (5.10)) ist äußerst komplex. Durch geschicktes Umformen der Gain-Gleichung können sowohl die Wurzel-Funktionen als auch die modifizierte Besselfunktion (5.11) als eine einzige Funktion mit einem Argument dargestellt werden, so dass sich die drei problematischen Funktionen durch eine einzige Tabelle implementieren lassen.

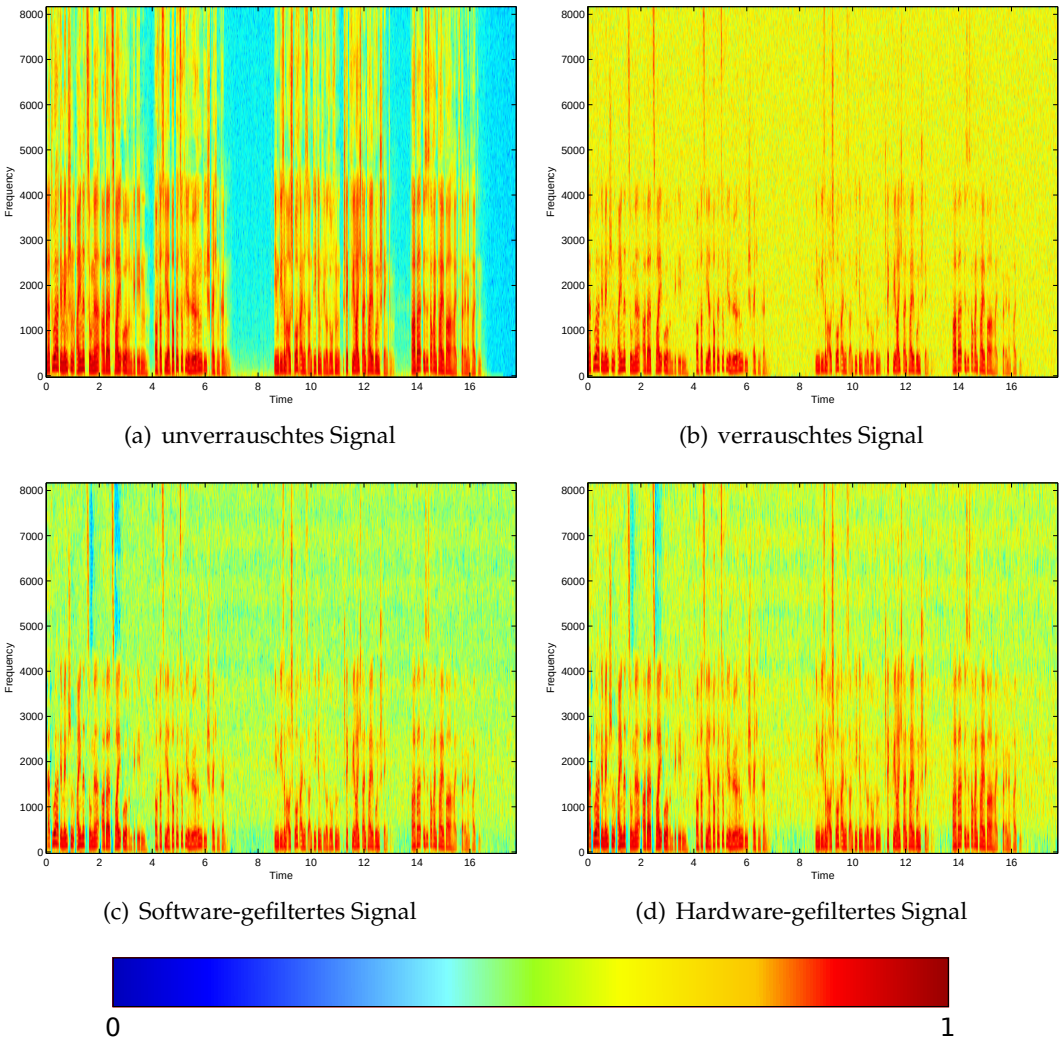
   Für die Tabelle wurde die in Abschnitt 4.6.3 beschriebene logarithmische Lookup-Tabelle verwendet. Für Details sei wiederum auf [82] verwiesen.
4. Für das System wurde ein Demonstrator auf Basis des in Abschnitt 4.7 vorgestellten Rapid-Prototyping System erstellt, mit dem die im folgenden dargestellten Messergebnisse ermittelt wurden.

5.4.3. Messwerte

Im folgenden werden die mit der Hardwareumsetzung der Ephraim-Malah Störgeräuschunterdrückung erzielten Messwerte aufgelistet.

Die Abbildungen 5.14 und 5.15 geben Spektrogramme eines Sprachsamples wieder, das durch den implementierten Algorithmus entrauscht wurde. Es sind im Vergleich das unverrauschte und das verrauschte, sowie das durch Hardware (Rapid-Prototyping System) und Software (Matlab-Referenz) gefilterte Signal wiedergegeben.

Die dargestellten Spektrogramme und Tabellenwerte bestätigen die Äquivalenz der in Hardware respektive in Software durchgeführten Filterung bezüglich des psychoakustisch validen Qualitätsmaßes und sind gleichzeitig ein Beleg für das korrekte Funktionieren der Umsetzung.



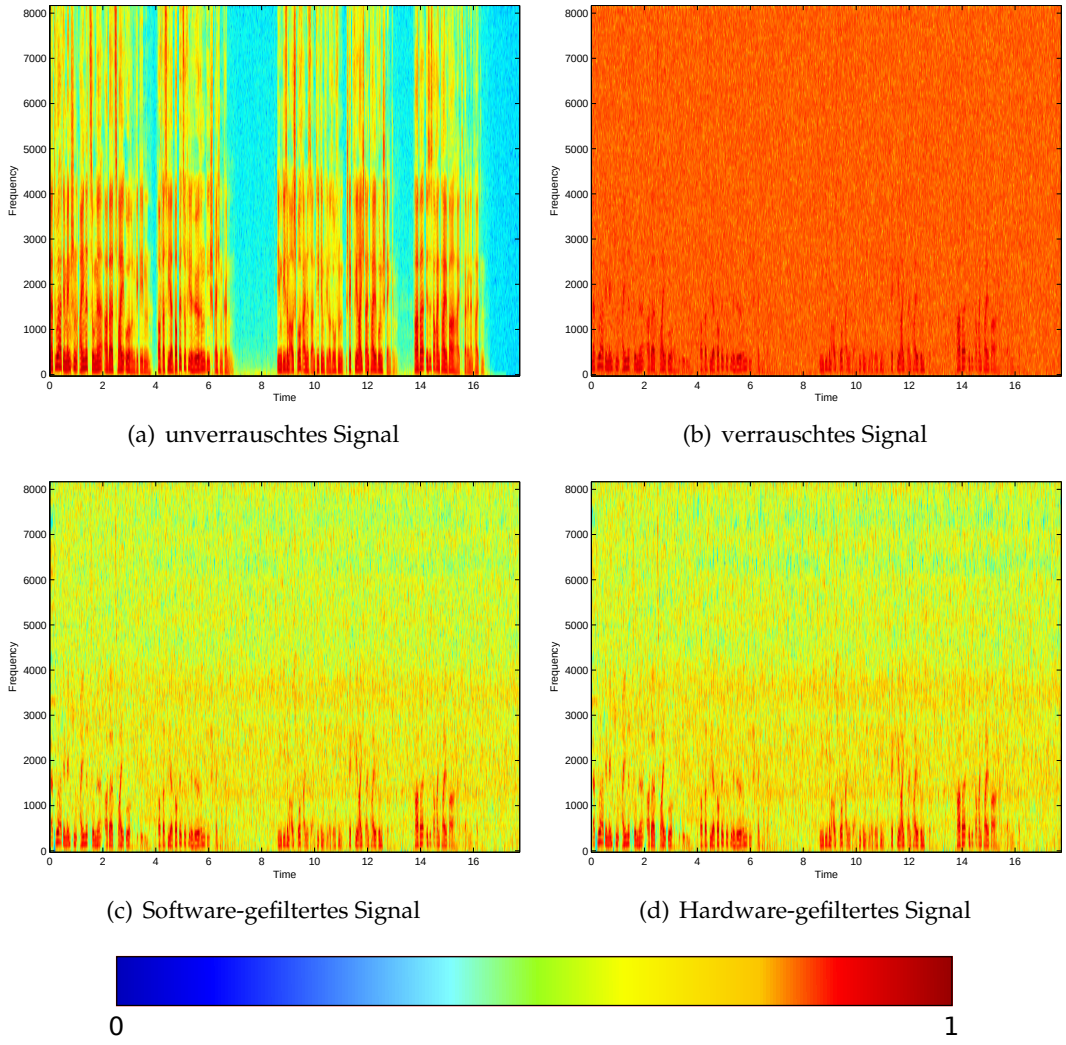
(a) unverrauschtes Signal

(b) verrauschtes Signal

(c) Software-gefiltertes Signal

(d) Hardware-gefiltertes Signal

Abbildung 5.14.: Spektrogramme für Rauschunterdrückung (12 dB Rauschabstand) des unverrauschten, verrauschten und gefilterten Signals (aus [82]). Der Signal-Rauschabstand zwischen dem ursprünglichen (a) und dem verrauschten Signal (b) beträgt 12 dB.
Die blauen Blöcke sind Sprachpausen im Audiostrom (es handelt sich um ein Sprachsample, Zeitachse in Sekunden, Frequenzachse in Hertz). Der Algorithmus kann das Rauschen sehr gut unterdrücken, dennoch gibt es deutliche sichtbare Unterschiede zum Originalsignal (a), die sich schon aus theoretischen Gründen nicht vollständig eliminieren lassen.
Die (geringen) Unterschiede zwischen Software- und Hardwarefilterung sind durch die Quantisierung bedingt, sie sind nicht hörbar.



