Modellbasierte Effizienzanalyse grobgranularer rekonfigurierbarer Prozessorarchitekturen

von der Fakultät

Elektrotechnik und Informationstechnik

der Universität Dortmund

genehmigte

Dissertation

zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften

von

Hendrik Lange

Dortmund, 2007

Tag der mündlichen Prüfung: 25.10.2007

Hauptreferent: Prof. Dr.-Ing. H. Schröder

Korreferent: Prof. Dr.-Ing. H. Fiedler

Vorwort

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Arbeitsgebiet „Schaltungen der Informationsverarbeitung“ der Universität Dortmund.

Dem Leiter des Arbeitsgebietes, Herrn Prof. Dr.-Ing. H. Schröder, möchte ich für die Anregung, Betreuung und Begutachtung der Arbeit meinen ganz besonderen Dank aussprechen. Durch fruchtbare Diskussionen, seine Hilfsbereitschaft, und durch das von ihm geschaffene hervorragende Arbeitsklima hat er erheblich zum Gelingen der Arbeit beigetragen.

Herrn Prof. Dr.-Ing. H. Fiedler vom Lehrstuhl für Intelligente Mikrosysteme danke ich für sein Interesse an der Arbeit und für die Übernahme des Korreferats.

Herrn Dr.-Ing H. Blume und Herrn Prof. Dr.-Ing. T.G. Noll vom Lehrstuhl für Allgemeine Elektrotechnik und Datenverarbeitungssysteme der RWTH Aachen danke ich sehr herzlich für die Anregung dieser Arbeit, sowie für die mir entgegengebrachte Hilfsbereitschaft und die langjährige gute Zusammenarbeit.

Bei der wissenschaftlichen Arbeit habe ich wesentlich von Erfahrungen und Erkenntnissen profitiert, die ich im Vorfeld im Rahmen einer Industriekooperation mit der Firma Nokia gesammelt habe. Allen Mitarbeitern des Projekt-Teams, mit denen ich zusammenarbeiten durfte, möchte ich dafür ebenfalls sehr herzlich danken.

Weiterhin danke ich allen Kollegen, studentischen Mitarbeitern, Studien- und Diplomarbeitern, die durch ihren Einsatz und anregende Diskussionen zu dieser Arbeit beigetragen haben.

Nicht zuletzt und ganz besonders danke ich meinen Eltern dafür, dass sie mir das Studium ermöglicht haben, und mir bei dieser Arbeit sehr viel Unterstützung zukommen ließen.

Hendrik Lange

Inhaltsverzeichnis

Notation

Abkürzungen

ALU	Arithmetic Logical Unit
ARA	Array Analyzer
ASIC	Application Specific Integrated Circuit
ASIP	Application Specific Instruction Set Processor
CMOS	Complementary Metal Oxide Semiconductor
CPLD	Complex Programmable Logic Device
DCT	Diskrete Cosinustranformation
DRAM	Dynamic Random Access Memory
DSP	Digitaler Signalprozessor
EDA	Electronic Design Automation
eFPGA	Eingebetteter (embedded) FPGA
FFT	Fast Fourier Transform, schnelle Fouriertransformation
FPGA	Field Programmable Gate Array
HDTV	High Definition Television
IP	Intellectual Property
LUT	Lookup-Tabelle
MIMD	Multiple Instructions, Multiple Data
PAL	Programmable Array Logic
PE	Prozessorelement

PROM	Programmable Read Only Memory
QCIF	Quarter Common Intermediate Format
SIMD	Single Instruction, Multiple Data
SoC	System-on-a-Chip
SPICE	Simulation Program with Integrated Circuit Emphasis
SRAM	Static Random Access Memory
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VLIW	Very Long Instruction Word
VLSI	Very Large Scale Integrated

Operatorbezeichnungen

$\min\{\cdot\}$	Minimum-Operator
$\max\{\cdot\}$	Maximum-Operator

Operatordefinitionen

$\lceil x \rceil$	$\lceil x \rceil = \min\{y \in \mathbb{Z} \vert y \geq x\}$
$\lfloor x \rfloor$	$\lceil x \rceil = \max\{y \in \mathbb{Z} \vert y \leq x\}$

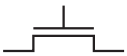
Formelzeichen

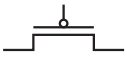
A_{gesamt}	Gesamtfläche
A_{PE}	Fläche einer Basiszelle
A_{RAM}	Fläche eines RAM-Blockes
A_{SB}	Fläche eines Switchblockes
$A_{SRAM-Zelle}$	Fläche einer SRAM-Zelle
$A_{VB,h}$	Fläche eines Verbindungsblockes für horizontale Routingleitungen

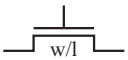
$A_{VB,v}$	Fläche eines Verbindungsblockes für vertikale Routingleitungen
$C_{Buf,in}$	Eingangskapazität eines Leitungstreibers
$C_{Buf,out}$	Ausgangskapazität eines Leitungstreibers
$C_{Leitung}$	Äquivalente Kapazität eines Leitungssegmentes
$C_{TG,leitend}$	Äquivalente Kapazität eines Transmission-Gates im leitenden Zustand
$C_{TG,sperrend}$	Äquivalente Kapazität eines Transmission-Gates im sperrenden Zustand
E	Energie
L_{min}	Minimale MOS-Transistor-Gatelänge
N_d	Globale Datenwortbreite
N_{HK}	Anzahl horizontaler globaler Routingkanäle
$N_{PE,s}$	Anzahl der PE-Spalten
$N_{PE,z}$	Anzahl der PE-Zeilen
$N_{PT,SB}$	Anzahl von Pass-Transistoren im Switchblock
$N_{PT,VB}$	Anzahl von Pass-Transistoren im Verbindungsblock
N_{VK}	Anzahl vertikaler globaler Routingkanäle
$\bar{P}$	Durchschnittsleistung
$Pfad(a \rightarrow b)$	Pfad innerhalb eines Netzwerkes von Knoten a nach Knoten b
$p(t)$	Augenblicksleistung
$P_V^{(i,b)}$	Leistungsaufnahme von Bitleitung b in Pfad i
R_{Buf}	Äquivalenter ohmscher Widerstand eines Leitungstreibers
$R_{Leitung}$	Äquivalenter ohmscher Widerstand eines Leitungssegmentes
R_{TG}	Äquivalenter ohmscher Widerstand eines Transmission-Gates
$\tau_{50\%}$	50%-Signalverzögerung
$\tau_{Buf,int}$	Intrinsische Verzögerung eines Leitungstreibers

τ_{Elmore}	Elmore-Verzögerung
τ_{SPICE}	Mit SPICE-Simulation gemessene Verzögerungszeit
V_{DD}	Betriebsspannung
W_{min}	Minimale MOS-Transistor-Gatebreite

Schaltsymbole


n-Kanal MOS-Transistor


p-Kanal MOS-Transistor



n-Kanal-Transistor mit Gate-Abmessungen $W_{Gate} = w \cdot W_{min}$ und $L_{Gate} = l \cdot L_{min}$


Buffer (Leitungstreiber)


Tristate-Treiber

Buffer mit n-facher minimaler Treiberstärke


Inverter

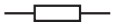

Diode


Transmission Gate

Bus mit n parallelen Leitungen

Bus-Slicing


Kondensator


Ohmscher Widerstand

Masse

Betriebsspannung


Eingangspin

Ausgangspin

Kurzfassung

Die vorliegende Arbeit befasst sich mit Analysemethoden für grobgranulare rekonfigurierbare Prozessorarchitekturen. Es wird ein modellbasiertes Verfahren vorgestellt, mit dessen Hilfe derartige Hardwarestrukturen auf einfache Weise bezüglich ihres Flächenbedarfs, der Datenrate und des Energieverbrauchs charakterisiert werden können. Weiterhin werden Untersuchungsergebnisse dargestellt, die einen Vergleich grobgranularer Architekturen mit anderen Implementierungsformen ermöglichen.

Field-Programmable Gate-Arrays (FPGAs) stellen aufgrund der Rekonfigurierbarkeit auf Bitebene flexible Bausteine mit hoher Verarbeitungsgeschwindigkeit dar, allerdings auf Kosten eines enormen Flächenoverheads und hohen Energieverbrauches. Demgegenüber enthalten grobgranulare Architekturen komplexere, anwendungsspezifisch optimierte Datenpfade, die eine Rekonfigurierbarkeit auf Wortebene gestatten.

Es ist ein parametrisierbares Modell für eine grobgranulare rekonfigurierbare Architektur entwickelt worden. Angenommen wird ein zweidimensionales Feld, dessen Basiszellen über lokale Datenleitungen, sowie über segmentierbare Busse miteinander kommunizieren können. Weiterhin sind auf lineare Algorithmen hin optimierte Prozessorelemente unterschiedlicher Komplexität entwickelt worden. Anhand von physikalischen Modellen zur Ermittlung des Flächenbedarfs, der maximalen Taktfrequenz und des Energieverbrauches werden die VLSI-Eigenschaften der Modelle abgeschätzt, und denen von FPGAs, DSPs und Semi-Custom-Entwürfen gegenübergestellt.

Die Ergebnisse zeigen, dass grobgranulare Architekturen im Vergleich zu FPGAs je nach Implementierung eine 7 bis 20 mal höhere Flächen- und Energieeffizienz aufweisen. Die Datenrate beider Varianten liegt dabei etwa in der gleichen Größenordnung. Bei Abbildung von Algorithmen, für welche die grobgranularen Datenpfade nicht optimiert wurden, sinkt die Effizienz dagegen erwartungsgemäß deutlich ab.

1 Einleitung und Übersicht

Die Komplexität der Algorithmen zur digitalen Signalverarbeitung in modernen Endgeräten hat in den letzten Jahren und Jahrzehnten stark zugenommen. Anwendungen wie das hochauflösende Fernsehen (HDTV), oder Mobilfunksysteme der dritten Generation (z.B. UMTS) erfordern eine hohe Rechenleistung bei gleichzeitig möglichst platzsparender Implementierung. Insbesondere mobile Endgeräte erfordern zudem einen möglichst geringen Energieverbrauch. Die Realisierbarkeit solcher Systeme wurde ermöglicht durch den seit Jahrzehnten exponentiell zunehmenden Integrationsgrad von Transistoren in mikroelektronischen Schaltungen bei gleichzeitig immer geringer werdenden Transistorschaltzeiten. So prädiziert das als Moore'sches Gesetz bekannt gewordene Postulat von Intel-Mitbegründer Gordon E. Moore aus dem Jahr 1965 eine Verdoppelung der Integrationsdichte auf integrierten Schaltungen etwa alle 18 Monate. Tatsächlich wurde der 1971 eingeführte Intel 4004-Prozessor mit 2300 Transistoren realisiert, der 1993 eingeführte Pentium mit 3.100.000 Transistoren, und der Itanium 2 mit 9MB Cache im Jahr 2004 bereits mit mehr als einer halben Milliarde Transistoren (592.000.000). Eine derart hohe Transistorendichte erlaubt seit einigen Jahren die Implementierung kompletter Systeme auf einem einzigen Chip statt wie bisher auf einer Platine mit mehreren in getrennten Gehäusen untergebrachten Einzelkomponenten. Für einen solchen Baustein ist die Bezeichnung *System-on-a-Chip* eingeführt worden (häufig auch *System-on-Chip* oder abgekürzt *SoC*). Vorteile gegenüber platinenbasierten Lösungen ergeben sich unter anderem durch eine erhebliche Flächenersparnis, höhere Geschwindigkeiten durch breite chipinterne Datenbusse und eine bessere Energieeffizienz.

Ein SoC ist demzufolge naturgemäß eine heterogene Struktur, die sich aus einer Vielzahl möglicher Systemkomponenten zusammensetzt. So können neben Prozessoren, Coprozessoren und Speichereinheiten auch analoge Komponenten wie A/D-Wandler auf einem einzelnen Chip integriert werden. Ein Problem tritt auf, wenn rechenintensive Algorithmen implementiert werden müssen, die aufgrund ihrer Komplexität nicht mit der geforderten Verarbeitungsgeschwindigkeit in Software auf einem der integrierten Prozessoren ausgeführt werden können. Der klassische Ansatz zur Lösung dieses Problems ist die Anbindung eines dedizierten Architekturblockes für den betreffenden Algorithmus auf dem SoC. In einer vollkundenspezifischen Realisierung lassen sich hier die besten Ergebnisse in Bezug auf den Flächenbedarf, den Datendurchsatz und den Energieverbrauch erzielen. Nachteilig bei dieser Lösung ist jedoch, dass sich eine solche dedizierte Schaltung, und damit der Algorithmus, nach Produktion und Serienauslauf des Chips nicht mehr ändern lässt. Dies ist insbesondere bei Systems-on-a-Chip von großer Tragweite, da eine Änderung eines Teils

des Systems eine Neuproduktion des kompletten Chips nach sich zieht, was mit hohem Zeitaufwand und erheblichen Kosten einhergeht.

Wünschenswert ist demzufolge eine Architekturkomponente, dessen Datendurchsatzrate einem kundenspezifischen Schaltkreis nahe kommt, dabei jedoch flexibel genug ist, um nach Fertigstellung des Chips noch an Änderungen der jeweiligen Anwendung anpassbar zu sein. Strukturen, die eine solche Möglichkeit bieten, werden als *rekonfigurierbare Architekturen* bezeichnet. Diese unterscheiden sich von programmierbaren Komponenten wie Prozessoren unter anderem dadurch, dass die Struktur der zur Berechnung erforderlichen Hardwareressourcen nach der Produktion des Chips geändert (*rekonfiguriert*) werden kann. Ansätze hierzu wurden bereits mehrere Jahrzehnte vor Beginn der SoC-Ära zunächst in Form programmierbarer Logikbausteine (PALs, PLAs oder PROMs) entwickelt. Von praktischer Bedeutung für Systems-on-a-Chip sind jedoch die 1986 von der amerikanischen Firma Xilinx erstmals vorgestellten *Field Programmable Gate Arrays (FPGAs)*. Diese Bausteine bestanden in ihrer ursprünglichen Form aus mehreren Dutzend oder einigen hundert programmierbaren Logikzellen, die über ein ebenfalls programmierbares Verbindungsnetzwerk miteinander kommunizierten. Jede Zelle konnte logische Funktionen mit bis zu vier Variablen realisieren. Nach stetiger Weiterentwicklung der FPGAs sind heute Bausteine erhältlich, die neben mehreren hunderttausend Logikzellen auch dedizierte Logik wie Multiplizierer oder ganze Prozessoren enthalten. Aufgrund ihrer Flexibilität werden FPGAs beispielsweise zur schnellen Erstellung von Prototypen digitaler Schaltungen eingesetzt. In SoCs eingebettete (*embedded*) FPGAs sind geeignet, die Funktionalität des Chips flexibel zu erweitern. Beispielsweise hat STMicroelectronics im Jahr 2003 einen Aufsatz über einen SoC veröffentlicht, welcher neben einem Mikroprozessor zusätzlich ein daran angekoppeltes eingebettetes FPGA enthält, der zur flexiblen Erweiterung des Befehlssatzes des Prozessors dient (siehe [BLFC03]). Das Thema eingebettete FPGAs für SoCs ist heute ein aktuelles Forschungsgebiet.

FPGAs bieten eine Rekonfigurierbarkeit auf Bit-Ebene, da theoretisch jede einzelne Bit-Leitung der Verbindungsstruktur unabhängig konfiguriert werden kann. Aus diesem Grunde werden FPGAs sehr häufig auch als *feingranulare* rekonfigurierbare Architekturen bezeichnet. Allerdings ist mit der hohen Flexibilität gleichzeitig eine sehr hohe Flächenineffizienz verbunden, da ein Großteil der Fläche des Bausteins bereits durch die Routingressourcen vereinnahmt wird. Mit dem hohen Zusatzaufwand zur Gewährleistung der Rekonfigurierbarkeit geht eine gleichfalls schlechte Energieeffizienz einher, was den Einsatz allgemein verwendbarer eingebetteter FPGAs insbesondere für mobile Endgeräte problematisch macht.

Die schlechten Eigenschaften dieser FPGAs liegen also ursächlich in der hohen Flexibilität begründet. Für einige Einsatzbereiche von FPGAs wie beispielsweise die Erstellung von Prototypen, ist eine solche Flexibilität unabdingbar, da nahezu jede beliebige Schaltung implementierbar sein muss. Bezogen auf einen SoC, dessen Anwendungsgebiet bereits vor der Entwurfsphase feststeht, ist eine solche allgemeine Flexibilität jedoch in vielen Fäl-

len nicht erforderlich. Es stellt sich daher die Frage, ob und wie die Strukturen bei für ein bestimmtes Anwendungsgebiet noch ausreichender Flexibilität optimiert werden können. Hier sind zwei unterschiedliche Ansätze denkbar, die beide in verschiedenen Forschungsprojekten untersucht werden. Eine Möglichkeit ist die Optimierung von FPGA-Strukturen für bestimmte Anwendungen unter Beibehaltung der feinen Granularität. Arbeiten hierzu sind beispielsweise in [NBFN03] zu Anwendungen bei elektronischen Medien oder in [vNBN06] veröffentlicht worden. Die andere Möglichkeit liegt in der Verwendung spezialisierter Datenpfade mit höherer Wortbreite. Derartige Strukturen werden als *grobgranulare* rekonfigurierbare Architekturen bezeichnet. Diese bilden seit etwa Anfang der 1990er Jahre ein heute noch immer aktuelles Forschungsthema, aus dem zahlreiche Veröffentlichungen und Vorschläge zu Architekturen für unterschiedliche Anwendungsgebiete hervorgegangen sind. Viele dieser Architekturen bestehen im Kern aus einem zweidimensional angeordneten Feld aus rekonfigurierbaren Basiszellen, die durch ein geeignetes Verbindungsnetzwerk miteinander kommunizieren. Die Datenpfade enthalten meist einen oder mehrere Multiplizierer, arithmetisch-logische Einheiten und Register, sowie Multiplexer zum Umschalten der Konfiguration der Zelle. Die Wortbreiten der in der veröffentlichten Literatur vorgeschlagenen Architekturen liegen in einem breiten Spektrum zwischen 2 und 32 Bit. Einige dieser Architekturen werden in dieser Arbeit noch näher beschrieben. Allen grobgranularen rekonfigurierbaren Architekturen gemein ist eine applikationsspezifische Ausrichtung. Die Flexibilität ist durch die vorgegebenen Datenpfad-Strukturen eingeschränkt, jedoch bezogen auf das betrachtete Anwendungsgebiet noch ausreichend hoch.

Definition der Zielsetzung

In Abbildung 1.1 ist qualitativ die subjektive Flexibilität verschiedener SoC-Architekturkomponenten gegenüber deren Rechenleistung aufgetragen. Durch Software programmierbare Lösungen wie Universalprozessoren, digitale Signalprozessoren (DSPs) oder Prozessoren mit anwendungsspezifischem Instruktionssatz (*Application Specific Instruction Set Processors*, ASIPs) bieten die größte Flexibilität, jedoch nur eine begrenzte Datenrate. Das andere Extrem bilden dedizierte vollkundenspezifische Entwürfe (Full Custom Makros), die bis auf Transistor- und Maskenebene auf maximale Datenrate optimiert werden können, dafür jedoch nach der Fertigung des Chips nicht mehr änderbar sind. Einen Kompromiss zwischen Datenrate und Flexibilität bieten die oben angesprochenen FPGAs. Interessant ist in diesem Zusammenhang die Frage, wie Architekturen mit grober Granularität im Vergleich zu den anderen Implementierungsformen bezüglich Datenrate, Flächenbedarf, Energieverbrauch und Flexibilität einzuordnen sind.

Seit das mögliche Potenzial grobgranularer rekonfigurierbarer Architekturen erkannt wurde, haben sich eine Vielzahl von Forschungsprojekten mit diesem Thema befasst. Frühe Arbeiten beschäftigten sich vor allem mit der Entwicklung erster Prototypen für die neuartigen Rechenstrukturen. Bekannt sind in diesem Zusammenhang beispielsweise die Pro-

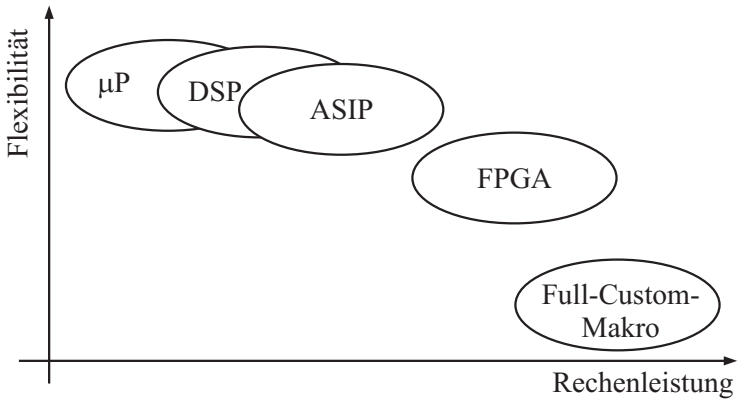


Abbildung 1.1: Qualitative Gegenüberstellung von Flexibilität und Rechenleistung verschiedener SoC-Architekturblöcke

jekte PADDI (erstmals veröffentlicht 1990, siehe z.B. [CR92]), MATRIX (1996,[MD96]), RaPiD (1996, [ECF96]) oder RAW (1997, [Wai97]). Einhergehend mit einer Weiterentwicklung der Architekturen und der Miniaturisierung der Prozesstechnologien sind die in jüngerer Zeit veröffentlichten Arbeiten und aktuell laufende Forschungsprojekte thematisch weiter gefächert. Neben der Weiterentwicklung der Strukturen selbst bilden derzeit beispielsweise die Entwicklung von Software-Tools und Entwurfsmethodiken für grobgranulare Architekturen, die SoC-Integration der Komponenten oder die Untersuchung dynamischer, zur Laufzeit rekonfigurierbarer Architekturen aktuelle Forschungsgebiete. Auch sind zahlreiche analytische Arbeiten zu Grundlagen oder zur Optimierung grobgranularer Architekturen veröffentlicht worden. Ein bekanntes Beispiel ist die in ([Nag01]) veröffentlichte Entwurfsraumexploration für grobgranulare Strukturen. In [BGDN03] werden die Ergebnisse von Untersuchungen zum Einfluss unterschiedlicher Basiszell-Konfigurationen auf die Performance eines rekonfigurierbaren Feldes dargestellt. Eine Untersuchung des Optimierungspotenzials des Flächenbedarfs anhand automatisch generierter Basiszellarchitekturen wird in [CH01] vorgestellt. Der Einfluss der Architektur der Prozessorzellen, des Algorithmus und der Compilereigeschaften auf Fläche, Datenrate und Leistungsaufnahme wird in [OSKR04] untersucht.

Im momentan veröffentlichten Stand der Forschung fehlen dagegen Untersuchungen, bei denen systematisch die Eigenschaften grobgranularer Architekturen wie Flächenbedarf, Geschwindigkeit, Energieverbrauch und Flexibilität den Merkmalen anderer Realisierungsformen gegenübergestellt werden. In Veröffentlichungen zu neuen Architekturen werden zwar die physikalischen Eigenschaften dargestellt, eine Gegenüberstellung mit anderen Architekturen (FPGAs, DSPs) findet jedoch entweder nicht, oder nur bestimmte Eigenschaften und einzelne Algorithmen betreffend statt. Beispielsweise wird in [MSK+99] die Flächeneffizienz einer 4-Bit-Struktur mit einem FPGA verglichen, allerdings nur für

einzelne arithmetische Grundoperationen. Untersuchungen zu komplexeren Algorithmen oder zum Energieverbrauch werden nicht dargestellt. Die Effizienz eines grobgranularen Feldes mit 16-Bit Datenpfaden wird in [GSL+99] durch Abbildung einer DCT und eines Verschlüsselungsalgorithmus gezeigt. Die Performance wird durch die benötigte Anzahl von Taktzyklen angegeben, während Untersuchungen zu Sampleraten, Flächenbedarf und Energieverbrauch nicht dargestellt sind. In Veröffentlichungen und Datenblättern zu aktuellen kommerziell vermarkteten grobgranularen Architekturen finden sich zwar Angaben zu Flächenbedarf, Performance und Energieverbrauch, diese beziehen sich allerdings aus Gründen des Marketings in den meisten Fällen auf die in besonders günstigen Fällen erreichbaren Werte (siehe z.B. [PAC05], [DPT03]) oder [Sil05]), weshalb diese Daten für eine kritische Gegenüberstellung der Stärken und Schwächen im Vergleich zu Implementierungsalternativen ausscheiden. Ebenso verfolgen die bisher veröffentlichten analytischen Arbeiten zu grobgranularen Architekturen nicht das primäre Ziel eines systematischen Vergleichs grob- und feingranularer Architekturen. So bezieht sich beispielsweise die in [Nag01] vorgestellte Arbeit auf die Suche nach Optima innerhalb des von grobgranularen Architekturen aufgespannten Entwurfsraumes. Absolute Angaben zu Fläche, Datenrate und Energie, die einen Vergleich mit Architekturen außerhalb dieses Entwurfsraumes erlauben würden, erfolgen dagegen nicht. Ähnliches gilt für den in [BGDN03] präsentierten Ansatz.

In dieser Arbeit wird eine neue Methodik zur Analyse und Bewertung grobgranularer rekonfigurierbarer Architekturen vorgestellt. Es ist ein Software-Werkzeug entwickelt worden, mit dessen Hilfe Flächenbedarf, Datenrate und Energieverbrauch parametrisierbarer grobgranularer Modellarchitekturen ermittelt werden können. Zu diesem Zweck sind Verfahren zur Abschätzung dieser Eigenschaften implementiert worden, so dass ohne aufwändige reale Implementierung der zu untersuchenden Hardware eine physikalische Charakterisierung der Architekturen erfolgen kann.

Weiterhin verfolgt die vorliegende Arbeit das Ziel, mit ihren Ergebnissen den derzeit veröffentlichten Erkenntnisstand zu grobgranularen Architekturen zu erweitern, indem diese Strukturen mit bereits existierenden Implementierungsalternativen wie FPGAs, DSPs und dedizierten Architekturen verglichen werden. Dies erfolgt unter Zuhilfenahme des erwähnten Software-Werkzeugs. Interessant ist vor allem die Frage, inwieweit sich durch die anwendungsspezifische Ausrichtung grobgranularer Architekturen Verbesserungen gegenüber feingranularer FPGA-Strukturen ergeben.

Die Untersuchungen erfolgen auf Basis eines Modells für grobgranulare rekonfigurierbare Architekturen für Anwendungen der digitalen Signalverarbeitung. Als Beispielanwendungen werden vor allem Algorithmen gewählt, die sehr häufig in der Bildsignalverarbeitung anzutreffen sind, wie beispielsweise lineare Filter oder mehrdimensionale Transformationen.

Es muss erwähnt werden, dass sich die Ergebnisse aus den Simulationen stets auf das in dieser Arbeit zugrunde gelegte Architekturmodell mit den jeweils gewählten Parametern beziehen. Aufgrund der Vielzahl der Parameter und der daraus resultierenden Größe des

bei grobgranularen Architekturen zur Verfügung stehenden Entwurfsraumes ist die Beschränkung auf ein Modell erforderlich. Während bei FPGAs ein Teil des Entwurfsraumes sinnvoll parametrisiert werden kann (siehe z.B. [BRM99]), ist es bezogen auf grobgranulare Architekturen nicht möglich, eine relevant große Menge aller denkbaren Varianten über eine Parametrisierung zu erfassen, wodurch letztendlich jede Architektur einzeln zu betrachten ist. Die bereits veröffentlichten Vorschläge für grobgranulare rekonfigurierbare Architekturen unterscheiden sich teilweise derart fundamental voneinander, dass es nicht möglich ist, ein Modell zu entwickeln, in dessen Parameterraum alle enthalten sind.

Das in dieser Arbeit verwendete Modell erhebt weiterhin nicht den Anspruch, eine für den gegebenen Anwendungsfall optimale Lösung darzustellen. Gleichwohl sind die Ergebnisse in der Hinsicht allgemeingültig, als dass sie einen Anhaltspunkt liefern, welche Möglichkeiten grobgranulare Architekturen bieten, und welche Verbesserungen in Bezug auf einen nicht-optimierten feingranularen Entwurf und in Bezug auf andere Implementierungsformen zu erzielen sind. Neben den Chancen grobgranularer Architekturen werden ebenso Probleme und Grenzen dieser Realisierungsform diskutiert.

Übersicht

Die vorliegende Arbeit gliedert sich wie folgt:

In *Kapitel 2* wird das dieser Arbeit zu Grunde liegende Konzept hinter modernen Systems-on-a-Chip aufgezeigt. Nach einer kurzen Darstellung der möglichen Systemkomponenten eines SoC werden zunächst die für diese Arbeit relevanten Begriffe wie Rekonfigurierbarkeit und Granularität definiert. Anschließend werden die Grundlagen für fein- und grobgranulare rekonfigurierbare Architekturen dargestellt, und anhand ausgewählter Beispiele illustriert. Es werden Projekte vorgestellt, bei denen rekonfigurierbare Architekturen in heterogene SoCs integriert wurden. Eine Darstellung existierender Ansätze zur Entwurfsraumexploration für fein- und grobgranulare Architekturen findet sich in *Kapitel 3*.

Das dieser Arbeit zugrunde liegende Architekturmodell und die neue Methodik zur Analyse grobgranularer Strukturen wird in *Kapitel 4* beschrieben. Nach einer Darstellung des Analyseablaufs werden alle in das Software-Werkzeug integrierten Systemblöcke erklärt. In *Kapitel 5* werden die implementierten und untersuchten Basiszellstrukturen dargestellt. Schließlich werden in *Kapitel 6* die angewandten Methoden bei der Abschätzung der VLSI-Eigenschaften Fläche, Datenrate und Energieverbrauch erklärt. Hier wird ebenfalls dargestellt, wie die Eigenschaften der in dieser Arbeit zum Vergleich mit den grobgranularen Architekturen herangezogen Implementierungsalternativen FPGA, DSP und dedizierter Architekturblock ermittelt wurden.

Nach einer kurzen Beschreibung von bekannten und teilweise neu definierten Kosten- und Gütemaßen für die untersuchten Architekturen werden in *Kapitel 7* die Simulationsergebnisse dargestellt und diskutiert.

2 SoCs und rekonfigurierbare Architekturen

Die Realisierung hochkomplexer Systeme zur Signalverarbeitung erfordert in der Regel die Integration einer Vielzahl unterschiedlicher Komponenten, die zudem geeignet miteinander kommunizieren müssen. Üblich waren und sind hier Multichip-Lösungen, bei denen Prozessoren, Coprozessoren, Speicher, A/D-Wandler und andere Komponenten in physikalisch getrennten Gehäusen auf einer Platine integriert sind. Mittlerweile allerdings ermöglicht die seit mehreren Jahrzehnten exponentiell wachsende Integrationsdichte von Transistoren bei modernen Prozesstechnologien die Realisierung eines kompletten Systems auf einem einzelnen Chip. Eine derartige Struktur wird als *System-on-a-Chip (SoC)* bezeichnet. Allen SoCs gemeinsam ist eine heterogene Struktur, da Architekturkomponenten wie Mikroprozessoren, DSPs, dedizierte Logik, Speicherblöcke, aber auch analoge Komponenten wie A/D-Wandler auf einer einzelnen Siliziumfläche integriert werden müssen. Dies macht einerseits den Entwurf eines SoC äußerst komplex, führt auf der anderen Seite jedoch durch die Flächenersparnis, sowie höhere Performance und geringere Verlustleistungsaufnahme zu teilweise erheblich effizienteren Realisierungen gegenüber Multichip-Lösungen.

Im SoC-Bereich bieten *rekonfigurierbare* Architekturen die Möglichkeit, den Einsatzbereich des Chips unter Beibehaltung einer hohen Datendurchsatzrate flexibel zu erweitern. Unter Rekonfigurierbarkeit wird die Möglichkeit verstanden, die Funktionalität der Hardware auch nach Produktion und Serienauslauf eines Chips nachträglich an sich verändernde Randbedingungen und Algorithmen anzupassen, oder gar eine vollständig andere Anwendung mit der Architektur betreiben zu können. Hier wird grundsätzlich unterschieden zwischen *feingranularen* und *grobgranularen* rekonfigurierbaren Architekturen. Feingranulare Architekturen wie FPGAs oder CPLDs bieten eine Rekonfigurationsmöglichkeit auf Bit-Ebene. Diese Strukturen bieten demzufolge die größtmögliche Flexibilität, besitzen jedoch durch die Konfigurationsmöglichkeiten auf sehr niederiger Ebene einen hohen Flächenbedarf und, ebenfalls damit verbunden, eine hohe Verlustleistungsaufnahme. Gerade der hohe Energieverbrauch macht feingranulare Strukturen kritisch für Systeme, die überwiegend im mobilen Bereich eingesetzt werden sollen. Demgegenüber sind grobgranulare Architekturen auf Wortebene rekonfigurierbar. Diese Architekturen beinhalten komplexere Datenpfade, die beispielsweise dedizierte Multiplizierer, ALUs oder Komparatoren enthalten können. Vorteile gegenüber feingranularen Strukturen liegen in einer erheblich verbesserten Flächen- und Energieeffizienz, was allerdings naturgemäß mit einem Verlust an Flexibilität einhergeht. Um die Vor- und Nachteile beider Varianten zu kombinieren,

werden von den führenden FPGA-Herstellerfirmen wie Xilinx und Altera hybride Architekturen angeboten, welche neben einem feingranularen Logikzellenfeld auch Multiplizierer, Prozessoren und andere dedizierte Architekturkomponenten beinhalten können.

Wie bereits in der Einleitung erwähnt, ist es das Ziel dieser Arbeit, die Eigenschaften grobgranularer rekonfigurierbarer Architekturen gegenüber feingranularen Strukturen und anderen Implementierungsformen zu bewerten. In diesem Kapitel werden daher zunächst das Konzept, mögliche Systemkomponenten und die Eigenschaften moderner System-on-a-Chip-Architekturen beschrieben. Da insbesondere grobgranulare rekonfigurierbare Architekturen nur anhand eines Vergleiches zu traditionellen Prozessorarchitekturen schlüssig zu motivieren sind, wird nach einer Darstellung des grundlegenden SoC-Konzeptes kurz auf traditionelle SoC-Komponenten wie Universalprozessoren, DSPs und dedizierte Architekturblöcke eingegangen. Anschließend folgt nach einer Definition relevanter Begriffe wie Rekonfigurierbarkeit und Granularität ein Überblick über den aktuellen Stand in Forschung und Technik bezogen auf rekonfigurierbare Architekturen. Neben einer Darstellung des grundsätzlichen Prinzips feingranularer Strukturen werden grobgranulare Architekturen anhand einer Konzeptdarstellung und durch ausgewählte Beispiele beschrieben.

2.1 Systems-on-a-Chip

2.1.1 Konzept

Abbildung 2.1 zeigt ein mögliches Blockschaltbild für einen aus mehreren Architekturkomponenten zusammengesetzten heterogenen SoC. Wie in der Einleitung dieses Kapitels bereits dargestellt, können unterschiedliche Blöcke wie Standardprozessoren, DSPs, eingebettete FPGAs, dedizierte Architekturen, grobgranulare Architekturen und Speicher auf dem Chip integriert sein. Der Aufbau der Kommunikationsstruktur zum Verbinden der Komponenten untereinander ist aktuell Gegenstand umfangreicher Forschungsaktivitäten. Neben klassischen Bussystemen erscheinen hier vor allem die aus der Netzwerktechnik auf on-Chip-Kommunikation adaptierte *Networks-on-Chip (NoCs)* interessant.

Welche der in Abbildung 2.1 dargestellten Blöcke tatsächlich implementiert sind, hängt von der jeweiligen Anwendung und dem vorgesehenen Einsatzbereich des Chips ab. In den folgenden Abschnitten werden die wichtigsten und für diese Arbeit relevanten Systemkomponenten kurz dargestellt.

2.1.2 Architekturkomponenten

Universalprozessoren

Die größte Flexibilität unter allen verfügbaren Rechenkomponenten für SoCs bieten Universalprozessoren. Diese Bausteine bieten die Möglichkeit, nahezu jeden beliebigen Algo-

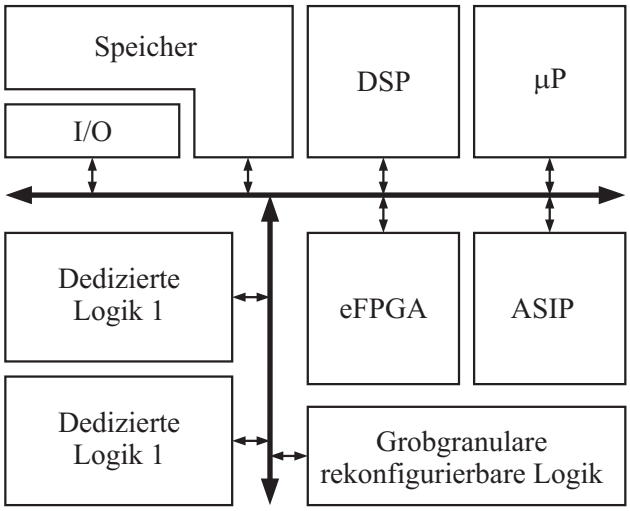


Abbildung 2.1: Blockschaltbild eines heterogenen System-on-a-Chip (SoC)

rithmus zu implementieren, desweiteren existieren für die meisten Universalprozessoren Compiler, die eine bequeme Abbildung des Codes aus einer Hochsprache wie C oder C++ heraus erlauben. Die wichtigsten Komponenten eines Universalprozessors sind die Ausführungseinheit, Steuerlogik, Speicher und Registerfiles, die schnellen Zugriff auf benötigte Operanden ermöglichen. Universalprozessoren sind im Allgemeinen nach der von-Neumann-Rechnerstruktur aufgebaut, dessen wichtigste Eigenschaft in einem gemeinsamen Speicherbereich für Programm und Daten besteht (siehe Abbildung 2.2). Allerdings besitzen die meisten Universalprozessoren getrennte Daten- und Instruktions-Caches, so dass intern von einer Harvard-Architektur gesprochen werden kann. Ferner wird in allen modernen Prozessoren eine Instruktions-Pipeline als Form der Parallelverarbeitung zur effizienteren Ausnutzung der Hardware verwendet. Superskalare Prozessoren können weiterhin durch mehrere parallele Pipelines mehrere Instruktionen gleichzeitig laden.

Sehr häufig wird unterschieden zwischen CISC- und RISC-Prozessoren. Während CISC-Prozessoren einen mächtigen Befehlssatz zur Verfügung stellen, besitzen klassische RISC-Prozessoren einen einfachen Befehlssatz mit relativ wenigen Instruktionen. Die Unterscheidung zwischen RISC und CISC verschwimmt bei aktuellen Architekturen, da einerseits die Anzahl der Instruktionen moderner RISC-Prozessoren bereits ähnlich derer von CISC-Strukturen ist, und andererseits moderne Prozessoren, wie beispielsweise der Intel Pentium IV, die Instruktionen vor der Ausführung in RISC-Instruktionen umwandeln.

In einen SoC integrierte Universalprozessoren werden als eingebettete Universalprozessoren bezeichnet. Viele Firmen bieten Prozessoren als *IP (Intellectual Property)*-Komponenten an, entweder als "Hard-IP" mit bereits fest vorgegebenem Layout, oder als "Soft-IP" in Form synthetisierbarer VHDL- oder Verilog-Komponenten. So ist beispiels-

weise die Firma ARM Ltd. mit 80% Marktanteil im Jahr 2005 führend bei eingebetteten 32-Bit RISC-Prozessoren (siehe z.B. [ARM03]).

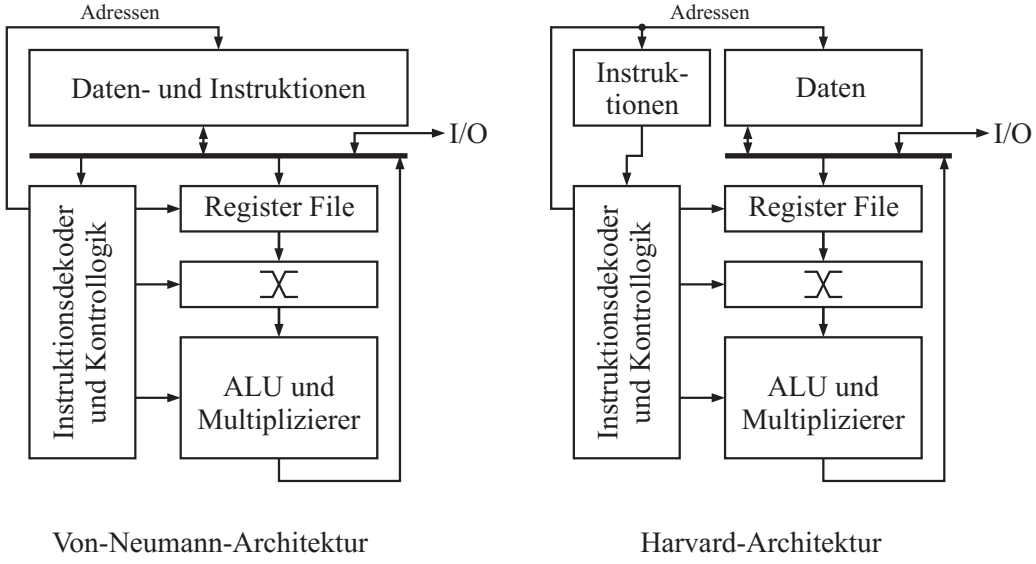


Abbildung 2.2: Vereinfachte Darstellung der von-Neumann- und Harvard-Prozessorarchitekturen

Digitale Signalprozessoren

Ein digitaler Signalprozessor (DSP) ist eine für Anwendungen der digitalen Signalverarbeitung optimierte Prozessorarchitektur. Alle DSPs besitzen dedizierte Funktionseinheiten beispielsweise zur effizienten Berechnung von Multiply-Add-Operationen, und sind nach der Harvard-Architektur mit getrennten Daten- und Instruktionsspeichern aufgebaut (Abbildung 2.2). Häufig kommt auch eine Dual-Harvard-Struktur mit zwei getrennten Datenspeichern zum Einsatz, wie beispielsweise der auch in Abschnitt 6.2.3 beschriebene Adelante RD16025 DSP Core von Philips (siehe z.B. [Moe02b], [Moe02a]). Neben dem großen Markt für diskrete Einzelchip-DSPs konnten sich bereits eine Reihe von Firmen mit eingebetteten DSP-Lösungen etablieren, wie beispielsweise Philips mit dem RD16025, Tensilica mit dem Xtensa (siehe z.B. [Ten04]), einer erweiterbaren, durch den Benutzer konfigurierbaren eDSP-Lösung, Infineon mit dem TriCore (siehe z.B. [Inf02]), oder 3DSP mit der SP-Serie (siehe z.B. [3DS03]). Alle eDSPs dieser Firmen lassen sich zur Verwendung in neuen Produkten lizensieren.

ASIPs

Ein ASIP (Application Specific Instruction Set Processor) ist ein Prozessor mit einem speziell für eine bestimmte Anwendung entwickelten Instruktionssatz. Während auf einem DSP theoretisch auch allgemeine Anwendungen ausgeführt werden können, bietet ein ASIP diese Flexibilität definitionsgemäß nicht mehr, da bereits in der Entwurfsphase des Bausteins auf die anwendungsspezifische Ausrichtung zu achten ist. Vorteilhaft sind die im Vergleich zu Universalprozessoren und DSPs besseren Eigenschaften in Bezug auf Flächenbedarf, Energieverbrauch und Datendurchsatz. Negativ wirken sich allerdings die unter Umständen hohen Entwicklungskosten aus, da neben der Hardware auch ein auf die Eigenschaften der Architektur ausgerichteter Compiler zu entwerfen ist (siehe z.B. [SB00]).

Eingebettete FPGAs

Ein FPGA (*Field Programmable Gate Array*) ist ein integrierter Schaltkreis, dessen Funktionalität durch Laden eines Konfigurationsdatenstroms nach der Entwurfsphase flexibel festgelegt werden kann. Die grundlegenden Architekturkonzepte von FPGAs werden in Abschnitt 2.3.1 dieses Kapitels näher beschrieben. In Abschnitt 2.3.2 folgt eine Übersicht über die Eigenschaften aktueller FPGAs der in diesem Bereich führenden Firmen Xilinx und Altera.

Wie in der Einleitung zu diesem Kapitel bereits dargestellt, bieten eingebettete FPGAs (eFPGAs) die Möglichkeit einer flexiblen Befehlssatzerweiterung eines angekoppelten Prozessors auf einem SoC. Theoretisch können die auch aus dedizierten IC-Lösungen bekannten FPGA-Strukturen als eingebettete FPGAs verwendet werden, wobei der Ring aus Ein- und Ausgangs-Pads (IO-Pads) entfällt. Derzeit ist unklar, inwieweit sich das Konzept, die Flexibilität eines SoC durch ein eFPGA zu erhöhen, durchsetzen wird. So hat beispielsweise die Firma Actel die Weiterentwicklung und Vermarktung des Varicore, eines lizenzierbaren eFPGAs, aufgrund fehlender Nachfrage wieder eingestellt. Dagegen hat sich die Firma M2000 auf die Entwicklung von eFPGA-Makros spezialisiert, und bietet mit dem FlexEOS einen lizenzierbaren Core an.

Insbesondere im SoC-Bereich ist die hohe Flächenineffizienz ein Kriterium, aufgrund dessen sich eFPGAs im kommerziellen Bereich bislang noch nicht durchsetzen konnten. Lediglich in Forschungsprojekten ist die Einbettung einer feingranularen Struktur in SoCs untersucht worden, wie beispielsweise in [BLFC03]. Aktuelle Forschungen im Bereich der eFPGAs konzentrieren sich daher vor allem auf eine Verbesserung der Flächeneffizienz durch Spezialisierung auf bestimmte Anwendungsgebiete (siehe z.B. [NBFN03]).

Grobgranulare rekonfigurierbare Architekturen

Im Gegensatz zu FPGAs enthalten grobgranulare rekonfigurierbare Architekturen spezialisierte Datenpfadelemente, die zudem eine höhere Wortbreite besitzen. Dadurch bieten

diese Strukturen eine wesentlich verbesserte Flächen- und Energieeffizienz als FPGAs, die allerdings mit einem Verlust an Flexibilität einhergeht. Für diese Architekturklasse wird in dieser Arbeit auch abkürzend der Begriff *grobgranulare Architekturen* verwendet.

Grobgranulare Architekturen sind bislang vor allem in Forschungsprojekten entwickelt und untersucht worden. Es existieren jedoch bereits einige Firmen, die sich auf eine kommerzielle Vermarktung dieser Strukturen spezialisiert haben. Die bekanntesten sind Silicon Hive (siehe z.B. [Sil05] und Abschnitt 2.4.2), PACT Informationstechnologie GmbH (siehe z.B. [PAC05] und Abschnitt 2.4.2), Elixent (z.B. [Eli01]), Morpho Technologies mit dem MS-1 (siehe z.B. [Mor04]), oder picoChip (siehe z.B. [pic04]).

Die Erarbeitung der Eigenschaften grobgranularer Architekturen im Vergleich zu feingranularen ist Gegenstand dieser Arbeit. Aus diesem Grunde werden das hinter grobgranularen Architekturen stehende Konzept, sowie einige ausgewählte Beispiele zu kommerziellen und nicht-kommerziellen Strukturen in Abschnitt 2.4 genauer dargestellt.

Häufig kommt es vor, dass eine Architekturkomponente nicht nur einer der hier vorgestellten Klassen zugeordnet werden kann. So besitzt beispielsweise die in Abschnitt 2.4.1 (Seite 32) dargestellte Accelerator-Architektur sowohl Eigenschaften eines ASIPs, als auch einer grobgranularen rekonfigurierbaren Architektur, bedingt durch die Wortebenen-Rekonfigurierbarkeit der Prozessorelemente.

Dedizierte Logik

Falls die Flächen-, Geschwindigkeits- oder Energieanforderungen eines Systems die Implementierung einer Anwendung auf einem programmierbaren oder rekonfigurierbaren Architekturblock ausschließen, muss die Anwendung in Form dedizierter Logik implementiert werden. Diese bietet unter allen Entwurfsalternativen die besten Performanceeigenschaften, ist demgegenüber jedoch unveränderbar, und erfordert zudem den höchsten Entwurfsaufwand. Daher ist in einem möglichst frühen Stadium des SoC-Entwurfs abzuwägen, mit welcher Entwurfsmethode jeweils gearbeitet wird.

Im günstigsten Fall ist der zu implementierende dedizierte Architekturblock in Form eines IP-Cores in einer firmeninternen Bibliothek verfügbar, und kann unmittelbar eingesetzt werden. Falls das nicht der Fall ist, muss der Block neu entworfen werden, was mit Zeitaufwand und entsprechenden Kosten einhergeht. Zur Implementierung einer dedizierten Systemkomponente stehen weiterhin verschiedene Entwurfsverfahren zur Verfügung, welche sich grob in zellbasierte "halbkundenspezifische" (*Semi-Custom-*) Entwürfe, und "vollkundenspezifische" (*Full-Custom-*) Entwürfe klassifizieren lassen. Semi-Custom-Entwürfe verwenden Zellbibliotheken, in denen fertige einfache Bausteine wie beispielsweise Gatter und Flipflops in Form so genannter Standardzellen vorliegen. Mit Hilfe von Entwurfswerkzeugen wie Hardwarebeschreibungssprachen (z.B. VHDL oder Verilog), geeigneten Synthesewerkzeugen und Speichergeneratoren, können so relativ einfach, d.h. ohne spezielle Kenntnisse des transistorbasierten Schaltungsentwurfs, dedizierte Komponeten ent-

worfen werden. Das durch die verwendete Prozesstechnologie implizierte Optimum lässt sich dagegen nur durch einen Full-Custom-Entwurf annähern, bei welchem im Extremfall kundenspezifische Maskensätze manuell editiert werden. Dies lohnt sich in der Regel nur für Schaltungsteile mit extremen Performanceanforderungen und Chips mit sehr hohen Stückzahlen, wie beispielsweise für Datenpfadelemente in Mikroprozessoren.

Dedizierte Logik kann ggf. auch *schwach konfigurierbar* (*weakly configurable*) ausgeführt werden. Bei diesen Implementierungen bleibt der dedizierte Charakter erhalten, jedoch ergänzt um die Möglichkeit, in eingeschränktem Maße benutzerdefinierte Anpassungen vorzunehmen, beispielsweise durch Laden neuer Koeffizientensätze oder durch Umschalten zwischen verschiedenen Betriebsmodi.

Eine Einführung und Gegenüberstellung verschiedener Entwurfsverfahren findet sich beispielsweise in [WH05].

Speicher

Da eine hohe Verarbeitungsgeschwindigkeit einen effizienten Zugriff auf Daten und Instruktionen erfordert, werden auf SoCs neben den Ausführungseinheiten auch Speicherblöcke integriert. Beispielsweise wird in [BLFC03] gezeigt, dass verschiedene Einheiten wie ein RISC-Prozessor und ein eFPGA über ein chipinternes Kommunikationsnetzwerk auf gemeinsame Speicherbereiche zugreifen können.

Da SRAM-Speicher sehr viel Fläche belegt, ist in der Entwurfsphase abzuwägen, wieviel Speicher auf dem Chip integriert werden soll, um die Produktionskosten möglichst gering zu halten. Ansätze, flächeneffizienteren DRAM-Speicher auf einem SoC zu integrieren, haben sich unter anderem aufgrund der erforderlichen zusätzlichen Fertigungsschritte und niedrigeren Zugriffszeiten lange Zeit nicht durchsetzen können. Für neuere Produkte ist der Einsatz von eingebettetem DRAM-Speicher jedoch wieder angekündigt worden (siehe z.B. [Hei05] und [EET05]).

Bussysteme und Networks-on-a-Chip

Ein besonderes Problem beim Entwurf heterogener Systems-on-a-Chip ist die Auswahl einer geeigneten Kommunikationsstruktur, da unter Umständen eine Vielzahl höchst unterschiedlicher Komponenten miteinander verbunden werden müssen. Mit der steigenden Anzahl der auf einem SoC zu integrierenden IP-Cores bekommt dieses Problem eine stark zunehmende Qualität. Neben den Geschwindigkeitsanforderungen ist für eine kurze Entwicklungszeit eine skalierbare und wiederverwendbare Verbindungsarchitektur von höchster Wichtigkeit. Wiederverwendbarkeit bezeichnet in diesem Zusammenhang die Möglichkeit, eine möglichst große Zahl neuer IP-Komponenten an bereits existierende Kommunikationsstrukturen anbinden zu können.

Punkt-zu-Punkt-Verbindungen zwischen zwei Komponenten bieten zwar die größte Datentransferrate, sind jedoch sehr spezifisch und führen bei wachsender Zahl von Komponenten zu unstrukturierten, zur Anbindung weiterer Komponenten schlecht wiederverwendbaren Architekturen. Zur Lösung dieses Problems werden Bussysteme verwendet, über die verschiedene Cores miteinander kommunizieren können. Die Verbindung zweier Komponenten erfolgt leitungsvermittelt, es muss also zunächst ein Request- und Acknowledge-Protokoll durchlaufen werden, um den Bus zu reservieren, bevor anschließend Daten übertragen werden können. Während der Datenübertragung ist die verwendete Busleitung für andere Komponenten gesperrt. Nach [CJT03] können auf diese Weise bis zu zehn Komponenten effizient auf einem SoC miteinander kommunizieren. Bei Anbindung weiterer Einheiten treten zunehmend Synchronisations- und Clock-Skew-Probleme auf.

Ein neuer Ansatz zur Realisierung von SoC-Kommunikationsstrukturen ist in den letzten Jahren unter dem Namen *Network-on-a-Chip (NoC)* bekannt geworden. Im Gegensatz zu verbindungsbehafteten Busstrukturen werden hier aus der Netzwerktechnik bekannte paketvermittelte Protokolle eingesetzt. Ähnlich wie beispielsweise beim Ethernet werden Daten zu Paketen mit einer festgelegten maximalen Größe zusammengefasst, die über das Netzwerk übertragen werden. Auf diese Weise können mehrere SoC-Komponenten gleichzeitig die Verbindungsleitungen nutzen. Die Skalierbarkeit und Wiederverwendbarkeit von paketbasierten NoC-Architekturen bleibt theoretisch für eine beliebige Anzahl von Komponenten erhalten. Nachteilig hierbei ist einerseits, dass Quality-of-Service- (QoS-) Anforderungen bei Netzwerken naturgemäß schwieriger zu erfüllen sind. Außerdem müssen die Kommunikationsschnittstellen aller an das Netzwerk anzubindenden SoC-Komponenten an das verwendete Netzwerkprotokoll angepasst werden. Dies erfordert einen zusätzlichen Entwurfsaufwand, da die Schnittstellen aktuell lizenzierbarer IP-Cores in den meisten Fällen auf ein herkömmliches Bussystem ausgerichtet sind (siehe z.B. [Wei04]). Trotz dieser und zahlreicher weiterer Probleme erscheinen Networks-on-a-Chip insbesondere für zukünftige SoCs interessant, und bilden daher aktuell ein breit diskutiertes Forschungsthema.

2.1.3 SoC-Entwurf

Wie in den obigen Abschnitten dargestellt, ist ein SoC ein häufig extrem komplexer Baustein, da Komponenten wie Prozessoren, Speicher, Ein-/Ausgabeinterfaces und dedizierte Logik auf einer Siliziumfläche integriert werden, die zudem geeignet miteinander kommunizieren müssen. Dies bedingt neue Entwurfsverfahren, was sich sowohl auf die Arbeitsweise einzelner Entwicklungsteams, als auch auf das globale Management einer Entwicklungsmannschaft auswirkt.

Um die Vorteile, die ein SoC gegenüber Board-basierten Multichip-Lösungen beispielsweise in Bezug auf Geschwindigkeit, Flächen- und Energieeffizienz bietet, voll auszuschöpfen, ist ein extrem komplexer Entwurfsablauf erforderlich. Die größte Herausforderung beim Entwurf eines SoC ist die Handhabung der Systemkomplexität. Der Entwurf erfolgt

in der Regel in großen Entwicklungsteams, die weiterhin in kleinere Teams aufgeteilt sind, von denen jedes für einen bestimmten Teil oder eine Komponente des Gesamtsystems zuständig ist. In [WH05] wird dargestellt, dass ein solcher Entwurf zu schlechter Performance des Systems führt, wenn jede Systemkomponente einzeln für sich entworfen und optimiert wird, während demgegenüber bei zu starkem Fokus auf die Interoperabilität die Entwicklungszeit zu groß werden kann, und das Projekt dadurch unwirtschaftlich wird. Das Finden eines geeigneten Tradeoffs wird in [WH05] als eines der Hauptprobleme beim SoC-Entwurf dargestellt.

Weiterhin ist das Problem zu lösen, auf welche Weise ein aus einer Vielzahl unterschiedlicher Systemblöcke zusammengesetztes System, wie beispielsweise eine Signalverarbeitungskette zur Bildsignaldecodierung und Nachverarbeitung, optimal auf die zur Verfügung stehenden Architekturkomponenten zu partitionieren ist. Daher sind zu einem frühen Zeitpunkt des Entwurfsablaufs Entscheidungen zu treffen, welche Systemblöcke beispielsweise in Software auf einem Universalprozessor oder DSP, und welche Komponenten in Form dedizierter Architekturen oder auf rekonfigurierbaren Architekturen zu implementieren sind, um ein in Bezug auf die geforderten Eigenschaften optimales Ergebnis zu erzielen. Die Entwicklungskosten sind umso geringer, je eher eine geeignete Partitionierung bekannt ist. In [BHFN02] wird ein Verfahren vorgestellt, mit welchem anhand von Bibliotheksmodellen die Kosten eines großen Suchbereiches innerhalb des zur Verfügung stehenden Entwurfsraumes abgeschätzt werden können.

Ist die Wahl auf eine bestimmte Systemkonfiguration gefallen, ist es nicht mehr ausreichend, wie in traditionellen Design-Flows zuerst die Hardware, und zu einem späteren Zeitpunkt die Software zu entwickeln. Vielmehr müssen in einem auch als Hardware-Software-Codesign bezeichneten Entwurfsablauf beide parallel zueinander entwickelt werden, was eine enge Kooperation zwischen Hard- und Software-Entwicklungsteams erfordert. Von besonderer Bedeutung beim SoC-Entwurf ist in diesem Zusammenhang das Prinzip des Design-Reuse, der Wiederverwendbarkeit von Architektur- und Softwarekomponenten. Es ist mit vertretbarem Aufwand nicht möglich, in einem aus mehreren hundert Millionen Transistoren bestehenden Chip jede Komponente von Grund auf neu zu entwerfen. Stattdessen ist nach Möglichkeit auf IP-Cores oder fertige Bibliotheksmodelle zurückzugreifen, um die Entwurfskosten unter Kontrolle zu halten. Beim Erstellen von IP-Cores ist weiterhin auf eine gute Parametrisierbarkeit zu achten.

2.2 Begriffsbestimmungen und Definitionen

Zum Verständnis der in dieser Arbeit verwendeten Ausdrücke werden im Folgenden einige Begriffe definiert und näher erläutert.

2.2.1 Programmierbarkeit, Rekonfigurierbarkeit und Granularität

Unter *Programmierbarkeit* wird in dieser Arbeit die grundsätzliche Möglichkeit verstanden, das Verhalten eines vorhandenen Systems an verschiedene Anwendungen oder auszuführende Algorithmen anzupassen. Diese Anpassung erfolgt in Abhängigkeit von der jeweils zu Grunde liegenden Architektur auf unterschiedliche Weise. Mikroprozessoren und DSPs werden durch Laden der Software programmiert, durch welche die jeweilige Anwendung ausgeführt wird. Bei feldprogrammierbaren Bausteinen wie FPGAs erfolgt dagegen die Programmierung durch Laden der Konfigurationsdaten, durch welche die Verbindungsstruktur und Logikzellen applikationsspezfisch konfiguriert werden. Um diese Art der Programmierung von dem ansonsten üblichen Begriff der Software-Programmierung zu unterscheiden, wird hier von *Rekonfigurierbarkeit* gesprochen. Da die Anpassung der Struktur nicht oder nicht ausschließlich durch Software, sondern zusätzlich durch Konfiguration der Hardware erfolgt, wird in dieser Arbeit in Anlehnung an [TB01] die folgende Definition für Rekonfigurierbarkeit verwendet:

> *Definition (Rekonfigurierbarkeit):* Der Begriff Rekonfigurierbarkeit bezeichnet die Möglichkeit, die logische Funktionalität und die Verbindungsstruktur einer vorhandenen Architektur kundenspezifisch an eine bestimmte Anwendung anzupassen.

Wenn eine Rekonfiguration des Systems eine Unterbrechung der laufenden Anwendung erfordert, wird von *statischer Rekonfigurierbarkeit* gesprochen. Ist dagegen eine Umkonfigurierung zur Laufzeit möglich, wird dies als *dynamische Rekonfigurierbarkeit* bezeichnet. Diese beiden Begriffe werden in Abschnitt 2.2.3 näher erläutert.

Von besonderer Bedeutung für diese Arbeit ist eine genaue Definition des Begriffs der *Granularität* einer rekonfigurierbaren Architektur. Es existieren im Wesentlichen zwei Möglichkeiten, den Granularitätsgrad zu definieren, von denen beide hier vorgestellt werden sollen. Die erste Definition definiert Granularität anhand der Wortbreite der Datenpfade (siehe z.B. [DW99]):

> *Definition (durch Wortbreite bestimmter Granularitätsgrad):* Die Granularität einer Architektur wird bestimmt durch die Anzahl der Bits, die innerhalb eines Wortes durch eine einzelne Operation angesprochen werden.

In Prozessoren ist der Granularitätsgrad nach dieser Definition identisch mit der Datenpfadbreite (z.B. 16 oder 32 Bit), während bei FPGAs der Granularitätsgrad 1 Bit beträgt, da theoretisch jedes Bit unabhängig von allen anderen Bits durch das Logikzellenfeld geroutet werden kann. Häufig werden Architekturen, deren Granularitätsgrad nach obiger Definition ein bis vier Bit beträgt, als *feingranular*, und alle anderen Architekturen als *grobgranular* bezeichnet.

Alternativ besteht die Möglichkeit, Granularität anhand der Komplexität der zugrunde liegenden Basiszellstruktur zu definieren:

> *Definition (durch Basiszellkomplexität bestimmter Granularitätsgrad):* Die Granularität einer Architektur wird bestimmt durch die Komplexität der jeweils zu Grunde liegenden Basiszellen.

Diese alternative Definition fasst den Begriff der Granularität stärker flächenorientiert. Demnach werden Architekturen mit sehr einfachen und flächenmäßig kleinen Basiszellen, wie beispielsweise FPGAs, als *feingranular*, und Architekturen mit komplexeren Basiszellen, die beispielsweise DSP-Einheiten wie Multiplizierer und ALUs beinhalten, als *grobgranular* bezeichnet. In den meisten Fällen sind beide Definitionen äquivalent, es gibt allerdings Ausnahmen. So enthalten beispielsweise CPLDs (Complex Programmable Logic Devices) Basiszellen in Form großer PAL- (Programmable Array Logic) Blöcke, deren Basiszellen aufgrund ihrer Komplexität nach der zweiten Definition als grobgranular bezeichnet werden können. Dennoch ist durch die programmierbare Verbindungsstruktur jede Bitleitung unabhängig von den anderen konfigurierbar, so dass nach der ersten Definition CPLDs üblicherweise als feingranular bezeichnet werden.

In dieser Arbeit werden Basiszell-Architekturen mit unterschiedlicher Komplexität, und beispielsweise verschiedener Anzahl enthaltener Multiplizierer, betrachtet. Die Wortbreite der Datenpfade ist in allen Fällen identisch. Da die Architekturen dennoch als unterschiedlich granular bewertet werden sollen, wird in dieser Arbeit die Granularität nach der zweiten Definition, also der Basiszell-Komplexität, bewertet.

2.2.2 Rekonfigurationsprinzip

Das Prinzip der Rekonfigurierbarkeit ist in Abbildung 2.3 schematisch skizziert. Aus dieser Darstellung geht auch der Unterschied zwischen fein- und grobgranularen Architekturen hervor.

Eine rekonfigurierbare Architektur besteht demzufolge aus einer theoretisch beliebigen Anzahl von Prozessorzellen, die beispielsweise, wie in Abbildung 2.3 dargestellt, in einem zweidimensionalen Array mit jeweils geeigneter Verbindungsstruktur angeordnet sein können. Andere Anordnungen sind ebenfalls denkbar, so sind die Basiszellen bei der Chess-Architektur schachbrettartig miteinander verbunden (siehe z.B. [MSK+99] und Abschnitt 2.4.1), während die RaPiD-Struktur aus einem eindimensionalen Feld besteht (siehe z.B. [ECF96] und Abschnitt 2.4.1). FPGAs sind in den meisten Fällen in zweidimensionaler Feldstruktur aufgebaut. Ausnahmen bilden beispielsweise die Antifuse-basierten FPGAs der Firma Actel, deren Logikelemente stärker zeilenorientiert verbunden sind.

Abbildung 2.3 zeigt weiterhin schematisch den Unterschied zwischen fein- und grobgranularer Logik. Abgebildet ist eine einfache feingranulare Basiszelle in Form einer Lookup-

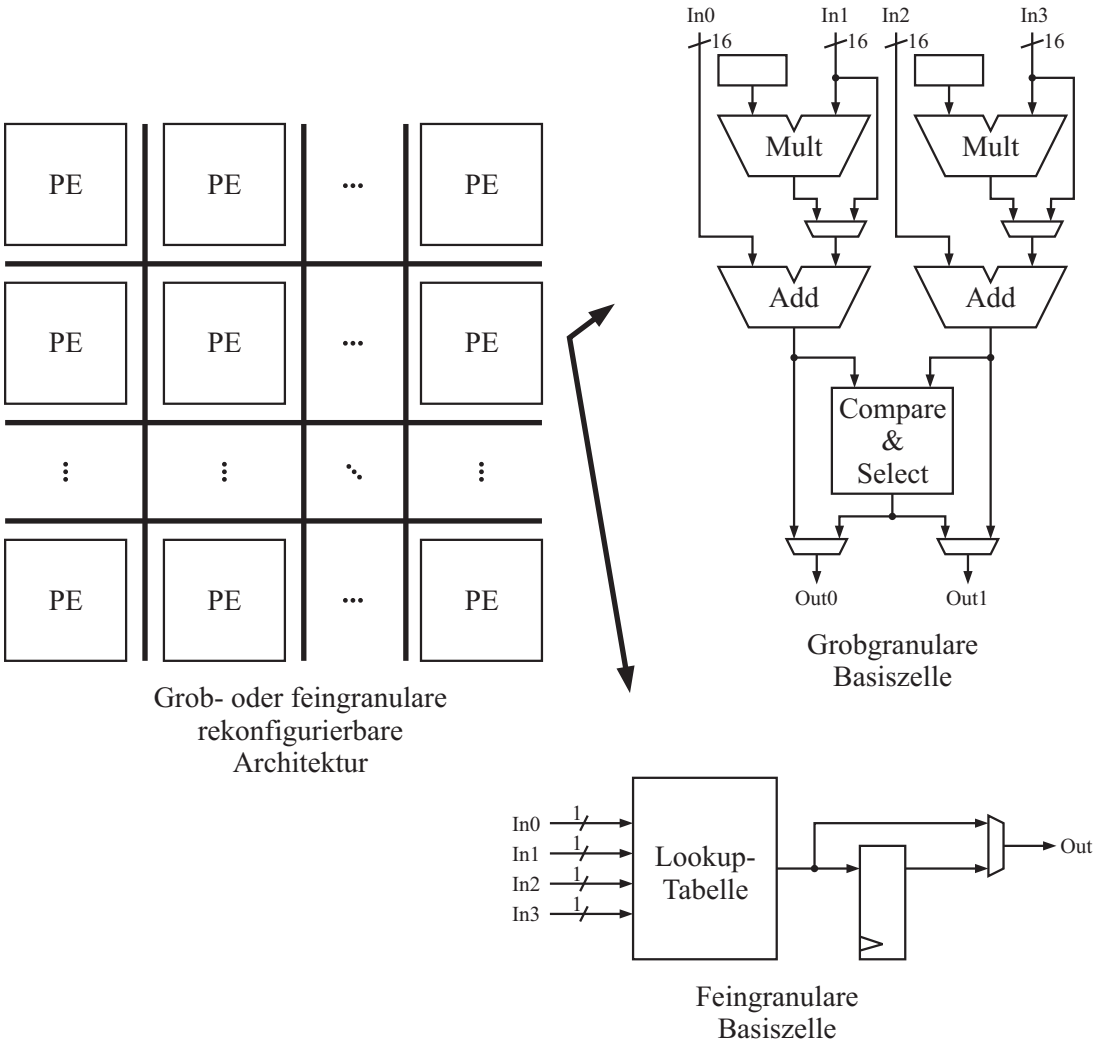


Abbildung 2.3: Grob- und feingranulare rekonfigurierbare Architekturen

Tabelle und einem Register. Dies ist eine Anordnung, deren Prinzip allen aktuell erhältlichen SRAM-basierten FPGAs zu Grunde liegt. Dagegen ist in dem grobgranularen PE ein kompletter Datenpfad bestehend aus Multiplizierern und ALUs implementiert. Die Konfiguration erfolgt durch kundenseitig programmierbare Multiplexer.

2.2.3 Statische, dynamische und partielle Rekonfigurierbarkeit

Für den Fall, dass der komplette Datenflussgraph eines Systems vollständig parallel auf eine rekonfigurierbare Architektur abbildbar ist, und der Algorithmus weiterhin nicht adaptiv ist, und daher keine datenabhängigen Änderungen der Ausführung erforderlich sind, ist keine Umkonfiguration zur Laufzeit erforderlich. Sehr häufig tritt jedoch der Fall auf, dass ein Algorithmus zu viele logische Operationen enthält, um vollständig parallel implementiert werden zu können. In diesem Fall ist die Berechnung auf mehrere Takte aufzuteilen, in denen die Architektur unter Umständen abhängig vom Datenflussgraphen umkonfiguriert werden muss. Auch datenadaptive Algorithmen können ein häufiges Wechseln der Konfiguration erfordern. In diesen Fällen ist ein *statisches* Konfigurationsmodell nicht mehr ausreichend, stattdessen muss *dynamisch* zur Laufzeit umkonfiguriert werden. Strukturen, die eine solche adaptive Umkonfiguration erlauben, heißen dementsprechend *dynamisch rekonfigurierbare Architekturen*, während im anderen Fall von *statisch rekonfigurierbaren Architekturen* gesprochen wird.

Aus diesen Überlegungen gehen verschiedene Rekonfigurationsmodelle hervor, die auch beispielsweise in [CH02] behandelt werden. Abbildung 2.4 zeigt das Prinzip eines *Einzelkontext*-Systems. In einer solchen Struktur kann nur eine Konfigurationsebene, ein Kontext, gespeichert werden. Ein Umkonfigurieren bedingt ein vollständiges Neuladen der Konfigurationsdaten, wodurch Rechenzeit verlorengeht.

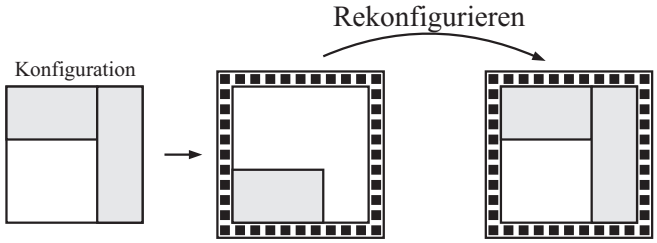


Abbildung 2.4: Rekonfiguration eines Einzelkontext-Systems

Demgegenüber können in einem Multikontextsystem mehrere Konfigurationsebenen bereits zur Konfigurationszeit geladen werden, so dass eine Umkonfiguration lediglich das Ändern einer Zeigeradresse erfordert, und zur Laufzeit erfolgen kann (Abbildung 2.5). Nachteilig ist der höhere Speicherbedarf gegenüber Einzelkontextsystemen, außerdem ist die Konfiguration innerhalb der verschiedenen Kontextebenen weiterhin statisch, und kann nicht datenabhängig adaptiert werden. Dennoch wird ein Multikontextsystem bereits als dynamisch rekonfigurierbare Architektur bezeichnet.

Wenn nur ein Teil der Architektur umkonfiguriert werden kann, während ein anderer Teil der Ressourcen in Betrieb bleibt, wird von partieller Rekonfigurierbarkeit gesprochen (Abbildung 2.6).

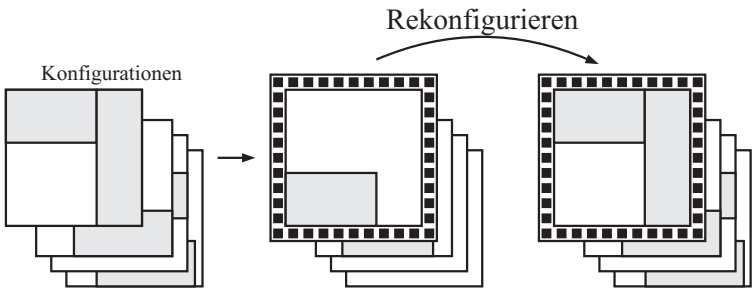


Abbildung 2.5: Rekonfiguration eines Multikontext-Systems

Problemetisch bei dynamisch rekonfigurierbaren Architekturen ist häufig der zusätzlich erforderliche Speicher zur Aufnahme der Konfigurationsdaten. Daher sind Mechanismen entwickelt worden, mit denen die Umkonfigurierung beschleunigt werden kann, ohne alle Konfigurationsdaten ständig auf dem Chip verfügbar halten zu müssen. Beispielhaft sei hier das *Configuration Caching* genannt, bei dem eigens für die Konfigurationsdaten Cache-Speicher zur Verfügung gestellt werden. Beim *Configuration Compression* dagegen werden die Konfigurationsdaten zusätzlich vor der Übertragung komprimiert und auf dem Chip dekomprimiert. Nachteilig ist jedoch auch hier der Bedarf an zusätzlichen Hardwareressourcen.

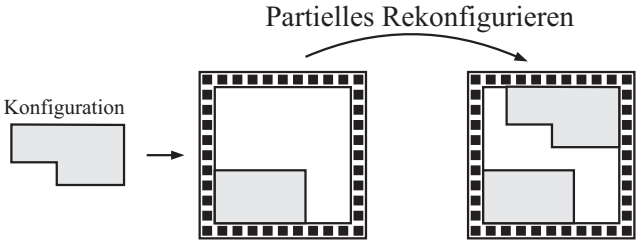


Abbildung 2.6: Prinzip der partiellen Rekonfiguration

2.3 Feingranulare Architekturen

Mit dem Begriff *feingranulare Architektur* wird in dieser Arbeit im Allgemeinen ein rekonfigurierbarer Baustein aus der Familie der FPGAs (Field Programmable Gate Arrays) bezeichnet. Es soll erwähnt werden, dass neben FPGAs auch andere Bausteine existieren, die als feingranulare Architekturen zu bezeichnen sind. Dies sind beispielsweise die bereits lange vor Markeinführung der FPGAs entwickelten programmierbaren Logikbausteine wie

PALs oder PROMs, oder die CPLDs (Complex Programmable Logic Devices). Diese sollen in dieser Arbeit lediglich der Vollständigkeit halber erwähnt werden (Abschnitt 2.3.3).

Nach einer Darstellung des allgemeinen Rekonfigurationsprinzips auf Hardware-Ebene, das allen feldprogrammierbaren Bausteinen zu Grunde liegt (Abschnitt 2.3.1) folgt in Abschnitt 2.3.2 eine kurze Übersicht über die grundsätzlichen Eigenschaften moderner FPGAs der führenden Herstellerfirmen Xilinx und Altera.

2.3.1 Allgemeine FPGA-Struktur

Zum Verständnis und zur Bewertung der in dieser Arbeit präsentierten Ergebnisse bezüglich feingranularer Architekturen ist eine grundlegende Kenntnis des allgemeinen Aufbaus eines FPGA-Logikzellenfeldes erforderlich. In Abbildung 2.7 ist die grundsätzliche Struktur eines solchen Feldes in einer Übersicht dargestellt. Die skizzierte Architektur bezieht sich auf ein FPGA nach der *Island-Style*-Topologie. Dieser Begriff bezeichnet eine Topologie, in der die Logikzellen ähnlich wie Inseln in eine sie umgebende Verbindungsstruktur eingebettet sind, wie dies bei FPGAs der Firmen Xilinx und Altera der Fall ist.

Jede Logikzelle besitzt Ein- und Ausgangspins, die mit den benachbarten Leitungen verbunden werden können. Über horizontale und vertikale Verbindungskanäle, die im Allgemeinen aus einer Vielzahl von Leitungen bestehen, können Logikzellen untereinander kommunizieren. Abbildung 2.7 verdeutlicht das Rekonfigurationsschema der Verbindungsstruktur. Auf der linken Seite der Abbildung ist exemplarisch der Aufbau eines Verbindungsblockes für einen Ausgangspin skizziert. In SRAM-basierten FPGAs werden Pass-Transistoren eingesetzt, um einen Pin mit einer Leitung zu verbinden, oder hochohmig von ihr zu trennen. Für jeden Pass-Transistor ist eine SRAM-Zelle erforderlich, die das Konfigurationsbit speichert. Um Fläche zu sparen, können normalerweise nicht alle Ausgangspins jeder Logikzelle auf alle Verbindungsleitungen zugreifen. Stattdessen sind beim Entwurf eines FPGAs umfangreiche Analysen durchzuführen, um einen guten Kompromiss zwischen Flächenbedarf und Flexibilität zu erzielen. Auf der rechten Seite von Abbildung 2.7 ist der Aufbau eines programmierbaren Eingangspins einer Logikzelle skizziert. Da sinnvollerweise nur eine Leitung mit einem Eingangspin verbunden werden kann, werden bei FPGAs üblicherweise Multiplexer zur Auswahl der Leitung eingesetzt. Die Multiplexer-Adresse muss wiederum in SRAM-Zellen gespeichert werden.

An den Kreuzungspunkten der Verbindungskanäle sind Switch-Blöcke eingefügt. Diese bestehen im Wesentlichen aus einer Vielzahl programmierbarer Verbindungspunkte für die Leitungen. Auch hier sorgen über SRAM-Zellen programmierte Pass-Transistoren entweder für eine Verbindung oder eine hochohmige Trennung zweier Leitungssegmente. In allen modernen FPGAs existieren weiterhin Verbindungsleitungen unterschiedlicher Länge. Über kurze, stark segmentierte Leitungen können benachbarte Logikzellen verbunden werden, während über Leitungen, die über eine längere physikalische Distanz nicht segmentiert sind, weiter voneinander entfernt liegende Logikzellen verbunden werden können.