(a) unverrauschtes Signal

(b) verrauschtes Signal

(c) Software-gefiltertes Signal

(d) Hardware-gefiltertes Signal

Abbildung 5.15.: Spektrogramme für Rauschunterdrückung (1,3 dB Rauschabstand) des unverrauschten, verrauschten und gefilterten Signals (aus [82]).
Der Signal-Rauschabstand zwischen dem ursprünglichen (a) und dem verrauschten Signal (b) beträgt 1.3 dB, ist also sehr viel geringer als in Abb. 5.14.
Die Störgeräuschunterdrückung kann auch dieses Signal immer noch relativ gut rekonstruieren.

5.4.4. Bewertung

Die psychoakustisch korrekte Quantisierung und Implementierung des Algorithmus' von Ephraim und Malah hat die Nützlichkeit und Funktion des durch das Entwicklungsframework vorgegebenen Entwicklungs- und Synthese-Fluss gezeigt. Der Algorithmus konnte trotz seiner Komplexität im Rahmen einer Diplomarbeit sehr zügig umgesetzt werden. Die Umsetzung erfolgte dabei zieltechnologieunabhängig, so dass das System einfach auf andere Zieltechnologien portierbar ist, ohne das die Optimierungsfähigkeit eingeschränkt wurde.

Zudem konnte die Funktion des Rapid-Prototyping-Systems belegt werden, das ein wirkungsvolles Hilfsmittel für die Verifikation und Validierung einer Framework-basierten Umsetzung ist.

Auf eine absolute Energieabschätzung für den implementierten Algorithmus wurde verzichtet, da die Aussagekraft gering ist. Sowohl die Quantisierung als auch die Implementierung der vorberechneten Tabellen erfolgte mit Hinblick auf eine möglichst geringe Leistungsaufnahme. Ein Vergleich mit einer schlechteren Variante erscheint nicht seriös, da die gewählte Implementierung gleichzeitig mit einem Minimum an Ressourcen auskommt. Andere Architekturen wären nicht effizient auf Hardware umzusetzen.

6. Diskussion

Es folgt eine knappe Zusammenfassung und Bewertung der Arbeit (6.1). Im Abschnitt 6.2 werden die im Verlaufe der Arbeit aufgetretenen Probleme diskutiert.

Zum Abschluss folgt ein Ausblick auf mögliche Erweiterungen und Modifikation des vorgestellten Entwicklungsrahmenwerks.

6.1. Zusammenfassung und Bewertung

Die vorliegende Arbeit beschreibt ein Framework, das eine **Systematik zur durchgängigen Entwicklung von Signalverarbeitungsalgorithmen** zur Verfügung stellt.

Durchgängigkeit bedeutet, dass von der Software-Spezifikation bis zum fertig platzierten und verdrahteten Chip möglichst **wenig redundante Arbeit** geleistet werden soll, ohne auf einen **hohen Grad an Optimierung** zu verzichten. Dies ist wegen der Heterogenität der in der Chip-Entwicklung üblichen Design-Flows sehr schwierig, daher wurde ein **Software-Framework** entwickelt, welches neben einer **Modulbibliothek** mit energieoptimierten, aber trotzdem zieltechnologieunabhängigen Modulen (Abschnitt 4.6) ein umfangreiches Skriptsystem zur Steuerung der Vielzahl der nötigen Werkzeuge enthält. Dieses Framework ermöglicht einen **automatisierten Entwurfsfluss**, der in Abschnitt 4.1.4 beschrieben wurde.

Bei der Modulbibliothek wurde auf eine hohes Maß an **Wiederverwendbarkeit** geachtet. Das bedeutet konkret, dass die Module für die **Synthese auf Verhaltensebene** und nicht auf der in der Chip-Entwicklung immer noch üblichen RT-Ebene spezifiziert wurden. Die hohe Abstraktionsebene der Spezifikation ermöglicht ferner ein höheres Optimierungsmaß in der nachfolgenden Hardware-Synthese, da die Eigenschaften der Zieltechnologie und die Anforderungen des Modul-benutzenden Algorithmus schon für die Generierung einer RT-Darstellung aus der Verhaltensspezifikation berücksichtigt werden können.

Im Unterschied zu anderen integrierenden Ansätzen (Kapitel 3) wurden bestehende **kommerzielle Werkzeuge** nahtlos in das Framework integriert, wobei die **standardisierten Schnittstellen** zwischen den einzelnen Implementierungsstufen so allgemein wie möglich gehalten wurden. Für die Modulbibliothek wurde dabei der Kompromiss eines **zweischichtigen Modulaufbaus** gewählt: Die Module bestehen aus einem (bitgenauen) C++-Prototypen, mit dem die Analyse und Partitionierung des zu implementierenden Algorithmus' vorgenommen wird und einem VHDL-Backend für die Hardware-Synthese, die durch ein kommerzielles Synthesewerkzeug weiterverarbeitet wird.

Die **Analyse und Quantisierung** des zu implementierenden Algorithmus' innerhalb des Frameworks wird durch zwei **integrierte Gütemaße** (Abschnitt 4.4) ermöglicht:

zum einen das einfache **Signal-Rausch Verhältnis**, das eine schnelle Abschätzung ermöglicht, zum anderen ein **psychoakustisch valides Gütemaß**, das von unserem Projektpartner entwickelt wurde und eine quantitative Aussage bezüglich der Hörbarkeit von vorhandenen Verzerrungen ermöglicht.

Die Anwendung des Frameworks wurde an ausgewählten **Audio-Algorithmen** demonstriert: Die Modulquantisierung sowie Architekturexploration und Verwendung anderer Energie-Minimierungsansätze wurde anhand des **Gammatonresynthese** Algorithmus' demonstriert (Abschnitt 5.2). Der Energiebedarf konnte gegenüber einem Referenzdesign ohne manuelle Eingriffe in den Quellcode um über 90% gesenkt werden, ohne eine mögliche V_{dd}-Absenkung zu berücksichtigen. Diese Ergebnisse wurden durch die im Framework implementierte **Energiebewertungsmethodik** gewonnen.

Der **modulare Aufbau** eines Audio-Signalverarbeitungssystems innerhalb des Frameworks wurde anhand der Implementierung **Ephraim-Malah Störgeräuschunterdrückung** (Abschnitt 5.4) mit dem Kurzzeit-FFT-Framework (Abschnitt 5.3) demonstriert. Weiterhin wurde gezeigt, dass dieses System auf der in das Entwurfsframework integrierten **Rapid-Prototyping**-Plattform lauffähig ist.

Das vorgestellte Entwicklungsframework liefert einen Beitrag zur Realisierung eines **durchgängigen Entwurfsfluss**es von der Systemspezifikation bis zur Hardware auf Basis der heute verfügbaren Werkzeuge und Hardwarebeschreibungsmöglichkeiten. Dabei wurde die zukünftig immer wichtiger werdende Aspekt der **Energieeffizienz** von vornherein berücksichtigt. Es wurden verschiedene Strategien zur Energieminimierung in das Framework integriert.

6.2. Probleme

Die während der vierjährigen Bearbeitungszeit des Projektes aufgetretenen Probleme sind typisch für die Hardwareentwicklung:

1. Alle drei Monate geben die Hersteller der EDA-Werkzeuge zumindest ein „ServicePack“ heraus. Neben den oft zahlreichen Erweiterungen, die man auch gern nutzt, hat dies oft auch subtile Änderungen an der Synthese-Maschine zur Folge, die zunächst dafür sorgen, dass der Implementierungsfluss angepasst werden muss. Die schlechte Dokumentation sorgt dabei für einen erheblichen Zeitaufwand.
2. Die enorme Komplexität der Synthese-Werkzeuge und viele „undokumentierte Features“ und versteckte Befehle erzeugen ebenfalls einen erheblichen Zusatzaufwand.
3. Die Limitierungen der gängigen Hardwarebeschreibungssprachen machen es generell schwer, einen in der Softwaretechnik längst etablierten modularen Entwurf mit abstrakten Schnittstellendefinitionen durchführen zu können. Die Entwicklung der Module, die den Anspruch der Wiederverwendbarkeit bei gleichzeitiger Hardware-Unabhängigkeit haben, war dadurch sehr zeitaufwendig und durch zahlreiche Iterationen geprägt.