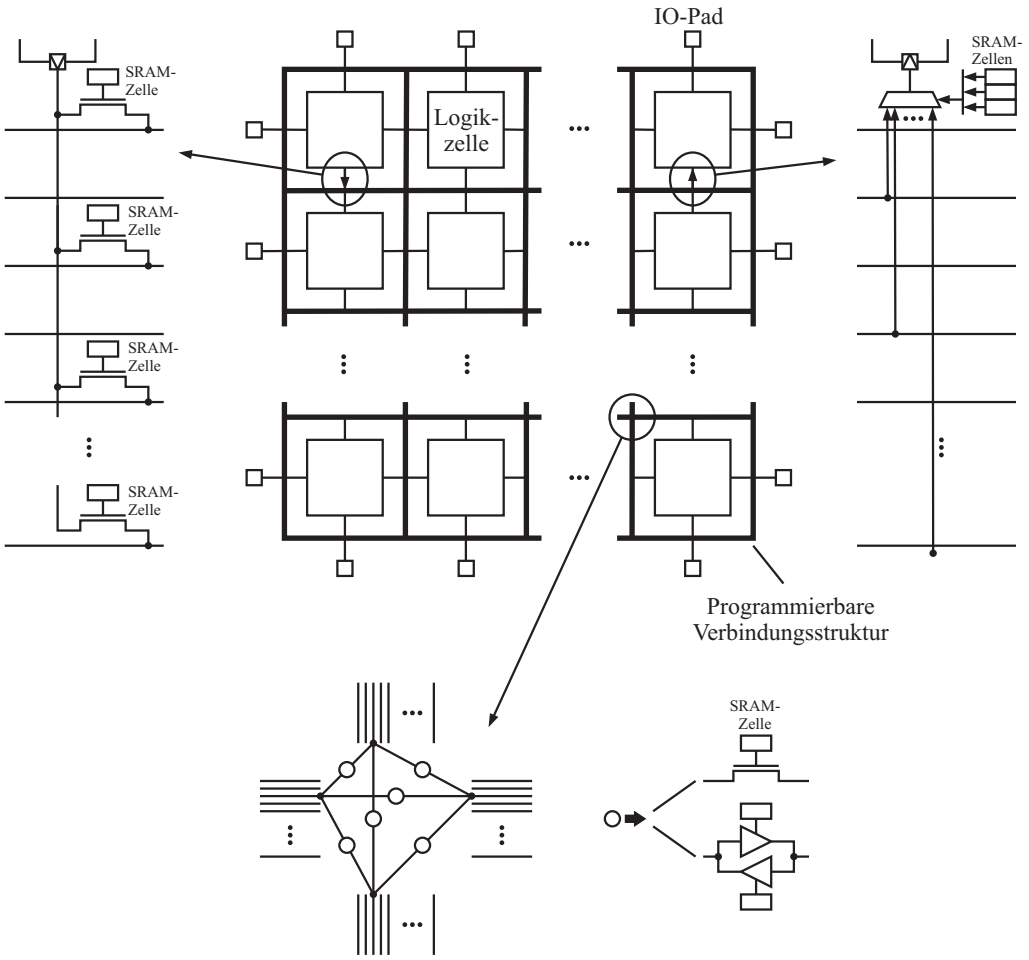


Abbildung 2.7: Allgemeiner Aufbau eines Island-Style FPGA

Die Logikellen sehr einfacher FPGAs bestehen prinzipiell lediglich aus einer programmierbaren Lookup-Tabelle zu Realisierung boolescher Funktionen, sowie einem Register zur Zwischenspeicherung von Daten. In modernen FPGAs werden solche Logikzellen in *Clustern* organisiert, die eine von der globalen Verbindungsstruktur unabhängige interne Interconnect-Architektur besitzen. Diese zusätzliche Hierarchieebene ist eingeführt worden, um der hohen Anzahl lokaler Operationen bei der Implementierung beliebiger Logik Rechnung zu tragen, und Ressourcen der globalen Verbindungsstruktur zu sparen. Abbildung 2.8 zeigt ein Beispiel für ein Cluster mit zehn externen Dateneingängen und vier Logikzellen. Die Ausgänge jeder Zelle können sowohl mit dem globalen Interconnect verbunden, als auch intern auf die Eingänge zurückgekoppelt werden. Dazu sind in jeder Logikzelle für die LUT-Eingänge zusätzliche Multiplexer und SRAM-Zellen erforderlich.

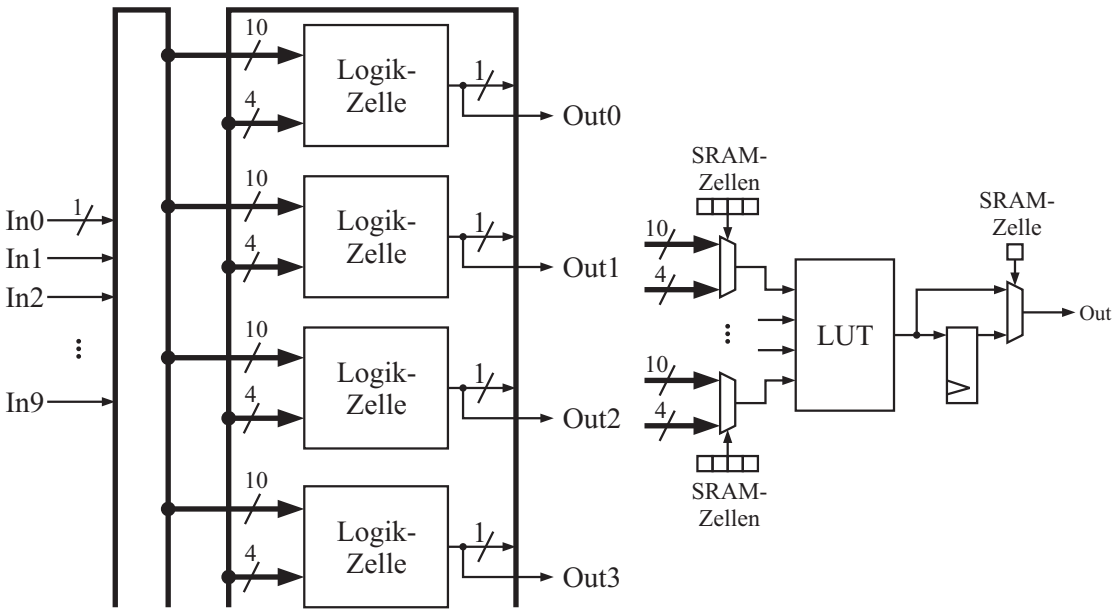


Abbildung 2.8: Beispiel für ein FPGA-Cluster mit zehn Dateneingängen und vier Logikzellen, Logikzelle mit 4-Eingangs-LUT

Die Logikzellen selbst enthalten eine oder mehrere Lookup-Tabellen, die wiederum in den meisten kommerziell erhältlichen FPGAs vier Dateneingänge besitzt, da sich diese Anzahl als guter Kompromiss zwischen Flächeneffizienz und Flexibilität erwiesen hat (siehe z.B. [BRM99]. Desweiteren ist üblicherweise in jeder Zelle zusätzliche Carry-Logik zur effizienteren Implementierung arithmetischer Operationen integriert.

Wie bereits aus dieser kurzen Darstellung des allgemeinen Aufbaus eines FPGAs ersichtlich ist, sind sehr viele Ressourcen allein zur Bereitstellung der Konfigurationsmöglichkeiten zu reservieren. So hat beispielsweise eine Untersuchung der Energieeffizienz eines FPGAs vom Typ Xilinx XC4003A ergeben, dass 65% der Energie durch das Verbindungsnetzwerk, 21% durch das Taktsignal, 9% durch Ein- und Ausgabeblöcke, und lediglich 5% durch die Logikzellen verbraucht wird ([KR98]). Aus diesem Grunde werden in moderne FPGAs neben dem feingranularen Logikzellenfeld zunehmend dedizierte Komponenten integriert, wie beispielsweise Multiplizierer oder eingebettete Prozessoren.

Neben SRAM-basierten FPGAs existiert weiterhin die Klasse der Antifuse-basierten FPGAs, die beispielsweise von der Firma Actel produziert werden. Bei diesen Bausteinen werden Verbindungen zwischen zwei Leitungen durch Anlegen einer Spannung realisiert, mit der das Metall zum Schmelzen gebracht wird. Die hergestellten Verbindungen sind dementsprechend nicht reversibel. Vorteilhaft ist eine wesentlich verbesserte Flächeneffizienz. Desweiteren existieren "semi-permanente" FPGAs, deren Konfigurationsdaten in

nicht-flüchtigen Flash-Speichern gehalten werden. Antifuse-basierte und semi-permanente FPGAs seien hier nur der Vollständigkeit halber erwähnt. Stattdessen bezieht sich der Begriff FPGA in dieser Arbeit immer auf SRAM-basierte Typen.

2.3.2 Ausgewählte Beispiele kommerzieller FPGAs

Um den Stand der Technik kommerziell erhältlicher FPGAs darzustellen, ist ein Blick auf die Produktlinien der beiden führenden Hersteller Xilinx und Altera sinnvoll. Ausgewählte Bausteine sollen hier kurz mit ihren wichtigsten Eigenschaften vorgestellt werden.

Xilinx Virtex-II Pro

Xilinx-FPGAs der Virtex-II-Serie besitzen neben dem rein feingranularen Logikzellenfeld zusätzliche dedizierte Komponenten wie Multiplizierer, eingebettete Prozessoren, Speicherbänke und Gigabit-Transceiver-Einheiten. Eine Betrachtung dieser Bausteine zeigt, dass derartige aktuelle FPGAs heterogene Strukturen sind, die demzufolge bereits den Charakter eines System-on-a-Chip aufweisen.

Virtex-II Pro FPGAs werden in einer $0.13\mu m$-Technologie mit neun Kupfer-Metallebenen gefertigt. Es sind verschiedene Ausbaustufen erhältlich. Die kleinste Stufe (XC2VP2) enthält insgesamt 352 konfigurierbare Logikblöcke (CLBs, Configurable Logic Blocks), die in einem 16×22-Array angeordnet sind, während die größte Stufe (XC2VP100) 11280 CLBs in einem 120×94-Array enthält. Ein CLB ist ein Cluster aus jeweils vier Logikzellen (*Slices*), die wiederum unter anderem zwei 4-Eingangs-LUTs, zwei Register und Carry-Logik beinhalten.

Eine wichtige Eigenschaft aktueller FPGAs ist die Bereitstellung von on-Chip-Speicher. Die größte Ausbaustufe des Virtex-II Pro enthält insgesamt 7992Kb Speicher, verteilt auf 444 18kb Dual-Port SRAM-Bänke. Desweiteren können die Lookuptabellen der Logikzellen als Speicherbänke konfiguriert werden, wodurch maximal weitere 1378kb Speicher verfügbar sind. An jede Block-Speicherbank ist ein dedizierter 18×18-Bit-Multiplizierer angekoppelt, der VC2VP100 enthält somit 444 dedizierte Multiplizierer. Desweiteren sind zwei PowerPC 405 RISC-Prozessor-Cores integriert. Diese Prozessoren können beispielsweise Verwaltungs- und User-Interface-Operationen übernehmen, während mit dem Logikzellenfeld und den dedizierten Multiplizierern rechenintensive Operationen mit hoher Parallelität und Geschwindigkeit ausgeführt werden können.

Interessant ist weiterhin ein Blick auf die in Virtex-II Pro FPGAs zur Verfügung stehenden Routing-Ressourcen, da hierdurch der große durch das Verbindungsnetzwerk entstehende Flächenbedarf deutlich wird. Jeder der zwischen den CLBs angeordneten horizontalen und vertikalen Routingkanäle ist hierarchisch in verschiedene Arten von Verbindungsleitungen aufgeteilt, die benachbarte oder weiter voneinander entfernt liegende CLBs miteinander verbinden können. Jeder Kanal enthält 24 lange bidirektionale Leitun-

gen, die über die gesamte Höhe und Breite des Chips verlaufen, 120 "Hex"-Leitungen, die jeweils jeden dritten oder jeden sechsten entlang des Kanals liegenden CLB miteinander verbinden, 40 "Double"-Leitungen zur Verbindung jedes ersten und zweiten horizontal und vertikal benachbart liegenden CLBs, sowie 16 direkte Verbindungen zu horizontal, vertikal und diagonal benachbart liegenden CLBs.

Für weitere Informationen sei auf [Xil05] verwiesen.

Altera Stratix II

FPGAs vom Typ Stratix II der Firma Altera werden mit einer $90nm$-Technologie mit Kupfermetallisierung hergestellt. Ähnlich wie Xilinx-FPGAs werden auch hier neben dem Logikzellenfeld dedizierte DSP-Komponenten zur Verfügung gestellt. Die größte Ausbaustufe EP2S180 bietet insgesamt 9600 Logik-Cluster (Logic Array Blocks, LABs), die in einer 96×100-Matrix angeordnet sind. Jeder Cluster besteht weiterhin aus acht Logikzellen (Adaptive Logic Module, ALM), die wiederum jeweils zwei 4-Eingangs-LUTs, Register und Carry-Logik beinhalten. Insgesamt stehen 9163kb Speicher in über den Chip verteilten, unterschiedlich großen Blöcken zur Verfügung. Zur Beschleunigung arithmetischer Operationen dienen "DSP-Blöcke", im im Wesentlichen konfigurierbare Multiplizierer bereitstellen. Jeder DSP-Block kann acht 9×9-Bit-Multiplizierer, vier 18×18-Bit-Multiplizierer oder einen 36×36-Bit-Multiplizierer realisieren. Bei 96 über den Chip verteilten DSP-Blöcken stehen somit 384 18×18-Bit-Multiplizierer zur Verfügung. Dedizierte Prozessoren sind in Stratix-FPGAs nicht integriert.

Für weitere Informationen sei auf [Alt05] verwiesen.

2.3.3 Weitere Beispiele für feingranulare Strukturen

Neben FPGAs existieren weitere feingranulare rekonfigurierbare Architekturen, die hier jedoch nicht näher beschrieben, sondern lediglich der Vollständigkeit halber erwähnt werden sollen.

Zur Familie der programmierbaren Logikbausteine (Programmable Logic Devices, PLDs) zählen beispielsweise Strukturen, deren Funktionalität über programmierbare UND- und ODER-Felder konfigurierbar ist, wie z.B. PROMs (Programmable Read-Only Memory) oder PALs (Programmable Array Logic). In den letzten Jahren sind CPLDs (Complex Programmable Logic Devices) populär geworden. Diese bestehen im Kern aus mehreren größeren PAL-Blöcken, mit denen logische Funktionen realisiert werden können, und einer konfigurierbaren Verbindungsstruktur. Im Unterschied zu FPGAs werden bei CPLDs weniger, dafür jedoch mächtigere Logikzellen eingesetzt.

2.4 Grobgranulare Architekturen

Das grundlegende Prinzip grobgranularer rekonfigurierbarer Architekturen ist bereits in Abschnitt 2.2 anhand einer schematischen Darstellung erläutert worden. Definitionsgemäß bestehen solche Bausteine aus mehr oder weniger komplexen Prozessorzellen, in denen konzentrierte Datenpfadelemente integriert sein können. Allerdings ist es für grobgranulare Architekturen nicht möglich, allgemeine, in allen Bausteinen vorhandene Strukturkomponenten anzugeben, wie dies für FPGAs in Ansätzen möglich ist (siehe z.B. Abbildung 2.7). Stattdessen unterscheiden sich vorhandene und veröffentlichte grobgranulare Architekturen teilweise fundamental voneinander. Es werden daher in den folgenden Abschnitten ausgewählte Beispiele bekannter grobgranularer Architekturen präsentiert. Die Auswahl wurde mit dem Ziel getroffen, eine große Bandbreite der Variationsmöglichkeiten zu zeigen. So arbeitet das im Folgenden dargestellte CHESS-Array auf 4-Bit-Ebene mit vergleichsweise einfachen Prozessorzellen, während die MorphoSys-Architektur (Seite 28) weniger, dafür jedoch mächtigere Basiszellen beinhaltet, und mit 16-Bit-breiten Datenpfaden arbeitet. Der RAW-Prozessor (Seite 31) besteht aus 16 hochkomplexen identischen "Tiles", von denen jedes einen eigenen Prozessor inklusive Daten- und Instruktionscache, und Steuereinheiten beinhaltet.

Über die hier näher dargestellten Strukturen hinaus sind viele weitere rekonfigurierbare Architekturen entwickelt und veröffentlicht worden. Diese können hier nicht alle vorgestellt werden, stattdessen wird in Abschnitt 2.6 eine Übersicht über die wichtigsten Eigenschaften der bekanntesten und am häufigsten referenzierten Architekturen präsentiert.

2.4.1 Beispiele forschungsnaher Projekte zu grobgranularen Architekturen

CHESS

Das 1999 veröffentlichte CHESS-Array ist aus einem akademisch-industriellen Forschungsprojekt mit Hewlett-Packard, der Ecole Normale Superieure in Frankreich und der Brigham Young University hervorgegangen (siehe [MSK+99]). Es handelt sich um ein rekonfigurierbares Feld, das zur flexiblen Erweiterung eines ASIC oder eines Prozessor-Datenpfads eingesetzt werden kann.

Kern des CHESS-Arrays sind 4-Bit breite ALUs mit einem 16 Operationen umfassenden Instruktionssatz. Die ALUs sind über konfigurierbare Switchboxen miteinander verbunden, und sind mit diesen in einer schachbrettartigen Topologie angeordnet. Abbildung 2.9 zeigt einen Ausschnitt aus dem Feld und den logischen Aufbau einer ALU, die in Bitslice-Technik realisiert ist. Demzufolge besitzt ein ALU-PE zwei 4-Bit Dateneingänge, einen Carry-Eingang, einen 4-Bit Datenausgang und einen Carry-Ausgang. Jedes PE kann Additionen, Subtraktionen sowie logische Operationen ausführen. Aufgrund der lediglich

4 Bit breiten Datenwortbreite und des einfachen Aufbaus der ALUs müssen komplexere Operationen wie Additionen und Subtraktionen höherer Wortbreite oder Multiplikationen aus mehreren ALU-PEs zusammengesetzt werden. Speicher wird einerseits durch in das Array eingefügte dedizierte RAM-Blöcke bereitgestellt, andererseits können die Konfigurationsspeicher der Switchboxen auch als 128-Bit-Speicherblock verwendet werden.

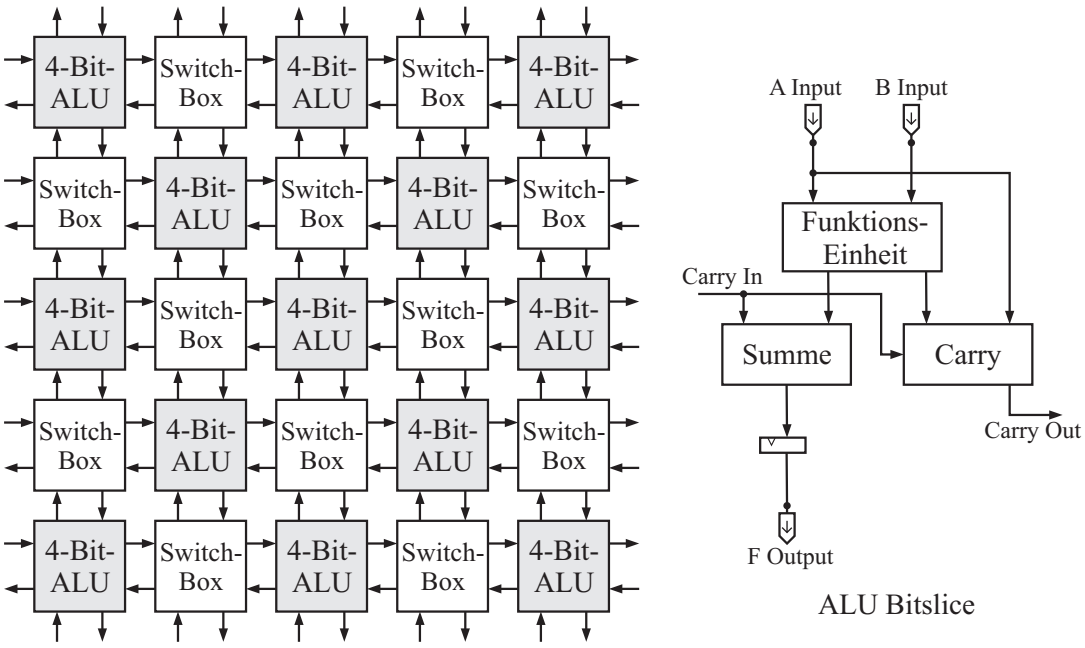


Abbildung 2.9: Topologie des CHESS-Arrays, ALU Bitslice bei CHESS

Das Routing-Netzwerk stellt eine Nearest-Neighbor-Konnektivität her, da jede ALU mit jeweils acht direkten Nachbarn kommunizieren kann. Desweiteren bieten segmentierbare Busleitungen die Möglichkeit, weiter voneinander entfernt liegende ALUs miteinander zu verbinden.

Das CHESS-Array soll laut [MSK+99] vor allem von einer besseren Flächeneffizienz als FPGAs durch die 4-Bit breiten Datenpfade profitieren. Es wird angegeben, dass eine ALU und eine daran angegliederte Switchbox mit 100 Konfigurationsbits programmiert werden kann, was eine schnelle Umkonfigurierung des Feldes ermöglicht. Dynamische Rekonfiguration zur Laufzeit wird dagegen nicht unterstützt. Für ein CHESS-Array mit 512 ALUs und 32 RAM-Blöcken mit jeweils 2048 Bit wird ein Flächenbedarf von $30mm^2$ in einem $0.35\mu m$-Prozess angegeben.

MATRIX

Ein Beispiel für eine aus 8-Bit breiten Datenpfaden aufgebaute Struktur ist die bereits 1996 veröffentlichte MATRIX-Architektur (siehe z.B. [MD96]). MATRIX ist aus einem zweidimensionalen Feld gleichartiger 8-Bit "BFUs" (Basic Functional Units) aufgebaut. Jede BFU beinhaltet eine ALU, welche neben logischen Operationen auch Additionen, Subtraktionen und Multiplikationen durchführen kann. Desweiteren enthält jede BFU einen Controller, der lokale Kontrollsignale aus den ALU-Ausgangswerten berechnen kann, wodurch dynamische Rekonfigurierbarkeit eingeführt wird. Eine weitere Besonderheit ist die Integration eines flexibel einsetzbaren Speichers. Jede BFU enthält einen 256 Byte großen Speicherblock, der sowohl Daten, als auch Instruktionen aufnehmen kann. Auf diese Weise können mehrere Kontextebenen initialisiert werden, um ein Multikontextsystem zu erhalten, das taktweise umgeschaltet werden kann (siehe Abschnitt 2.2.3). Jede BFU kann demzufolge unabhängig von allen anderen konfiguriert werden (MIMD-Charakteristik).

Die Verbindungsstruktur bietet Nearest-Neighbor-Konnektivität benachbarter BFUs, Leitungen, die jede vierte BFU miteinander verbinden, sowie lange globale Verbindungsleitungen.

MorphoSys

An den Universitäten UC Irvine und Rio de Janeiro wurde der MorphoSys-Chip entwickelt und 1999 erstmals veröffentlicht (siehe [GSL+99]). MorphoSys ist ein System-on-a-Chip, das neben einem rekonfigurierbaren PE-Feld unter anderem einen RISC-Prozessor, Instruktions- und Daten-Cache und einen DMA-Controller enthält. Dadurch kann der Baustein prinzipiell für beliebige, universelle Anwendungen eingesetzt werden, ist jedoch in erster Linie zur Beschleunigung und effizienten Ausführung von Algorithmen geeignet, die einen hohen Grad an Parallelität und Regularität aufweisen.

Das rekonfigurierbare PE-Feld besteht aus 64 Prozessorelementen ("Reconfigurable Cells", RCs), die in einem 8×8-Array angeordnet sind (siehe Abbildung 2.10). Ein Prozessorelement beinhaltet unter anderem eine ALU, einen 16×12-Bit-Multiplizierer, sowie Register zur Zwischenspeicherung von Daten. Die ALU arbeitet mit 16-Bit-Werten, lediglich der Addierer ist auf 28-Bit-breite Worte ausgelegt um Multiply-Add-Operationen ohne Genauigkeitsverlust durchführen zu können. Insgesamt kann die ALU 25 verschiedene Funktionen ausführen.

Das Array ist untergliedert in vier Quadranten zu je 4×4 Prozessorelementen. Die Verbindungsstruktur weist drei Hierarchieebenen auf: Alle Prozessorelemente sind jeweils mit ihrem oberen, unteren, linken und rechten Nachbarn verbunden. Innerhalb eines Quadranten ist zudem die Kommunikation eines PEs mit den drei übrigen PEs einer Zeile und mit den drei übrigen PEs einer Spalte möglich. Die obere, globale Hierarchieebene wird durch horizontal und vertikal angeordnete Datenbusse gebildet. Über diese Busse können

alle vier Prozessorelemente in einer Zeile, bzw. Spalte innerhalb eines Quadranten den Ausgangswert eines der vier PEs des benachbarten Quadranten in derselben Zeile, bzw. Spalte, lesen. Entweder Zeilen oder Spalten des rekonfigurierbare PE-Feldes von MorphoSys arbeiten jeweils nach dem SIMD-Prinzip. Demzufolge sind alle PEs einer Zeile oder Spalte gleich konfiguriert. Es ist auch die PE-interne Rückführung von Daten auf Eingänge der selben Basiszelle vorgesehen, so dass jede RC als eigenständiger Prozessor fungieren kann.

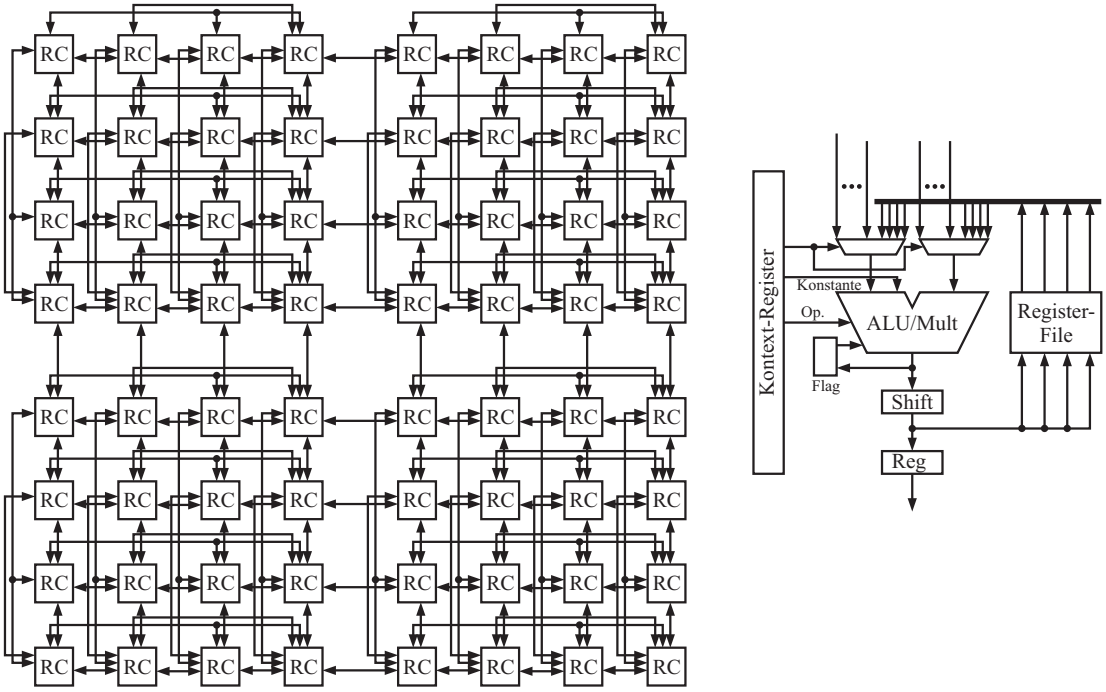


Abbildung 2.10: Rekonfigurierbares Array des MorphoSys-SoC, rekonfigurierbare Zelle (RC)

Im laufenden Betrieb auftretende Zwischenwerte, die von verschiedenen RCs weiterverarbeitet werden sollen, werden im "Frame-Buffer" gespeichert. Der Frame-Buffer ist ein linksseitig des Arrays angeordneter Speicher, der einen Schreib- und einen Lesebereich zur Verfügung stellt, die jeweils umgeschaltet werden können. Jede Zeile aus acht RCs ist über einen 16 Bit breiten Datenbus mit dem Frame-Buffer verbunden.

Die Software-Umgebung für den MorphoSys-SoC besteht aus einem Compiler für den eingebetteten RISC-Prozessor, und einer Entwicklungsumgebung für das RC-Feld ([Nag01]). Die Partitionierung des Codes in Teile, die der RISC-Prozessor ausführen soll, und Teile, die vom RC-Array übernommen werden sollen, muss manuell durch Eingabe

einer Präfix-Anweisung im entsprechenden Software-Code durchgeführt werden. Daraus werden Anweisungen zur Aktivierung des RC-Feldes an der entsprechenden Stelle generiert. Das RC-Feld selbst kann unter Zuhilfenahme einer grafischen Benutzeroberfläche konfiguriert werden. Aus einem fertigen Programm kann VHDL-Code generiert werden, der mit einem entsprechenden Simulator zur Verifikation des Systems herangezogen werden kann.

RaPiD

Ein Beispiel für eine grobgranulare rekonfigurierbare Architektur mit eindimensionaler Feldstruktur ist der 1996 von der Universität Washington veröffentlichte RaPiD-Chip [ECF96]). Die Architektur dient ebenfalls zur Beschleunigung rechenintensiver und regulärer Algorithmen, die zudem stark von Daten-Pipelining profitieren.

Der Prototypenchip RaPiD-1 besteht aus 16 Basiszellen, die jeweils einen 16-Bit-Multiplizierer, drei 16-Bit ALUs, sechs Register, sowie drei RAM-Blöcke für 32 Datenworte beinhalten (siehe Abbildung 2.11). Das Feld wiederum wird durch lineare Verkettung dieser Basiszellen gebildet. Die Register können Konstanten oder Zwischenwerte speichern, oder als Pipeline-Register dienen.

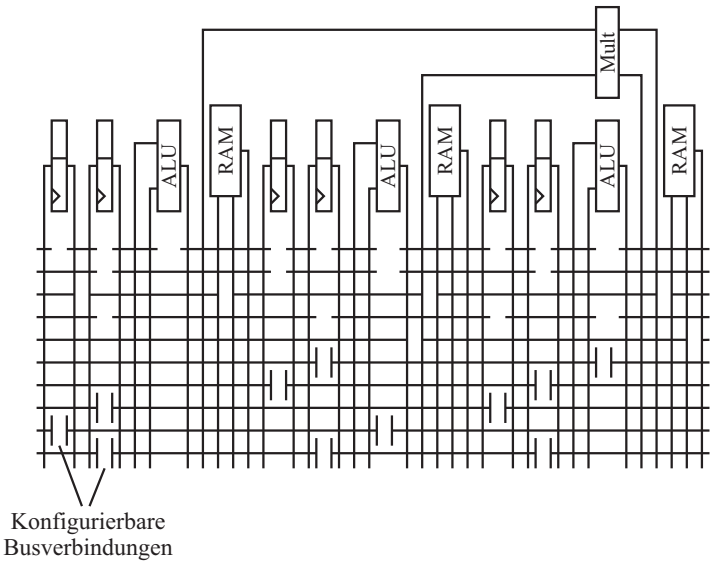


Abbildung 2.11: Beispiel für eine RaPiD-Basiszelle. Der in [ECF96] vorgestellte RaPiD-1-Chip ist ein lineares Feld aus 16 Basiszellen

Die Verbindungsstruktur des RaPiD-1-Chips besteht aus segmentierten 16-Bit-Bussen, die über die gesamte Breite des Feldes hinweg verlaufen. Die Länge der Busse variiert in

jedem Kanal. Einige der segmentierten Busse können über konfigurierbare Busverbindungen miteinander verbunden werden, die zusätzlich die Möglichkeit bieten, die Daten zwischenzuspeichern. Jede Funktionseinheit kann auf alle Busleitungen lesend und schreibend zugreifen. Die Auswahl der zu lesenden Leitung erfolgt über konfigurierbare Multiplexer. Auf diese Weise lassen sich mit RaPiD konfigurierbar applikationsspezifische Datenpfade realisieren. Das RaPiD-Array ist zudem an ein programmierbares Speicherinterface angebunden.

RAW

Das RAW- (*Reconfigurable Architectures Workstation*) Projekt ist ein Beispiel für eine Architektur mit sehr komplexen und mächtigen Prozessorelementen. Die Architektur wurde vom Massachusetts Institute of Technology (MIT) im Jahre 1997 erstmals veröffentlicht ([Wai97]), und seitdem weiterentwickelt (siehe z.B. [Tay02], [Tay04]). Der Prototyp des RAW-Prozessors besteht aus 16 identischen Prozessorzellen ("Tiles"), die in einem 4×4-Array angeordnet sind (Abbildung 2.12a). Jedes Tile wiederum enthält einen Prozessor mit modifizierter MIPS-R2000-Architektur, eine Floating-Point-Einheit, 32kB Daten-Cache, 96kB Instruktions-Cache, sowie einen statisch und zwei dynamisch konfigurierbare Daten-Router. Vor Realisierung des Prototypen-Chips war auch die Implementierung eines kleines Blocks feingranularer rekonfigurierbarer Logik vorgesehen ([Tay04]), was jedoch nicht umgesetzt wurde.

Die Tiles kommunizieren miteinander über Nearest-Neighbor-Verbindungen, von denen jede aus vier bidirektionalen 32-Bit-Leitungen besteht (siehe Abbildung 2.12b). Jedes RAW-Tile besitzt einen konfigurierbaren Switch, der die Recheneinheit des Tiles mit dem Netzwerk verbindet, und über den Daten durch das Array geroutet werden können. Abbildung 2.12c zeigt die Topologie eines RAW-Tiles.

Der RAW-Chip ist von Anfang an unter der Prämisse eines weitgehend statischen Schedulings entwickelt worden. So enthält der Tile-interne Prozessor keine aus modernen superskalaren Prozessoren bekannten dedizierten Hardware-Ressourcen zur Steuerung von Register-Renaming, spekulativen Ausführungen von Instruktionen oder Cache-Vorgängen. Diese dynamischen Prozeduren werden stattdessen vom Compiler übernommen, und bereits vor dem Laden des Programms festgelegt. Der Compiler ist auch weitgehend verantwortlich für die Konfiguration des Verbindungsnetzwerkes, dessen Verhalten ebenfalls durch statisches Scheduling festgelegt wird. Jedes Tile enthält einen Switch-Prozessor mit entsprechendem Speicher, durch welchen die statische Konfiguration der Datentransfers in jedem Takt festgelegt wird. Lediglich in den Fällen, in denen kein statisches Scheduling möglich ist, können durch einen dynamischen Router in vom statischen Netzwerk unbenutzten Taktzyklen Datentransfers durchgeführt werden ([Nag01], [Tay04]).

RAW ist vor allem zur Beschleunigung rechenintensiver Aufgaben durch Bereitstellung massiv paralleler Rechenressourcen ausgelegt. Die Fläche eines 4×4-RAW-Prozessors wird

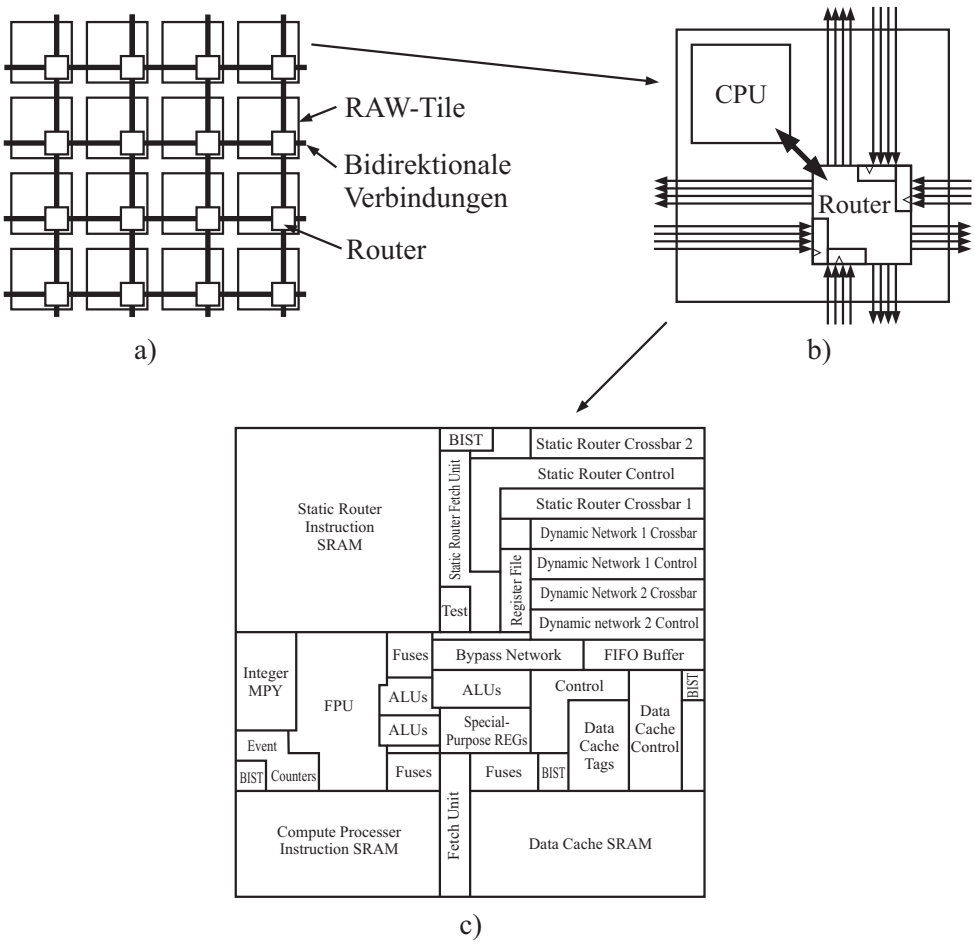


Abbildung 2.12: Reconfigurable Architectures Workstation (RAW): a) 4×4-RAW Prozessor, b) Struktur eines RAW-Tiles mit Prozessor und Routing-Ressourcen, c) Floorplan eines RAW-Tiles

in [Tay02] mit $16{\times}16mm^2$ angegeben, bezogen auf eine $0.15\mu m$-Technologie. Die maximale Taktfrequenz dieses Prototypen beträgt 225 MHz, und die Leistungsaufnahme soll für typische Anwendungen bei 25W liegen.

Rekonfigurierbarer ASIP-Accelerator

Im Vorfeld zu dieser Arbeit ist in einem gemeinsamen Projekt mit der Firma Nokia eine rekonfigurierbare Accelerator-Architektur entwickelt worden, die für verschiedene rechenintensive Anwendungen einen flexiblen Rahmen bilden kann. Kern dieser Architektur sind jeweils Prozessorzellen mit anwendungsspezifischem Instruktionssatz, die für verschiedene

Applikationen ausgetauscht, und in ein und denselben Rahmen aus Speicher- und Kontrolleinheiten eingesetzt werden können.

Abbildung 2.13 zeigt ein Blockdiagramm dieser Architektur (siehe z.B. [OBU+02], [LFS+02]). Im Rahmen des Projektes sind mehrere Prozessorzellen für unterschiedliche Klassen von Algorithmen entwickelt worden. Dies sind ein PE mit CORDIC-Prozessor zur Berechnung matrixbasierter Algorithmen, ein PE für Galois-Feld-Arithmetik, ein PE zur Decodierung faltungscodierter Sequenzen mit dem Viterbi-Code, sowie eine Zelle zur Berechnung Multiply-Accumulate-basierter Algorithmen. Das PE für die Galois-Feld-Arithmetik ist in Bezug auf die Decodierung verschiedener mit Reed-Solomon-Codes blockcodierter Sequenzen getestet worden, grundsätzlich ist jedoch auch der allgemeine Einsatz dieses PEs für Algorithmen, die auf Endlichen Feldern beruhen, denkbar [RBWP02]. Das Viterbi-PE ist dagegen ausschließlich zur Decodierung faltungscodierter Sequenzen mit dem Viterbi-Code optimiert worden [LOS04].

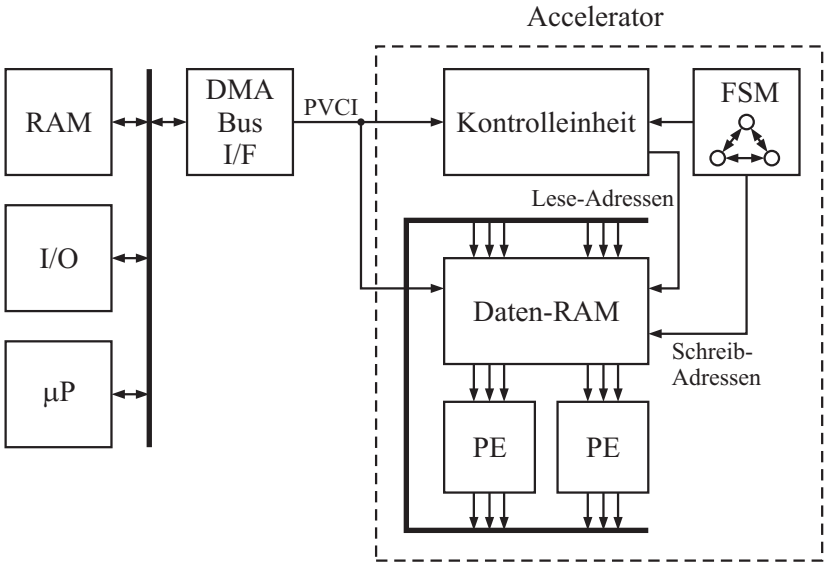


Abbildung 2.13: Rekonfigurierbarer ASIP-Accelerator

Alle zur Berechnung eines Algorithmus erforderlichen Operanden werden im Daten-RAM gespeichert, und von den Prozessorzellen direkt aus diesem gelesen, und wieder in den Speicher zurückgeschrieben (Speicher-Speicher-Maschine). Alle Schreib-/Lesezugriffe sind bereits vor dem Start der Programmausführung in Form eines statischen Schedulings festgelegt, so dass keine Hardwareressourcen zur Verwaltung dynamischer Instruktionsausführungen erforderlich sind. Im Daten-RAM werden sowohl alle Ein- und Ausgangswerte, als auch bei der Berechnung auftretende Zwischenwerte gespeichert. Zur Vermeidung von Zugriffskonflikten besteht der Datenspeicher aus zwei globalen Speicherbänken, die

im Rahmen des Projektes als *Active*- und *Shadow*-Speicherbank bezeichnet wurden. Die Prozessorelemente lesen aus der Active-Bank, und schreiben gleichzeitig Ausgangs- und Zwischenwerte in die Shadow-Bank. Der Active- und Shadow-Status der Bänke wird vertauscht, sobald die Zwischenwerte zur weiteren Berechnung wieder benötigt werden. Zwei weitere Speicherbänke dienen weiterhin zur Aufnahme von Ein- und Ausgangswerten.