Die Etablierung einer neuen Eingabesprache für die Hardwaresynthese erscheint unbedingt nötig, um die Entwurfslücke zu schließen.

6.3. Ausblick

Aufgrund seines modularen Aufbaus und der weitgehend generischen Schnittstellen bietet das vorgestellte System zahlreiche Anknüpfungspunkte für zukünftige Erweiterungen und Modifikationen, im folgenden wird eine Auswahl angerissen:

Ein wichtiger Gesichtspunkt bei der Entwicklung des Systems war die Durchgängigkeit der Entwicklung. Das Framework weist unter diesem Aspekt zwei Schwachstellen auf:

Zum einen die nötige manuelle Übersetzung einer Matlab-Spezifikation in eine C++-Beschreibung, die erforderlich ist, um das zu implementierende System in das Framework zu „importieren“. Da eine Entwicklung und Spezifikation in Matlab/Simulink von den Vorteilen des „Modellbasierten Designs“ profitieren kann, erscheint es sinnvoll, diese Spezifikation in das Framework zu integrieren. Es müsste dazu eine Abbildung des Simulink-Modells auf die Framework-Module möglich sein und für den Rahmenalgorithmus müsste eine synthesefähige Verhaltensbeschreibung generiert werden können. Das PRO-DASP-Nachfolgeprojekt AVSy (*Architecture for Automatic Power Minimization of Signalprocessing Systems*) befasst sich mit diesen Problemstellungen.

Zweitens haben die Framework-Module einen zweischichtigen Aufbau mit C++-Prototyp und Behavioral-VHDL Beschreibung. Diese doppelte Spezifikation der Funktionalität bedingt neben dem erhöhten Erstellungs- und Verifikationsaufwand auch eine erschwerte Pflege und Verbesserung der Module. Dieser Kompromiss musste zur Spezifikationszeit des Frameworks gemacht werden, da noch kein geeignetes Werkzeug verfügbar war, das die Hardwaresynthese aus einer C++-Spezifikation heraus ermöglichte. Da es inzwischen ein solches Werkzeug gibt [3], würde eine Umstellung der Behavioral-Synthese auf dieses Werkzeug das Framework dem Ziel der Durchgängigkeit noch ein Stück näher bringen.

Der einfachste denkbare Ausblick, weitere Algorithmen mit dem Framework umzusetzen, ist leider dadurch verwehrt, dass der zur Behavioral-Synthese eingesetzte Behavioral Compiler von Synopsys nicht mehr verfügbar ist. Quantisierungsuntersuchungen und eine Partitionierung auf bestehende (und neue) Module kann aber für neue Algorithmen ohne Einschränkungen durchgeführt werden. Auch sind inzwischen Alternativen zum Behavioral Compiler vorhanden. Ein Beispiel ist das in Abschnitt 3.8 kurz umrissene PowerOpt, ein anderes der Cynthesizer von Forte [4], der ebenfalls eine Synthese aus einer SystemC-Verhaltensbeschreibung ermöglicht.

Für die Umsetzung neuer Algorithmen ist aber eine Umstellung des momentan noch auf dem Behavioral Compiler basierenden Synthesesystems in jedem Falle notwendig.

Die in das Framework integrierte Modulbibliothek kann um zusätzliche Module erweitert werden. Eine Möglichkeit wäre die Integration eines Mikroprozessor-Moduls, das die Ablaufsteuerung für Algorithmen mit komplexem Kontrollfluss übernehmen und zudem ein Power-Management durchführen könnte. Aber natürlich sind auch viele einfache Module denkbar, die weitere Signalverarbeitungsfunktionen umsetzen.

Das Framework wurde mit Audioalgorithmen evaluiert und die (komplexeren) integrierten Module erfüllen Funktionen, die für die Audiosignalverarbeitung gedacht sind. Zudem ist ein speziell auf Audiosignalverarbeitung zugeschnittenes Gütemaß integriert. Der grundsätzliche Aufbau des Frameworks ist jedoch unabhängig von der Anwendungsdomäne. Durch die Integration eines geeigneten Gütemaßes und entsprechender Module könnte der Anwendungsbereich z. B. auf Bild- und Videoverarbeitung ausgedehnt werden. Bildverarbeitungsoperationen sind schon aufgrund des hohen Datenaufkommens so rechenzeitintensiv, dass eine Umsetzung auf Hardware sie oftmals erst echtzeitfähig macht. In Hinblick auf mobile Systeme spielt zudem der Energiebedarf eine große Rolle, so dass eine entsprechende Erweiterung des Frameworks hier neue Perspektiven öffnen würde.

A. Literaturverzeichnis

[1] *Blackfin DSP Datasheet*. Analog Devices, 2006. 2.8.5

[2] *WSTS Market Monitoring*. http://www.wsts.org/, 2007. 2.2.1

[3] *Catapult SL Pressemitteilung*. Mentor Graphics, 2008. 2.4.4, 4, 6.3

[4] *Cynthesizer (Forte Design Systems)*. http://www.forteds.com/, 2008. 6.3

[5] *DSP Builder*. Altera Corp., 2008. 4

[6] *Open SystemC Initiative*. http://www.systemc.org/, 2008. 2.4.2

[7] *PowerOpt*. Chipvision, 4 2008. 3.8.1, 14

[8] Adleman, Leonard M.: *Rechnen mit DNA*. Spektrum der Wissenschaft, 11 1998. 2.2.3

[9] Alcatel-Mietec: *0.25um Standard Cell Library*, 2002. 1

[10] Anderson, M. S. und S. Summerfield: *Power-Time Tradeoffs in Digital Filter Design and Implementation*. In: *IEEE Colloquium Digest Low Power VLSI Techniques*, 1995. 28

[11] Anis, Mohab, Shawki Areibi, Mohamed Mahmoud und Mohamed Elmasry: *Dynamic and Leakage Power Reduction in MTCMOS Circuits Using an Automated Efficient Gate Clustering Technique*. In: *DAC-2002: 39th Design Automation Conference, New Orleans, Lousiana*, June 2002. 3

[12] Arnold, M., T. Bailey, J. Cowles und C. Walter: *Fast Fourier Transforms using the Complex Logarithm Number System*. J. VLSI Signal Proc., 33, 2003. 2.8, 2.8.5

[13] Arnold, Mark: *Reduced Power Consumption for MPEG Decoding with LNS*. In: *Application Specific Architectures and Processors, San Jose*, July 2002. 2.8.5

[14] Arnold, Mark G. und Mark D. Winkel: *A Single-Multiplier Quadratic Interpolator for LNS Arithmetic*. In: *19th International Conference on Computer Design (ICCD 2001)*, 2001. 2.8.1

[15] Bailey, John C.: *A Comparison of Rechargeable Batteries for Portable Devices*. In: *Advances in R&D for the Commercialization of Small Fuel Cells and Battery Technologies For Use in Portable Applications*, April 1999. 2.3

[16] Benz, Benjamin: *Triebwerk a la carte*. c't Magazin für Computertechnik, Mai 2005. 2.2

[17] Berry, Gerard: *The foundations of Esterel*. Technischer Bericht, Ecole des Mines de Paris and INRIA, 2004. 3.6.1

[18] Boemo, E., S. López-Buedo, C. Santos Pérez, J. Jáuregui und J. Meneses: *Logic Depth and Power Consumption: A Comparative Study Between Standard Cells and*

FPGAs. In: *Proc. XIII DCIS Conference (Design of Circuit and Integrated Systems)*, November 1998. 2.6.4, 4.7.2

[19] Briggs, W. und V. Henson: *The dft: An owner's manual for the discrete fourier transform*. Philadelphia, 1995. 2.9.6

[20] Brown, Theodore L.: *Chemistry - The Central Science*. Prentice Hall, 1994. 2.2.2

[21] Buyuksahin, K. M. und F. N. Najm: *High-level power estimation with interconnect effects*. In: *IEEE International Symposium on Low Power Electronics and Design*, July 2000. 3

[22] Cardarilli, Gian Carlo, Alberto Nannarelli und Marco Re: *Reducing Power Dissipation in FIR Filters using the Residue Number System*. In: *Proceedings of 43rd IEEE Midwest Symposium on Circuits and Systems*, Band 1, Seiten 320–323, 2000. 2.8.2

[23] Chamberlain, Roger, Eric Hemmeter, Robert Morley und Jason White: *Modeling the Power Consumption of Audio Signal Processing Computations Using Customized Numerical Representations*. In: *Proc. of the 36th Annual Simulation Symposium*, April 2002. 2.8, 2.8.1

[24] Chandrakasan, A., M. Potkonjak, R. Mehra, J. Rabaey und R. Brodersen: *Optimizing Power Using Transformations*. IEEE Transactions on CAD, 14(1):12–31, 1995. 3.3.1

[25] Chandrakasan, A., S. Sheng und R. Brodersen: *Low Power CMOS Digital Design*. IEEE Journal of Solid-State Circuits (JSSC), 27(4), April 1992. 2.2.3, 2.5.1, 3.3

[26] Chandrakasan, A. P., M. Potkonjak, J. Rabaey und R. W. Brodersen: *Hyper-lp : a system for power minimization using architectural transformations*. In: *Proc. ICCAD*, 1992. 3.3

[27] Choo, H., K. Roy und K. Muhammad: *MRPF: An Architectural Transformation for Synthesis of High-Performance and Low-Power Digital Filters*. In: *DAC-2003: 40th Design Automation Conference*, March 2003. 4.6.3

[28] Clarke, John: *SQUIDs*. Spektrum der Wissenschaft, 10 1994. 2.2.3

[29] Cooley, J.W. und J.W. Tukey: *An algorithm for the machine calculation of complex Fourier series*. Math. Comp., Seiten 297–301, April 1965. 2.9.6

[30] Coudert, O., R. Haddad und K. Keutzer: *What is the state of the art in commercial EDA tools for low power?* In: *Proc. Low Power Electronics and Design*, 1996. 4.2.2

[31] Cölln, Gerd von: *Modellierung und Simulation der Verlustleistung von integrierten Scahltungs-Makros*. Doktorarbeit, Universität Oldenburg, Germany, Fachbereich Informatik, 2001. 13

[32] D'Amora, Angelo, Alberto Nannarelli, Marco Re und Gian Carlo Cardarilli: *Reducing Power Dissipation in Complex Digital Filters by using the Quadratic Residue Number System*. In: *Proc. 34th Asilomar Conference on Signals, Systems, and Computers*, 2000. 2.8.2

[33] Dau, T., D. Puschel und A. Kohlrausch: *A quantitative model of the 'effective' signal processing in the auditory system. I. Model structure*. Journal of the Acoustical Society of America, 99(6), 1996. 4.4.2, 5.2, 5.2.1

[34] Dau, T., D. Puschel und A. Kohlrausch: *A quantitative model of the 'effective' signal processing in the auditory system. II. Simulations and measurements*. Journal of the Acoustical Society of America, 99(6):3623–31, 1996. 4.4.2, 5.2

[35] Davis, W. R., N. Zhang, K. Camera, D. Markovic, T. Smilkstein, C. Chan, M. J. Ammer, E. Yeo, B. Nikolic und R. W. Brodersen: *An Automated Design Flow for Low-Power, High-Throughput, Dedicated Signal Processing Systems*. In: *Proc. IEEE Custom Integrated Circuits Conf.*, Mai 2001. 3.4, 4.6.3

[36] Davis, W. Rhett, Ning Zhang, Kevin Camera, Dejan Markovic, Tina Smilkstein, M. Josie Ammer, Engling Yeo, Stephanie Augsburger, Borivoje Nikolic und Robert W. Brodersen: *A Design Environment for High-Throughput Low-Power Dedicated Signal Processing Systems*. IEEE Journal of Solid State Circuits, 37(3), March 2002. 3.4, 3.2

[37] Demtröder, Wolfgang: *Experimentalphysik*. Springer Verlag, Berlin, 1995. 10

[38] Duhamel, P.: *Implemenation of "split-radix" FFT algorithms for complex, real and real-symmetric data*. IEEE Trans. Acoust. Speech Sig. Process., ASSP-34(2):285–295, April 1986. 2.9.6

[39] Duhamel, P. und H. Hollmann: *Existence of a 2^n FFT algorithm with a number of multiplications lower than 2^{n+1}*. Electron. Lett., 20(17):690–692, August 1984. 2.9.6

[40] Duhamel, P. und H. Hollmann: *Split-radix FFT algorithm*. Electron. Lett., 20(1):14–16, January 1984. 2.9.6

[41] Ebergen, Jo und Ivan E. Sutherland: *Taktlose Computer*. Spektrum der Wissenschaft, 12 2002. 2.2.3

[42] Economakos, G., P. Oikonomakos, I. Panagopoulos, I. Poulakis und G. Papakonstantinou: *Behavioral Synthesis with SystemC*. In: *Proceedings of the International Conference on Computer-Aided Design (ICCAD 2001)*, 2001. 2.4.2

[43] Eisenbach, Thomas, Bärbel Mertsching, Nikolaus Voss und Frank Schmidtmeier: *Optimization of Modules for Digital Audio Processing*. In: *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation 15th International Workshop (PATMOS 2005), Leuven, Belgium, September 21-23, 2005, Proceedings*, Nummer 3728 in *Lecture Notes in Computer Science*. Springer Verlag, September 2005. 4, 4.6

[44] Ephraim, Y. und D. Malah: *Speech enhancement, using a minimum mean-square error log-spec tral amplitude estimator*. IEEE Trans. Acoust. Speech Signal Processing, 32(6):1109–1121, 1984. 5.4.1

[45] Frank, Michael P.: *Reversibility for efficient computing*. Doktorarbeit, MIT, 1999. 2.1, 2.2.3

[46] Frey, Thomas und Martin Bossert: *Signal- und Systemtheorie*. Teubner, 2004. 2.9, 2.9.3, 2.17, 2.18, 2.19

[47] Friedman, T.D. und S.C. Yang: *Methods used in an automatic logic design generator (ALERT)*. IEEE Transactions in Computing, 1969. 2.7.3

[48] Gawollek, Nils-Holger, Eike Grimpe, Maik Weilert, Mark Hillers, Carsten Beth und Jochen Zurborg: *Entwurf eines eingebetteten Systems zur Sprachverarbei-*

tung. Technischer Bericht, Universität Oldenburg, Fachbereich Informatik, 1999. 5.2.2, 5.2.2, 5.2.2

[49] Gerlach, J. und W. Rosenstiel: *A Scalable Methodology for Cost Estimation in a Transformational High-Level Design Space Exploration Environment*. In: *Proc. DATE*, 1998. 3.7.1

[50] Gerlach, Joachim: *Transformationale Entwurfsraum-Exploration für den Entwurf eingebetteter Systeme*. Doktorarbeit, Universität Tübingen, 2000. 3.7.1

[51] Gerlach, Joachim und Wolfgang Rosenstiel: *A Methodology and Tool for Automated Transformational High-Level Design Space Exploration*. In: *Proc. IEEE International Conference On Computer Design*, 2000. 3.7, 3.5

[52] Girod, Bernd, Rudolf Rabenstein und Alexander Stenger: *Einführung in die Systemtheorie*. Teubner, 1997. 2.9.2

[53] Goel, M. und N. Shanbhag: *Dynamic Algorithm Transformations (DAT) for Low-Power Adaptive Signal Processing*. In: *International Symposium on Low Power Electronics and Design (ISLPED'97)*, 1997. 4.6.3

[54] Goel, M. und N. R. Shanbhag: *Dynamic algorithm transforms (DAT): A Systematic approach to low-power reconfigurable signal processing*. IEEE Transactions on VLSI Systems, 7, 4 1999. 4.6.3

[55] Goering, Richard: *ESL tools: Are EDA giants in the game?* EETimes http://www.eetimes.com/news/design/showArticle.jhtml?articleID=47204415, 9 2004. 4.1.1

[56] Gold, B. und C.M. Rader: *Effects of Parameter Quantization on the Poles of a Digital Filter*. Proc. IEEE, May 1967. 2.9.5, 32

[57] Goldberg, David: *What Every Computer Scientist Should Know About Floating-Poin Arithmetic*. ACM Computing Surveys (CSUR), 23(1), March 1991. 2.8.3

[58] Gonzalez, Ricardo, Benjamin M. Gordon und Mark A. Horowitz: *Supply and Threshold Voltage Scaling for Low Power CMOS*. IEEE Journal of Solid-State Circuits, 32(8), August 1997. 2.5.1, 1

[59] Gunther, Stephen H., Frank Binns, Douglas M. Carmean und Jonathan C. Hall: *Managing the Impact of Increasing Microprocessor Power Consumption*. Intel Technology Journal, (1), 2001. 2.7

[60] Halbwachs, N.: *A Synchronous Language at Work: the Story of Lustre*. In: *Proc. 3thrd ACM-IEEE Int. Conf. on Formal Methods and Models for Codesign*, 7 2005. 8

[61] Haldar, Malay, Anshuman Nayak, Alok Choudhary und Prith Banerjee: *A System for Synthesizing Optimized FPGA Hardware from MATLAB*. In: *Proceedings of the International Conference on Computer-Aided Design (ICCAD2001)*, 2001. 3.5

[62] Han, Chang-Young, Hyoung-Joon Park und Lee-Sup Kim: *A Low-Power Array Multiplier Using Separated Multiplication Technique*. IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, September 2001. 4.6.3