Der Mikrocode zur Ablaufsteuerung wird im Konfigurationsspeicher abgelegt, der die Lese-Adressen des Datenspeichers und die Instruktionen für die Prozessorzellen bei jeder PE-Aktivierung enthält. Der in Abbildung 2.13 mit FSM gekennzeichnete Block dient wiederum als globale Kontrolleinheit, und stellt unter anderem die Lese-Adressen des Konfigurationsspeichers und die Schreib-Adressen des Datenspeichers zur Verfügung.

Abbildung 2.14 zeigt beispielhaft das RMAC-PE ("Reconfigurable Multiply-Accumulate-based Processing Element) für die Klasse der Multiply-Accumulate-basierten Algorithmen. Dem PE liegt das Prinzip zugrunde, dass Algorithmen wie die FFT, DCT, FIR-Filterungen oder Matrix-Vektor-Multiplikationen zwar dieselbe arithmetische Grundoperation teilen, die implementierten Datenpfade sich jedoch für eine effiziente Realisierung unterscheiden. Demzufolge lässt sich die Struktur des Datenpfads über Multiplexer anwendungsspezifisch umkonfigurieren.

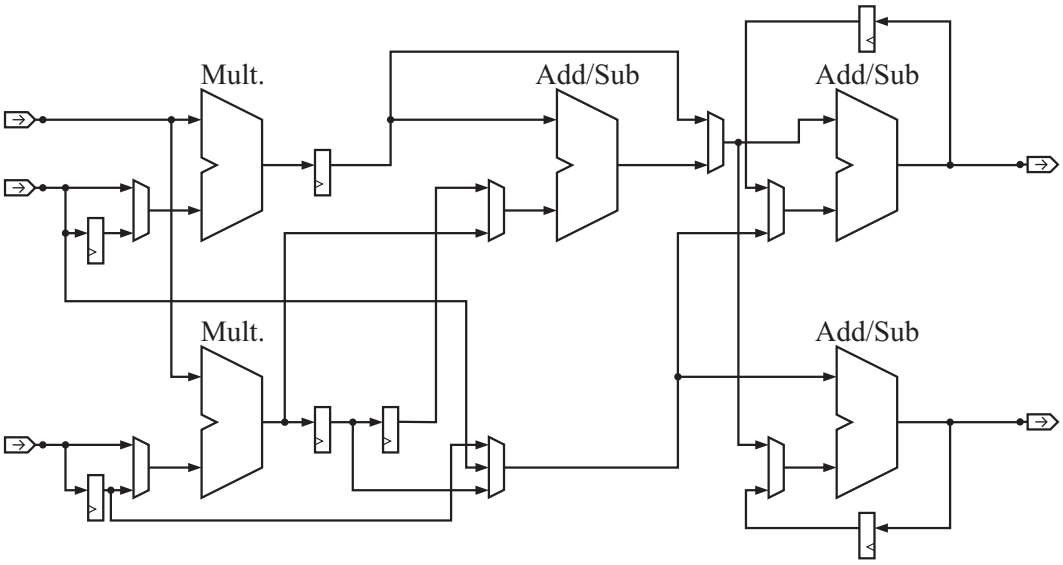


Abbildung 2.14: Rekonfigurierbares Multiply-Accumulate-basiertes Prozessorelement (RMAC-PE, siehe [LFS+02])

Weitere Informationen zu diesem Projekt finden sich beispielsweise in [OBU+02], [LFS+02] oder [LOS04].

2.4.2 Beispiele für kommerzielle grobgranulare Architekturen

Neben dem rein akademischen Interesse an grobgranularen Architekturen bieten seit einigen Jahren auch einige Firmen Bausteine an, die im Kern aus grobgranularen rekonfigurierbaren Prozessorfeldern bestehen. Zu den bekanntesten Firmen gehören beispielsweise PACT Informationstechnologie GmbH, Picochip und Silicon Hive, deren Produkte alle auf mehr oder weniger universalen grobgranularen Array-Strukturen fußen. Diese werden im folgenden kurz dargestellt.

PACT

Die 1996 in Deutschland gegründete Firma PACT Informationstechnologie GmbH bietet mit dem *XPP-Prozessor* (eXtreme Processing Platform) ein massiv paralleles Array aus grobgranularen Rechenelementen, den Processing Array Elements (PAEs) an (siehe z.B. [PAC05]). Diese können jeweils anwendungsspezifisch entworfen werden, und werden von PACT derzeit in zwei Ausführungsformen angeboten, den ALU-PAEs und den RAM-PAEs, die auch I/O-Einheiten enthalten können. Die Prozessorelemente kommunizieren untereinander über ein paketorientiertes Verbindungsnetzwerk in Form eines einfachen Network-on-a-Chip. Abbildung 2.15a zeigt beispielhaft den Aufbau eines Clusters aus 3×5 ALU-PAEs und sechs RAM-PAEs. Der Prototypenchip XPP64 besteht aus einem Cluster mit insgesamt 64 ALU-PAEs. Ein kompletter XPP-Prozessor kann weiterhin aus mehreren Clustern hierarchisch aufgebaut sein.

Die Struktur eines ALU-PAEs ist in Abbildung 2.15b dargestellt. Die Zelle enthält die drei Recheneinheiten "Forward Register Object" (FREG), "ALU Object" und das "Backward Register Object" (BREG). Eingangsseitig enthält jede Komponente ein Register, sowie einen FIFO-Speicher zum eventuellen Synchronisieren von Pipelinetakten. Jedes ALU-Object stellt neben logischen Operationen auch Addierer und Subtrahierer, sowie Komparatoren und Multiplizierer zur Verfügung. Die Forward- und Backward-Register-Objects übernehmen Routing-Aufgaben, und bieten zusätzlich neben Addierern und Subtrahierern eine Lookup-Tabelle für Boolesche Operationen. Alle ALU-PAEs arbeiten mit einer Datenwortbreite von 16, 24 oder 32 Bit, abhängig von der Konfiguration bei der Implementierung.

Der XPP-Prozessor ist dynamisch rekonfigurierbar. Bei der Implementierung eines Algorithmus werden die Knoten des Datenflussgraphen mit größtmöglicher Parallelität auf arithmetische Operationen abgebildet. Wenn der Datenflussgraph aufgrund seiner Komplexität nicht vollständig parallel gemappt werden kann, wird das Array zwischen verschiedenen Konfigurationen umgeschaltet. XPP stellt hierfür schnelle Rekonfigurationsmechanismen wie Configuration Caching zur Verfügung (siehe Abschnitt 2.2.3). Eine Kontrollstruktur sorgt mit entsprechenden Signalen zudem für die chipinterne Synchronisation der paketvermittelten Daten. Jedes PAE führt die Berechnung durch, sobald es alle Daten-

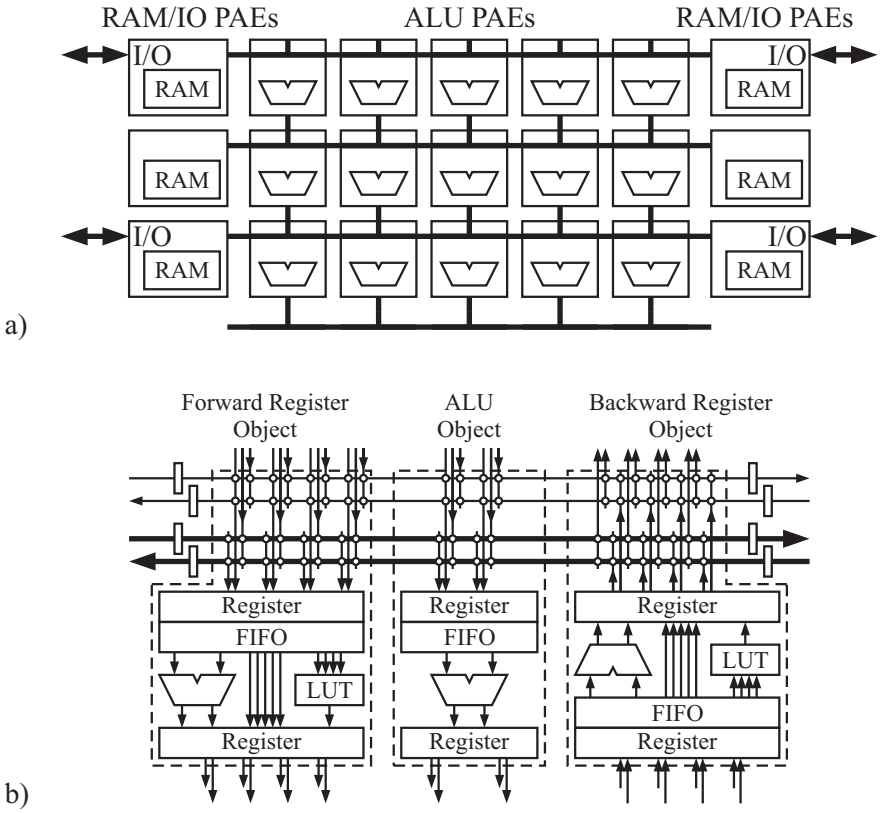


Abbildung 2.15: PACT eXtreme Processing Platform (XPP), a) Beispielkonfiguration eines XPP Processing Array Clusters (PAC), b) Struktur eines ALU-PAEs

wörter zur Verfügung hat, und schickt die Daten weiter, sobald die Operation beendet ist.

PACT stellt für die XPP-Plattform eine Softwareumgebung inklusive Compiler und grafischer Benutzeroberfläche zur Verfügung. Algorithmen lassen sich mit der proprietären, HDL-ähnlichen Programmiersprache NML ("Native Mapping Language") auf das Feld abbilden.

Für weitere Informationen sei auf [PAC05] verwiesen.

picoChip

Einen ähnlichen Ansatz wie PACT verfolgt die 2002 im englischen Bath gegründete Firma picoChip Designs Ltd. Das von dieser Firma entworfene *picoArray* ist ebenfalls eine massiv parallele Anordnung von Prozessoren (siehe z.B. [DPT03]). Ein picoArray ist ein

eigenständiger Chip, der insbesondere für rechenintensive Algorithmen in Basisstationen von Mobilfunksystemen der dritten Generation entworfen wurde, wie z.B. UMTS oder CDMA2000. Der Chip enthält insgesamt 430 16-Bit-Prozessoren, die jeweils für unterschiedliche Aufgaben optimiert sind. Abbildung 2.16 zeigt einen Ausschnitt aus dem picoArray und den allgemeinen Aufbau eines der eingesetzten Prozessoren. Insgesamt existieren vier verschiedene Prozessortypen, die zwar einen gemeinsamen Instruktionssatz, jedoch unterschiedliche Speichergrößen beinhalten, und zusätzliche dedizierte Spezialinstruktionen besitzen können. Insgesamt existieren im Chip 240 Prozessoren mit jeweils 768 Byte verteiltem Speicher und einem speziell für 3G-Anwendungen optimierten Instruktionssatz beispielsweise zur Berechnung von Spreizcodes, 120 Multiply-Accumulate-basierte PEs mit ebenfalls 768 Byte großem Speicher, 68 weitere MAC-Zellen mit 8kByte Speicher, sowie zwei speziell für Kontrollflussaufgaben vorgesehene Prozessoren mit Multiplizierer und 32kB großem Daten- und Instruktionsspeicher.

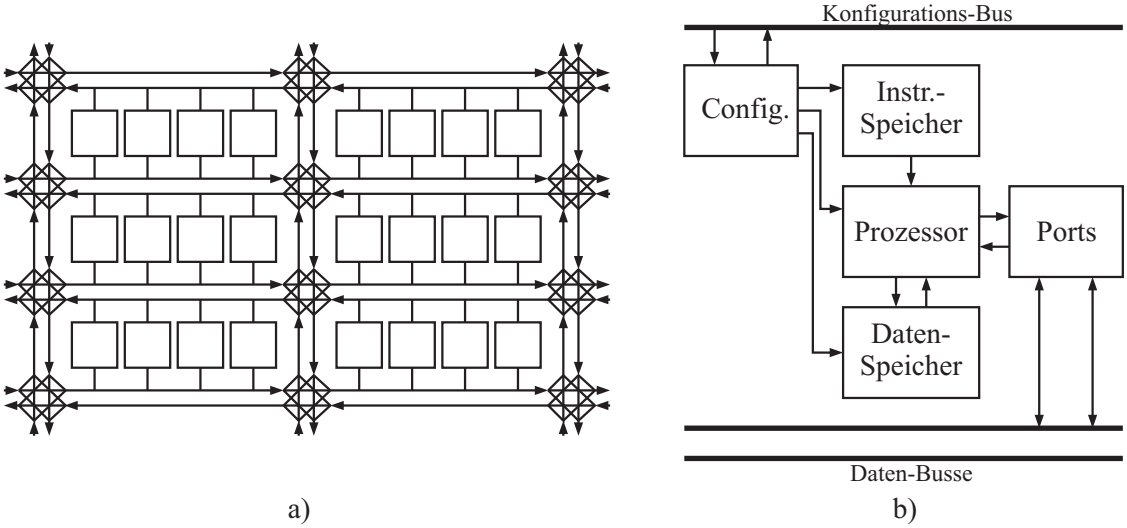


Abbildung 2.16: Picochip's picoArray, a) Ausschnitt aus dem picoArray mit Verbindungsstruktur, b) Struktur eines Prozessorelementes

Alle Prozessoren schreiben und lesen von gemeinsamen Datenbussen, deren Zugriff über einen statisch festelegten Zeitmultiplex geregelt ist. So ist keine Laufzeit-Arbitrierung erforderlich, und zudem steht jedem Prozessor eine garantierte Bandbreite zur Datenübertragung zur Verfügung. Im Unterschied zur paketvermittelten Kommunikationsstruktur beispielsweise beim XPP-Prozessor von PACT werden bei picoChip die TDM-Zeitslots bereits beim Compile-Vorgang festgelegt. Der Benutzer hat die Möglichkeit, die für eine bestimmte Verbindung erforderliche Bandbreite zu spezifizieren. Entsprechend werden vom

Compiler für schnelle Verbindungen mehr Timeslots reserviert als für Verbindungen mit geringeren Anforderungen.

Die Implementierung von Algorithmen erfolgt über einen komplexen Tool-Flow, der die Verwendung von VHDL, ANSI C und Assembler vorsieht. Über VHDL wird zunächst die Struktur des Systems einschließlich der Kommunikation zwischen den Prozessen beschrieben, während anschließend jeder einzelne Prozess in C oder Assembler programmiert wird.

Silicon Hive

Die Philips-Ablegerfirma Silicon Hive bietet konfigurierbare Prozessor-Arrays an, die eine flexible Alternative zu festverdrahteten Komponenten in SoCs darstellen sollen ([Sil05]). Da Silicon-Hive-Prozessoren im Allgemeinen die Aufgabe von Accelerator-Bausteinen für rechenintensive Algorithmen übernehmen sollen, ist die Kopplung an einen Standardprozessor vorgesehen. Gegenwärtig existieren drei grundsätzliche Varianten, die jeweils für verschiedene Einsatzbereiche optimiert sind. Das Grundelement bildet in allen Architekturen eine als "Processing and Storage Element" (PSE) bezeichnete Basiszelle (siehe Abbildung 2.17a). Ein PSE ist ein VLSI-Datenpfad mit konfigurierbarer Anzahl von Rechenelementen, Verbindungs- und Speicherblöcken. Eine Zelle (CELL, Abbildung 2.17b) besteht wiederum aus mehreren PSEs, die über "Communication Lines" miteinander verbunden sind, und bildet bereits einen eigenständigen Prozessor. In einer höheren Hierarchiestufe können weiterhin mehrere Zellen zu einem Cluster zusammengefasst werden ("Streaming Array", Abbildung 2.17c).

Die von Silicon Hive angebotenen Varianten sind der Avispa-, Moustique- und Bresca-Prozessor. Der Avispa besteht typischerweise aus einer einzelnen Zelle mit mehreren parallelverarbeitenden PSEs, und soll eine gute Rechenleistung bei geringem Energieverbrauch bieten. Der Moustique-Prozessor ist auf einen geringen Flächenbedarf ausgerichtet, und beinhaltet ebenfalls nur eine Zelle, die auch nur ein oder wenige PSEs beinhaltet. Demgegenüber ist der Bresca-Prozessor mit mehreren Zellen auf maximale Rechenleistung ausgelegt.

Die von Silicon Hive zur Verfügung gestellte Programmierumgebung enthält unter anderem einen "Partitioning Compiler", der dem Benutzer bei der Aufteilung des Codes zwischen Host-Prozessor und Accelerator behilflich ist. Der Programmiervorgang wiederum erfolgt mit dem HIVECC-Compiler, dessen Befehle eine Untermenge von ANSI C bilden, und der auf die jeweils zu Grunde liegende Zielarchitektur angepasst ist. Für weitere Informationen sei auf [Sil05] verwiesen.

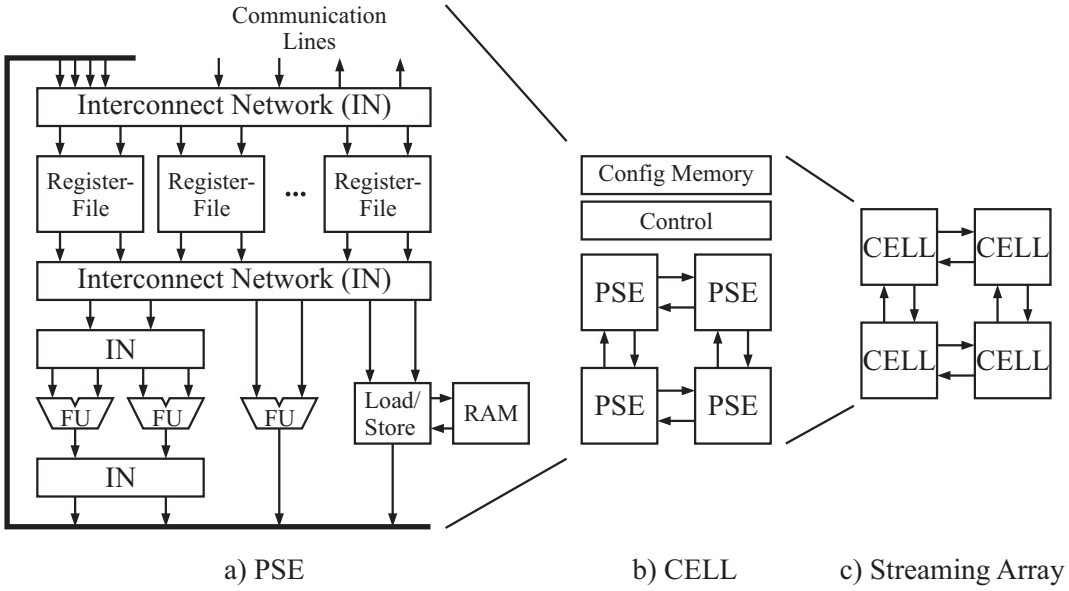


Abbildung 2.17: Darstellung der Hierarchie in Silicon-Hive-Prozessoren (s.a. [Sil05]): a) Processing and Storage Element (PSE), b) Zelle (Cell) aus mehreren PSEs, c) Feld aus mehreren Zellen

2.5 Rekonfigurierbare SoCs

Die Untersuchungen und Ergebnisse dieser Arbeit beziehen sich auf *eingebettete* rekonfigurierbare Architekturen, die eine Komponente eines heterogenen System-on-a-Chip (SoC) darstellen. So können diese Strukturen beispielsweise zur flexiblen Befehlssatzerweiterung eines angekoppelten Host-Prozessors eingesetzt werden.

Der Begriff *rekonfigurierbarer SoC* bezeichnet eine heterogene Architektur, die entweder rekonfigurierbare Bausteine als Komponenten enthält, oder bei der die verschiedenen Einzelkomponenten selbst auf einer rekonfigurierbaren Struktur abgebildet werden. Bezüglich der Programmier- oder Rekonfigurierbarkeit lassen sich verschiedene Klassen unterscheiden. Ein *ASIC-SoC* ist ein aus mehreren dedizierten Komponenten zusammengesetzter Chip, der sich nach der Entwurfsphase nicht mehr ändern oder programmieren lässt. Demgegenüber wird unter einem *PLD-SoC* ein System-on-a-Chip verstanden, dessen Einzelkomponenten komplett auf einer rekonfigurierbaren Architektur, beispielsweise einem FPGA oder einem CPLD, abgebildet werden. Zwischen diesen beiden Extremen liegen die kombinierten *ASIC-PLD-SoC-Lösungen*, auf denen neben dedizierten Einzelkomponenten auch rekonfigurierbare Bausteine integriert werden. Diese auch als *System-Level-Integration-Chips* bezeichneten Bausteine sind derzeit im Wesentlichen Gegenstand

von Forschungsarbeiten. Drei dieser Projekte sollen in den nächsten Abschnitten beispielhaft vorgestellt werden.

2.5.1 STMicroelectronics: SoC mit eFPGA

Die Firma STMicroelectronics hat im Jahr 2003 eine Arbeit über eine Architektur veröffentlicht, bei der ein eingebettetes FPGA zur flexiblen Erweiterung des Befehlssatzes eines Host-Prozessors verwendet wird (siehe [BLFC03]). Der Chip setzt sich in der veröffentlichen Implementierung aus Standard-IP-Komponenten zusammen. Der Aufbau ist in Abbildung 2.18 dargestellt.

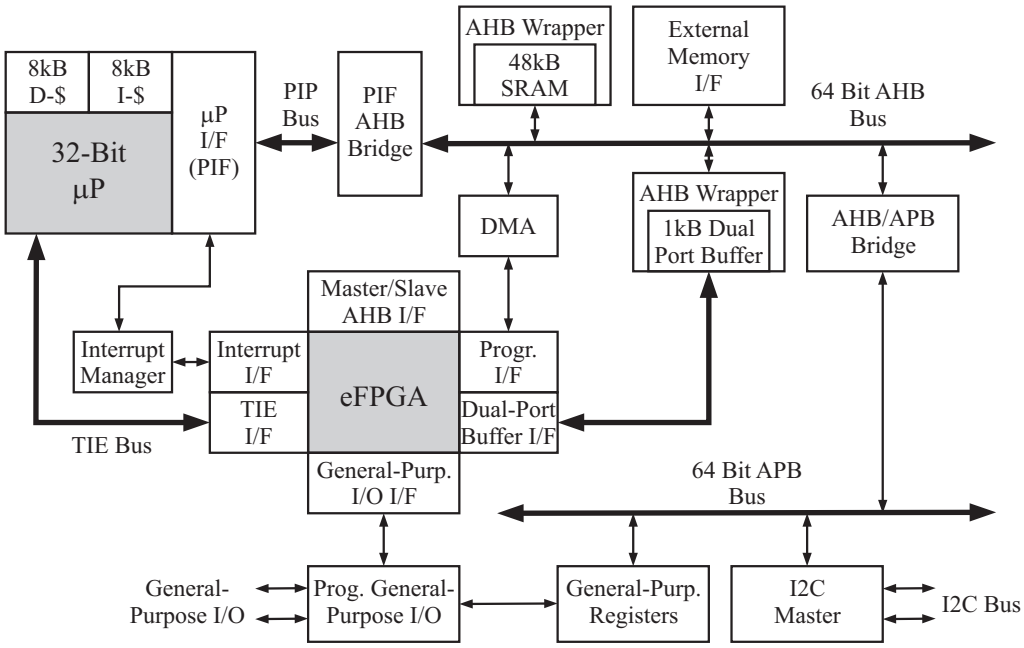


Abbildung 2.18: Block-Diagramm des ASIC-PLD-SoCs von STMicroelectronics ([BLFC03])

Bei dem Prozessor handelt es sich um einen für das Projekt modifizierten 32-Bit Tensilica Xtensa mit fünfstufiger Pipeline und angekoppelte 8 kByte großen Instruktions- und Daten-Caches. Das FPGA ist ein von M2000 lizenzierter eFPGA-Baustein der FlexEOS-Familie, der um einige Interfaces zu den anderen SoC-Komponenten wie dem Prozessor oder den Speichereinheiten erweitert wurde. Das eFPGA setzt sich aus insgesamt 3000 MFCs (Multi-Function Logic) Zellen zusammen, die wiederum in 24 größeren Clustern organisiert sind. Jede MFC enthält ihrerseits eine 4-Input-LUT und ein Flipflop. Die Ein-

zelkomponenten des Chips kommunizieren über ebenfalls als IP-Bausteine lizenzierte Bussysteme wie AMBA AHB/APB oder PIF (Processor Interface) Busse.

Der in einer $0.18\mu m$-Technologie gefertigte Chip belegt $20mm^2$ Chipfläche, von welcher 40% von dem eFPGA eingenommen wird. Die maximale Taktfrequenz liegt bei 175MHz, die mittlere Leistungsaufnahme wird mit 300mW bei 100 MHz Taktfrequenz angegeben.

2.5.2 Uni Bologna: XiSystem

Eine über den in Abschnitt 2.5.1 beschriebenen Ansatz zur Einbettung eines FPGA in einen programmierbaren SoC weiterentwickelte Struktur ist 2006 von der University of Bologna veröffentlicht worden (siehe [Lod06]), ebenfalls in Zusammenarbeit mit STMicroelectronics. Es handelt sich um den mit XiSystem bezeichneten SoC, der zwei anwendungsspezifische feingranulare Einheiten enthält. Das Blockdiagramm von XiSystem ist in Abbildung 2.19a gezeigt.

XiSystem besteht im Kern aus einem Prozessor, dem XiRisc-Core, sowie aus zwei unterschiedlich aufgebauten rekonfigurierbaren Blöcken, die für verschiedene Zwecke eingesetzt werden. XiRisc ist ein RISC-Core mit VLIW-Architektur mit zwei parallelen Datenpfad-Einheiten und Einheiten zur effizienten Ausführung von Signalverarbeitungs-Operation. Eine dritte parallele Datenpfad-Einheit wird durch das *PiCoGA*-Array (Pipelined Configurable Gate Array) bereit gestellt (siehe Abbildung 2.19b und c). PiCoGA ist ein FPGA-ähnlich aufgebautes Array, deren Basiszell-Elemente jedoch neben Carry-Logik zwei 4-Input Lookup-Tabellen mit jeweils zwei Bit breiten Ausgängen enthalten, so dass eine höhere Granularität vorliegt als bei Standard-FPGA-Bausteinen. Die 2-Bit-Granularität trägt der Aufgabe des PiCoGA Rechnung, im Wesentlichen zur schnellen Ausführung von DSP-Operationen verantwortlich zu sein.

Weiterhin ist in XiSystem ein eFPGA integriert, das als rekonfigurierbares IO-Modul dient. Dieses eFPGA ist ebenfalls applikationsspezifisch optimiert, und beinhaltet einfache Logikzellen mit jeweils einer 4-Input-LUT, einem Multiplexer und einem Register. Dieser sehr feingranulare Ansatz wird den Anforderungen der IO-Schnittstellen gerecht, die im Wesentlichen aus 1-Bit-Logikoperationen zusammengesetzt sind. Die Einzelkomponenten von XiSystem kommunizieren über Standard-Bus-IP-Blöcke wie AMBA AHB/APB.

XiSystem belegt laut [Lod06] eine Chipfläche von $42mm^2$ inklusive IO-Pads bei einer $0.13\mu m$-Technologie. PiCoGA belegt dabei $11mm^2$ Chipfläche, während das eFPGA $6mm^2$ belegt. Die Taktfrequenz wird mit 166MHz angegeben, und die mittlere Leistungsaufnahme mit 180mW bei 100MHz. In [Lod06] wird gezeigt, dass ein mit dem PiCoGA-Array erweiterter XiRisc-Prozessor verglichen mit einem XiRisc, der lediglich einen einzelnen Datenpfad enthält, für bestimmte Anwendungen bis zu 15 mal schneller ist, und 89% weniger Energie verbraucht.

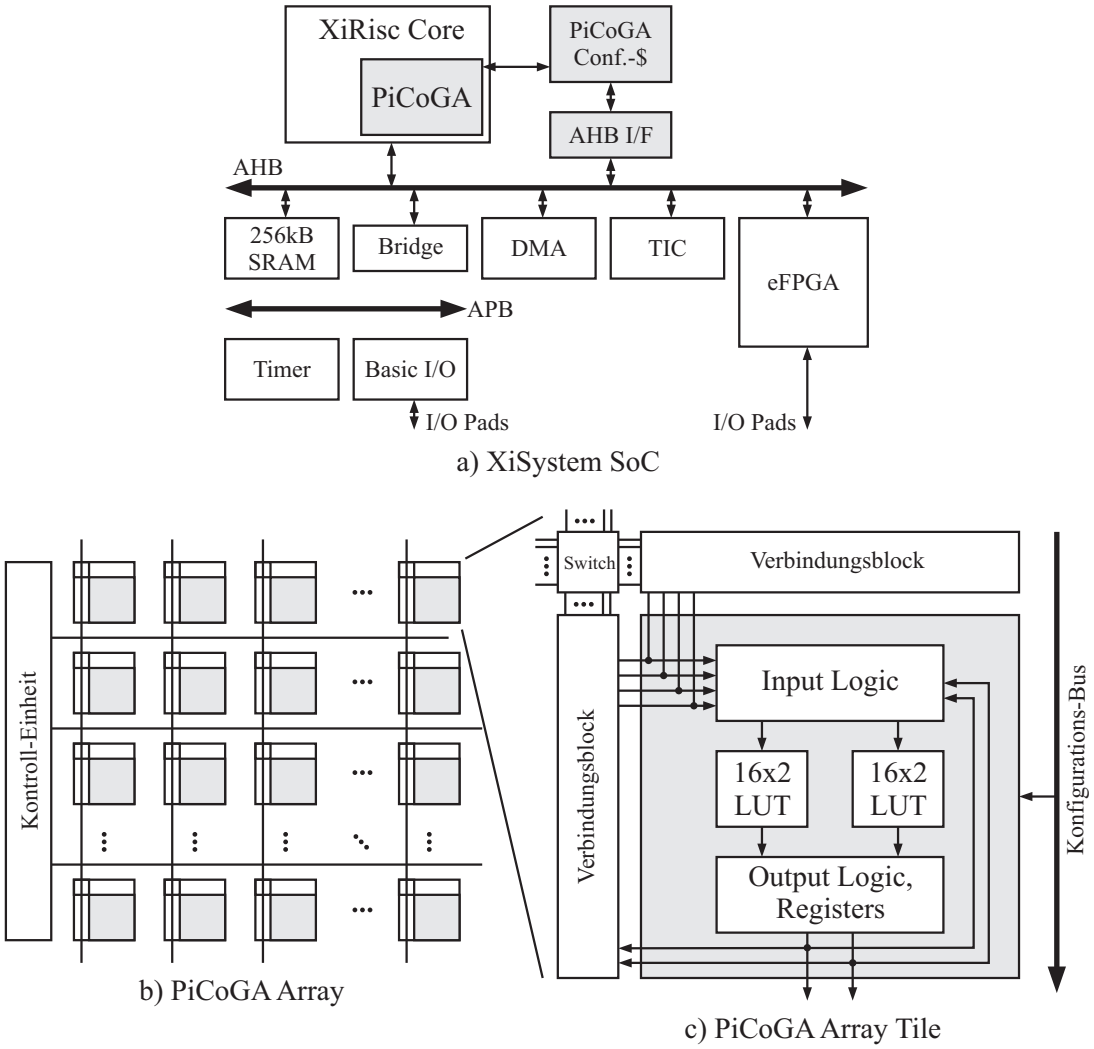


Abbildung 2.19: XiSystem-Architektur aus [Lod06]. a) System-Blockdiagramm, b) Struktur des PiCoGA-Arrays für applikationsspezifische Instruktionen, c) PiCoGA-Basiszelle

XiSystem ist somit ein Beispiel für eine Architektur, die aus anwendungsspezifisch optimierten rekonfigurierbaren Architekturen besonderen Nutzen zieht. Für weitere Einzelheiten sei auf [Lod06] verwiesen. Das PiCoGA-Array wird in [LTC03a] näher behandelt.

2.5.3 Uni Twente: Chameleon SoC Template

In den letzten beiden Abschnitten sind Beispiele für SoCs vorgestellt worden, die durch teilweise anwendungsspezifische feingranulare Bausteine flexibel erweitert worden sind. Ein Beispiel für ein Projekt, welches auch grobgranulare Elemente auf einem SoC integriert, ist das Chameleon-Projekt, welches von der Universität Twente veröffentlicht wurde. Dieses Projekt ist nicht zu verwechseln mit dem Reconfigurable Communications Processor (RCP) der Firma Chameleon Systems, die mittlerweile nicht mehr am Markt vertreten ist.

Das Chameleon-Projekt (siehe z.B. [HSM03] und [Hey04]) stellt ein SoC-Template, mit dessen Hilfe verschiedene Architekturblöcke zu einem heterogenen System-on-a-Chip zusammengesetzt werden können. Die Architekturblöcke, wie z.B. ASIC-Einheiten, eFPGAs, Standard-Prozessoren (GPPs), DSPs oder grobgranulare Einheiten (Domain Specific Reconfigurable Arrays, DSRAs) sind untereinander über ein paketvermitteltes Network-on-a-Chip (NoC) verbunden (siehe Abbildung 2.20).

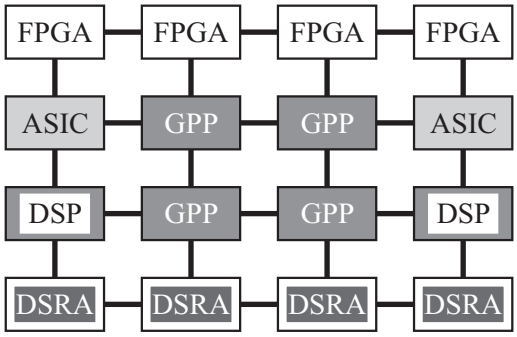
Eine grobgranulare Einheit, die als Teil eines Chameleon-SoCs eingesetzt werden kann, wird in [Hey04] veröffentlicht. Der *Montium-Tile-Processor* ist in Abbildung 2.20b skizziert. Der Datenpfad besteht aus fünf parallel arbeitenden ALUs, die über ein komplexes Interconnect-Netzwerk an zehn unabhängig addressierbare Speicherbänke angekoppelt sind. Weiterhin werden die für das NoC-Netzwerk abzuarbeitenden Protokolle von der *Communication and Configuration Unit* übernommen. In einer $0.13\mu m$-Technologie belegt ein Montium-Prozessor eine Fläche von etwa $2mm^2$, und kann eine FIR-Filterung mit einer Taktfrequenz von 140MHz, und eine FFT mit 100Mhz ausführen. Die durchschnittliche Leistungsaufnahme wird mit 57.7mW bei 100MHz angegeben.

2.5.4 Weitere Arbeiten

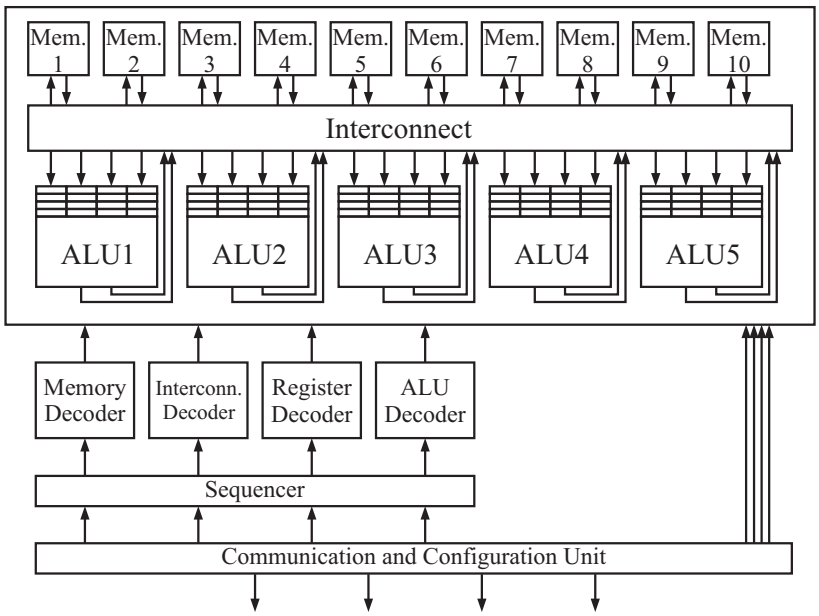
Der bereits in Abschnitt 2.4 vorgestellte MorphoSys-Chip ist ein weiteres Beispiel für einen SoC, auf dem eine grobgranulare rekonfigurierbare Architektur mit einem RISC-Prozessor gekoppelt wird (siehe z.B. [GSL+99]). In [ZPG+00] wird eine Forschungsarbeit präsentiert, bei der ein eFPGA mit einem ARM-Prozessor auf einem Silizium-Die integriert wurde.

2.6 Übersicht zu grobgranularen Architekturen

Neben den in diesem Kapitel vorgestellten Beispielen für nicht-kommerzielle rekonfigurierbare Architekturen mit grobgranularen Datenpfaden existieren zahlreiche weitere publizierte Forschungsprojekte, die sich mit diesem Thema befassen, und aus denen eigene Vorschläge für Architekturen entstanden sind. Diese können hier nicht alle erläutert werden, stattdessen seien hier nur die bekanntesten und am häufigsten zitierten Projekte namentlich erwähnt. Ohne Anspruch auf Vollständigkeit seien beispielsweise das *PADDI*-Projekt



a) Chameleon-SoC Architektur-Template



b) Montium Processing Tile

Abbildung 2.20: Chameleon-Projekt. a) Chameleon SoC-Template, b) Montium-Prozessor als *Domain Specific Reconfigurable Array* (DSRA) im Chameleon-Chip

genannt (Berkeley, 1990), weiterhin *Colt* (Virginia State University, 1996), *Pleiades* (Berkeley, 1996), *Garp* (Berkeley, 1997), *PipeRench* (Carnegie Mellon University, 1998), sowie *REMARC* (Stanford University, 1998). Eine Übersicht zu Details aus diesen Projekten findet sich beispielsweise in [Nag01].

Darüber hinaus existieren einige Firmen, die sich auf die Entwicklung und Vermarktung grobgranularer rekonfigurierbarer Prozessorstrukturen spezialisiert haben. Neben den bereits genannten Firmen PACT, picoChip und Silicon Hive seien hier noch die Firmen QuickSilver Technology und Elixent Ltd. erwähnt, die ebenfalls grobgranulare IP-Cores anbieten.

In Tabelle 2.1 sind einige Eigenschaften der in den obigen Abschnitten dargestellten grobgranularen rekonfigurierbaren Architekturen zusammengefasst.

Wie in der Einleitung bereits erwähnt, wird der bei der Entwicklung einer rekonfigurierbaren Architektur zur Verfügung stehende Entwurfsraum von einer Vielzahl von Parametern aufgespannt. Um die Größe dieses Entwurfsraumes zu verdeutlichen, sind in Abbildung 2.21 einige der in diesem Kapitel vorgestellten Architekturen in einem von den drei Parametern *Wortbreite*, *Basiszell-Komplexität* und *Instruktionstiefe* aufgespannten Diagramm eingetragen. Der Begriff *Instruktionstiefe* bezeichnet die Anzahl der lokal speicherbaren Instruktionen oder Konfigurationen einer Basiszelle oder eines Logikblockes.

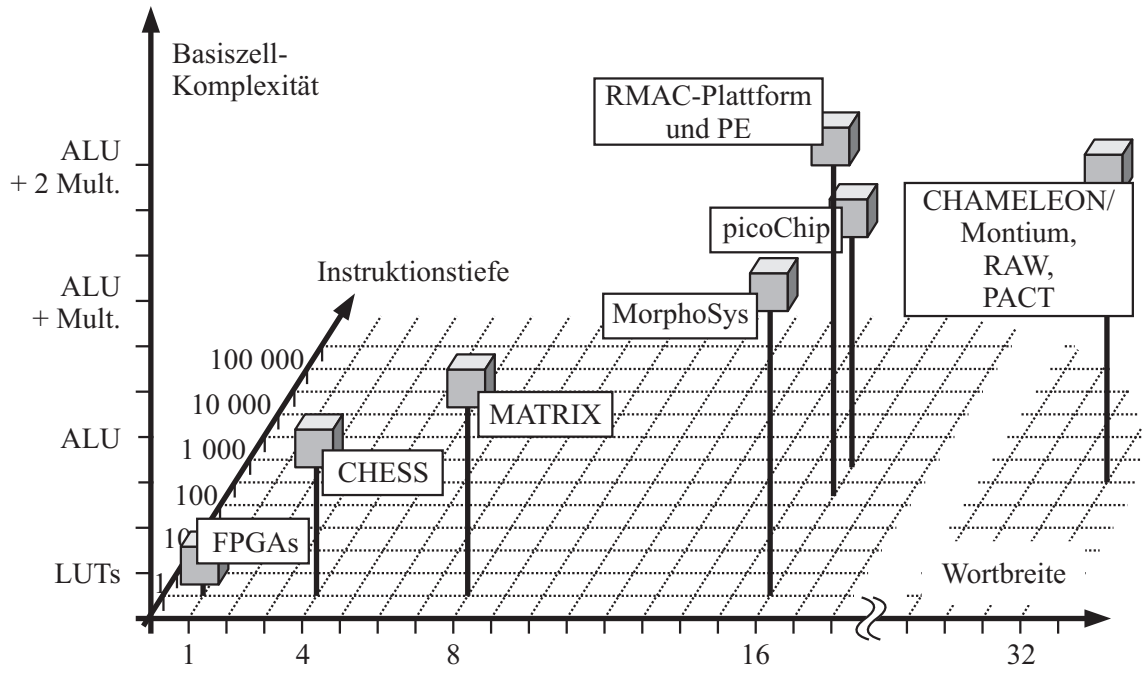


Abbildung 2.21: Entwurfsraum für rekonfigurierbare Architekturen

Tabelle 2.1: Übersicht zu Eigenschaften der vorgestellten grobgranularen Architekturen

Name	Erstmals veröffentlicht	Wortbreite	Technologie	Eigenschaften
CHESS	1999	4 Bit	0.35μm 4LM	PEs mit 4-Bit-ALUs Array aus 512 PEs Fläche: 30mm^2 Taktrate: 200MHz
MATRIX	1996	8 Bit	0.5μm	Array aus 10×10 BFUs BFUs enthalten ALUs, Speicher und Kontrollmodule Fläche: 238mm^2 Taktrate: 100MHz
MorphoSys	1999	16 Bit	0.35μm	Array aus 8×8 RCs SoC aus Array, RISC-Core und RAM PEs mit ALU, Multiplizierer und Register
RaPiD-1	1996	16 Bit	0.5μm	1D-Feld aus 16 RaPiD-Zellen Fläche: 144mm^2 Taktrate: 100MHz
ASIP-Accelerator	2002	16 Bit		Zwei applikationsspezifische PEs Instruktions- und Daten-RAM, Kontrolleinheiten
RAW	1997	32 Bit	0.15μm	Array aus 4×4 RAW-Tiles Komplexe Tiles mit MIPS-Prozessor, Caches und Routing Fläche: 256mm^2 Taktrate: 225MHz Leistungsaufnahme: 25W
PACT	2002	16, 24 oder 32 Bit		Array aus ALU- und RAM-PAEs, paketorientierte Verbindungsstruktur, dynamisch rekonfigurierbar

Name	Erstmals veröf-fentlicht	Wort-breite	Tech-nologie	Eigenschaften
picoChip picoArray	2003	16 Bit		Array aus 430 Prozessoren Busstruktur mit Zeitmultiplex mit statischem Scheduling
Silicon Hive	2003			Rekonfigurierbare Akzeleratoren Variable Anzahl hierarchisch verbundener Prozessorzellen

3 Entwurfsraum-Exploration und Optimierung rekonfigurierbarer Architekturen

Komplexe Hardware-Strukturen wie Prozessoren und rekonfigurierbare Architekturen zeichnen sich meist durch einen sehr großen Entwurfsraum aus, da beim Design der Architektur eine Vielzahl von Parametern festgelegt werden müssen. Diese Parameter sind dabei so zu wählen, dass eine in Hinblick auf die Spezifikation und die auszuführenden Anwendungen optimale Schaltung entsteht. Je nach Komplexität der Schaltung ist die Wahl der korrekten Parameter ein höchst komplexer Prozess, der eine Kenntnis des Einflusses jedes einzelnen Parameters auf die Performance des Gesamtsystems erfordert. Da sich die Einflüsse einzelner Parameter teilweise untereinander beeinflussen, ist ein systematisches Verfahren zur Suche nach Optima im Entwurfsraum, eine Entwurfsraum-Exploration, erforderlich.

Bezüglich fein- und grobgranularer Architekturen sind verschiedene Projekte veröffentlicht worden, die sich mit der Entwurfsraumexploration befassen. Alle Verfahren beziehen ihre Ergebnisse aus mehr oder weniger abstrakten Modellen, da nicht für jeden Parametersatz eine Architektur implementiert und simuliert werden kann. Da auch das in dieser Arbeit präsentierte Verfahren zur Untersuchung grobgranularer Architekturen auf einem Modell fußt, werden in den folgenden Abschnitten diesbezüglich einige relevante Forschungsarbeiten präsentiert. Größtenteils beschränkt sich die Darstellung hier auf eine kurze Präsentation der jeweiligen Zielsetzung und der Vorgehensweise. Die Ergebnisse können nicht in allen Einzelheiten präsentiert werden, hier sei auf die jeweiligen Literaturstellen verwiesen.

3.1 Entwurfsraumexploration für feingranulare Architekturen

3.1.1 Uni Toronto: Versatile Place-and-Route (VPR)

Ein an der University of Toronto betriebenes Forschungsprojekt befasst sich mit der Entwurfsraumexploration für universelle FPGA-Architekturen (siehe z.B. [BRM99] oder [BR99]). Wie aus der Beschreibung des allgemeinen Aufbaus eines FPGAs in Abschnitt

2.3.1 hervorgeht, wird der Entwurfsraum derartiger feingranularer Architekturen von einer Vielzahl von Parametern aufgespannt. Um eine flächen-, energie- und geschwindigkeitsoptimale Struktur zu finden, die zudem eine ausreichend hohe Flexibilität bietet, sind beispielsweise die Anzahl horizontaler und vertikaler Verbindungsleitungen, die Länge der Leitungen, die Konnektivität der Switch- und Verbindungsblöcke, die Cluster-Größe, die LUT-Größe und weitere Parameter geeignet zu wählen.

In dem von der Uni Toronto veröffentlichten Projekt werden die Fläche und die Datenrate generischer FPGA-Architekturen untersucht. Die Analysen werden mit dem *VPR (Versatile Place-and-Route)* getauften Tool durchgeführt. Basis dieses Tools ist ein in sehr weiten Bereichen parametrisierbares FPGA-Modell, welches den Aufbau jeder einzelnen Komponente, beispielsweise eines Logik-, Verbindungs- oder Switchblockes, kennt. Mit Hilfe von VPR können zudem die mit einer proprietären Sprache beschriebenen Algorithmen auf das generische Feld abgebildet werden.

Zur Abschätzung von Fläche und Datenrate sind jeweils analytische Modelle entwickelt worden. Um die Fläche abzuschätzen, werden aus einem durch den Benutzer vorgegebenen Parametersatz die Zahl der zur Implementierung des FPGA erforderlichen Transistoren ermittelt. Unter Berücksichtigung der Größe jedes einzelnen Transistors wird anschließend die Fläche ermittelt. Zur Bestimmung der Datenrate werden alle in dem abgebildeten Routing vorhandenen Pfade untersucht, und deren Pfadverzögerung mit dem Elmore-Delay-Modell ermittelt. Aus der Signallaufzeit des langsamsten Pfades wird die maximal mögliche Taktfrequenz bestimmt. Das Elmore-Modell wird auch in dieser Arbeit zur Ermittlung der Verzögerungszeit der Verbindungsstruktur verwendet (siehe Abschnitt 6.1.2).

Eine Untersuchung des Energieverbrauchs ist nicht nativ in VPR implementiert, es existieren allerdings Erweiterungen, mit deren Hilfe auch Leistungsaufnahme und Energieverbrauch der generischen Architekturen ermittelt werden können (siehe z.B. [PWY05]).

3.1.2 RWTH Aachen: eFPGA-Analyse und Optimierung

Wie in der Einleitung zu dieser Arbeit bereits dargestellt, besteht der Nachteil eingebetteter FPGAs in der Ineffizienz bezüglich des Flächenbedarfs und Energieverbrauches dieser Bausteine. Es sind zwei grundsätzlich verschiedene Verfahren denkbar, um dennoch die Vorteile bezüglich der Verarbeitungsgeschwindigkeit rekonfigurierbarer Komponenten in SoCs ausnutzen zu können, nämlich entweder durch Erhöhung der Granularität und Verwendung anwendungsspezifischer Datenpfade in Form von grobgranularen rekonfigurierbaren Architekturen, oder durch anwendungsspezifische Optimierung der feingranularen Architekturen. In der vorliegenden Arbeit wird das Potenzial der grobgranularen Strukturen untersucht. Demgegenüber sind in [vNBN06] und [vKN$^+$06] Untersuchungen vorgestellt worden bezüglich des Optimierungspotentials eingebetteter FPGAs, und zu Möglichkeiten der SoC-Integration dieser Bausteine, inklusive einer quantitativen Analyse der Effizienzsteigerung durch die vorgeschlagenen Verfahren. Da die Ergebnisse zu

optimierten feingranularen Architekturen zu den in der vorliegenden Arbeit vorgestellten Analysen grobgranularer Strukturen in unmittelbarem Zusammenhang stehen, sollen die beiden Aufsätze in den folgenden Abschnitten detaillierter dargestellt werden.

eFPGA für arithmetische Operationen

Kommerzielle FPGAs müssen Benutzern ausreichend Flexibilität zur Verfügung stellen, um Applikationen aus einem breiten Anwendungsspektrum abbilden zu können. Problematisch wird dies durch die Tatsache, dass sich Hardwarerealisierungen von Anwendungen aus verschiedenen Anwendungsklassen in ihren Charakteristika teilweise fundamental voneinander unterscheiden. So weisen beispielsweise Datenpfad-orientierte Hardwarearchitekturen eine höhere Regularität auf als Strukturen zur Realisierung von Kontrollfluss-Aufgaben wie Automaten (FSMs). In [vNBN06] wird anhand eines Beispiels gezeigt, dass der Kontrollblock eines Korrelators für einen GPS-Empfänger bei einer FPGA-Implementierung zu etwa jeweils einem Viertel kurze und lange Verbindungsleitungen belegt, während mehr als die Hälfte der Daten über Leitungen mittlerer Länge geroutet wird (zur allgemeinen Struktur von FPGAs siehe z.B. Abschnitt 2.3). Demgegenüber sind im Datenpfad des Korrelators mehr als zwei Drittel aller Verbindungen kurze Leitungen. Die Ineffizienz kommerzieller FPGAs bezüglich des Flächenbedarfs und des Energieverbrauchs entsteht daraus, dass alle Anwendungsklassen mit etwa gleicher Effizienz abbildbar sein müssen.

Aus der hohen Flexibilität allgemeiner FPGAs und der damit verbundenen Ineffizienz entsteht die Frage, welches Optimierungspotenzial feingranulare Architekturen bei Ausrichtung auf eine bestimmte Anwendungsklasse besitzen. In [vNBN06] werden eFPGA-Strukturen untersucht, die für arithmetische Operationen optimiert sind. Möglichkeiten zur Optimierung bieten die Verbindungsstruktur, die bis zu 90% der Fläche eines Standard-FPGAs belegt (siehe z.B. Kapitel 7), sowie die Logikzellen.

Abbildung 3.1 zeigt eine in [vNBN06] vorgestellte optimierte Zelle bestehend aus einem Cluster mit acht Logikelementen (LE), zwei Verbindungsblöcken (*Connection Blocks*, CB) und einem Routing-Switch (RS). Um der starken Lokalität der Elementaroperatioen bei arithmetischen Strukturen Rechnung zu tragen, ist die Anzahl mittlerer und langer Leitungen gegenüber allgemeinen FPGAs stark reduziert worden. In der vorgeschlagenen Architektur wird eine Kanalbreite von 4 Bit verwendet, die innerhalb der Verbindungsblöcke und Routing-Switches unabhängig voneinander verarbeitet werden können. Lokale Verbindungsblöcke (LCB, s.a. Abbildung 3.2) stellen die für arithmetische Operationen erforderliche hohe Konnektivität lokal benachbarter Zellen sicher.

Das optimierte Logikelement (Abbildung 3.2) besteht neben dem angesprochenen lokalen Verbindungsblock aus der datenverarbeitenden Einheit (*Core Logic*) und einer Registerstufe. Das Logikelement enthält zwei dedizierte Volladdierer (VA) und zwei Lookup-Tabellen (LUT) mit jeweils zwei Eingängen, die zu einer LUT mit drei Eingängen zusammengeschaltet werden können. Demzufolge beträgt die Granularität der Zelle zwei Bit

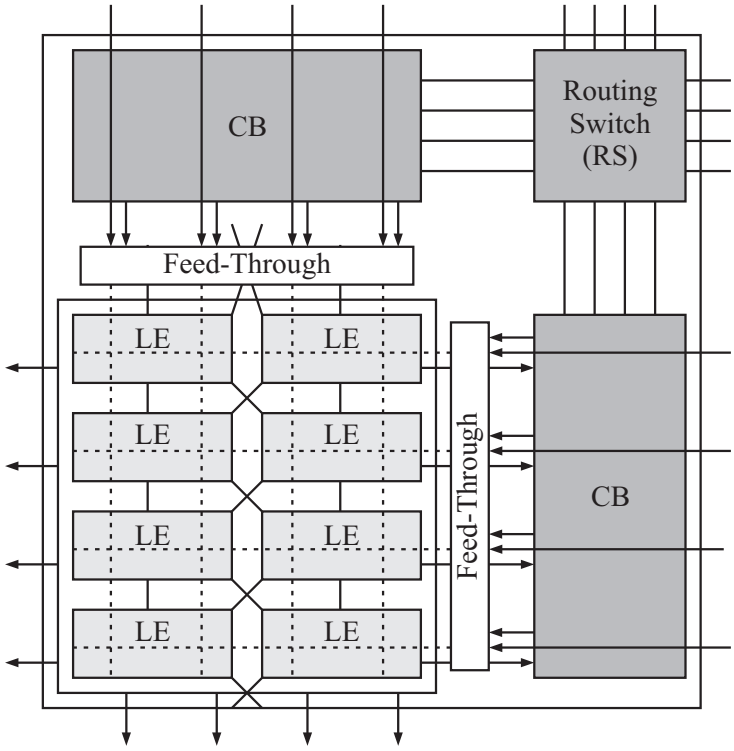


Abbildung 3.1: Für arithmetische Operationen optimierte eFPGA-Makrozelle bestehend aus acht Logikelementen (LE), zwei Verbindungsblöcken (Connection Blocks, CB) und einem Routing Switch (RS), s.a. [vNBN06]

für arithmetische Operationen, und ein Bit für allgemeine Operationen. Die Registerstufe enthält sechs Register, mit denen jeweils die vier Ausgänge der Logikzelle, und zusätzlich zwei globale Leitungen verzögert werden können. Die im Vergleich zu Standard-FPGAs hohe Registerzahl trägt der Tatsache Rechnung, dass zur Realisierung von arithmetischen Operationen mit vielen Pipelinestufen mehr Register erforderlich sind. Logikzellen von Standard-FPGAs enthalten demgegenüber meist nur ein oder zwei Register.

Es sind Simulationen auf Basis des Layouts eines 2×2-Arrays von eFPGA-Makrozellen nach Abbildung 3.1 in 130nm-Technologie bezüglich einer 8×8-Multiplikation, einer 4-Bit-MAC-Operation und eines 4-Bit-Butterflies durchgeführt worden. Die gleichen Operationen sind auch auf einen Altera Cyclone-I-FPGA, der ebenfalls in 130nm-Technologie gefertigt ist, abgebildet worden. Die Ergebnisse sind in Abbildung 3.3 gezeigt.

Das für arithmetische Operationen vorgeschlagene eFPGA-Makro ist dem Cyclone-I-FPGA bei allen abgebildeten Operationen bezüglich Fläche, Verzögerungszeit und Energieverbrauch überlegen. Die Verzögerungszeiten betragen 50% bis 80% des Standard-FPGAs. Signifikanter sind die Verbesserungen bezüglich des Flächenbedarfs, der je nach Operation

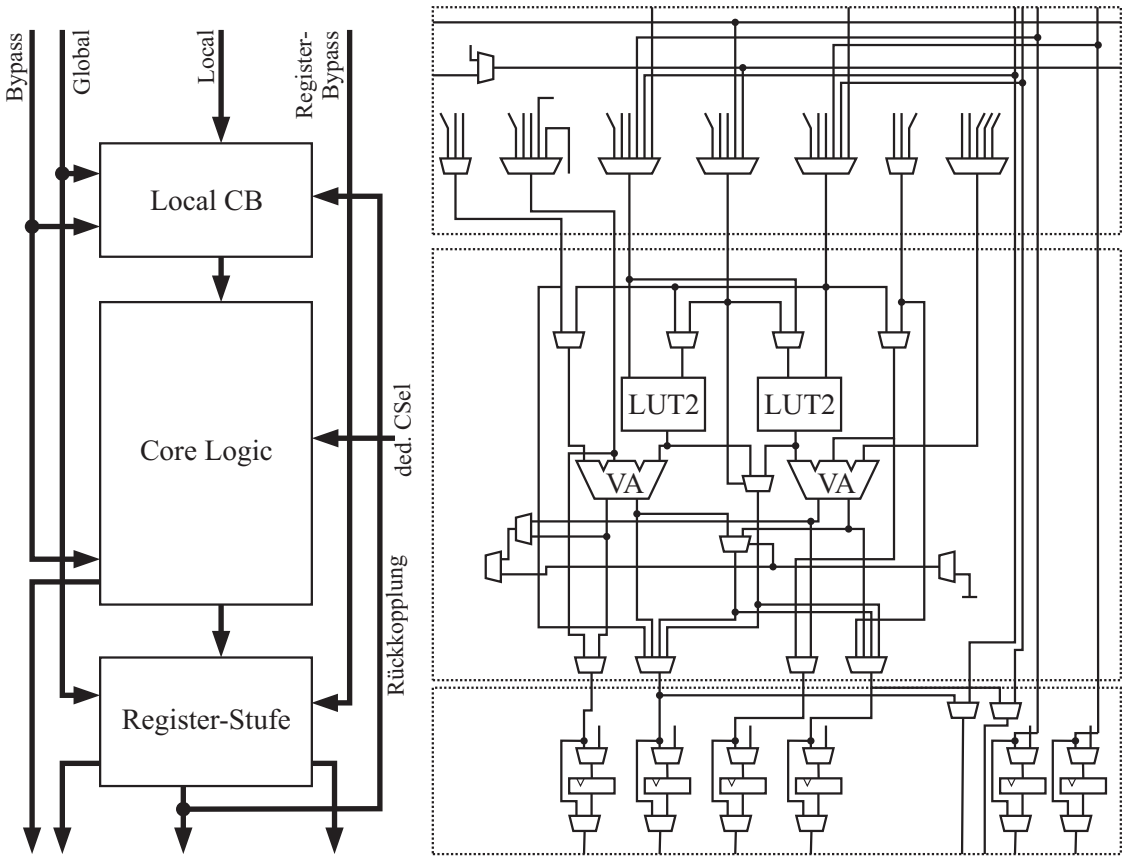


Abbildung 3.2: Für arithmetische Operationen optimiertes Logikelement, s.a. [vNBN06]

um den Faktor 3 bis 5 geringer ausfällt, und der Energie, welche um den Faktor 5 bis 10 niedriger ist als beim Cyclone-I.

Für weitere Informationen sei auf [vNBN06] verwiesen.

Analyse von eFPGAs als rekonfigurierbare Funktionseinheiten

Im obigen Abschnitt sind die Untersuchungen zu der in [vNBN06] vorgestellten optimierten eFPGA-Struktur dargestellt worden. Die Ergebnisse beziehen sich auf einzelne arithmetische Operationen, die auf das eFPGA abgebildet wurden. Kontrollfluss-Einheiten, sowie die Integration des eFPGA in einen SoC wurden nicht betrachtet. Demgegenüber werden in [vKN+06] Möglichkeiten, einen eFPGA-Kern an einen eingebetteten Prozessor anzubinden, untersucht und quantitativ ausgewertet.

In [vKN+06], wie auch in [LTC+03b], werden zunächst zwei grundsätzlich verschiedene Möglichkeiten der Anbindung einer rekonfigurierbaren Architekturkomponente an einen

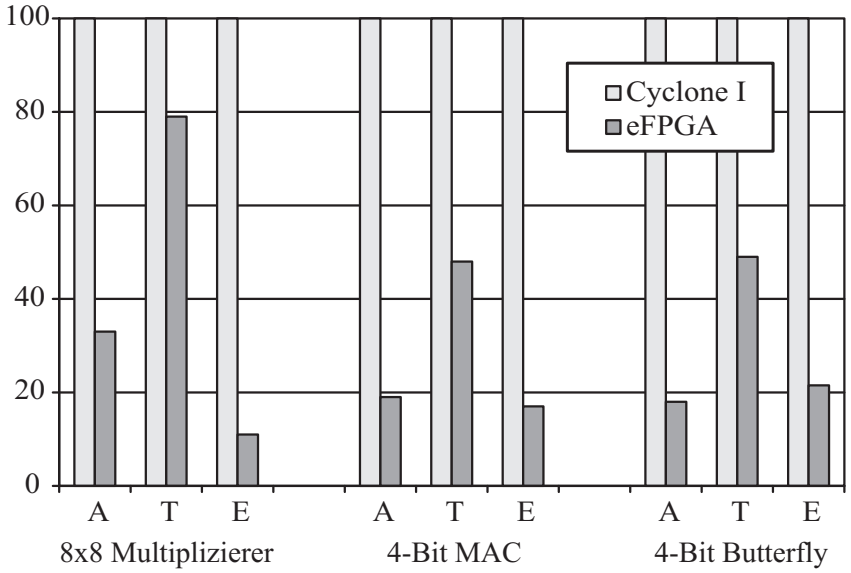


Abbildung 3.3: Fläche (A), Verzögerungszeit (T) und Energie (E) für verschiedene Operationen mit einem Altera Cyclone I FPGA, und mit der optimierten eFPGA-Struktur, s.a. [vNBN06]

Prozessor unterschieden. Dies sind die lose Kopplung und die enge Kopplung. Bei der losen Kopplung, dargestellt in Abbildung 3.4a, kommunizieren ein eFPGA und ein Prozessor über einen Standard-Bus (z.B. AMBA) miteinander. Eine Modifikation des Prozessors ist nicht erforderlich. Ein solches Bussystem stellt eine begrenzte Performance zur Verfügung, insbesondere dann, wenn noch andere Komponenten außer dem eFPGA angekoppelt sind. Daher bietet sich eine derartige Konfiguration in den Fällen an, in denen Algorithmen ausgeführt werden sollen, die in voneinander unabhängige Teile partitioniert werden können. Es müssen explizite Kommunikations- und Synchronisationsmechanismen eingeführt werden, was den Software-Entwurf entsprechend erschwert. Beim Prinzip der losen Kopplung wird in [vKN+06] weiterhin die Möglichkeit aufgezeigt, das eFPGA statt über einen Standard-Bus wie AMBA über einen dedizierten und schnelleren Prozessor-internen Bus mit der CPU zu verbinden (Abbildung 3.4b).

Der losen Kopplung gegenüber steht die enge Kopplung (Abbildung 3.4c). Bei dieser wird eine rekonfigurierbare Architekturkomponente in die Prozessorpipeline integriert. Das eFPGA hat Zugriff auf die internen Prozessorregister bei minimalem Mehraufwand für Kommunikationsaufgaben zwischen eFPGA und Prozessor. Problematisch bei dieser Anordnung ist laut [vKN+06] der Umstand, dass die Taktfrequenz, mit der auf eFPGAs abgebildete Schaltungen betrieben werden, typischerweise deutlich niedriger sind als der Prozessortakt. Dieses Problem lässt sich entweder durch Reduktion des Prozessortaktes

auf die eFPGA-Frequenz lösen, was eine Verlangsamung der Ausführungsgeschwindigkeit des Prozessors zur Folge hat, oder durch Einfügen von zusätzlichen Synchronisations-Mechanismen bei Verwendung unterschiedlicher Taktfrequenzen, was wiederum den Hardwareaufwand erhöht.

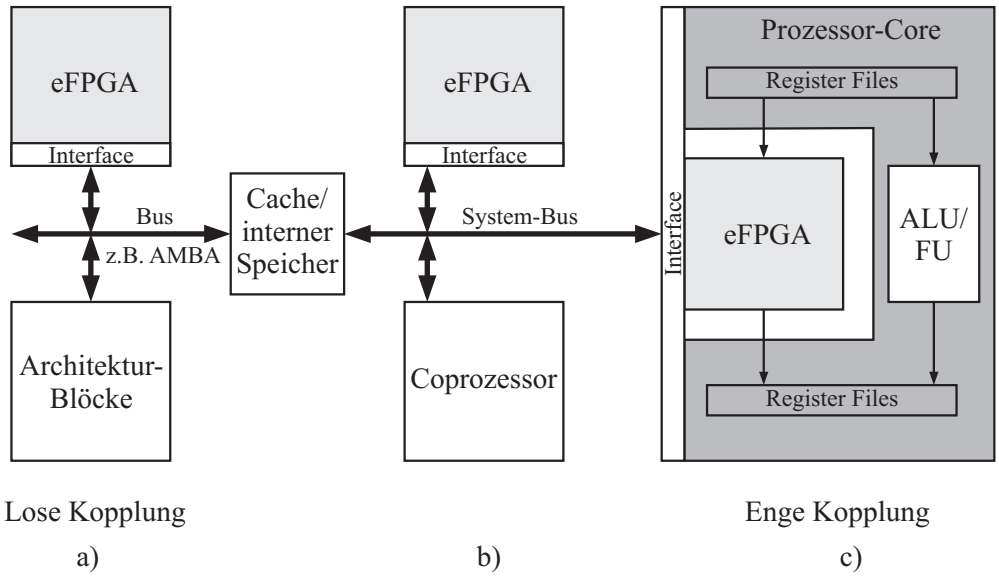


Abbildung 3.4: Möglichkeiten zur Anbindung eines eFPGA an eines Prozessor nach [vNBN06]. a) Lose Kopplung über einen Standard-Bus wie z.B. AMBA, b) Anbindung des eFPGA über einen System-internen Bus wie den Prozessor-Bus, c) enge Kopplung, Integration des eFPGA in die Prozessor-Pipeline

In [vKN+06] werden Architekturen untersucht, bei denen ein in [vNBN06] und im obigen Abschnitt vorgestelltes für arithmetische Operationen optimiertes eFPGA-Makro über die enge Kopplung in die Pipeline von RISC-Prozessoren eingebunden wird. Für die Analysen werden ein Prozessor mit MIPS-IV-Architektur mit 64-Bit Instruktions- und Datenwortbreite, und ein ARM940T mit 32-Bit-Architektur verwendet. Die Anbindung des eFPGA an einen der beiden Prozessoren ist in Abbildung 3.5 dargestellt. Das eFPGA ist in mehrere Teile aufgespalten, für die in [vKN+06] der Begriff *Supercluster* verwendet wird. Ein Supercluster enthält eine parametrisierbare Anzahl von eFPGA-Makrozellen nach Abbildung 3.1. Die Prozessorregister sind über einen Switch mit den Ein- und Ausgängen jedes Superclusters verbunden.

Zur Abschätzung der Performance wird ein heterogener Simulationsablauf verwendet, bei dem unterschiedliche Verfahren zur Bestimmung von Fläche, Datenrate und Energieverbrauch von Prozessor und eFPGA zum Einsatz kommen. Ausgehend vom C-Code für

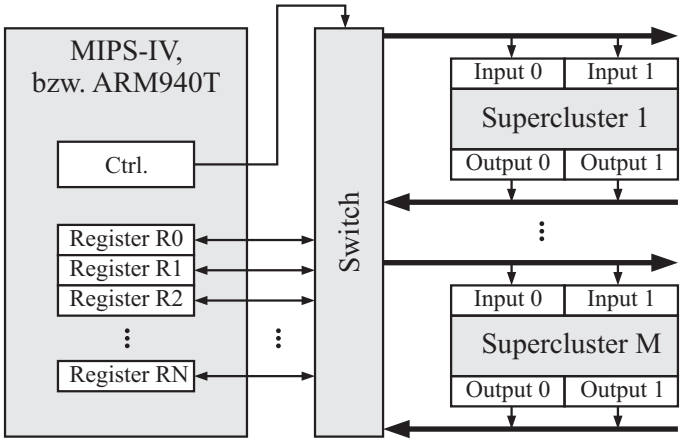


Abbildung 3.5: Hardware-Konfiguration für die in [vKN+06] vorgestellten Analysen

den Prozessor werden mit Hilfe taktgenauer Simulatoren die Ausführungszeiten bestimmt. Mit Hilfe eines Leistungsmodells für den Prozessor auf funktionaler Ebene (s.a. [BBB+06]) wird der Energieverbrauch abgeschätzt. Algorithmen-Kerne mit hohem Rechenaufwand werden manuell auf das eFPGA abgebildet. Davon ausgehend werden Fläche, Verzögerungszeiten und Energieverbrauch des eFPGA-Kerns bestimmt.

Es werden zwei verschiedene Algorithmen betrachtet, dies sind eine Implementierung des Data Encryption Standard (DES) und eine 3×3-Medianfilterung. Bei der DES-Verschlüsselung werden 64-Bit-Blöcke ein- und ausgangsseitig permutiert, und im Kern durch eine 16-fach iterierte Funktion, bei der im Wesentlichen XOR-Verknüpfungen und weitere Permutationen stattfinden, verschlüsselt. Die Grundoperation bei der angewandten Medianfilterung ist die Compare-and-Swap-Operation, siehe hierzu auch Abschnitt 5.3.2.

Die beiden Algorithmen wurden auf den betrachteten Hardwarearchitekturen implementiert. Untersucht wurde jeweils die Implementierung auf einem MIPS-IV- und einem ARM9-Prozessor, jeweils mit und ohne eingekoppeltem eFPGA zur Beschleunigung der Algorithmen. Desweiteren ist die in [LTC+03b] dargestellte Variante eines VLIW-Prozessors mit ebenfalls nach enger Kopplung integriertem PiCoGA-Feld (siehe hierzu auch Abschnitt 2.5.2) in die Ergebnisübersicht mit einbezogen worden. Die in [vKN+06] präsentierten Ergebnisse bezüglich der beiden Algorithmen sind in Tabelle 3.1 zusammengestellt.

Im Vergleich zur reinen Software-basierten Implementierung des DES-Algorithmus lassen sich durch die Erweiterung des Prozessors mit einem eFPGA Performance-Gewinne von bis zu drei Größenordnungen erzielen. Die MIPS-eFPGA-Lösung arbeitet zudem mehr als doppelt so schnell wie die VLIW-PiCoGA-Architektur. Die Energie lässt sich im Vergleich zur MIPS-IV-basierten Software-Lösung ebenfalls um drei Größenordnungen reduzieren.

Tabelle 3.1: Flächenbedarf (A), Ausführungszeit (T) und Energieverbrauch (E) für die in [vKN+06] untersuchten Architekturvarianten

	DES			Median-Filter		
	A/mm^2	$T/\mu s$	$E/\mu Ws$	A/mm^2	$T/\mu s$	$E/\mu Ws$
MIPS	9.80	359.60	338.88	9.80	38.70	17.50
MIPS+eFPGA	25.98	0.51	0.30	25.60	3.70	4.40
ARM9	4.20	1013.88	46.61	4.20	40.80	3.20
ARM9+eFPGA	20.38	1.03	0.13	20.00	4.30	2.40
VLIW	10.20	18.05	2.47	10.20	54.80	7.20
VLIW+PiCoGa	13.20	1.33	0.19	29.40	7.10	1.10

Im Vergleich zum ARM9 fällt der Gewinn aufgrund der hohen Energieeffizienz dieses Prozessors geringer aus, dennoch ist eine deutliche Verbesserung zu erkennen.

Bezüglich des Medianfilters fällt auf, dass die Performance der Prozessor-eFPGA-Lösungen etwa um den Faktor zehn höher liegt als die der reinen Software-Implementierungen ohne eFPGA-Beschleunigung, zudem sind beide Geschwindigkeiten höher als bei der PiCoGA-Architektur. Bezüglich der Energieeffizienz wird eine Verbesserung um den Faktor 4, bzw. 1.5 gegenüber der Software-basierten Lösungen erzielt. Der im Vergleich zum DES-Algorithmus geringere Gewinn wird in [vKN+06] mit dem höheren Kommunikationsaufwand bei der Medianfilterung erklärt, da der Algorithmus ein fortlaufendes Lesen und Schreiben neuer Daten erfordert. Dieser Overhead lässt sich durch einen direkten Speicherzugriff des eFPGAs verringern.

Für weitere Informationen sei auf [vKN+06] verwiesen.

3.2 Entwurfsraumexploration für grobgranulare Architekturen

3.2.1 UC Irvine: Grobgranulare Arrays

Ein Ansatz zur Analyse grobgranularer Prozessorarchitekturen, der ohne ein physikalisches Modell auskommt, ist an der Universität Kalifornien in Irvine entwickelt worden (siehe [BGDN03]). Es wird der Einfluss unterschiedlicher Basiszell-Konfigurationen und Array-Dimensionen auf die Performance eines rekonfigurierbaren Feldes untersucht. Basis ist der in Abbildung 3.6a gezeigte Aufbau. Alle Prozessorzellen einer Zeile, bzw. Spalte sind miteinander verbunden. Die Anzahl der Verbindungsleitungen kann über Parameter

festgelegt werden. Weiterhin parametrisierbar ist die Konfiguration der zugrunde liegenden Basiszellstruktur. Abbildung 3.6 zeigt eine mögliche Konfiguration mit einem Register-File und zwei ALUs. Der Benutzer hat die Möglichkeit, die Anzahl der Funktionseinheiten zu variieren, sowie deren Funktionalität und Latenzzeit.

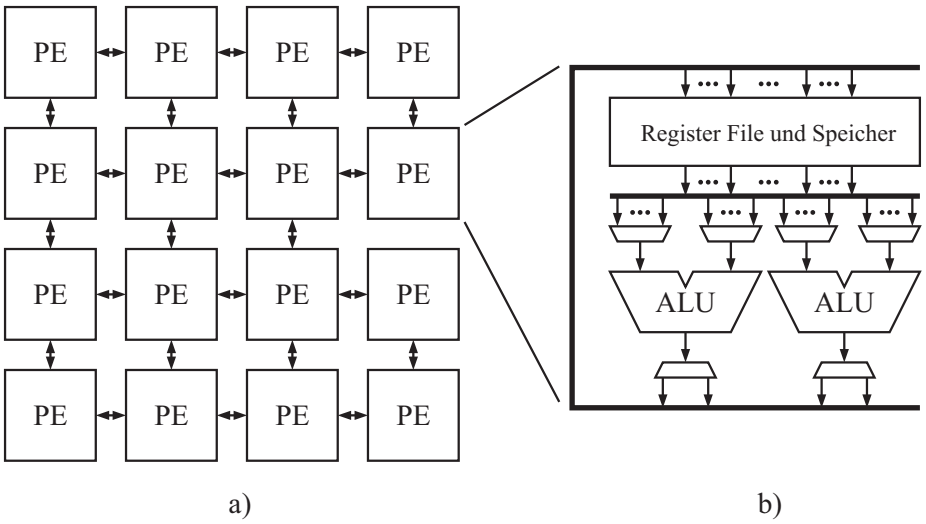


Abbildung 3.6: a) Globales Architekturmodell aus [BGDN03], b) Beispiel für eine mögliche Datenpfad-Implementierung

Das funktionale Architekturmodell ist in ein Softwaretool integriert worden, welches über eine Benutzerschnittstelle zunächst die Parameter einliest. Weiterhin enthält das Software-Framework einen einfachen C-Compiler. Mit dessen Hilfe und einem Scheduling- und Mapping-Algorithmus lassen sich Algorithmen automatisiert auf das Feld abbilden.

Es sind Experimente mit unterschiedlichen Feldgrößen, PE-Konfigurationen und Verbindungsstrukturen, sowie verschiedenen Latenzzeiten der Basiszellen und Busse durchgeführt worden. Die Resultate zeigen eine Verbesserung der in Anzahl der Taktzyklen gemessenen Performance, wenn die Basiszellen entweder mehr Funktionseinheiten enthalten, oder die Anzahl der Verbindungsleitungen zwischen den PEs erhöht wurde, was mit einem höheren Freiheitsgrad beim Scheduling und Mapping begründet wird. Für eine umfangreichere Darstellung der Ergebnisse sei auf [BGDN03] verwiesen.

Das Verfahren liefert Aussagen über die Performance einer Architektur bezogen auf den CPI-Wert (*Cycles per Instruction*). Eine Untersuchung des absoluten Flächenbedarfs, des Energieverbrauches und der maximalen Taktfrequenz ist mit dem Verfahren nicht möglich, da es, wie eingangs erwähnt, nicht auf einem physikalischen Modell beruht.

3.2.2 Northwestern University: Totem-Projekt

Das an der Northwestern University und der University of Washington entwickelte Totem-Projekt (siehe [CH01]) befasst sich mit der Untersuchung anwendungsspezifischer grobgranularer Architekturen. Grundlage der Analysen ist die Basiszellstruktur des RaPiD-Projektes, das bereits in 2.4.1 beschrieben wurde. Die Untersuchungen im Totem-Projekt werden über einen kundenseitig programmierbaren *Array-Generator* automatisiert. Dieses Tool liest RaPiD-Netzlisten für verschiedene Algorithmen ein, und erzeugt daraus automatisch eine für das dadurch definierte Anwendungsspektrum optimierte RaPiD-Basiszelle.

In [CH01] sind die Verbesserungen bezüglich des Flächenbedarfs dokumentiert, die sich durch die automatische Optimierung ergeben. Die Ergebnisse beziehen sich auf eine für FIR- und Medianfilterung, sowie auf Matrixmultiplikationen optimierte RaPiD-Basiszellstruktur. Demnach lassen sich, in Abhängigkeit vom jeweils verwendeten Algorithmus bei der Optimierung, Architekturen erhalten, die teilweise um den Faktor 2 kleiner sind als die Original-RaPiD-Struktur, jedoch alle geforderten Algorithmen ausführen können.

3.2.3 University of British Columbia: Verbindungsstrukturen

Ein Ansatz, der sich mit der Exploration und Optimierung der Verbindungsstruktur einer grobgranularen Architektur beschäftigt, wurde an der University of British Columbia und am IMEC in Leuven entwickelt und veröffentlicht (siehe [WKMV04]). Grundlage der Untersuchungen ist das in Abbildung 3.7a gezeigte 4×4-Basiszell-Array. Jede Basiszelle (*CFU, Configurable Functional Unit*) kann eine oder mehrere arithmetisch-logische Operationen pro Taktzyklus auf 32-Bit-Daten ausführen. Die Struktur einer CFU ist in Abbildung 3.7b dargestellt. Die CFUs der dritten Spalte des Feldes können neben den ALU-Operationen zusätzlich Multiplikationen berechnen. Das Feld ist weiterhin an einen Standard-VLIW-Prozessor angekoppelt.