[63] Hansen, M. und B. Kollmeier: *Objective Modeling of Speech Quality with a Psychoacoustically Validated Auditory Model*. J. Audio Eng. Soc., 48(5), 2000. 4.4.2

[64] Hansen, Martin: *Assessment and Prediction of Speech Transmission Quality with an Auditory Processing Model*. Doktorarbeit, University of Oldenburg, Germany, 1998. 4.4.2

[65] Harding Energy, Inc., One Energy Centre, Norton Shores, MI 49441: *The Quest Advantage Nickel Metal Hydride Batteries Handbook*, 12 2007. 2.4

[66] Hohmann, V. und M. Hansen: *Implementation der Gammatone Filterbank*. Technischer Bericht, Universität Oldenburg, 1997. 5.2.2

[67] Huber, Rainer: *Objective assessment of audio quality using an auditory processing model*. Doktorarbeit, University of Oldenburg, Germany, 2003. 4.10

[68] Huber, Rainer und Birger Kollmeier: *PEMO_Q - Audio Quality Assessment Using a Model of Auditory Perception*. In: *14. Konferenz Elektronische Sprachsignalverarbeitung ESSV*, 2003. 4.4.2

[69] Huber, Rainer und Birger Kollmeier: *Temporal Aspect in the Prediction of perceived Audio Quality Differences*. In: *30. Deutsche Jahrestagung für Akustik*, 2004. 4.4.2

[70] Hwang, Kai: *Advanced Computer Architecture - Parallelism, Scalability, Programmability*. McGraw-Hill, New York, 1993. 2.8.3

[71] IBM: *SOI Technology: IBM's next advance in chip design*. Technischer Bericht, Whitepaper, 2006. 2.2.2

[72] Iman, Sasan und Massoud Pedram: *POSE: Power Optimization and Synthesis Environment*. In: *33rd Design Automation Conference*, 1996. 3.2, 2, 3.1

[73] Intel Corporation: *Intel Pentium 4 Processor with 512KB L2 Cache on .13 Micron Process at 2 GHz and 2.20 GHz*, January 2002. 2.2.2

[74] Intel Corporation: *Intel Datasheets*, January 2008. 1.2

[75] ITRS: *ITRS Roadmap 2007*. Technischer Bericht, ITRS, 2007. 2.2.4

[76] Jochens, G., L. Kruse, E. Schmidt, A. Stammermann und W. Nebel: *Power Macro-Modelling for Firm-Macros*. In: *Int. Works. Power and Timing Modelling of Integrated Circuits (PATMOS)*, 2000. 3.8.1

[77] Jung, Seong-Ook, Ki-Wook Kim und Sung-Mo Kang: *Low-Swing Clock Domino Logic Incorporating Dual Supply and Dual Threshold Voltage*. In: *DAC-2002: 39th Design Automation Conference, New Orleans, Lousiana*, June 2002. 3

[78] Jähne, Bernd: *Digitale Bildverarbeitung*. Springer, 2005. 2.21, 2.22

[79] Kadayif, I., M. Kandemir, N. Vijaykrishnan, M. Irwin und A. Sivasubramaniam: *EAC: A Compiler Framework for High-Level Energy Estimation and Optimization*. In: *DATE 2002: Design, Automation and Test in Europe, Paris, France*, March 2002. 3

[80] Katkoori, S. und R. Vemuri: *Architectural Power Estimation Based on Behavioral Profiling*. Special Issue on Low Power Design, Journal on VLSI DESIGN, 1996. 3

[81] Kim, C. und K. Roy: *Dynamic VTH Scaling Scheme for Active Leakage Power Reduction*. In: *DATE 2002: Design, Automation and Test in Europe, Paris, France*, March 2002. 4

[82] Klöckner, Felix: *Entwurf und Implementierung eines energieeffizienten psychoakustisch motivierten Störgeräuschunterdrückugns-Algorithmus' auf einem Rapid-Prototyping System*. Diplomarbeit, Universität Hamburg, 2004. 4.6.3, 4.6.3, 4.24, 5.4, 5.4.2, 3, 5.14, 5.15

[83] Knodel, Hans und Horst Bayrhuber (Herausgeber): *Linder Biologie*. Metzlersche Verlagsbuchhandlung, 1983. 5.1

[84] Kollmeier, Birger, Michael Kleinschmidt, Bärbel Mertsching, Alexander Schwarz, Wolfgang Nebel und Matthias Brucke: *Endbericht zum Projekt System- und Schaltungstechnik einer integrierten Cochlea für Sprachanalyse, Spracherkennung, und Sprachcodierung*. Technischer Bericht, Universität Hamburg und Universität Oldenburg, Juli 2000. 5.3, 5.2.3

[85] Kruse, L., E. Schmidt, G. Jochens und A. Stammermann: *Lower bounds on the power consumption in scheduled data flow graphs with resource constraints*. In: *DATE'00, Design, Automation and Test in Europe*, 2000. 3.8.1

[86] Langdo, T. A.: *Strained Silicon on Insulator Technology: From Materials to Devices*. Solid-State Electronics, 48(8), 2004. 2.2.2

[87] Lapalme, James, El Mostapha Aboulhamid und Gabriela Nicolescu: *A new efficient EDA tool design methodology*. ACM Transactions on Embedded Computing Systems, 5, May 2006. 2.4.4

[88] Lehmann, Gunther, Bernhard Wunder und Manfred Selz: *Schaltungsdesign mit VHDL*. Franzis, 1994. 2.12

[89] Lin, James: *Challenges for SoCDesign in Very Deep Submicron Technologies*. In: *Proc. CODES*, 2003. 2.1

[90] Lindner, Albrecht: *Grundkurs Theoretische Physik*. Teubner Studienbücher, 1997. 28

[91] LSI Logic Inc.: *LSI G11 0.25-micron ASIC Technology Manual*, 1998. 4.2.2

[92] Macii, E., M. Pedram und F. Somenzi: *High-Level Power Modeling, Estimation and Optimization*. In: *Proceedings of DAC*, 1997. 3

[93] Man, H. De, J. Rabaey, P. Six und L. Claesen: *Cathedral-II: A silicon compiler for digital signal processing*. Design & Test of Computers, 12 1986. 2.7.3

[94] Marzinzik, Mark: *Noise Reduction Schemes for Digital Hearing Aids and their Use for for the Hearing Impaired*. Doktorarbeit, Universität Oldenburg, Germany, Fachbereich Physik, 2000. 5.4.1

[95] Mathworks Corp.: *Simulink*, 2008. 3.5.1

[96] Matousek, Rudolf, Milan Tichy, Zdenek Pohl, Jiri Kadlec, Chris Softley und Nick Coleman: *Logarithmic Number System and Floating-Point Arithmetics on FPGA*. In: *Proc. of FPL 2002*, 2002. 2.8, 2.8.1

[97] Mehra, Renu und Jan Rabaey: *Behavioral Level Power Estimation and Exploration*. In: *Proc.First International Workshop on Low Power Design*, 1994. 3

[98] Menard, D. und O. Sentieys: *Automatic Evaluation of the Accuracy of Fixed-Point Algorithms*. In: *DATE 2002: Design, Automation and Test in Europe, Paris, France*, March 2002. 2.8.5

[99] Mertsching, Bärbel, Thomas Eisenbach, Frank Schmidtmeier, Nikolaus Voss und Hongyu Wang: *High Level Circuit Design For Low Power Audio Signal Processing*. In: *INFORMATIK 2005 - Informatik Live! 35. Jahrestagung der GI*, Band 1 der Reihe *Lecture Notes in Informatics*, Seite 453, Bonn, September 2005. Gesellschaft für Informatik e.V., GI. 8

[100] Moore, Gordon E.: *Cramming more components onto integrated circuits*. Electronics, 38(8), April 1965. 2.2.2

[101] Morley, R. E., G. L. Engel und T. J. Sullivan: *VLSI Based Design of a Battery Operated Digital Hearing Aid*. In: *ICASSP*, 1988. 2.8.5

[102] Morley, R.E., G.L. Engel, T.J. Sullivan und S.M. Natarajan: *VLSI based design of a battery-operated digital hearing aid*. In: *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 1988. 2.8.1

[103] Morris, L.R.: *A comparative study of time efficient FFT and WFTA programs for general purpurse computers*. IEEE Trans. Acoust. Speech Signal Process., ASSP-26:74–78, 2 1977. 2.9.6

[104] Muhammad, K. und K. Roy: *Low Power Digital Filters Based On Constrained Least Squares Solution*. In: *Proceedings of the 31st Asilomar Conference on Signals, Systems and Computers*, 1997. 4.6.3

[105] Muhammad, K. und K. Roy: *On Complexity Reduction of FIR Digital Filters Using Constrained Least Squares Solution*. In: *Proceedings of 1997 IEEE International Conference on Computer Design (ICCD '97)*, 1997. 4.6.3