Ziel der Arbeit ist das Finden einer optimalen Verbindungsstruktur für das Feld. Zu wenig Verbindungsmöglichkeiten zwischen den PEs geben dem Compiler und Mapper laut [WKMV04] zu wenig Freiheitsgrade beim Abbilden der Algorithmen, während zu viele Verbindungsleitungen eine flächenineffiziente Architektur zur Folge haben kann. Es werden Verbindungsstrukturen variiert zwischen einem Netzwerk aus Punkt-zu-Punkt-Verbindungen bis zum MorphoSys-ähnlichen Netzwerk (s.a. Abschnitt 2.4, Seite 28), bei dem alle PEs einer Zeile oder Spalte kommunizieren können. Es zeigt sich, dass ab einer bestimmten Flexibilität einer Basiszelle bei der Auswahl der Verbindungen, die Rechenleistungsdichte (AT-Produkt, s.a. Abschnitt 7.1.4) erheblich besser als bei einem vollvermaschten Netzwerk werden kann.

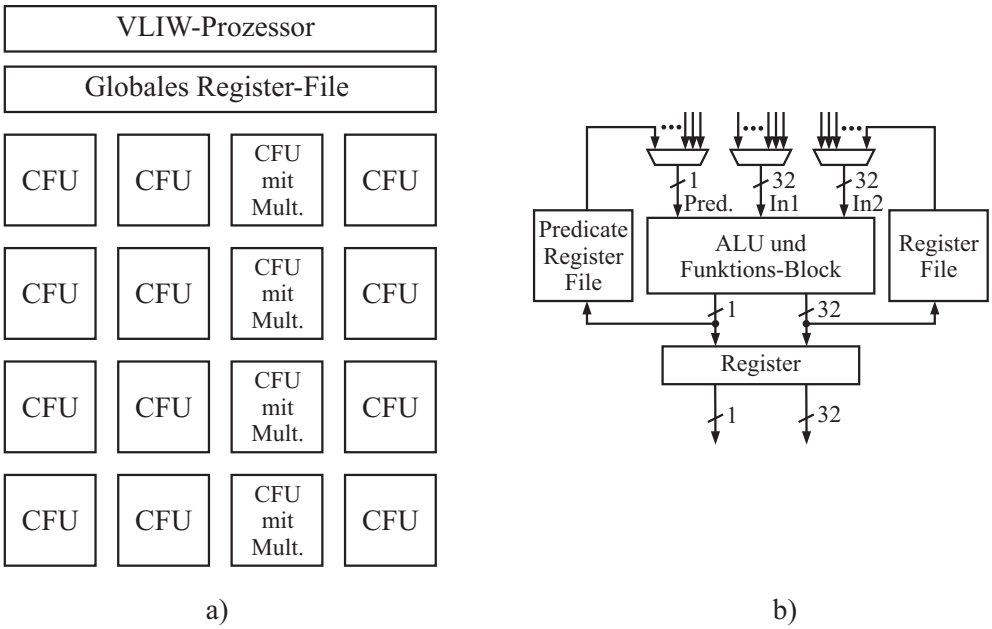


Abbildung 3.7: Architekturmodell für die in [WKMV04] vorgestellten Analysen

Der Energieverbrauch wird in der in [WKMV04] veröffentlichten Arbeit nicht untersucht. Zwar werden absolute Werte zur Fläche und zum AT-Produkt angegeben, die Ergebnisse werden jedoch nicht mit anderen Implementierungsformen verglichen.

3.3 Weitere Arbeiten

Weitere bekannte Ansätze zur Entwurfsraumexploration rekonfigurierbarer Architekturen sind beispielsweise das KressArray-Projekt (siehe z.B. [HHHN00] oder [Nag01]), sowie der in [BGP03] präsentierte analytische Ansatz.

4 Modellierung grobgranularer Architekturen

Rekonfigurierbare Architekturen grober Granularität besitzen, wie in Kapitel 2 dargestellt, Vorteile gegenüber feingranularen Strukturen bezüglich Rechengeschwindigkeit, Flächen- und Energieeffizienz, sofern ein Algorithmus ausgeführt wird, für dessen Klasse die Architektur entworfen wurde. Diese Chancen grobgranularer Architekturen sind absehbar, da anwendungsspezifische Datenpfade zum Einsatz kommen, welche beispielsweise dedizierte Multiplizierer, ALUs oder Komparatoren enthalten können. Die Effizienz feingranularer Strukturen ist bei Algorithmen, die sich aus den angesprochenen Operatoren zusammensetzen, naturgemäß niedriger, da beispielsweise ein Multiplizierer aus mehreren kleinen Logikzellen zusammengesetzt werden muss. An dieser Stelle ergeben sich zwei wichtige Fragen, die in der vorhandenen Literatur bislang nur unzureichend geklärt wurden, und denen in der vorliegenden Arbeit nachgegangen wird. Eine der Fragen beschäftigt sich mit dem Granularitätsgrad einer Architektur, und damit, welche Auswirkungen eine Variation der Basiszell-Komplexität auf Datendurchsatz, Flächen- und Energieeffizienz und die Flexibilität besitzt, bzw. welche Granularität für eine bestimmte Anwendungsklasse jeweils optimal ist. Die andere Frage beschäftigt sich mit den Chancen grobgranularer Architekturen im Vergleich mit anderen Implementierungsformen wie FPGAs, Standardzell-Entwürfen oder DSPs.

In diesem Kapitel wird ein Verfahren vorgestellt, mit welchem die beiden dargestellten Fragestellungen in Bezug auf grobgranulare rekonfigurierbare Architekturen untersucht werden können. Dieses Verfahren basiert auf einem allgemeinen parametrisierbaren Modell, welches eine Analyse rekonfigurierbarer Architekturen mit unterschiedlichem Granularitätsgrad der Basiszellen ermöglicht. Da die Parameter des Modells mit physikalischen Simulationsergebnissen abgeglichen sind, ist es weiterhin möglich, absolute Werte für Fläche, Datendurchsatz und Verlustleistung abzuschätzen, und somit die Effizienz der betrachteten Architekturen mit anderen Implementierungsformen zu vergleichen.

Nach einer kurzen Übersicht über den angewandten Analyse-Ablauf wird das zu Grunde liegende physikalische Modell vorgestellt, welches aus Basiszellen und einer globalen Verbindungs- und Speicherstruktur besteht. Im Anschluss daran wird die Modellierung von Flächenbedarf, Datendurchsatzrate und Energieeffizienz beschrieben.

4.1 Übersicht zum Analyse-Ablauf

Es soll zunächst die allgemeine Vorgehensweise bei der Untersuchung der VLSI-Eigenschaften der rekonfigurierbaren Modellarchitekturen dargestellt werden. Der genaue Aufbau des physikalischen Modells wird bei dieser Darstellung noch nicht betrachtet.

Abbildung 4.1 zeigt einen schematischen Überblick über den Analyse-Ablauf. Es kommen eine Reihe von Software-Tools zum Einsatz, welche jeweils unterschiedlichen VLSI-Entwurfsebenen zuzuordnen sind. Bei den Tools handelt es sich, wie nachfolgend beschrieben, um eine Sammlung MATLAB-basierter Werkzeuge zur Architektur-Analyse, welche im Rahmen dieser Arbeit entstanden sind, sowie um kommerzielle Entwurfs- und Simulations-Programme. Die Sammlung der MATLAB-Tools soll in dieser Arbeit unter dem Namen "Array Analyzer" (ARA) zusammengefasst werden. Die Tools bieten die Möglichkeit, Algorithmen auf die rekonfigurierbaren Modelle abzubilden, und anschließend in Bezug auf Zeitverhalten, Flächenbedarf und Energieverbrauch zu untersuchen. Eine grafische Benutzeroberfläche ermöglicht außerdem eine visuelle Darstellung des Routings.

Ausgangspunkt einer jeden Untersuchung ist stets die in der Einleitung dieses Kapitels bereits gestellte Frage nach dem optimalen Granularitätsgrad einer Architektur für eine bestimmte Algorithmenklasse, sowie der Wunsch, grobgranulare Architekturen mit anderen Implementierungsformen zu vergleichen. Vor Beginn der Analyse ist daher zunächst die Klasse der Algorithmen festzulegen. Dabei sind auch hybride Strukturen denkbar, mit der Möglichkeit mehrere Klassen zu implementieren, wie beispielsweise lineare und nichtlineare Filteroperationen. Daran anschließend sind diejenigen Algorithmen innerhalb der gewählten Klasse auszuwählen, für welche die rekonfigurierbare Struktur entworfen und eingesetzt werden soll.

Nach Festlegung der Anwendungsklasse beginnt der Systementwurf der Architektur, vgl. Abbildung 4.1. Bei diesem Schritt werden Basiszellen entworfen, und in Bezug auf die ausgewählten Algorithmen optimiert. Durch den Entwurf verschiedener Basiszellen mit jeweils unterschiedlicher Komplexität lassen sich nach der Analyse Aussagen über die Auswirkungen der Variation des Granularitätsgrades treffen. Desweiteren sind das Verbindungsnetzwerk und die Speicherstruktur, in welches die Prozessorelemente eingebettet sind, so zu parametrisieren, dass alle in der Spezifikation festgelegten Algorithmen auf die Architektur abgebildet werden können. Dies betrifft beispielsweise die korrekte Wahl der Speichergröße und der Routingressourcen. Auf die jeweilige Bedeutung wird in Abschnitt 4.2 ausführlich eingegangen. Die gewählten Parameter werden dem MATLAB-Toolset in Form einer Parameter-Datei zur Verfügung gestellt.

Da es ein Ziel dieser Arbeit ist, die VLSI-Eigenschaften der Architekturen möglichst genau abzuschätzen, werden im nächsten Schritt die vorher lediglich auf Systemebene funktional definierten Prozessorzellen in VHDL implementiert. Anschließend werden die PEs mit Hilfe des Design Compilers von Synopsys synthetisiert. Zieltechnologie für die Synthe-

se der PEs ist in dieser Arbeit stets eine 0.18μm-Standardzellenbibliothek von Chartered Semiconductors [Cha04]. Das Syntheseergebnis erlaubt eine Abschätzung der zur Implementierung erforderlichen Chipfläche, welche vom Design Compiler in Gatteräquivalenten angegeben wird. Ein Gatteräquivalent entspricht der Fläche eines NAND-Gatters mit zwei Eingängen der jeweiligen Prozesstechnologie. Um eine Abschätzung der absoluten Fläche zu erhalten, ist daher die vom Synthesetool errechnete Anzahl der Gatteräquivalente in μm^2 umzurechnen. Der erforderliche Umrechnungsfaktor wurde durch Layout und Messung mit Hilfe der ICFlow EDA-Software von Mentor Graphics bestimmt. Wegen des NDAs mit Chartered wird dieser Faktor hier nicht als absoluter Wert angegeben.

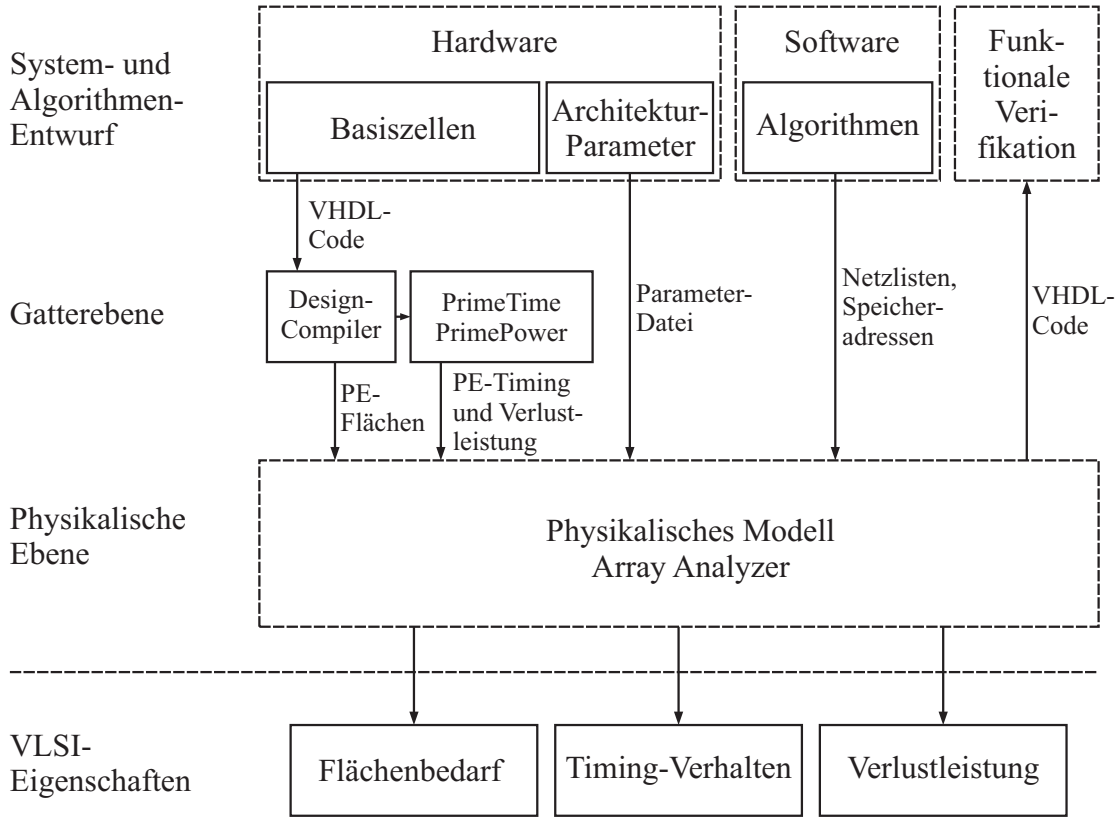


Abbildung 4.1: Schematische Darstellung der Vorgehensweise bei der Untersuchung der rekonfigurierbaren Modellarchitekturen

Die Analyse des Timingverhaltens der Prozessorelemente erfolgt ebenfalls mit Hilfe von Simulationen. Das Synopsys-Tool PrimeTime erlaubt eine statische Timinganalyse der synthetisierten Gatternetzlisten unter Verwendung der Timingwerte der Standardzellenbibliothek. Bei diesem Vorgang können neben dem längsten Pfad der Schaltung auch die Verzögerungen von den Eingangspins bis zum ersten nichtkombinatorischen Element be-

stimmt werden, oder die Signalverzögerung vom letzten kombinatorischen Element bis zu den Ausgangspins. Die Ergebnisse der Analysen sollen laut Angaben des Tool-Herstellers innerhalb von 5% der Genauigkeit von SPICE-Simulationen liegen (siehe [Syn03a]).

Zur Bestimmung der maximalen Sampleraten sind die Algorithmen auf die durch die Prozessorzellen und die Parameter festgelegten Architekturen abzubilden. Dies geschieht ebenfalls mit Hilfe des Array Analyzers. Das Routing kann von Hand unter der grafischen Oberfläche des Programms vorgenommen werden. Ebenfalls in diesem Software-Entwurfsschritt festzulegen sind die Konfigurationen der Prozessorelemente, sowie die Adressen der Ein- und Ausgangsdaten in jedem Takt. Da bei grobgranularen Architekturen in der Regel erheblich weniger Prozessorzellen zum Einsatz kommen als beispielsweise bei FPGAs, ist der manuelle Entwurfsvorgang mit vertretbarem Aufwand durchführbar.

Der Energieverbrauch der rekonfigurierbaren Modellarchitekturen wurde mit einem bibliotheksbasierten Verfahren abgeschätzt, indem alle relevanten Blöcke zunächst in Bezug auf die Leistungsaufnahme charakterisiert wurden. Die Daten sind in den Array Analyzer integriert worden, welcher unter Berücksichtigung der jeweiligen Schaltaktivität den Energieverbrauch berechnet.

Die Analyse der VLSI-Eigenschaften der Gesamtarchitektur erfolgt mit dem bereits erwähnten Array Analyzer. Die Ergebnisse der Hardware-Simulationen der Prozessorzellen, also Fläche und simuliertes Timingverhalten, sind in entsprechenden Parameterdateien zur Verfügung zu stellen, ebenso die beim manuellen Mapping entstandenen Netzlisten. Eine weitere Parameterdatei enthält Informationen zu Speichergröße und Routingstruktur der Architektur, sowie detaillierte physikalische Parameter, die zur Berechnung von Fläche, Timing und Energieverbrauch benötigt werden, wie beispielsweise die Flächen- und äquivalenten Widerstands- und Kapazitätswerte von Transistoren und Bustreibern. Letztere Parameter müssen aber in der Regel nur einmal festgelegt werden, und bleiben dann für alle Simulationen konstant. Eine ausführlichere Beschreibung der physikalischen Parameter und die Vorgehensweise zur Bestimmung der Werte erfolgt in den nächsten Abschnitten.

Der Array Analyzer bestimmt mit Hilfe der Parametersätze und des Routings Abschätzungen zu Flächenbedarf, Timingverhalten und Verlustleistung, bezogen auf den jeweils abgebildeten Algorithmus. Es ist weiterhin möglich, zur funktionalen Verifikation ein VHDL-Modell der Gesamtarchitektur generieren zu lassen.

4.2 Allgemeines Architektur-Modell

In dieser Arbeit wird ein in weiten Bereichen parametrisierbares Hardware-Modell verwendet, um die Eigenschaften grobgranularer rekonfigurierbarer Architekturen zu untersuchen. Die allgemeine Struktur des Modells wird in diesem Abschnitt vorgestellt. Nach einer Darstellung der Anforderungen an ein solches Modell unter den oben erwähnten Randbedingungen in Abschnitt 4.2.1 erfolgt zunächst eine Darstellung des globalen Auf-

baus (Abschnitt 4.2.2). In den darauf folgenden Unterabschnitten wird im Wesentlichen die detaillierte Struktur des Verbindungsnetzwerkes erläutert. Nach einer Darstellung der Eigenschaften der verwendeten VLSI-Elemente in Abschnitt 4.3.1 werden die aus diesen Elementen zusammengesetzten Architektur-Blöcke beschrieben (Abschnitte 4.3.2 - 4.4). Die in dieser Arbeit für ausgewählte Klassen von Algorithmen entworfenen und untersuchten Basiszellen mit unterschiedlichem Granularitätsgrad werden in Kapitel 5 beschrieben.

4.2.1 Anforderungen an ein allgemeines Modell

Bei einer modellbasierten Analyse rekonfigurierbarer Architekturen sind verschiedene Anforderungen an das verwendete Modell zu stellen, um eine möglichst hohe Aussagequalität zu erreichen. Bevor in den folgenden Abschnitten auf die Details des verwendeten Modells eingegangen wird, sollen daher zunächst die Ausgangsbedingungen dargestellt werden, anhand derer das Modell entwickelt wurde:

- *Parametrisierbarkeit.* Bei einer Entwurfsraumexploration wird untersucht, welche Auswirkungen die Architektur bestimmenden Parameter auf eine oder mehrere vorher festgelegte Kostenfunktionen besitzen. Ziel ist die Angabe von Minima der Kostenfunktionen innerhalb des von den Architekturparametern aufgespannten multidimensionalen Entwurfsraumes. Aufgrund der Vielzahl der Parameter ist es nicht möglich, für jede Kombination die Hardware neu zu entwerfen, und anschließend eine Simulation durchzuführen. Es ist daher erforderlich, die Struktur der zu analysierenden Hardware durch entsprechende Wahl globaler Parameter, z.B. der Anzahl der Basiszellen oder der Verbindungsleitungen, festlegen zu können, und auf diese Weise eine weitgehend automatisierte Generierung von Simulationsmodellen zu erreichen.

- *Erweiterbarkeit.* Um das Modell mit möglichst geringem Aufwand an Randbedingungen anpassen zu können, die vor der Modellerstellung noch nicht absehbar waren, ist auf leichte Erweiterbarkeit zu achten. Beispielsweise sollen neu entworfene Prozessorzellen ohne Redesign der Verbindungsstruktur in die Architektur integriert werden können.

Ein Problem bei der Analyse grobgranularer rekonfigurierbarer Architekturen ist der durch enorm viele Implementierungsparameter aufgespannte Entwurfsraum. In Kapitel 2 wurden einige ausgewählte grobgranulare Architekturen vorgestellt, die in den vergangenen Jahren in Forschungs- und Industrieprojekten entwickelt worden sind. Projekte wie MATRIX, MorphoSys, CHESS, DReAM, PipeRench, Rapid oder die XPP-Plattform von PACT sind zwar allesamt Beispiele für grobgranulare Architekturen, unterscheiden sich

jedoch in ihrem Aufbau fundamental voneinander. Bei Betrachtung der Vielzahl von Implementierungsmöglichkeiten wird schnell klar, dass der Parameterraum eines Analyse-Modells nicht alle erwähnten und denkbaren Architekturen enthalten kann. Es ist daher erforderlich, sich auf ein Modell zu beschränken, welches die allgemeinen Eigenschaften einer grobgranularen Architektur so gut wie möglich widerspiegelt, und gleichzeitig den oben dargestellten Anforderungen an die Parametrisierbarkeit und die Erweiterbarkeit Rechnung trägt. Ein derartiges Modell wird in den folgenden Abschnitten vorgestellt.

4.2.2 Globale Struktur

Ein großer Teil der existierenden rekonfigurierbaren Architekturen besteht im Kern aus einem zweidimensionalen Feld von Logikzellen, bzw. grobgranularen Rechenelementen. Die Basiszellen dieser Strukturen können durch ein den jeweiligen Anforderungen angepasstes Verbindungsnetzwerk miteinander kommunizieren. Dies ist der Grundaufbau der meisten kommerziellen FPGAs, aber auch die in Kapitel 2 exemplarisch vorgestellten grobgranularen Architekturen besitzen eine zweidimensionale Array-Struktur mit entsprechendem Verbindungsnetzwerk. Ausnahmen sind lediglich PipeRench und Rapid, welche eine eindimensionale Anordnung der PEs aufweisen. Für die Entwicklung eines allgemeinen Modells liegt es daher nahe, ebenfalls eine zweidimensionale Feldstruktur zu wählen, deren Dimensionen mit globalen Parametern eingestellt werden kann. Mit einem solchen Modell lassen sich die allgemeinen Eigenschaften einer großen Klasse grobgranular rekonfigurierbarer Architekturen nachbilden. Die dabei gefundenen Aussagen zum tendenziellen Verhalten dieser Strukturen lassen sich in Bezug auf die Klasse der grobgranularen Architekturen, die im Kern aus einer 2D-Feld-Struktur bestehen, verallgemeinern.

Abbildung 4.2 zeigt die globale Struktur des allgemeinen parametrisierbaren Architektur-Modells. Dargestellt ist ein 4×4-PE-Feld. Durch Setzen der entsprechenden Parameter in der Architektur-Beschreibungsdatei lassen sich theoretisch beliebige Feldgrößen realisieren, so unter anderem auch eindimensionale Felder, wie sie z.B. bei PipeRench und Rapid zum Einsatz kommen (siehe [GSM+99],[ECF96]).

Die Prozessorelemente sind untereinander durch ein Verbindungsnetzwerk verbunden, welches zwei Hierarchiestufen besitzt. Jede Basiszelle kann mit seinem jeweils nächsten Nachbarn kommunizieren, welcher sich ober- oder unterhalb, bzw. links oder rechts vom PE befindet. Weiterhin ist es möglich, Daten an einem oder mehreren Prozessorelementen vorbei zu routen, wie dies auch bei FPGAs implementiert wird. Zu diesem Zweck können die Ausgänge der PEs so konfiguriert werden, dass vertikale oder horizontale globale Verbindungsleitungen getrieben werden. Die Bustreiber sind innerhalb eines strukturalen Blockes angeordnet, der in dieser Arbeit als Verbindungselement bezeichnet wird. An jedem Kreuzungspunkt zwischen zwei globalen Verbindungsleitungen befindet sich eine Switchbox, mit deren Hilfe Daten auf die angrenzenden Verbindungskanäle verteilt werden können. Eine solche Switchbox kann auch verwendet werden, um einen Kanal zu unterbrechen,

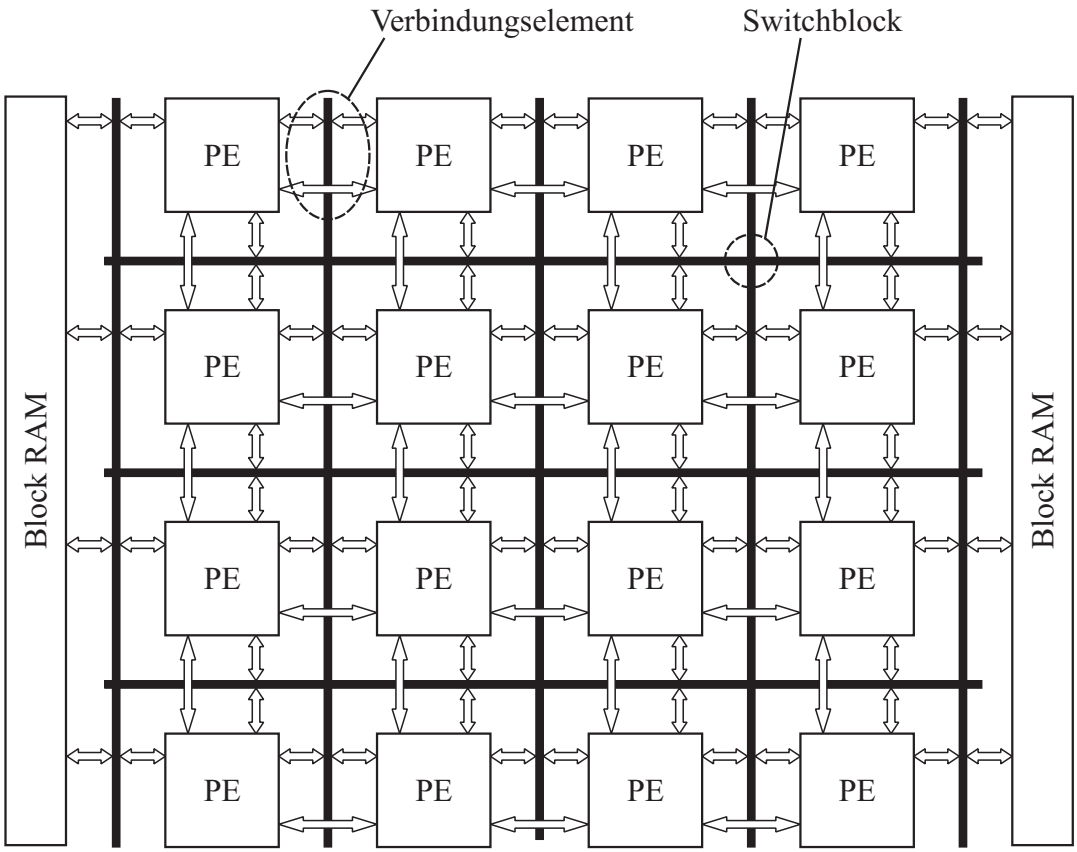


Abbildung 4.2: Globale Struktur des allgemeinen Architektur-Modells

und so mehrere segmentierte Verbindungen innerhalb derselben Zeile oder Spalte zu ermöglichen. Neben den oben erwähnten lokalen Verbindungen zwischen benachbarten PEs bilden die globalen Routing-Kanäle die zweite Hierarchiestufe der Verbindungsstruktur. Die Routing-Konfiguration eines bestimmten Algorithmus wird in lokalen SRAM-Zellen gespeichert. Die Hardwareimplementierung der Verbindungs- und Switchblöcke wird in den Abschnitten 4.3.2 und 4.3.3 näher beschrieben.

Da die Effizienz einer rekonfigurierbaren Hardwarestruktur nicht nur von den Prozessorzellen und dem Verbindungsnetzwerk abhängt, sondern auch in hohem Maße von der Fähigkeit, Daten effizient zwischenzuspeichern, besitzt das Modell zwei Speicher-Ebenen. Jedes Prozessorelement beinhaltet einen lokalen Speicher zur Speicherung von bei Berechnungen eventuell auftretenden Zwischenergebnissen. Weiterhin ist das rekonfigurierbare Feld an zwei Block-RAMs angekoppelt, welche sich an der linken und rechten Seite des Arrays befinden. Die Block-RAMs dienen ebenfalls zur Speicherung von Zwischenergebnissen, und weiterhin zur Speicherung von Ein- und Ausgangswerten. Zur Vermeidung

von Speicherzugriffskonflikten und Wartezyklen ist jedes Block-RAM unterteilt in eine Anzahl von unabhängigen Adressbereichen. Eine ähnliche Anordnung größerer dedizierter Speicherblöcke findet sich beispielsweise in FPGAs der Spartan-II-Reihe von Xilinx und in der XPP-Plattform von PACT [PAC05]. Die genaue Struktur der Block-RAM-Speicher wird in Abschnitt 4.4 dargestellt.

4.3 Verbindungsstruktur

Nach einer kurzen Beschreibung der für diese Arbeit relevanten Eigenschaften einiger VLSI-Schaltelemente und Speicherzellen, wird in diesem Abschnitt die Verbindungsstruktur beschrieben, in welche die Prozessorelemente eingebettet sind.

4.3.1 VLSI-Schaltelemente und Speicherzellen

Die im vorigen Abschnitt erwähnten Verbindungs- und Switch-Blöcke sind im Kern aus einfachen VLSI-Elementen wie Pass-Transistoren, Bustreibern und SRAM-Zellen aufgebaut. Die Extraktion der physikalischen Eigenschaften der rekonfigurierbaren Architektur (siehe Kapitel 6) basiert unter anderem auf einer Kenntnis des Verhaltens dieser Bauelemente. Bevor daher auf den genauen Aufbau der Verbindungs- und Switchblöcke, sowie des Block-RAM-Speichers eingegangen wird, ist es sinnvoll, zunächst kurz auf die Eigenschaften der VLSI-Bauelemente einzugehen, aus denen diese Architekturblöcke in dem in dieser Arbeit verwendeten Modell zusammengesetzt sind.

Pass-Transistoren und Transmission-Gates

Eine alternative Realisierungsform zur komplementären MOS-Logik ist die sogenannte Pass-Transistor-Logik (PTL, s.a. [WH05], [NPC02], [Rab93]). Hier werden komplexe Gatter nicht durch Verschaltung komplementärer nMOS- und pMOS-Netzwerke realisiert, sondern mit Hilfe bidirektionaler Übertragungsglieder wie Pass-Transistoren oder Transmission-Gates. Vorteilhaft gegenüber CMOS-Logik ist eine höhere Flächeneffizienz, da weniger Transistoren zur Realisierung eines Gatters benötigt werden. So lässt sich beispielsweise ein 2:1-Multiplexer mit nur zwei Pass-Transistoren aufbauen. Ein Pass-Transistor wird in diesen Schaltungen als Transfer-Element betrieben. Nachteilig gegenüber CMOS-Schaltungen ist die fehlende Signalpegel-Regenerierung, so dass bei PTL-Logik grundsätzlich Buffer zur Wiederherstellung des Pegels eingesetzt werden müssen.

Abbildung 4.3a zeigt einen nMOS-Pass-Transistor mit den zugehörigen Eingangs- und Ausgangssignalen A und B, sowie dem am Gate des Transistors anliegenden Kontrollsignal C. Wird am Gate die Betriebsspannung V_{DD} angelegt, schaltet der Transistor in den

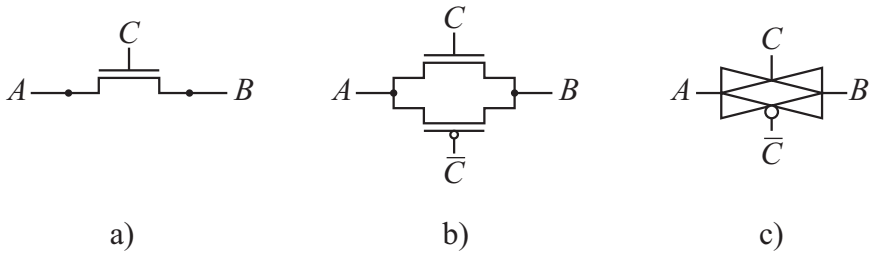


Abbildung 4.3: a) Pass-Transistor als bidirektionales Übertragungsglied, b) Transmission-Gate aus zwei komplementären Pass-Transistoren, c) Schaltsymbol eines Transmission-Gates

leitenden Zustand, so dass ein Signal von A nach B oder in umgekehrter Richtung übertragen werden kann. Wird C auf Masse gelegt, werden die beiden Eingänge hochohmig voneinander getrennt.

Beim Aufbau von PTL-Schaltungen ist das Übertragungsverhalten der Pass-Transistoren zu beachten. Auf eine detaillierte Erklärung der Effekte mit Hilfe physikalischer Transistormodelle wird hier verzichtet, vielmehr soll kurz auf die für diese Arbeit relevanten elektrischen Eigenschaften von Pass-Transistor-Schaltungen eingegangen werden. Dies ist unter anderem zur Beschreibung des angewandten analytischen Verzögerungsmodells für die Verbindungsstruktur von Bedeutung, welches in Abschnitt 6.1.2 vorgestellt wird.

Ein Simulationsaufbau zur Bestimmung des Schaltverhaltens eines nMOS-Pass-Transistors ist in Abbildung 4.4a dargestellt. In dieser Schaltung wird eine am Eingang anliegende Spannung U_E über einen einzelnen nMOS-Transistor übertragen. Mit der am Ausgang des Transistors anliegenden Spannung $U_{G,PT}$ wird der Eingang einer Inverterstufe getrieben, welche zur Wiederherstellung des Signalpegels auf die Betriebsspannung V_{DD} dient.

Die Zeitverläufe der in Abbildung 4.4 gezeigten Spannungen U_E, $U_{G,PT}$ und $U_{A,PT}$ sind in Abbildung 4.5 gezeigt. Angenommen wird für die Eingangsspannung U_E ein Verlauf mit endlicher Flankensteilheit.

Erkennbar erreicht die durch den nMOS-Transistor übertragene Spannung $U_{G,PT}$ nicht mehr das Niveau der Eingangsspannung, sondern liegt deutlich darunter. Dennoch ist die nachgeschaltete Inverterstufe in der Lage, nach einer bestimmten Verzögerungszeit die korrekte Ausgangsspannung von $0V$ zu liefern. Die logische Funktion der Schaltung ist also korrekt, allerdings hat die zu niedrige Spannung $U_{G,PT}$ am Eingang des Inverters zur Folge, dass der pMOS-Transistor einen kleinen Strom $I_{D,PT}$ durchlässt (Abbildung 4.5 unten), welcher auch nach dem Umschaltvorgang eine unnötig hohe statische Verlustleistung verursacht.

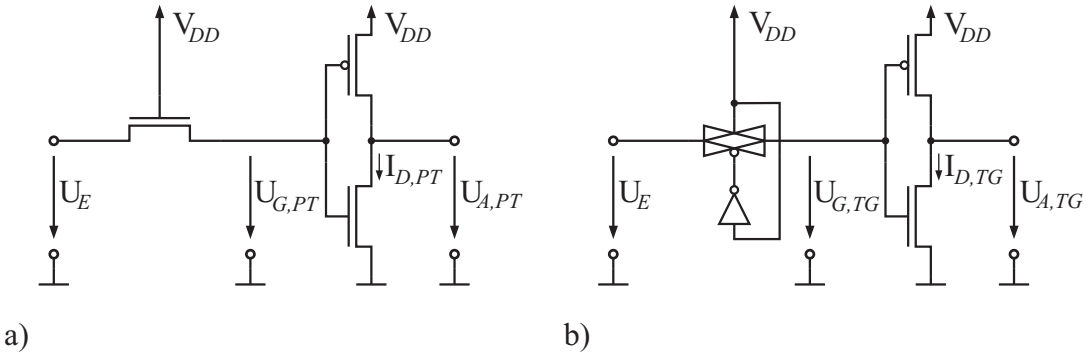


Abbildung 4.4: a) Bestimmung des Schaltverhaltens eines einzelnen nMOS-Pass-Transistors, der eine Inverterstufe treibt, b) die gleiche Schaltung mit einem Transmission-Gate statt des Pass-Transistors

Zur Lösung dieses Problems existieren verschiedene Ansätze. In älteren Technologieprozessen kann das Gate des nMOS-Pass-Transistors mit einer Spannung $U_G > V_{DD}$ angesteuert werden, wodurch die übertragene Spannung $U_{G,PT}$ ausreichend groß ist, um keinen Verluststrom durch den Inverter zu verursachen ("Gate Boosting", siehe z.B. [BRM99]). Eine andere Möglichkeit liegt in der Implementierung einer rückgekoppelten Pegelanhebungsschaltung (siehe z.B. [Rab93]). In dieser Arbeit werden statt einzelner nMOS-Pass-Transistoren Transmission-Gates angenommen. Diese bidirektionalen Übertragungsglieder bestehen aus einer Parallelschaltung eines nMOS- und eines pMOS-Transistors (siehe Abbildung 4.3b), und besitzen bessere Übertragungseigenschaften als einzelne nMOS-Transistoren. Wird statt des nMOS-Transistors in Abbildung 4.4a ein Transmission-Gate eingesetzt (Abbildung 4.4b), ergibt sich der in Abbildung 4.5 gezeigte Verlauf für die Eingangsspannung $U_{G,TG}$ am Inverter. Erkennbar wird der volle Spannungshub auf die Betriebsspannung erreicht, so dass im statischen Fall kein Verluststrom durch den Inverter fließt.

Buffer und Tristate-Treiber

Sehr häufig kommt es vor, dass der Ausgang einer Schaltung mit einer großen kapazitiven Last verbunden ist, wie dies bei den hier betrachteten Architekturen beispielsweise bei den Busleitungen der Fall ist. Zum Umladen der Kapazitäten sind entsprechende Treiber (Buffer) erforderlich. Diese werden sehr häufig aus einer Kaskade von Inverterstufen aufgebaut, bei denen jede nachfolgende Stufe um einen Faktor a größer ausgelegt wird als die davor liegende (s.a. [WH05]). Ein typischer Wertebereich für a liegt bei $4 \leq a \leq 5$. Abbildung 4.6a zeigt eine zweistufe Treiberschaltung und das dazugehörige Schaltsymbol. Die Zahlen geben das Verhältnis W/L der Gate-Dimensionen der Transistoren an.