[106] Muhammad, Khurram und Kaushik Roy: *A novel design methodology for high performance and low power digital filters*. In: *Proc. ICCAD*, 1999. 4.6.3

[107] Muhammad, Khurram und Kaushik Roy: *A Graph Theoretic Approach for Synthessizing Very Low-Complexity High-Speed Digital Filters*. IEEE Trans. on CAD of Int. Circ. and Systems, 21(2), February 2002. 4.6.3

[108] *http://www.nallatech.com*, 2008. 4.7.1

[109] Neau, C., K. Muhammad und K. Roy: *Low Complexity FIR Filters Using Factorization of Perturbed Coefficients*. In: *Design, Automation and Test in Europe Conference 2001*, 2001. 4.6.3

[110] Nemani, Mahadevamurty und Farid N. Najm: *High-Level Area and Power Estimation for VLSI Circuits*. In: *IEEE 1997 International Conference on Computer Aided Design*, 1997. 3

[111] Nielsen, Michael A.: *Spielregeln für Quantencomputer*. Spektrum der Wissenschaft, 4 2003. 2.2.3

[112] Oraintara, Soontorn, Ying-Jui Chen und Truong Q. Nguyen: *Integer Fast Fourier Transform*. IEEE Trans. on Signal Proc., 50(3), March 2002. 4.6.3

[113] Paliouras, V. und T. Stouraitis: *Logarithmic Number System for Low-Power Arithmetic*. In: *Proc. of PATMOS 2000*, 2000. 2.8.1

[114] Parhami, B.: *Computer Arithmetic – Algorithms and Hardware Designs*. Oxford University Press, New York, 2000. 4.6.4

[115] Park, N. und A. Parker: *Sehwa: A software package for synthesis of pipelines from behavioral specifications*. IEEE Transaction on Computer Aided Design, 3 1988. 2.7.3

[116] Patterson, R. D., J. Nimmo-Smith, J. Holdsworth und P. Rice: *An efficient auditory filterbank based on the gammatone function*. Technischer Bericht, IOC Speech Group, 1987. 4.4.2, 5.2.2

[117] Pellerin, David: *Getting started with Impulse-C*. In: *Tutorial at Reconfigurable Systems Summer Institute*. University of Illinois at Urbana-Champaign, 2008. 2

[118] Premkumar, A. Benjamin: *A formal framework for conversion from binary to residue numbers*. IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, 49(2):135–144, February 2002. 2.8, 2.8.2

[119] Proceedings of the 25th EOROMICRO Conference, Milano, Italy: *Implementing a quantitative model for the 'effective' signal processing in the auditory system on a dedicated digital VLSI hardware*, 1999. 4

[120] Qu, Gang: *What is the Limit of Energy Saving by Dynamic Voltage Scaling?* In: *Proceedings of the International Conference on Computer-Aided Design (ICCAD 2001)*, 2001. 4

[121] Rabaey, J.M. und M. Pedram (Herausgeber): *Low Power Design Methodologies*. Kluwer Boston, MA, 1996. 2.2.3

[122] Ramprasad, S., N. Shanbag und I. Hajj: *Decorrelating (DECOR) transformations for low-power digital filters*. IEEE Trans. on Circuits and Systems, June 1999. 4.6.3, 4.6.3

[123] Raudvere, Tarvo: *Design and Verification in the ForSyDe Methodology*. Talk, 9 2004. 3.4

[124] Reyneri, L. M., F. Cucinotta, A. Serra und L. Lavagno: *A Hardware/Software Codesign Flow and IP Library Based of Simulink*. In: *38th Design Automation Conference*, 2001. 3.5.1

[125] Rich, D. I.: *The Evolution of SystemVerilog*. IEEE Design and Test of Computers, 20, 2003. 2.4.2

[126] Risch, Lothar, Wolfgang Rösner und Thomas Schulz: *Transistor verkehrt*. Spektrum der Wissenschaft, 6 1999. 2.2.2

[127] Rossello, J.L. und Jaume Segura: *Power-Delay Modeling of Dynamic CMOS Gates for Circuit Optimization*. In: *Proceedings of the International Conference on Computer-Aided Design (ICCAD 2001)*, 2001. 2.2.3

[128] Sakamoto, T., T. Yamada, M. Mukuno, Y. Matsushita, Y. Harada und H. Yasuura: *Power analysis techniques for SoC with improved wiring models*. In: *Proc. ISLPED 02*, 2002. 4.2.2

[129] Sander, I., A. Jantsch und Z. Lu: *Development and Application of Design Transformations in ForSyDe*. In: *DAC-2003: 40th Design Automation Conference*, March 2003. 3.6.1

[130] Sander, Ingo und Axel Jantsch: *System modeling and transformational design refinement in ForSyDe*. IEEE Trans. on CAD of Integrated Circuits and Systems, 23, 1 2004. 3.6.1

[131] Sankarayya, N., K. Roy und Debashis Bhattacharya: *Algorithms for Low Power and High Speed FIR Filter Realization Using Differential Coefficients*. IEEE Transactions on Circuits and Systems, Part II, Analog and Digital Signal Processing, 44(6), June 1997. 4.6.3

[132] Scholz, Michael: *Untersuchung und Simulation von High-Level-Transformationen für die Entwicklung von leistungsarmen Schaltungen am Beispiel von Audiofiltern*. Diplomarbeit, Universität Hamburg, 2002. 4.6.1

[133] Schulte, Michael J. und James E. Stine: *Reduced Power Dissipation Through Truncated Multiplication*. In: *Proceedings of the IEEE Alessandro Volta Memorial Workshop on Low-Power Design*, 1998. 35

[134] Schwarz, Alexander: *VLSI - Entwurf für ein digitales Perzeptionsmodell des menschlichen Hörens*. Doktorarbeit, Universität Hamburg, 2002. 5.2.2, 5.2.2

[135] Shannon, Claude E.: *A Mathematical Theory of Communication*. Bell Telephone System Technical Publications, 1948. 34

[136] Sharma, M. und N. R. Shanbhag: *Architecture driven filter transformations*. In: *International Symposium on Circuits and Systems*, May 2000. 28

[137] Skurai, T. und A.R. Newton: *Alpha-power law MOSFET model and its application to CMOS inverter delay and other formulas*. IEEE Journal on Solid-State Circuits, 25, April 1990. 2.5.1, 2.5.1

[138] Slaney, Malcolm: *An Efficient Implementation of the Patterson-Holdsworth Auditory Filter Bank*. Technischer Bericht, Apple Computer, 1993. Technical Report 35 Perception Group Advanced Technology Group. 3

[139] Solbach, Ludger: *An Architecture for Robust Partial Tracking and Onset Localization in Single Channel Audio Signal Mixes*. Doktorarbeit, Technical University of Hamburg-Harburg, Germany, 1998. 3

[140] Solbach, Ludger, Rolf Wöhrmann und Jörg Kliewer: *The complex-valued continuous wavelet transform as a preprocessor for auditory scene analysis*. In: *IJCAI-95 Workshop on Computational Auditory Scene Analysis*, 1995. 3

[141] Soudris, D. J., V. Paliouras, T. Stouraitis und C. E. Goutis: *A VLSI Design Methodology for RNS Full Adder-Based Inner Product Architectures*. IEEE Transactions on Circuits and Systems, 44(4), April 1997. 2.8.2

[142] Spaniol, Otto: *Arithmetik in Rechenanlagen*. B. G. Teubner, 1976. 2.8.4, 4.6.3

[143] Sproch, J.: *High Level Power Analysis and Optimiziation*. In: *Tutorial Int. Symposium on Low Power Electronics and Design*, 1997. 4.1

[144] Stammermann, Ansgar, Lars Kruse, Eike Schmidt, Alexander Pratsch, Milan Schulte, Arne Schulz und Wolfgang Nebel: *ORINOCO: Verlustleistungsanalyse und Optimierung auf der algorithmischen Abstraktionsebene*. In: *10. E.I.S.-Workshop*, 2001. 3, 3.6

[145] Standards Committee of The IEEE Computer Society: *Std 754-1985 for Binary Floating-Point Arithmetic*, 1985. 4.6.3

[146] Steenis, Bernard und Christian Piguet: *Low-Power Gated-Clock Schemes Using High-Level VHDL Modelling*. In: *International Workshop - Power and Timing Modeling, Optimization and Simulation (PATMOS'98), Technical University of Denmark*, October 1998. 16

[147] Sterling, Thomas: *Supercomputer - die jüngsten Entwicklungen*. Spektrum der Wissenschaft, März 2005. 1.1, 1.4

[148] Stiller, Andreas: *Itanium auf dem Sprung*. c't - Magazin für Computertechnik, (20), 2005. 1.3

[149] Swan, Stuart: *An Introduction to System Level Modeling in SystemC 2.0*. Technischer Bericht, Cadence Design Systems, Inc., May 2001. 2.4.2