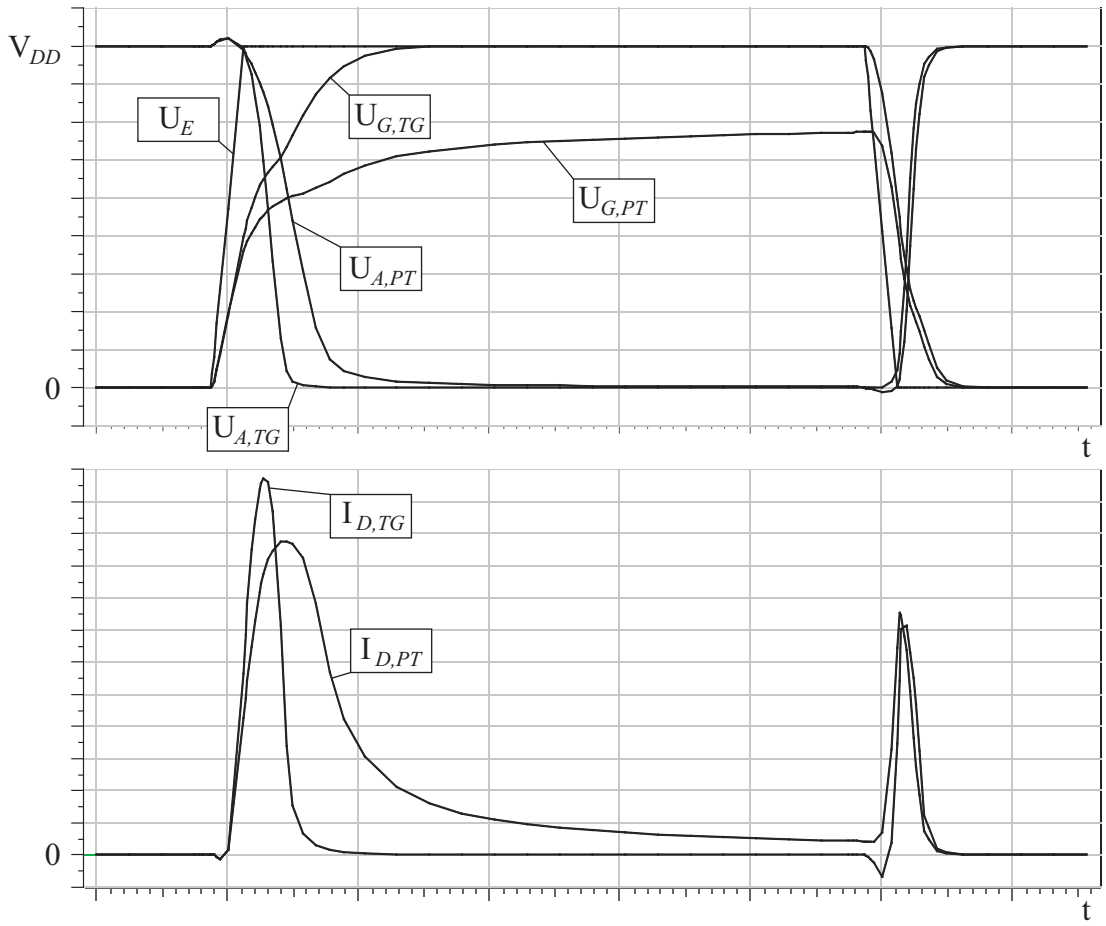


Abbildung 4.5: Oben: Verläufe der Ausgangsspannungen $U_{A,PT}$ und $U_{A,TG}$, sowie der Spannungen $U_{G,PT}$ und $U_{G,TG}$ an den beiden Gates des Inverters für die Pass-Transistor- und Transmission-Gate-Schaltungen aus Abbildung 4.4 in Abhängigkeit von der Eingangsspannung U_E, unten: Verlauf der Drain-Ströme $I_{D,PT}$ und $I_{D,TG}$ der beiden Schaltungen

Die Aufgabe von Bustreibern ist, den Ausgang einer Schaltung entweder mit der entsprechenden Busleitung zu verbinden, oder den Ausgang hochohmig von der Leitung zu trennen. Hierfür werden Tristate-Treiber eingesetzt. Ein solcher Tristate-Treiber wird häufig in Form einer möglicherweise mehrstufigen Treiberschaltung mit ausgangsseitigem Übertragungsglied, wie z.B. einem Pass-Transistor oder einem Transmission-Gate realisiert (Abbildung 4.6b). In Abhängigkeit vom Steuersignal für das Übertragungsglied kann der Ausgang entweder die logischen Pegel "High" oder "Low", oder einen hochohmigen "High-Impedance"-Zustand annehmen.

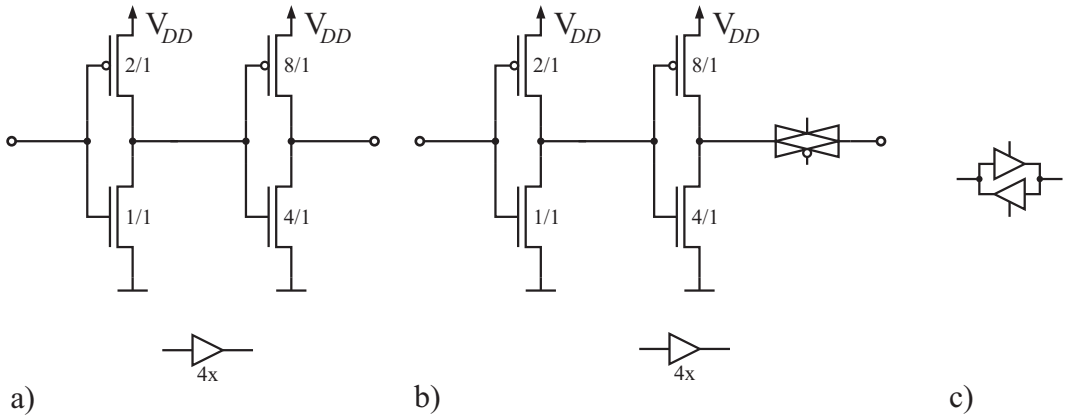


Abbildung 4.6: a) Buffer aus zwei hintereinander geschalteten Inverterstufen und Schaltsymbol, b) Tristate-Treiber und Schaltsymbol, c) bidirektionales Übertragungsglied mit Pegelregenerierung

Ein bidirektionales Übertragungsglied mit Pegelregenerierung kann aus zwei Tristate-Treibern aufgebaut werden, indem die beiden Treiber in entgegengesetzter Richtung parallel geschaltet werden (Abbildung 4.6c).

Bei den in dieser Arbeit betrachteten Architekturen tritt sehr häufig der Fall auf, dass in der Verbindungsstruktur mehrere Übertragungsglieder kaskadiert auf einem Signalpfad liegen (Abbildung 4.7). Daher ist es sinnvoll, die Übertragungseigenschaften von Transmission-Gates und Tristate-Treibern genauer zu betrachten. Es ist bekannt (siehe z.B. [BR99]), dass Pass-Transistoren und Transmission-Gates für den Fall, dass nur wenige Elemente kaskadiert werden, kürzere Verzögerungszeiten als Buffer besitzen. Allerdings steigt die Verzögerungszeit im Gegensatz zu einer Kette aus Buffern quadratisch mit der Anzahl der Elemente an, während die Verzögerungszeit einer Buffer-Kette nur linear mit der Zahl der Elemente steigt. Dieser Sachverhalt ist in Abbildung 4.8 für Transmission-Gates und Tristate-Treiber dargestellt. Angenommen wurde hier, dass die Transferglieder durch Leitungssegmente miteinander verbunden sind. Erkennbar ist die Verzögerungszeit einer Kaskade aus maximal 4 Transmission-Gates schneller als die Verzögerung einer Buffer-Kette, während für längere Ketten Buffer den schnelleren Signalpfad bilden. Auf der anderen Seite belegen Buffer eine größere Fläche als Transmisson-Gates. Für eine Entwurfsraumexploration ist daher stets nach einem Tradeoff zu suchen, für den eine möglichst minimale Signalverzögerung bei adäquatem Flächenbedarf erzielt wird. Für FPGAs existieren eine Vielzahl von Arbeiten, die sich mit diesem Problem beschäftigen (siehe z.B. [BR99], [LL02]). Das primäre Ziel der vorliegenden Arbeit ist dagegen nicht eine umfangreiche Entwurfsraumexploration unter Berücksichtigung sämtlicher physikalischer Parameter, sondern eine Kosten-/Nutzen-Analyse grobgranularer rekonfigurierbarer Archi-

tekturen. Daher wird für die verwendeten Modelle eine Verbindungsstruktur angenommen, bei der der Signalpfad nach maximal fünf in Reihe geschalteten Transmission-Gates durch Buffer von den dahinter liegenden Verbindungssegmenten kapazitiv entkoppelt wird. Der genaue Aufbau der hierzu verwendeten Verbindungs- und Switchblöcke wird den Abschnitten 4.3.2 und 4.3.3 vorgestellt.

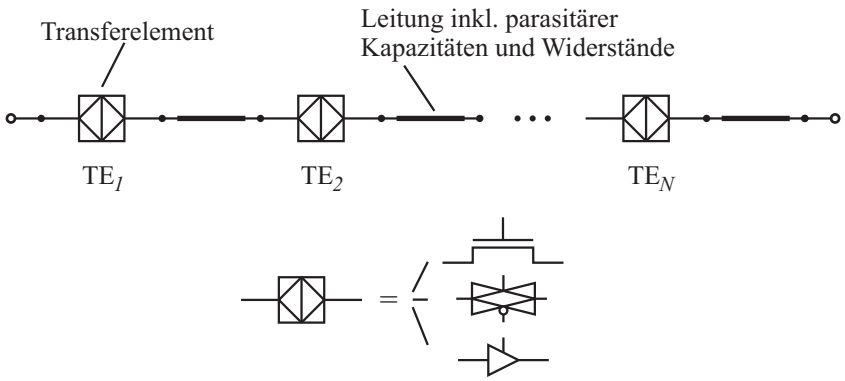


Abbildung 4.7: Kaskadierung von N Transferelementen und Leitungssegmenten

SRAM-Zellen

Wie in SRAM-basierten FPGAs werden auch bei den in dieser Arbeit betrachteten Architekturen zur Speicherung der Konfigurationsdaten SRAM-Zellen verwendet. Für SRAM-Zellen existieren verschiedene Realisierungsformen. Wie in [BRM99] wird auch in dieser Arbeit eine 6-Transistor-Zelle angenommen. Fläche, Zugriffszeiten und Leistungsaufnahme von einzelnen Zellen und wortweise organisierten SRAM-Speicherblöcken sind den entsprechenden Datenblättern entnommen.

4.3.2 Verbindungsblock

Als Schnittstelle zwischen den Datenausgängen eines Prozessorelementes und der globalen Routingstruktur dienen die bereits in den vorigen Abschnitten erwähnten Verbindungsblöcke. Die Architektur eines solchen Verbindungselementes ergibt sich aus den durch die Routingstruktur vorgegebenen Randbedingungen. In dieser Arbeit wird eine grobgranulare "Island Style" Routingstruktur für die rekonfigurierbaren Architekturmodelle angenommen. Bei ausreichend vorhandenen Routingressourcen sollen Daten von einem beliebigen Ausgangsport eines PEs oder eines Speicherblockes zu einem beliebigen Eingangsport geleitet werden können. Wie in Abschnitt 4.2.2 dargestellt, erfolgt der Datentransport durch das Feld über segmentierbare horizontale und vertikale Busleitungen. Aus diesen Randbedingungen ergeben sich folgende Anforderungen an einen Verbindungsblock:

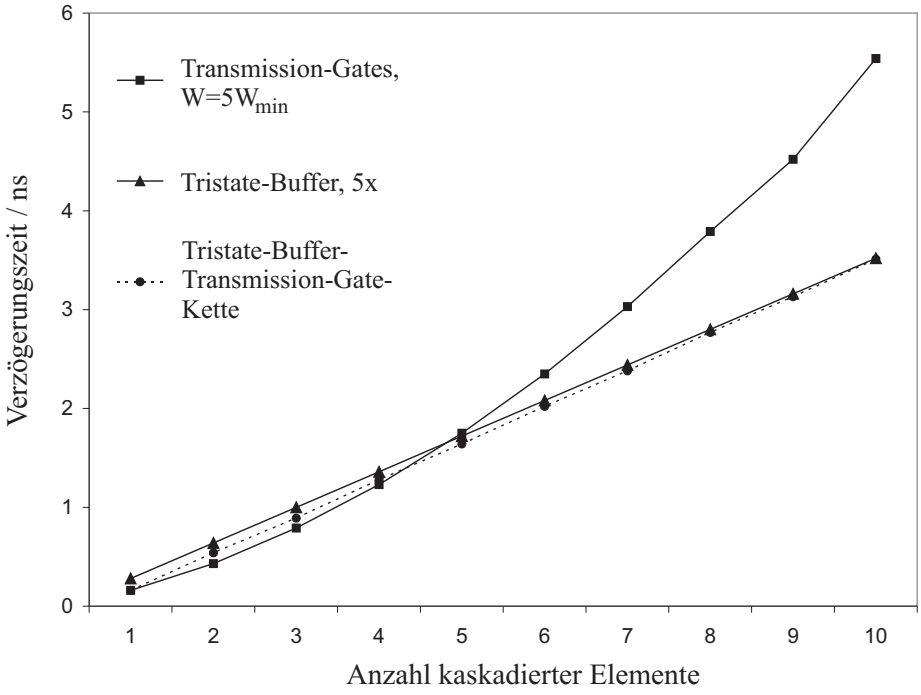


Abbildung 4.8: Gemessene Verzögerungszeiten der Pass-Transistor- bzw. Buffer-Kette aus Abbildung 4.7

- Ein PE-Datenausgang soll auf eine globale Busleitung schreiben können
- Ein PE-Dateneingang soll von einer globalen Busleitung lesen können
- Zur Gewährleistung der Nearest-Neighbor-Konnektivität sollen Ein- und Ausgangsports benachbarter Basiszellen miteinander verbunden werden können

Diese Anforderungen ähneln denen eines feingranularen Island-Style FPGAs. Abbildung 4.9 zeigt das sich hieraus ergebende Verbindungselement exemplarisch für eine Routingstruktur mit drei vertikalen globalen Leitungen, und Prozessorelementen mit jeweils zwei Eingangs- und Ausgangsports. Bei allen in der Abbildung eingezeichneten Leitungen handelt es sich um Busleitungen mit einer Datenwortbreite von N_d Bits. Über die schematisch dargestellten programmierbaren Verbindungen kann ein Ausgangsport einer Basiszelle auf jede der drei globalen Busleitungen schreiben. Um Zugriffskonflikte zu vermeiden, versteht es sich, dass jeweils nur ein Treiber pro Leitung gleichzeitig aktiv sein darf. Ein Basiszell-Eingang kann entweder von einer der globalen Leitungen, oder von dem Ausgang des gegenüberliegenden PEs lesen, oder auch offen bleiben.

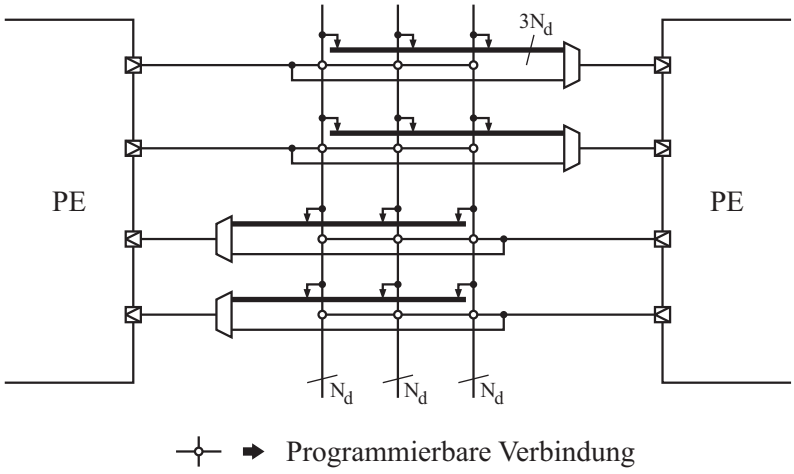


Abbildung 4.9: Schematische Darstellung eines Verbindungsblocks für drei vertikale globale Leitungen und jeweils zwei Basiszell-Ein-/Ausgänge

Zur Modellierung der Eigenschaften der Architektur wie Zeitverhalten, Flächenbedarf und Energieverbrauch ist es erforderlich, neben dem in Abbildung 4.9 skizzierten schematischen Aufbau des Verbindungsblockes die Struktur auf Schaltelement-Ebene zu betrachten. Abbildung 4.10 zeigt die Realisierung einer programmierbaren Verbindung eines Basiszell-Ausgangs mit den globalen Busleitungen. Nach einer Pegelregenerierung durch Signaltreiber wird jede einzelne Bitleitung des Datenausgangs über ein Transmission-Gate programmierbar mit den globalen Leitungen verbunden. Durch Programmierung einer SRAM-Zelle wird festgelegt, ob der Datenausgang mit der Busleitung verbunden oder hochohmig von ihr getrennt werden soll. Im Gegensatz zu FPGAs, bei denen für jedes einzelne Bit eine SRAM-Zelle vorgesehen werden muss, genügt bei grobgranularen Architekturen eine Speicherzelle mit entsprechend großer Treiberleistung für ein N_d Bit breites Leitungsbündel.

Ein Basiszell-Eingang soll nach den oben formulierten Anforderungen an die Routing-Funktionalität von einer globalen Leitung oder von dem Ausgangsport des gegenüberliegenden PEs lesen können. Bei FPGAs tritt eine ähnliche Situation bei der Herstellung einer flexiblen Verbindung der Logikblock-Eingangspins mit den Routing-Segmenten auf. Diesbezüglich werden in [CSR+99] für FPGAs zwei Realisierungsmöglichkeiten vorgeschlagen. Da ein Eingangsport von maximal einer Datenleitung lesen kann, genügt ein Multiplexer zur Auswahl der entsprechenden Leitung. Diese Realisierung hat den Vorteil, dass bei $N_{VK} + 1$ zur Auswahl stehenden Leitungen nur $\lceil \mathrm{ld}\,(N_{VK} + 1) \rceil$ SRAM-Speicherzellen zur Adressierung benötigt werden. Nachteilig ist, dass zusätzlich $2 \cdot N_{VK} \cdot N_d$ Transmission-Gates zur Implementierung des Multiplexers erforderlich sind. Ungünstig ist ferner, dass

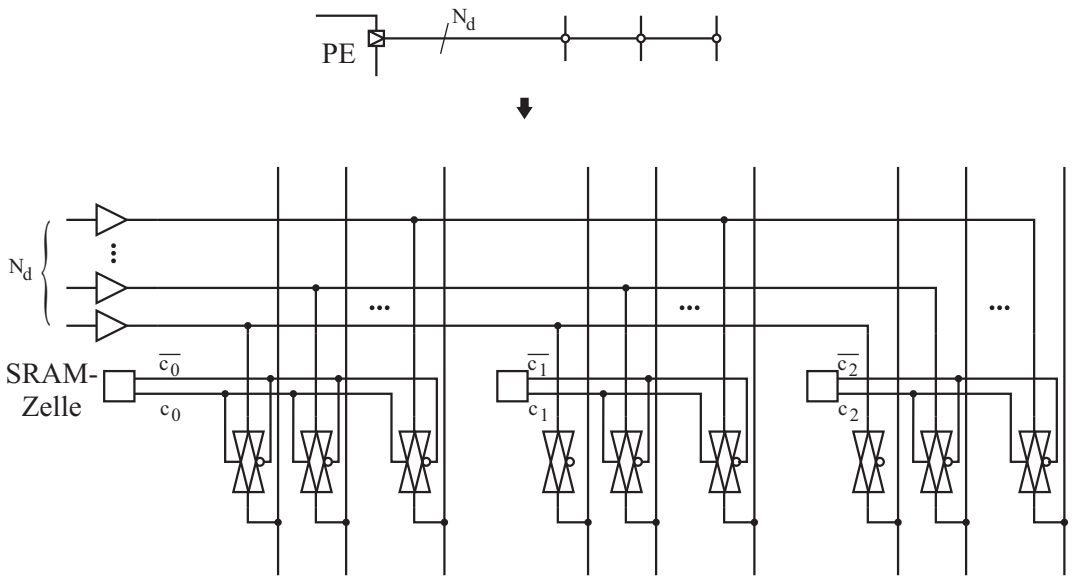


Abbildung 4.10: Herstellung einer programmierbaren Verbindung eines PE-Ausgangs mit den Busleitungen

aufgrund der in Baumstruktur angeordneten Transmission-Gates mehrere Transferglieder kaskadiert auf dem Signalpfad liegen, wodurch sich die Zeitverzögerung des Pfades signifikant erhöht. Bei der anderen in [CSR+99] vorgeschlagenen Variante werden alle zur Auswahl stehenden Datenleitungen über Transmission-Gates mit dem Eingangspin verbunden. Dies hat zur Folge, dass im Gegensatz zur Multiplexer-Realisierung für jedes Transmission-Gate eine SRAM-Zelle bereitgestellt werden muss. Für FPGAs gilt, dass die Multiplexer-Variante in den meisten Fällen aufgrund der geringeren Zahl erforderlicher SRAM-Zellen weniger Fläche belegt (siehe [CSR+99]). Dagegen ist bei den hier betrachteten grobgranularen Architekturen die zweite Variante flächengünstiger. So werden bei einer Datenwortbreite von $N_d = 8$ Bit und 3 globalen Routingkanälen für die Multiplexer-Variante unter Berücksichtigung der SRAM-Zellen 108 Transistoren benötigt, bei der Transmission-Gate-Variante jedoch nur 88. Bei $N_d = 16$ Bit werden für die Multiplexer-Alternative 204, und für die Transmission-Gate-Realisierung 152 Transistoren benötigt. In dieser Arbeit wird daher für die Implementierung der Eingangspin-Schaltung die zweite Variante auf Basis von Transmission-Gate-Schaltern angenommen.

Abbildung 4.11 zeigt die zu diesem Schema passende Schaltung für einen Eingangspin und 3 globale Routingkanäle. Wie bei der in Abbildung 4.10 gezeigten Schaltung werden Transmission-Gates eingesetzt, um eine Verbindung oder hochohmige Trennung der Routing-Kanäle mit dem Port zu realisieren. Leitungstreiber sorgen zudem für eine kapazitive Entkopplung der Busleitungen vom Eingangsport. Da inklusive der innerhalb der

Basiszelle vorhandenen Wrapper-Schaltung (siehe Abschnitt 4.3.4) nur zwei Transmission-Gates kaskadiert auf dem Signalpfad liegen, wird ein einstufiger invertierender Treiber als ausreichend betrachtet. Innerhalb des Wrappers sorgt ein zusätzlicher Inverter für eine Wiederherstellung des korrekten logischen Signals.

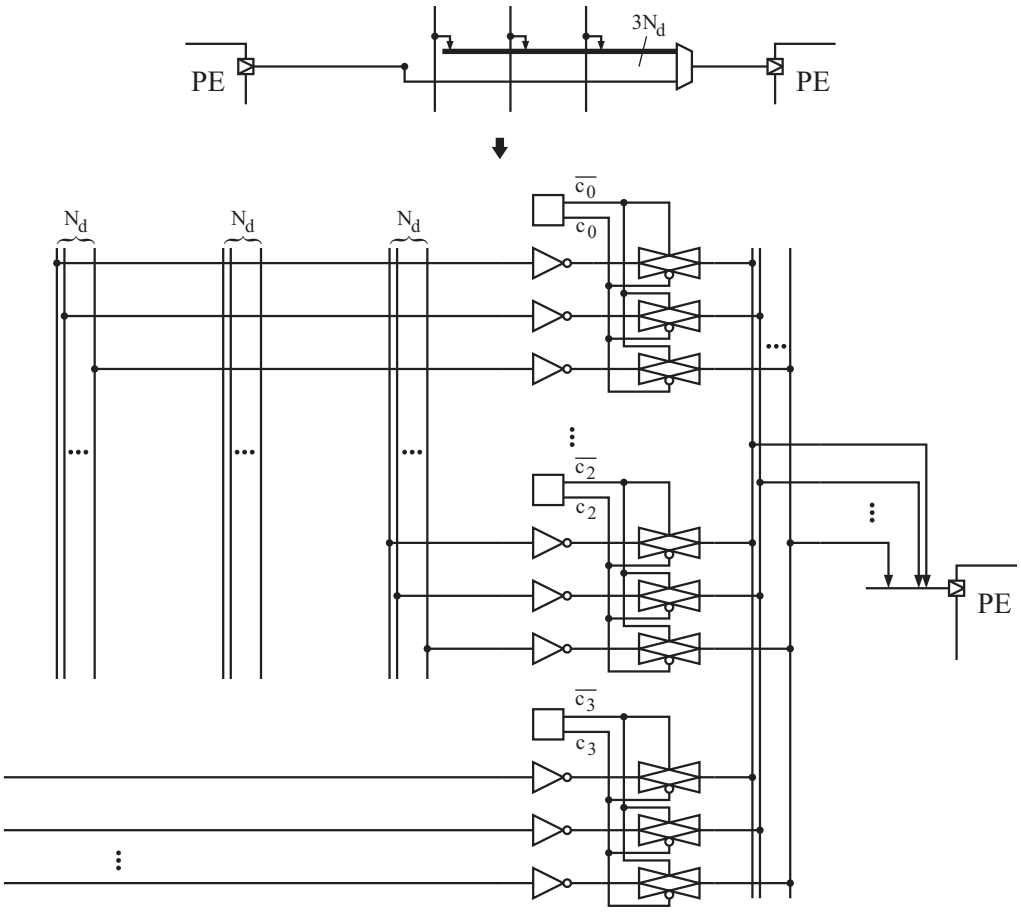


Abbildung 4.11: Realisierung einer programmierbaren Auswahlschaltung für einen PE-Eingang

4.3.3 Switchblock

Die Aufgabe der im vorigen Abschnitt angesprochen Verbindungsblöcke ist die Herstellung einer Schnittstelle zwischen den Ein- und Ausgängen der Basiszellen und dem globalen Routingnetzwerk. Letzteres besteht aus im Allgemeinen mehreren Kanälen horizontaler

und vertikaler Busleitungen (siehe Abbildung 4.2). Um den oben definierten Ansprüchen an das Verbindungsnetzwerk Rechnung zu tragen, nach denen Daten nach Möglichkeit von einem beliebigen Start- zu einem beliebigen Endpunkt durch das Feld geroutet werden können, ist an den Kreuzungspunkten zwischen den horizontalen und vertikalen Routingkanälen eine flexible Verbindungsmöglichkeit vorzusehen. Für einen solchen Switchblock lassen sich die folgenden Ansprüche formulieren:

- Ein ankommendes Datensignal soll in beliebige Richtungen (nach oben, unten, rechts oder links) geroutet werden können
- Die horizontalen und vertikalen Busleitungen sollen durch den Switchblock segmentiert werden können

Beim Design eines FPGAs ist der Entwurf einer entsprechenden optimalen Switchblock-Architektur ein hochkomplexes Problem, da ein Tradeoff gefunden werden muss zwischen einem möglichst minimalen Flächenbedarf und hoher Geschwindigkeit bei gleichzeitiger Gewährleistung der Flexibilität. Bei allen modernen FPGAs, die nach der Island-Style-Topologie aufgebaut sind, werden zwischen den Logikzellen segmentierte Busleitungen verschiedener Länge implementiert, um sowohl für kurze als auch für lange physikalische Distanzen schnelle Signalpfade zu ermöglichen. Aufgrund der groben Granularität der in dieser Arbeit betrachteten Architekturen ergeben sich andere Ansprüche an die Routingstruktur, und damit auch an die Switchbox, als bei FPGAs. So ist die Anzahl der Prozessorzellen erheblich geringer. Auch die Anzahl der unabhängig voneinander zu routenden Verbindungskanäle, die zur Abbildung der in dieser Arbeit betrachteten Algorithmen typischerweise zwischen 3 und 5 liegt, ist um eine volle Größenordnung kleiner als die Zahl der in einem Switchblock zu schaltenden Bitleitungen bei modernen FPGAs.

Aufgrund der im Vergleich zu FPGAs geringen Anzahl an Prozessorzellen wird auf längere, mehrere Zellen passierende Leitungen verzichtet. Stattdessen kann eine globale Leitung in jedem Switchblock segmentiert und umgeleitet werden. Dies vereinfacht in hohem Maße die Entwurfskomplexität für die Routingstruktur. In Abbildung 4.12 sind zwei einfache, in der Literatur häufig zitierte Switchblock-Varianten gezeigt. Bei dem in Abbildung 4.12a dargestellten "Disjoint"-Switchblock (siehe z.B. [SC05]) kann ein Kanal mit der Nummer i ausschließlich mit anderen Kanälen mit der gleichen Indexnummer i verbunden werden. Dagegen gibt es beim "Universal"-Switchblock [FLW01] die Möglichkeit, auch Verbindungen von Kanälen mit unterschiedlicher Indexnummer zu implementieren, woraus sich laut [FLW01] eine erhöhe Routing-Flexibilität ergibt. In modernen FPGAs werden dagegen weitaus höher optimierte Switchblöcke eingesetzt. Mit dem Analyse-Tool "ARA" (siehe Abschnitt 4.1) können durch einfaches Editieren einer den Switchblock beschreibenden Matrix theoretisch beliebige Varianten untersucht werden. Da eine umfangreiche Entwurfsraumexploration in Bezug auf den Switchblock jedoch nicht das Ziel dieser Ar-

beit ist, wird, sofern nicht anders erwähnt, der Disjoint-Switchblock angenommen. Diese Variante bietet für alle betrachteten Algorithmen eine ausreichende Flexibilität.

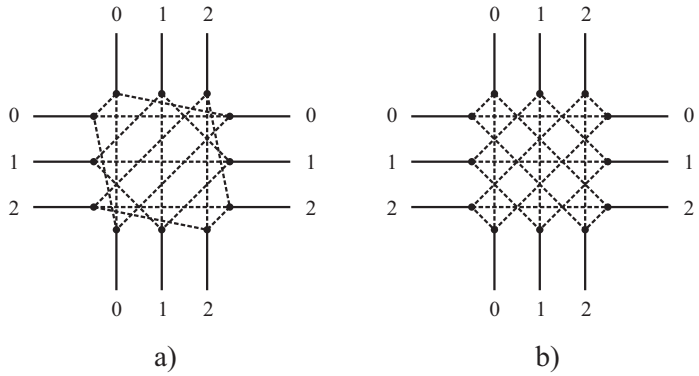


Abbildung 4.12: Zwei einfache Implementierungsvarianten eines Switchblockes für drei horizontale und vertikale Leitungen. a) Disjoint-Switchblock, b) Universal-Switchblock

Ein Disjoint-Switchblock für jeweils N horizontale und vertikale Kanäle beinhaltet N programmierbare Kreuzungspunkte. Ein solcher Kreuzungspunkt muss derart konfigurierbar sein, dass ein Leitungssegment mit den drei anderen Leitungssegmenten entweder verbunden oder von ihnen getrennt werden kann. Abbildung 4.13 zeigt die Struktur eines derartigen Verbindungspunktes auf Basis von Transmission-Gates. Für jedes Transmission-Gate ist eine SRAM-Zelle zur Speicherung der Routinginformation vorzusehen.

In Abschnitt 4.3.1 wurde gezeigt, dass die Verzögerungszeit einer Kette von Transmission-Gates quadratisch mit der Zahl kaskadierter Elemente zunimmt. Um eine zu große Anzahl kaskadierter Transmission-Gates auf dem Signalpfad durch die Verbindungsstruktur zu vermeiden, sind daher in regelmäßigen Abständen Buffer einzusetzen, welche für eine kapazitive Entkopplung der Leitungssegmente sorgen, und außerdem den Signalpegel regenerieren. Für die Einbindung von Leitungtreibern in die Verbindungsstruktur gibt es wiederum mehrere Alternativen. Eine Möglichkeit ist die Verwendung zweier parallel in entgegengesetzter Richtung geschalteter Tristate-Treiber zum Verbinden zweier Leitungssegmente anstelle der Transmission-Gates. Nachteilig dabei ist ein recht hoher Flächenbedarf. In [LL02] werden daher Alternativen für eine effizientere Implementierung vorgeschlagen. Eine Implementierungsmöglichkeit für eine gebufferte programmierbare Verbindung ist in Abbildung 4.14 gezeigt. Der Aufbau besteht zunächst aus der in Abbildung 4.13 gezeigten Schaltung auf Basis von Transmission-Gates. Zusätzlich ist an die vier Ausgänge des Verbindungspunktes jeweils ein Tristate-Treiber geschaltet, der mit einem weiteren Transmission-Gate überbrückt werden kann.

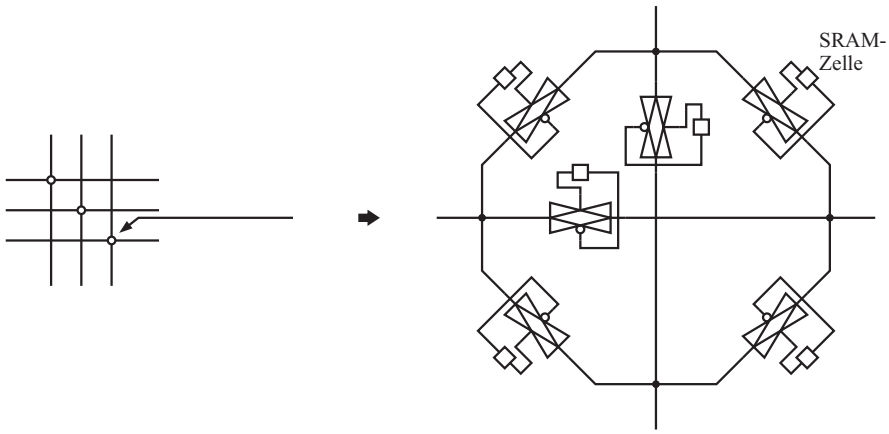


Abbildung 4.13: Realisierung eines programmierbaren Kreuzungspunktes mit Transmission-Gates im Disjoint-Switchblock

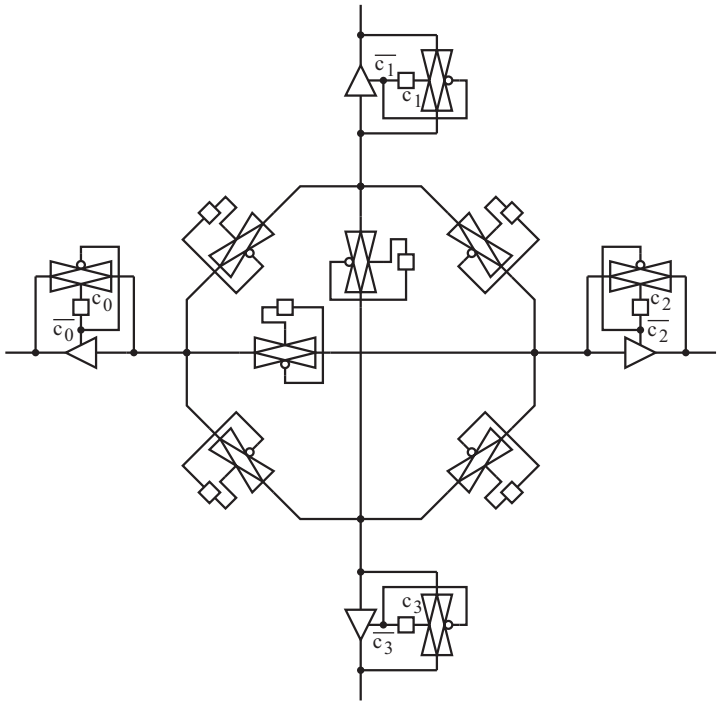


Abbildung 4.14: Realisierung eines gebufferten programmierbaren Kreuzungspunktes

Die in Abbildung 4.13 gezeigte Realisierung eines Verbindungspunktes mit Transmission-Gates ist flächeneffizient, hat aber den Nachteil, dass keine Pegelregenerierung und keine kapazitive Entkopplung stattfindet, was bei längeren Pfaden zu großen Signalverzögerungen führt. Der gebufferte Switch gemäß Abbildung 4.14 eliminiert diese Nachteile, weist jedoch einen höheren Flächenbedarf auf. In FPGAs wird daher eine Mischung aus gebufferten und ungebufferten Switchpunkten implementiert. In dieser Arbeit wird angenommen, dass 50% der Verbindungspunkte ungebuffert, und 50% gebuffert sind. Die Mischung von gebufferten und ungebufferten Kreuzungspunkten ist konform mit der Implementierung der Routing-Netzwerke kommerzieller FPGAs bei Verwendung von Prozesstechnologien bis $0.18\mu m$. Es sei erwähnt, dass in modernen FPGAs bei neueren Prozessgenerationen ab $90nm$ fast nur noch gebufferte Kreuzungspunkte verwendet werden. Der Array Analyzer (ARA) erlaubt weiterhin die Analyse theoretisch beliebiger Kombinationen gebufferter und ungebufferter Kreuzungspunkte durch Editierung der entsprechenden Beschreibungsmatrix des Switches.

4.3.4 Basiszell-Wrapper

Jedes Prozessorelement besitzt Datenein- und Datenausgangsports, deren Anzahl jeweils vom Typ der Basiszelle abhängig sind (siehe hierzu Kapitel 5 . Um eine flexible Anbindung an die Verbindungsstruktur zu erreichen, ist es wichtig sicherzustellen, dass jeder Port mit Verbindungselementen, bzw. benachbarten Prozessorzellen verbunden werden kann, die sich rechts, links, oberhalb oder unterhalb der betrachteten Zelle befinden. Um dieser Forderung gerecht zu werden, ist jede Basiszelle in einen Container (*Wrapper*) eingebunden, deren Aufbau in Abbildung 4.15 gezeigt ist.

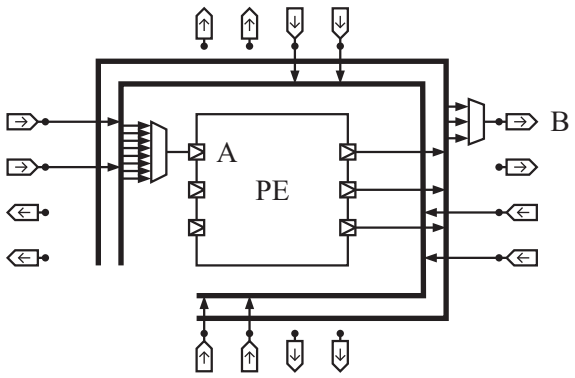


Abbildung 4.15: Realisierung des Basiszell-Wrappers

Jeder Wrapper besitzt eine durch die Parameterwahl der Verbindungsstruktur festgelegte Anzahl von Eingangs- und Ausgangsports zu jeder der vier Seiten. In Abbildung 4.15

ist eine Konfiguration mit jeweils zwei Eingangs- und zwei Ausgangsports an jeder Seite des Wrappers dargestellt. Für jeweils einen Eingangs- und einen Ausgangsport des eingebetteten Prozessorelementes ist die Anbindung an den Wrapper gezeigt. Für Eingangsport *A* des PEs lässt sich über einen Multiplexer die entsprechende Verbindung zu einem der insgesamt acht Wrapper-Eingänge auswählen. Ebenfalls über einen Multiplexer kann ausgewählt werden, von welchem Datenausgang der Prozessorzelle ein Wrapper-Ausgang lesen soll. In der Abbildung ist dies für den mit *B* gekennzeichneten Port verdeutlicht.

4.4 Block-Speicher

Von besonderer Bedeutung für die Funktionalität einer rekonfigurierbaren Architektur ist die Anbindung und die Struktur des Speichers. Wie bereits in Abschnitt 4.2.2 dieses Kapitels dargestellt, sind zwei Speicher-Hierarchieebenen in die in dieser Arbeit betrachteten Modellarchitekturen integriert. Dies sind lokale Speicher innerhalb der Basiszellen für bei der Berechnung auftretende Zwischenwerte, sowie der globale Blockspeicher für Ein-und Ausgangswerte oder Zwischenwerte, die von verschiedenen Prozessorzellen gemeinsam genutzt werden sollen. Da die Struktur des PE-internen Speichers in engem Zusammenhang mit der Anbindung an den jeweiligen Datenpfad steht, wird der lokale Speicher in Abschnitt 5.2.2 zusammen mit den Basiszellen näher beschrieben.

Die Speicherstruktur ist daraufhin zu entwerfen, dass der Ablauf eines Programms möglichst frei ist von Wartezyklen. Für diesen Fall wäre ein einzelner Single-Port SRAM-Block nicht ausreichend, da auf diesen in einem Taktzyklus nur entweder lesend oder schreibend zugegriffen werden könnte. Durch Speicherzugriffe bedingte Wartezyklen lassen sich hier jedoch nur vermeiden, wenn sowohl ein schreibender, als auch lesender Zugriff gewährleistet ist. So wird beispielsweise in FPGAs der Virtex-Familien das Problem durch die Integration von Dual-Port SRAM-Blöcken vermieden. Diese haben jedoch den Nachteil, bei gleicher Speicherkapazität bis zu doppelt so viel Fläche in Anspuch zu nehmen wie Single-Port SRAMs. In dieser Arbeit wird daher der in Abschnitt 2.4.1 beschriebene Ansatz mit zwei Single-Port-Blöcken verwendet, von welchen auf einen jeweils schreibend, und auf den anderen lesend zugegriffen werden kann.

In Abbildung 4.16 ist der Aufbau des linksseitig des PE-Feldes angekoppelten BlockRAM-Speichers dargestellt. Für jeden der mit *in* gekennzeichneten Eingangsports sind zwei Single-Port SRAM-Bänke vorgesehen, die jeweils mit *A* oder *B* beschriftet sind. Während des Ablaufs einer Berechnung wird auf eine der beiden Bänke schreibend, und auf die andere lesend zugegriffen. Nach einer bestimmten Anzahl von Takten können Schreib- und Leserechte vertauscht werden, so dass auf die im ersten Durchlauf gespeicherten Zwischenwerte zugegriffen werden kann.

Durch die Multiplexer-Struktur wird sichergestellt, dass die Zwischenwerte einer Berechnung nach Möglichkeit von allen Prozessorelementen des Feldes gelesen werden können.