[150] *Behavioral Compiler VHDL User and Modeling Guide*, 2003. 4.12, 4.13, 4.14

[151] *Module Compiler Manual*, 2005. 1

[152] *PowerCompiler Manual*, 2006. 4.2.1, 4.2.2, 4.2.2, 4.8

[153] *PrimePower Manual*, 2006. 4.9

[154] Szabo, N. und R. Tanaka: *Residue Arithmetic and its Applications to Computer Technology*. McGraw-Hill, 1967. 24

[155] Tchorz, J., M. Kleinschmidt und B. Kollmeier: *Noise suppression based on neurophysiologically motivated SNR estimation for robust speech recognition*. In: *Proceedings of Neural Information Processing Systems (NIPS2000)*, 2000. 4.27

[156] Tchorz, Jürgen: *Auditory-based signal processing for noise suppression and robust speech recognition*. Doktorarbeit, University of Oldenburg, Germany, 2000. 4.27

[157] Thompson, Scott, Paul Packan und Mark Bohr: *MOS Scaling: Transistor Challenges for the 21st Century*. Intel Technology Journal, (3), 1998. 5, 2.5.1

[158] V. Paliouras, T. Stouraitis: *Novel High-Radix Residue Number System Architectures*. IEEE Trans. on Circuits and Systems, 47(10), October 2000. 4, 2.8.2

[159] V. Paliouras, T. Stouraitis: *Considering the Alternatives in Low-Power Design*. IEEE Circuits & Devices, (23–29), July 2001. 2.8

[160] V. Paliouras, T. Stouraitis: *Low-power Properties of the Logarithmic Number System*. In: *Proc. 15th Symp. Computer Arithmetic (ARITH15)*, 2001. 2.8, 2.8.1

[161] Valluri, M. und L. John: *Is Compiling for Performance == Compiling for Power?*, Kapitel Interaction between Compilers and Computer Architectures. Kluwer Academic Publishers, 2001. 2

[162] Vaseghi, S.V.: *Advanced Signal Processing and Digital Noise Reduction*. Teubner, 1996. 5.4.1

[163] Vetterli, M. und C. Herley: *Wavelets and filter banks: theory and design*. IEEE Trans. on Signal Proc., 40(9), 1992. 6

[164] Virtual Silicon UMC: *L180 40um Staggered I/O Library*, 2002. 4.2.2, 2

[165] Voss, Nikolaus: *Entwurf und Implementation einer durch parallele Hardware beschleunigten Gabor-Filterbank*. Diplomarbeit, Universität Hamburg, 2000. 4.6.3

[166] Voss, Nikolaus, Thomas Eisenbach und Bärbel Mertsching: *A Rapid Prototyping Framework for Audio Signal Processing Algorithms*. In: *2004 IEEE International Conference on Field-Programmable Technology, Brisbane, Australia*, Seiten 375 – 378. IEEE, December 2004. 39

[167] Voss, Nikolaus und Bärbel Mertsching: *Design and Implementation of an Accelerated Gabor Filter Bank Using Parallel Hardware*. In: *Proc. Field-Programmable Logic and Applications (FPL)*, 2001. 39

[168] Voss, Nikolaus und Bärbel Mertsching: *PRO-DASP - Using Transformations to Implement Hardware-Macros for a Low Power Design Methodology*. In: *Proc. 3rd VIVA-Kolloquium*, 2002. 4.6.1, 4.18

[169] Voss, Nikolaus und Bärbel Mertsching: *A Framework for Low Power Audio Design*. In: *Proc. of 17th Int'l Conf. on VLSI Design (VLSI-2004)*, 2004. 4

[170] Walker, R. und D. Thomas: *A model of design representation and synthesis*. In: *Proc. 22nd Design Automation Conf.*, 1985. 2.7.2, 2.11

[171] Walters, E. G. und M. J. Schulte: *Design Tradeoffs Using Truncated Multipliers in FIR Filter Implementations*. In: *Proceedings of SPIE : Advanced Signal Processing Algorithms, Architectures, and Implementations*, 6 2002. 2.8, 35

[172] Wang, L. und N. R. Shanbhag: *Low-power signal processing via error cancellation*. In: *Proc. of IEEE Workshop on Signal Processing Systems*, October 2000. 4.6.3

[173] Wang, Zhongde, G. A. Jullien und W. C. Miller: *An efficient 3-modulus residue to binary converter*. In: *IEEE 39th Midwest symposium on Circuits and Systems*, Band 3, Seiten 1305–1308, August 1996. 2.8.2

[174] Wilk, G. D., R. M. Wallace und J. M. Anthony: *High-kappa gate dielectrics: Current status and materials properties considerations*. Journal of Applied Physics, 89(10), 2001. 2.2.2

[175] Winograd, S.: *On Computing the Discrete Fourier Transform*. Proc. Nat. Acad. Sci. USA, Seiten 1005–1006, April 1976. 2.9.6

[176] Wires, K. E., M. J. Schulte und J. G. Linebarger: *Potential Speedup with Decimal Floating-Point Hardware*. In: *Proceedings of the Thirty Sixth Asilomar Conference on Signals, Systems, and Computers*, 11 2002. 2.8

[177] Wires, K. E., M. J. Schulte und J. E. Stine: *Combined IEEE Compliant and Truncated Floating Point Multipliers for Reduced Power Dissipation*. In: *Proceedings of the International Conference on Computer Design*, 9 2001. 2.8, 35

[178] Wirth, Niklaus: *Program Development by Stepwise Refinement*. Communications of the ACM, 14(4), 1971. 20

[179] Wu, A.Y.: *Algorithm-Based Low-Power Digital Signal Processing System Designs*. Doktorarbeit, University of Maryland, 1995. 4.6.3

[180] Xilinx Corp.: *CORE Generator Guide*, 2008. 4.7.2

[181] Xilinx Corp.: *System Generator for DSP*, 2008. 3.5, 3.3

[182] Yu, T. Z., F. Chen und E. H.-M. Sha: *Loop Scheduling Algorithms for Power Reduction*. In: *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing*, May 1998. 4.3.1

[183] Zhang, Yumin, Xiaobo (Sharon) Hu und Danny Z. Chen: *Task Scheduling and Voltage Selection for Energy Minimization*. In: *DAC-2002: 39th Design Automation Conference, New Orleans, Lousiana*, June 2002. 15

[184] Zölzer, Udo: *Digitale Audiosignalverarbeitung*. Teubner, 1996. 2.13, 2.14, 2.20, 4.4.1

Index

A.1. Glossar

Wo immer möglich, habe ich Fachbegriffe direkt im Text erläutert. Zum Nachschlagen sind hier die wichtigsten Begriffe nochmals aufgelistet.

Allokation ist die Zuordnung einer Menge von Objekten zu einer anderen Menge, siehe auch Abschnitt 4.5.1. In der Hardware-Synthese bezeichnet Allokation das Zuordnen einer Menge von physikalischen Operatoren (Addieren, Multiplizieren, ...) zu der Menge der Operationen in einer Hardwarebeschreibung. Es muss für jede Art von Operationen mindestens einen physikalischen Operator geben. Die Zuordnung einer bestimmten Operation im Datenpfad auf einen bestimmten Operator bezeichnet man als *Binding*.

API *Application Programming Interface*, eine Programmierschnittstelle, die in der Regel aus standardisierten Funktionsaufrufen besteht. Die API stellte die Schnittstelle z. B. einer Bibliothek oder eines Moduls für den Benutzer dar.

ASIC Application Specific Integrated Circuit. Es handelt sich hierbei um ein IC, das im Gegensatz zu einfachen Logikchips oder Mikroprozessoren für eine bestimmte Anwendung entworfen wurden. ASICs werden in den allermeisten Fällen in kleineren Stückzahlen als die letztgenannten produziert und sind deshalb entsprechend teuer.

Binding ist die Zuordnung einer Operation zu einem bestimmten physikalischen Operator. Der Vorgang wird bei der Erzeugung einer RT-Beschreibung aus einer Verhaltensbeschreibung durchgeführt, siehe Abschnitt 4.5.1.

Butterfly Elementare Rechenzelle der FFT. Bei der einfachsten Zerlegung, der Radix-2 FFT, wird lediglich je eine Multiplikation, Addition und Subtraktion durchgeführt. Der Name (= Schmetterling) stammt aus der Überkreuzung der Signalwege im Signalflussgraphen.

Cache Schnelle Zwischenspeicher. Es handelt sich um einen SRAM-Speicher, der zwischen Prozessor und Hauptspeicher geschaltet ist. Er entkoppelt den schnellen Prozessor von dem relativ langsamen Hauptspeicher. Um den Zustand des Systems bei gleichzeitig hoher Leistung konsistent zu halten, muss u.U. viel zusätzlicher Aufwand getrieben werden.

Constraint (Randbedingung, Einschränkung) Dieser Begriff hat sich im Zusammenhang mit der Hardwaresynthese als Bezeichnung für die von der Designerin vorgegebenen Zwangsbedingungen etabliert. Dies sind meistens zeitliche Bedingungen (z. B. minimaler globaler Takt) können aber auch geometrische Parameter wie Abmessungen der Chip-Fläche oder die Lage bestimmter Anschlüsse (*Pads*) sein.

Core Ein „Core“ (Kern) ist ein schaltungstechnisch implementiertes Modul, das eine bestimmte Funktionalität hat. Ein Core hat eine Anzahl von Anschlüssen („Ports“), über die es mit der Außenwelt kommunizieren kann. Cores können in Schaltungsentwürfen als „Subdesigns“ eingebaut werden.

DRAM Dynamic RAM. Bei dynamischem RAM wird jedes Bit in einem kleinen Kondensator gespeichert. Die Kondensatoren (ihre Kapazität liegt in der Größenordnung von Atto-Farad) haben eine endliche Leckrate, so dass ihr Inhalt alle paar

Millisekunden aufgefrischt werden muss. DRAM ist i.A. langsamer als SRAM und verbraucht wesentlich mehr Energie.

DesignWare DesignWare ist eine sehr umfangreiche IP-Bibliothek von Synopsys. Sie enthält zum einen Standardimplementierungen für die arithmetischen Grundoperationen, z. B. Ripple-, CSA- und CLA-Addierer. Diese können automatisch inferiert werden, d. h. während der Synthese wird automatisch eine geeignete Implementierung synthetisiert. Zum anderen sind auch IP-Blöcke für komplexere Funktionen wie z. B. PCI-Interfaces oder UARTs vorhanden, die aber explizit im HDL-Quelltext instantiiert werden müssen.

DSP Digital Signal Processor. Digitale Signalprozessoren sind auf Signalverarbeitungsanwendungen spezialisierte Mikroprozessoren. Sie verfügen über einen Befehlssatz, der Matrixoperationen durch den bereits erwähnten MAC-Befehl unterstützt. Gegenüber einem Vielzweck-Mikroprozessor weisen DSPs eine einfachere Struktur auf, von der Rechenleistung übertreffen sie erstere durch ihren hohen Spezialisierungsgrad häufig.

Dual-Ported RAM Bei diesem RAM-Typ sind Adress- und Datenbus doppelt vorhanden, so dass gleichzeitig eine Schreiboperation auf eine, und eine Leseoperation auf eine andere Adresse erfolgen können. Einige Dual-Ported-RAM-Designs erlauben auch den gleichzeitigen Schreibzugriff auf zwei verschiedene Adressen. Da der schaltungstechnische Aufwand für solches RAM relativ hoch ist, verwendet man nach Möglichkeit Single-Ported-RAM.

ERB (*Effective Rectangular Bandwidth*). Die ERB eines Bandpass-Filters ist die Bandbreite eines Rechteck-Filters mit gleicher Maximalverstärkung und Energie der Impulsantwort.

FFT Fast Fourier Transform. Historisch zunächst die Bezeichnung für Cooley-Tukey-Algorithmen, hat sich die Abkürzung als Oberbegriff für alle beschleunigten Fouriertransformationsalgorithmen, die auf Multiplikation mit Twiddle-Faktoren beruhen, durchgesetzt. Andere Algorithmen wie die Polynomtransformationsmethode oder der Winograd-Algorithmus werden bei einigen Autoren auch als FFTs bezeichnet.

FIR-Filter Finite Impulse Response Filter. Dieser Filtertyp wird in der Signalverarbeitung sehr häufig eingesetzt. Die Filterkomponenten sind nicht rückgekoppelt, deshalb kann die Filterantwort einen endlichen Wert nicht überschreiten. Sie sind auf DSPs sehr einfach zu kodieren, haben aber den Nachteil, dass sie für große Eingangsvektorlängen sehr langsam werden, da ihre Zeitkomplexität $O(n^2)$ beträgt. Frequenzraumfilterung dagegen hat durch die FFT eine Zeitkomplexität von $0(n \log_2 n)$.

Floorplan Bei Standardzell-Entwürfen bezeichnet der Floorplan die geometrische Anordnung der Standardzellen auf dem ASIC nebst der benötigen Verdrahtung.

FPGA Field Programmable Gate Array. Programmierbarer Logikchip, der aus einer Vielzahl identischer logischer Blöcke aufgebaut ist. Die Blöcke können logische n-zu-1-Funktionen annehmen und sind durch eine konfigurierbare Verdrahtung miteinander verbunden.

Full-Custom-Entwurf Chipentwurf auf unterster Ebene. Die Geometrie des gesamten Chip-Layouts wird von Hand festgelegt. Dies ermöglicht das individuelle Anpassen von Treiberleistung, Leitungslängen, Transistorgeometrie usw.. Durch den hohen Zeitaufwand wird es nur dann gemacht, wenn es spezielle Anforderung wie höchste Geschwindigkeit oder geringster Energieverbrauch an das Design gibt. Normalerweise werden ASIC mit Hilfe von Standardzellen entworfen. Diese Zellen enthalten fertige Gatter, Flipflops, Latches usw. und müssen nur noch geeignet verdrahtet werden.

Handshake-Leitung Mit diesem Begriff bezeichnet man eine Leitung, die den Zustand einer Design-Komponente anzeigt und damit andere Teile des Designs zu Aktionen veranlassen kann. Beispielsweise wird dem I/O-Master über eine solche Leitung mitgeteilt, dass Daten zur Abholung bereitstehen.

IP *Intellectual Property*, bezeichnet ein meist kommerziell vertriebenes Funktionsmodul für die Integration in eigene Chip-Designs. Es werden hiermit z. B. Standardschnittstellen wie PCI oder USB implementiert aber auch spezielle Signalverarbeitungsmodule wie FFT oder Filterfunktionen. Weiterhin gibt es komplexe Module, die z. B. komplette CPUs einfach integrationsfähig machen. Die IP-Module werden oft als platzierter und verdrahteter Floorplan (z. B. ARM-Prozessoren), in einigen Fällen auch als RT-Komponenten (z. B. Synopsys DesignWare) vertrieben.

Kritischer Pfad ist in der Schaltungstechnik derjenige Logikpfad, der die maximale Taktfrequenz der Schaltung bestimmt. Diese ist durch die summierten Laufzeiten aller direkt miteinander verbundener Gatter gegeben.

Latenzzeit Wenn der Durchsatz einer Funktion mit Hilfe von Pipelining gesteigert werden soll, werden die Daten mindestens einen Takt in jeder Stufe bearbeitet. Dadurch vergeht eine gewisse Zahl von Takten, bis das Ergebnis der Gesamtoperation verfügbar ist. Diese Zeit wird mit Latenzzeit bezeichnet.

Pipelining Fließbandverarbeitung. Diese Technik erhöht den Durchsatz einer Funktion durch Zerlegen einer Funktion in sequentielle Schritte. Jeder dieser Schritte wird nun in einer eigenen Verarbeitungseinheit bearbeitet und von Stufe zu Stufe weitergereicht. Im Idealfall werden ständig neue Daten angeliefert, die Pipeline ist immer gefüllt. Dann steigt der Datendurchsatz um die Anzahl der Pipeline-Stufen. Von dieser Technik wird im modernen Chip-Design häufig Gebrauch gemacht. Der Nachteil ist, dass große Latenzzeiten auftreten können.

Synthetic Module Ein *Synthetic Module* enthält die abstrakte Definition eines Operators und Hardware-Beschreibungen des Operators auf Verhaltens- oder RT-Ebene, sog. Implementierungen. Weiterhin gibt es für jede Implementierung ein Synthese-Skript, das den physikalischen Operator für die spezifizierte Zieltechnologie synthetisiert. Auf diese Weise ist es möglich, technologieunabhängige Hardware-Module in den Synthese-Flow zu integrieren. *Synthetic Modules* werden in einer *Synthetic Library* zusammengefasst (siehe auch Abschnitt 4.5.1).

Toolchain Unter Toolchain versteht man die Verkettung verschiedener Werkzeuge, um eine Algorithmusbeschreibung auf einer hohen Abstraktionsebene auf eine funktional äquivalente Beschreibung auf niedriger Abstraktionsebene abzubilden. Für das Framework sind dies ein knappes Dutzend verschiedener Werkzeu-

ge, die durch Skripte in der richtigen Reihenfolge aufgerufen werden. Dabei sind an einigen Stellen Format-Konvertierungen nötig. Die Skriptsteuerung ermöglicht ferner eine (begrenzte) automatische Entwurfsraumexploration und verfügt über einen Fehler-Report Mechanismus, um Problem erkennen zu können, ohne die Vielzahl der Log-Dateien der einzelnen Werkzeuge durchsuchen zu müssen.

Top-Level Die höchste Stufe in einem hierarchischen Design. Der Top-Level umfasst den gesamten zu implementierenden Chip.