Die Schreibadressen für die vom PE-Feld zu speichernden Werte werden von der externen Kontrolleinheit geliefert, und von Takt zu Takt um den Wert 1 inkrementiert. Der Lesevorgang erfolgt dagegen anhand einer Adresse, welche für jeden Takt aus dem globalen Instruktionsspeicher gelesen wird. Aus diesem werden auch die Steuersignale für die Multiplexer gelesen.

Desweiteren enthält jeder Blockspeicher jeweils einen Schreib- und einen Lese-Port, über welchen externe Daten in den Speicher geschrieben, bzw. aus diesem gelesen werden können.

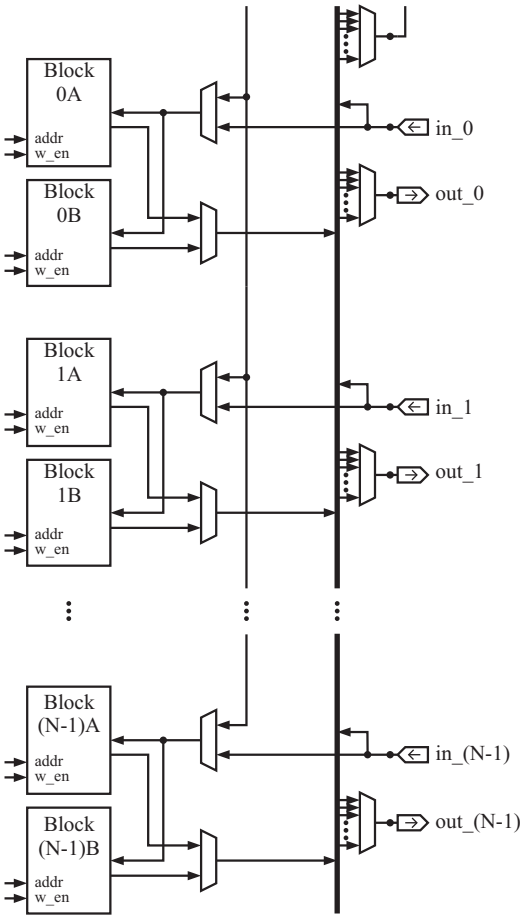


Abbildung 4.16: Realisierung des BlockRAM

5 Ausgewählte Modell-Basiszellen

Ein wesentliches Ziel dieser Arbeit ist die Untersuchung und Bewertung der qualitativen und quantitativen Eigenschaften grobgranularer rekonfigurierbarer Architekturen. In Kapitel 4 ist das in dieser Arbeit angewandte Analyseverfahren und das zugrunde gelegte Architekturmodell beschrieben worden. Die Struktur dieses Modells ist derart ausgelegt, dass durch geeignete Parameterwahl Basiszellen mit unterschiedlichen Schnittstellenkonfigurationen und für verschiedene Aufgabenbereiche instantiiert werden können. Um einen aussagefähigen Kosten/Nutzen-Vergleich verschiedener Implementierungsformen zu erhalten, erfolgt eine diesbezügliche Untersuchung am sinnvollsten vor dem Hintergrund konkreter Anwendungen aus dem Bereich der digitalen Signalverarbeitung. In diesem Kapitel werden die den Untersuchungen dieser Arbeit zu Grunde liegenden Basiszellstrukturen vorgestellt.

Kommerzielle FPGAs bieten in der Regel die Möglichkeit, nahezu beliebige Architekturbeschreibungen mehr oder weniger effizient abzubilden. Dagegen entfalten sich die Chancen grobgranularer Architekturen zumeist durch Konzentration auf eine bestimmte Anwendungsklasse. Es ist daher naheliegenderweise nicht möglich, eine grobgranulare Basiszelle zu entwerfen, die feingranularen Strukturen für alle denkbaren Algorithmen in allen Kriterien überlegen ist. Aus diesem Grunde ist vor dem Entwurf einer Zelle eine Klassifikation durchzuführen. Eine solche Klassifikation kann aus zwei Sichtweisen heraus erfolgen:

- *Aus Architektursicht.* Nach dieser Definition gehören all diejenigen Basiszellen zu einer Klasse, deren Hardwareimplementierungen sich aus den gleichen, wenn auch möglicherweise unterschiedlich verschalteten, Funktionsblöcken zusammensetzen. Als Beispiele seien hier die Familien der Multiply-Add-basierten Basiszellen und der Lookup-Table-basierten Basiszellen genannt.

- *Aus algorithmischer Sicht.* Diese Definition fasst alle Basiszellen zu einer Klasse zusammen, deren Zielalgorithmen eine bestimmte mathematische Eigenschaft teilen. Beispiele sind hier Basiszellen für lineare und nichtlineare Algorithmen.

Diese beiden Klassen sind sorgfältig zu unterscheiden. Zwar sind in vielen Fällen Basiszellen, die zu einer bestimmten Architekturklasse gehören, auch einer bestimmten Anwendungsklasse zuzuordnen. So sind beispielsweise Multiply-Add-basierte Zellen geeignet zur Berechnung linearer Operationen. Andererseits sind auch Basiszellen denkbar, die aus Architektursicht zwar zu unterschiedlichen Klassen gehören, wie zum Beispiel Multiply-Add-

und Lookup-Tabellen-basierte Zellen, die aber dennoch für die gleichen Zielalgorithmen entworfen wurden, also nach der zweiten Definition der gleichen Klasse zuzuordnen sind. In dieser Arbeit wird die zweite Definition verwendet, die Basiszellen nach der Klasse der auszuführenden Algorithmen einteilt.

In den folgenden Unterabschnitten werden die entwickelten und untersuchten grobgranularen Basiszellstrukturen vorgestellt. Es folgt zunächst eine Beschreibung des allen Basiszellentwürfen zugrunde liegenden Prinzips des virtuellen Feldes. Nach einer kurzen Beschreibung der jeweils zugrunde liegenden Algorithmen erfolgt anschließend die Darstellung der daraus resultierenden Architekturen.

5.1 Prinzip des virtuellen Feldes

Ein beim Entwurf eines massiv parallel datenverarbeitenden Prozessorfeldes zu lösendes Problem tritt auf, wenn der auszuführende Algorithmus mehr Operationen enthält, als physikalische Ausführungseinheiten in realer Hardware existieren. In diesem Fall muss der reale Signalflussgraph des Algorithmus in mehrere logische Teile aufgespaltet werden, welche von den zur Verfügung stehenden Hardwareressourcen serialisiert in mehreren Takten berechnet werden. Dies ist der allgemein anzutreffende Fall, welcher durch eine geeignete Organisation der Verbindungs- und Speicherstruktur gelöst werden muss. Der reale, nicht in einem Takt parallel zu berechnende Signalflussgraph wird hier als *virtuelles Feld* bezeichnet (s.a. [OBU+02] und [LOS04]).

Das Prinzip des virtuellen Feldes ist schematisch in Abbildung 5.1 skizziert. Dargestellt ist ein willkürlicher Signalflussgraph, der insgesamt zwölf Operationen (Knoten) umfasst, die untereinander in Abhängigkeit stehen (Kanten, dargestellt durch die Pfeile). Angenommen wird hier, dass durch die zur Verfügung stehende Hardware statt zwölf lediglich vier Operationen parallel ausgeführt werden können. Der Signalflussgraph muss also zu drei Sub-Graphen "zusammengefaltet" werden, die nacheinander abgearbeitet werden. In der Abbildung ist durch die grau schattierten Boxen eine mögliche Aufteilung angegeben. Die Operationen innerhalb der Boxen können jeweils parallel berechnet werden. Erkennbar entstehen Zwischenwerte an den Stellen, an denen die Kanten (Pfeile) die schattierten Boxen verlassen. In der Abbildung ist dies durch die schwarz ausgefüllten Punkte dargestellt. Diese Zwischenwerte müssen geeignet gespeichert, und beim Start der Berechnung des nächsten Sub-Graphen wieder geladen werden.

In der Praxis sind die zu berechnenden Signalflussgraphen in der Regel erheblich größer als der in diesem einfachen Beispiel angegebene. Die hierbei entstehenden Ansprüche an die Verbindungs- und Speicherstruktur sind beim Hardwareentwurf des PE-Feldes zu beachten. Die Verbindungsstruktur und der globale Block-Speicher wurden bereits in den Abschnitten 4.3 und 4.4 vorgestellt. Wie das hier angeführte Beispiel verdeutlicht, ist es erforderlich, zusätzlich zum globalen Speicher für Ein- und Ausgangswerte einen lokalen

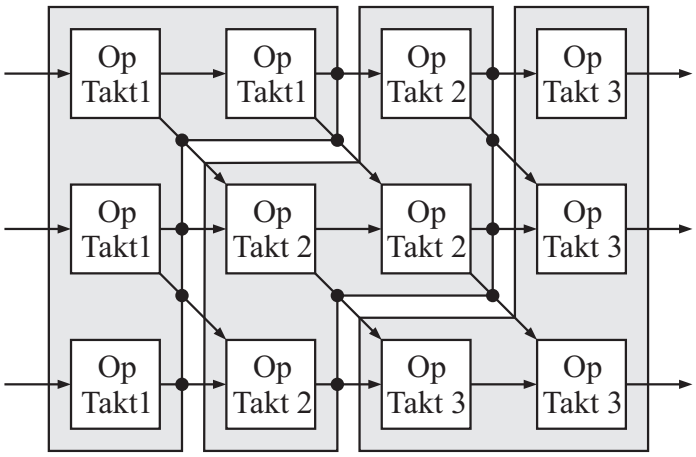


Abbildung 5.1: Darstellung des Prinzips des virtuellen Feldes: Ausführung eines zwölf Operationen enthaltenden Signalflussgraphen mit vier physikalischen Ausführungseinheiten

PE-internen Speicher vorzusehen, in dem bei der Berechnung großer Signalflussgraphen nach dem Prinzip des virtuellen Feldes auftretende Zwischenwerte gehalten werden können.

In Abschnitt 5.2.2 wird eine an die Datenpfade der Prozessorzellen angekoppelte lokale Speicherstruktur vorgestellt, die den oben formulierten Ansprüchen Rechnung trägt.

5.2 Basiszellen für lineare Operationen

Ein Operator $f(\cdot)$ heiß linear, wenn sich das Ergebnis einer Operation linear aus der Summe der Einzelergebnisse ergibt ($f\left(\sum_i k_i s_i\left(x\right)\right) = \sum_i k_i f\left(s_i\left(x\right)\right)$, Superpositionsprinzip). Bekannte Beispiele für lineare Operationen sind lineare Filter in verschiedenen Ausführungsformen, und lineare Transformationen wie die Fourier- und die Cosinus-Transformation. Derartige lineare Operationen finden in vielen Bereichen der Bild- und Videosignalverarbeitung Anwendung, wie beispielsweise in der Bildskalierung oder -interpolation, in Artefaktreduktionsverfahren und in Bild- und Videocodecs. Da die Architekturen sich direkt aus den Zielalgorithmen ergeben, erfolgt zunächst eine kurze Darstellung der in dieser Arbeit betrachteten linearen Operationen.

5.2.1 Algorithmen aus der Klasse der linearen Operationen

In diesem Abschnitt werden Grundlagen und allgemeine Realisierungsaspekte der betrachteten linearen Operationen dargestellt. Eine detaillierte Behandlung und Herleitung der

Algorithmen ist nicht Gegenstand dieser Arbeit, weshalb für tiefergehende Informationen jeweils auf die Literatur verwiesen sei.

FIR-Filter

Finite Impulse Response (FIR) Filter (siehe z.B. [Sch98], [SB00] oder [Wan99]) sind nichtrekursive Filter, deren Ausgangssequenz $g(n)$ sich durch diskrete Faltung der Impulsantwort $h(n)$ mit der Eingangswertefolge $s(n)$ bestimmen lässt:

$$g(n) = \sum_{m=0}^{N-1} h(m) \cdot s(n-m) \tag{5.1}$$

Die Faltungssumme führt unmittelbar zur direkten Form des Signalflussgraphen (Abbildung 5.2a). Aus der direkten Form ableiten lässt sich die transponierte Form (Abbildung 5.2b), welche die gleiche Übertragungsfunktion besitzt, bei der jedoch die Verzögerungsregister in die Addiererkette geschaltet sind.

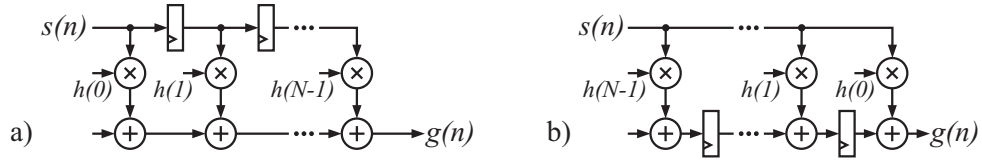


Abbildung 5.2: FIR-Filter in a) direkter und b) transponierter Realisierung

Aus Gleichung 5.1 und Abbildung 5.2 ist erkennbar, dass FIR-Filtern mit der Multiply-Add-Operation eine sehr einfache, sich wiederholende Grundoperation zugrunde liegt, welche vorteilhaft zur Realisierung mit regelmäßigen PE-Feldern geeignet ist. FIR-Filter mit sehr langer Impulsantwort werden im Allgemeinen nicht vollständig parallel, sondern serialisiert implementiert. Siehe hierzu Abschnitt 5.1 über das Prinzip des virtuellen Feldes.

Zweidimensionale Filter

In der Bild- und Videosignalverarbeitung liegen im Allgemeinen örtlich zweidimensionale Eingangssignale vor. Eine aufwandgünstige 2D-Filterung lässt sich durch separierte aufeinanderfolgende eindimensionale Filterungen der Zeilen und Spalten eines Bildes durchführen. Allerdings ist hier die Einstellbarkeit der Filterwirkung in den diagonalen Richtungen nur sehr eingeschränkt gegeben. Für eine allgemeine örtliche Filterung ist daher die im vorigen Abschnitt dargestellte Faltungssumme für 1D-Signale auf den 2D-Fall zu erweitern. Ein Ausgangsbild $g(n_x, n_y)$ ergibt sich aus der 2D-Eingangssequenz $s(n_x, n_y)$ und einer zweidimensionalen Impulsantwort $h(n_x, n_y)$ zu ([Sch98]):

$$g(n_x, n_y) = \sum_{m_x=0}^{N_x-1} \sum_{m_y=0}^{N_y-1} h(m_x, m_y) \cdot s(n_x - m_x, n_y - m_y) \tag{5.2}$$

Schnelle Fourier-Transformation

Die schnelle Fouriertransformation (Fast Fourier Transform, FFT), ist ein schnelles Berechnungsverfahren der diskreten Fouriertransformation (DFT). Letztere ist eine Frequenztransformation, bei welcher eine Anzahl von im Zeit- oder Ortsbereich gegebenen Werten in den Frequenzbereich transformiert werden. Sie ist eine komplexe Transformation, bei der komplexe Eingangswerte in komplexe Ausgangswerte transformiert werden. Es existieren eine Vielzahl unterschiedlicher Verfahren zur effizienten Berechnung der DFT. Für genauere Informationen sei hier auf die einschlägige Literatur verwiesen. An dieser Stelle sollen lediglich die für den Entwurf der Basiszell-Architekturen relevanten Eigenschaften dargestellt werden.

Abbildung 5.3a zeigt den Signalflussgraphen einer 8-Punkte Decimation-in-Time FFT auf Basis der in Abbildung 5.3b gezeigten Radix-2 Butterfly-Struktur. Diese wiederum besteht aus einer komplexen Multiplikation eines Eingangswertes mit einem komplexen Twiddle-Faktor, sowie aus vier zusätzlichen reellen Additionen, bzw. Subtraktionen.

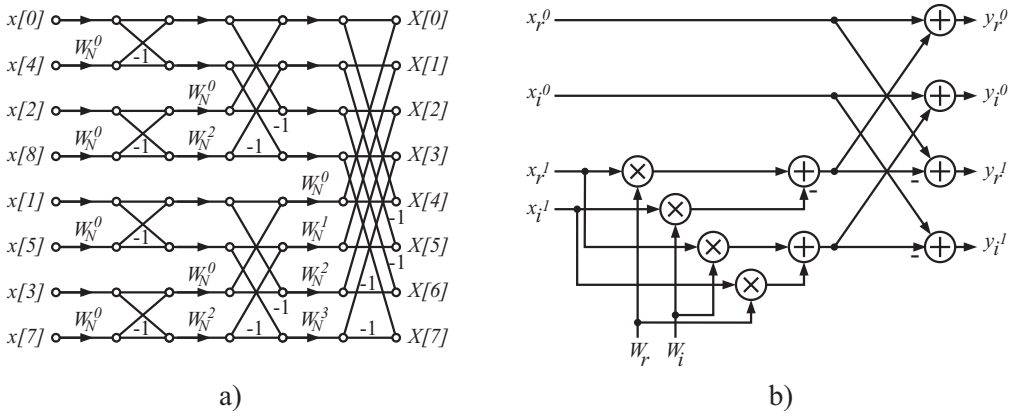


Abbildung 5.3: a) Signalflussgraph einer schnellen Fourier-Transformation (Radix-2, Decimation in Time), b) Signalflussgraph eines komplexen Radix-2 Butterflies

Diskrete Cosinus-Tranformation

Die diskrete Cosinustransformation (DCT) ist ähnlich wie die FFT eine Frequenztransformation, bei der allerdings reelle Eingangswerte in reelle Ausgangswerte transformiert wer-

den. Sie findet unter anderem in vielen Bereichen der Bildsignalverarbeitung Anwendung, da sie die Eigenschaft besitzt, eine sehr gute Energiekonzentration auf wenige Koeffizienten zu erreichen. Sie wird daher beispielsweise in den Bild- und Videocodierungsverfahren von JPEG und MPEG zur Bildkompression verwendet. Auch für die DCT existieren eine Vielzahl von schnellen Berechnungsformen. Das Verfahren zur schnellen DCT-Berechnung nach Chen wird in [CSF77] und [Rao80] vorgestellt. Der Signalflussgraph ist in Abbildung 5.4a gezeigt. Äquivalent zur FFT ist auch hier eine Butterfly-ähnliche Operation zugrunde gelegt. Abbildung 5.4b zeigt die allgemeine Butterfly-Operation mit vier reellen Multiplikationen und zwei Additionen zur Berechnung der DCT nach Chen.

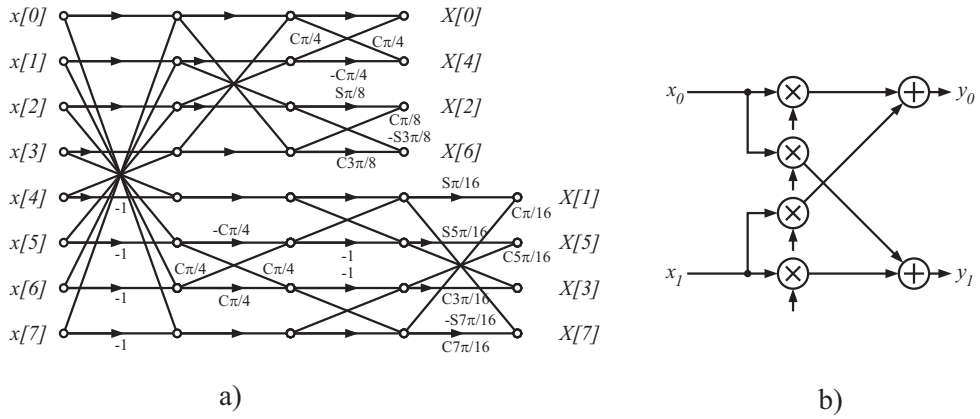


Abbildung 5.4: a) Signalflussgraph einer schnellen Cosinus-Transformation nach Chen, b) Signalflussgraph eines Chen-DCT Butterflies

Matrix-Vektor-Multiplikation

Viele Algorithmen in der digitalen Signalverarbeitung lassen sich auf eine Matrix-Vektor-Multiplikation zurückführen, wie beispielsweise die Farbraumkonversion von RGB nach YUV oder umgekehrt. Eine Matrix-Vektor-Multiplikation lässt sich durch sukzessives Ausführen von Multiply-Accumulate-Operationen berechnen:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ a_{M1} & a_{M2} & \cdots & a_{MN} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \tag{5.3}$$

5.2.2 PE-Architekturen für lineare Operationen

Die im vorigen Abschnitt beschriebenen linearen Algorithmen bilden die Grundlage für die im Nachfolgenden dargestellten Prozessorelement-Architekturen. Es sind vier verschiedene PEs für diese Algorithmen entworfen worden, welche sich im Wesentlichen durch den Granularitätsgrad unterscheiden. Der Granularitätsgrad einer Basiszelle wird in dieser Arbeit, wie in Abschnitt 2.2.1 beschrieben, an der Komplexität der zu Grunde liegenden Basiszellarchitektur gemessen. Bezogen auf Prozessorelemente für lineare Algorithmen wird hier angenommen, dass die Zahl der in der Zelle enthaltenen Multiplizierer die Granularität bestimmt. Dies begründet sich durch die Tatsache, dass die Multipliziererzahl, bzw. die Anzahl der Multiply-Accumulate-Einheiten, ein wesentlicher Faktor für die Rechenleistung eines DSP-Systems ist, da die inneren Schleifen vieler Algorithmen der digitalen Signalverarbeitung im Kern MAC-Operationen ausführen müssen.

Tabelle 5.1 zeigt einen Überblick über die in dieser Arbeit implementierten und untersuchten grundlegenden Basiszell-Strukturen für lineare Algorithmen. Weiterhin sind Varianten untersucht worden mit unterschiedlich tiefem Pipelining, sowie Prozessorelemente, die neben den linearen Datenpfaden zusätzlich Blöcke zur Ausführung nichtlinearer Algorithmen enthalten.

Tabelle 5.1: Übersicht über die untersuchten Basiszellen für lineare Algorithmen

Name	Anzahl Mult.	Anzahl Add./ Sub.	Anzahl Akk.-Reg.	Bemerkung
PE1M	1	3	2	
PE2M	2	3	2	Modifiziertes RMAC-PE, siehe Abschnitt 2.4.1
PE4M	4	6	4	Cluster aus zwei *PE2M*
PE8M	8	12	8	Cluster aus vier *PE2M*

Allen PEs gemeinsam ist der in Abbildung 5.5 gezeigte allgemeine Aufbau. Dieser setzt sich aus dem Datenpfad, dem PE-internen Speicher und einem Instruktions-Decoder zusammen. Die Datenpfad-Einheit kann zur Zwischenspeicherung vorgesehene Daten in den internen Speicher schreiben, und diesen auslesen. Die Lese-Adressen werden vom Instruktions-Decoder geliefert. Die Schreibadressen werden für alle Basiszellen der Gesamtarchitektur global zur Verfügung gestellt, und werden zu Beginn einer Berechnung

von 0 beginnend aufwärts gezählt. Nach Ende eines Durchlaufs wird der Zähler für die Schreibadressen wieder auf 0 zurückgesetzt. Siehe hierzu Abschnitt 5.1 zum Prinzip des virtuellen Feldes.

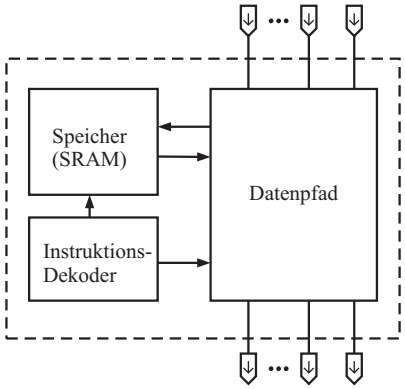


Abbildung 5.5: Allgemeiner Aufbau der Prozessorzellen

Wie bereits in Abschnitt 4.4 dargestellt, ist beim Entwurf der Speicherstruktur darauf zu achten, dass der Programmablauf möglichst wenige durch Speicherzugriffe bedingte Wartezyklen enthält, was nur dann möglich ist, wenn auf einen Speicherbereich in einem Taktzyklus sowohl lesend, als auch schreibend zugegriffen werden kann. Bei den BlockRAM-Speichern (Abschnitt 4.4) wird das Problem durch zwei unabhängig voneinander ansprechbare Single-Port SRAM-Bänke gelöst, von denen auf jeweils einen schreibend, und auf den anderen lesend zugegriffen wird. Ein ähnlicher Ansatz wird auch für den PE-internen Speicher verwendet, wobei die Speicherbänke durch eine schaltungstechnische Maßnahme zusätzlich als Lookup-Tabelle konfiguriert werden können.

Abbildung 5.6 zeigt exemplarisch die Speicherstruktur für das *PE2M*-Prozessorelement. Die Struktur besteht im Kern aus insgesamt vier Single-Port Speicherbänken (0A, 0B, 1A und 1B), von denen jeweils zwei zu einem Block zusammengefasst sind. Wird auf die mit A gekennzeichneten Bänke lesend zugegriffen, dann wird gleichzeitig in die mit B gekennzeichneten Bänke geschrieben, oder umgekehrt. Von den an den vier Dateneingängen des *PE2M* anliegenden Datenwörtern können in jedem Takt durch die beiden 4:1-Multiplexer jeweils zwei Wörter ausgewählt und in den Speicher geschrieben werden. Zu Beginn eines Rechendurchlaufes werden eventuell auftretende Zwischenwerte zunächst in die A-Bänke geschrieben. Werden die in Bank A gespeicherten Zwischenwerte wieder benötigt, dann wechselt der Zugriff auf diese Bank von schreibend auf lesend, und wiederum entstehende Zwischenwerte werden nun in Bank B geschrieben. Während eines Rechendurchlaufes können mehrere Wechsel auftreten, diese sind Teil des normalen Ausführungsvorganges. Durch die zwei getrennten Speicherbereiche lassen

sich Zugriffskonflikte vermeiden. Ein ähnliches Konzept wird in der in Abschnitt 2.4.1 dargestellten Accelerator-Architektur angewandt (s.a. [OBU+02], [LFS+02] und [LOS04]).

Vorteil dieser Anordnung ist die automatische Vermeidung von Schreib-/Lese-Konflikten, da stets in einen Speicherbereich nur geschrieben, und aus dem anderen nur gelesen wird. Alternativ können Dual-Port-RAMs verwendet werden. Bei diesem können Adresskonflikte durch geichzeitigen Schreib- und Lesezugriff auf eine Speicherzelle auftreten. In diesem Fall wird der neue Wert zwar korrekt in die Zelle geschrieben, jedoch kann der gleichzeitig aus der Zelle gelesene alte Wert fehlerhaft sein. Dual-Port-RAM-Blöcke belegen laut [Cha03a] und [Cha03b] bei gleicher Speichergröße in etwa den doppelten Flächenbedarf, und verbrauchen bis zu einer Speichergröße von 128 16-Bit-Werten auch etwa doppelt so viel Energie wie Single-Port-RAMs.

Einige Algorithmen zur Bildsignalverarbeitung, wie beispielsweise Gamma-Korrektur oder automatischer Weißabgleich, lassen sich zum Teil günstig mit Hilfe von Wertetabellen implementieren. Aus diesem Grunde erlaubt die in Abbildung 5.6 dargestellte Speicherstruktur zusätzlich die Konfiguration des Speichers als Lookup-Tabelle (LUT). Dabei lassen sich alle vier Bänke zu einer Lookup-Tabelle zusammenfassen. Die Lese-Adressen werden demzufolge nicht mehr vom im Instruktionsdecoder enthaltenen Speicher zur Verfügung gestellt, sondern von den 8 LSBs des PE-Dateneingangs 0 (hier als Bits 7 bis 0 bezeichnet). Die Bits 7 und 6 dienen hier zur Auswahl der korrekten Speicherbank, deren Lese-Attribut über die beiden Bits aus einem 1-aus-4-Decoder entsprechend gesetzt wird. Die Bits 5 bis 0 dienen weiterhin als Adresse für die ausgewählte Bank.

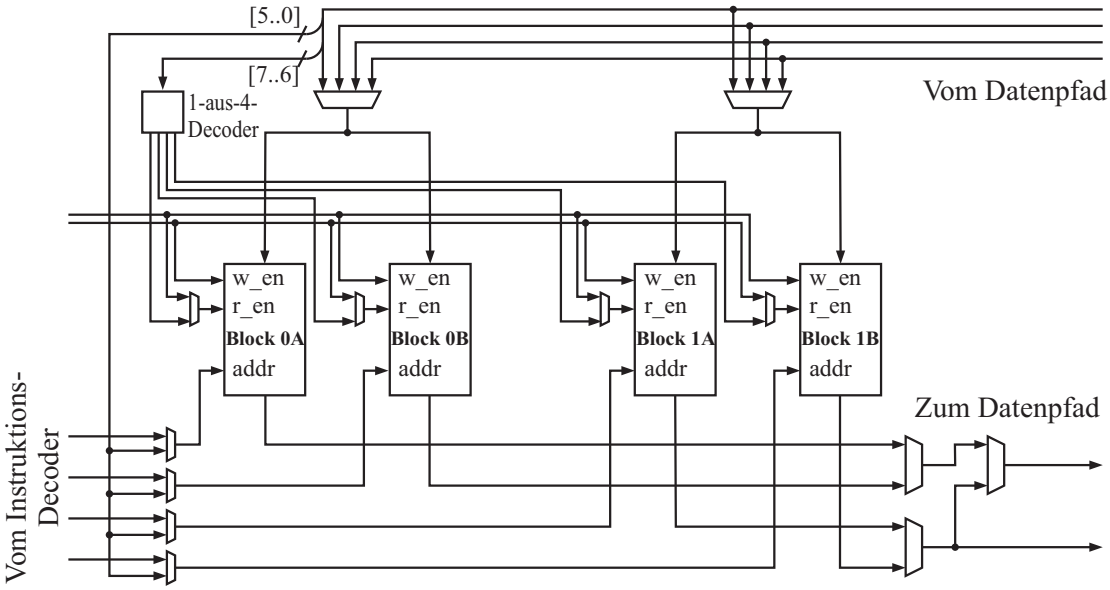


Abbildung 5.6: Speicherstruktur für das *PE2M*-Prozessorelement

PE1M

Die Struktur des Datenpfades für die Basiszelle mit einem Multiplizierer ist in Abbildung 5.7 dargestellt. Die Zelle dient zur Ausführung allgemeiner Multiply-Add- und Multiply-Accumulate-basierter Rechenoperationen, und ist insbesondere auf die in Abschnitt 5.2.1 dargestellten Algorithmen hin optimiert, ohne jedoch auf diese beschränkt zu sein. Das *PE1M* beinhaltet neben dem Multiplizierer zusätzlich drei Additions-/Subtraktions-Einheiten und ein Akkumulator-Register. Die Multiplexer dienen zum Umschalten der Funktionalität der Zelle.

Die Basiszelle trägt den Anforderungen der in Abschnitt 5.2.1 dargestellten Algorithmen Rechnung. So lässt sich eine FIR-Filterung unter Ausnutzung der Multiply-Add-Stufe durchführen. Falls die Länge des Filters die Anzahl der im gesamten PE-Feld enthaltenen Multiply-Add-Stufen übersteigt, wird das unter Abschnitt 5.1 beschriebene Prinzip des virtuellen Feldes angewandt. In diesem Fall werden die bei der Berechnung entstehenden Zwischenwerte in den lokalen Speicherbänken abgelegt, und bei Bedarf wieder gelesen. Weiterhin lässt sich das *PE1M* für eine effiziente Ausführung der Butterfly-Operationen zur FFT- und DCT-Berechnung konfigurieren (siehe Abschnitte 5.2.1 und 5.2.1). Matrix-Vektor- oder Matrix-Matrix-Operationen lassen sich unter Ausnutzung des Akkumulator-Registers durchführen. In allen Fällen werden die jeweils benötigten Koeffizienten in dem mit *TapRAM* gekennzeichneten Speicherblock abgelegt.

Abbildung 5.7 zeigt weiterhin die Anbindung des lokalen Speichers an den Datenpfad. Jeder der drei Dateneingänge des *PE1M* ist über die in Abbildung 5.6 (exemplarisch für das *PE2M*) gezeigte Struktur mit dem Speicherblock verbunden. Die vom Speicherblock gelesenen Daten können über die eingangsseitigen Multiplexer *mux0*, *mux1* und *mux2* dem nachfolgenden Datenpfad zugeführt werden.

Ausgangsseitig besitzt das *PE1M* für jeden Datenausgang jeweils ein konfigurierbares Verzögerungsregister. Über die Multiplexer *mux8*,, *mux9* und *mux10* kann entschieden werden, ob der Datenausgang um einen Takt verzögert werden soll, oder nicht. Dies ist in einigen Fällen zum Ausgleichen von Latenzzeitdifferenzen erforderlich. Ferner sind Varianten der Zelle mit jeweils unterschiedlicher Anzahl von Pipeline-Stufen untersucht worden (siehe hierzu Kapitel 7). Die Pipeline-Register sind aus Gründen der Übersichtlichkeit in Abbildung 5.7 nicht eingezeichnet. Alle untersuchten Basiszellen enthalten jedoch mindestens eine ausgangsseitige Pipeline-Stufe.

PE2M

Das *PE2M* ist eine modifizierte Variante des bereits in Abschnitt 2.4.1 kurz vorgestellten RMAC-PEs für eine rekonfigurierbare Accelerator-Architektur. Im Vergleich zum *PE1M* enthält der Datenpfad im Wesentlichen einen weiteren Multiplizierer, ein zusätzliches Akkumulator-Register, und einen weiteren Dateneingang. Auch dieses PE ist für die er-

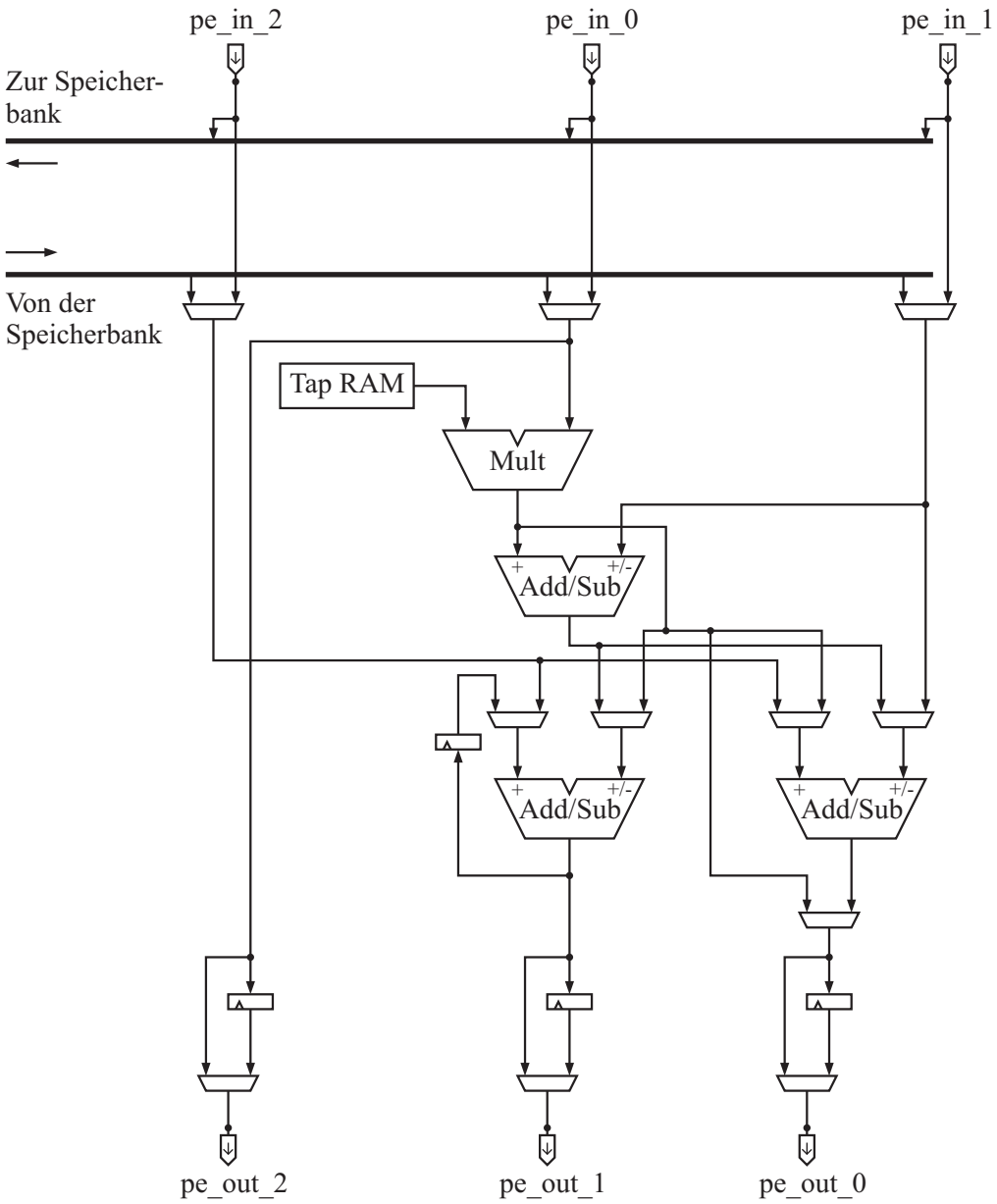


Abbildung 5.7: Datenpfad des Prozessorelementes für lineare Operationen mit einem Multiplizierer (*PE1M*)

wähnten Algorithmen optimiert, ohne auf diese beschränkt zu sein, da sich auch weitere, hier nicht dargestellte Multiply-Add- und Multiply-Accumulate-basierte Algorithmen ausführen lassen. Für die Speicheranbindung, die Rekonfigurierbarkeit und das Pipelining gilt das bereits zum *PE1M*-Datenpfad gesagte.

PE4M und *PE8M*

Um in die Untersuchung auch die Eigenschaften komplexerer Basiszellen einfließen zu lassen, sind zwei Prozessorzellen entworfen worden, welche jeweils vier (*PE4m*), bzw. acht (*PE8M*) Multiplizierer und Akkumulatorstufen enthalten. Dabei sind die gleichen Zielalgorithmen zugrunde gelegt wie für die beiden anderen PE-Varianten *PE1M* und *PE2M* für lineare Algorithmen. Der Datenpfad der *PE4M*-Basiszelle besteht daher aus einem Cluster aus zwei *PE2M*-Zellen mit lokaler Verbindungstruktur (siehe Abbildung 5.9). Der *PE8M*-Datenpfad ist ein Cluster aus vier *PE2M*-Zellen (Abbildung 5.10).

5.3 Erweiterungen für nichtlineare Operationen

5.3.1 Motivation

Grobgranulare rekonfigurierbare Architekturen können durch anwendungsspezifischen Schaltungsentwurf für die zu Grunde liegenden Algorithmen Effizienzsteigerungen gegenüber FPGA- oder DSP-Lösungen erzielen. Es ist jedoch auch der Fall denkbar, dass eine rekonfigurierbare Architektur zur Beschleunigung von Algorithmen oder Operationen verwendet werden soll, für welche die Struktur nicht ursprünglich optimiert wurde. Als einfaches Beispiel sei der Fall einer zunächst linearen zweidimensionalen Bildfilterung durch entsprechend für lineare Operationen optimierte Basiszellen genannt, wenn mit der gleichen Architektur die möglicherweise noch im Speicher liegenden Bilddaten einer nichtlinearen Medianfilterung unterworfen werden sollen. Für die nichtlineare Operation ist die Architektur entsprechend umzukonfigurieren. Beispielsweise stellt die in [PAC05] vorgestellte XPP-Architektur von PACT (s.a. Abschnitt 2.4.2) Prozessorzellen zur Verfügung, die neben Addierern und Multplizierern auch Komparatorbausteine enthalten. Die XPP-Architektur ist zudem dynamisch konfigurierbar, so dass lineare und nichtlineare Filterungen durch zeitlichen Multiplex auf derselben Hardware ausgeführt werden können.

Es soll der Frage nachgegangen werden, welche Effizienz eine hybride Prozessorzelle für zwei verschiedene Klassen von Algorithmen besitzt. Zu diesem Zweck werden die in Abschnitt 5.2.2 vorgestellten Basiszellen für lineare Algorithmen um die Möglichkeit erweitert, zusätzlich nichtlineare Operationen durchzuführen.

Im folgenden Abschnitt sollen zunächst die grundlegenden Architekturkonzepte für zwei bekannte nichtlineare Algorithmen, der Medianfilterung und der Add-Compare-Select-Operation für Viterbi-Decoder, vorgestellt werden. In Abschnitt 5.3.3 wird eine

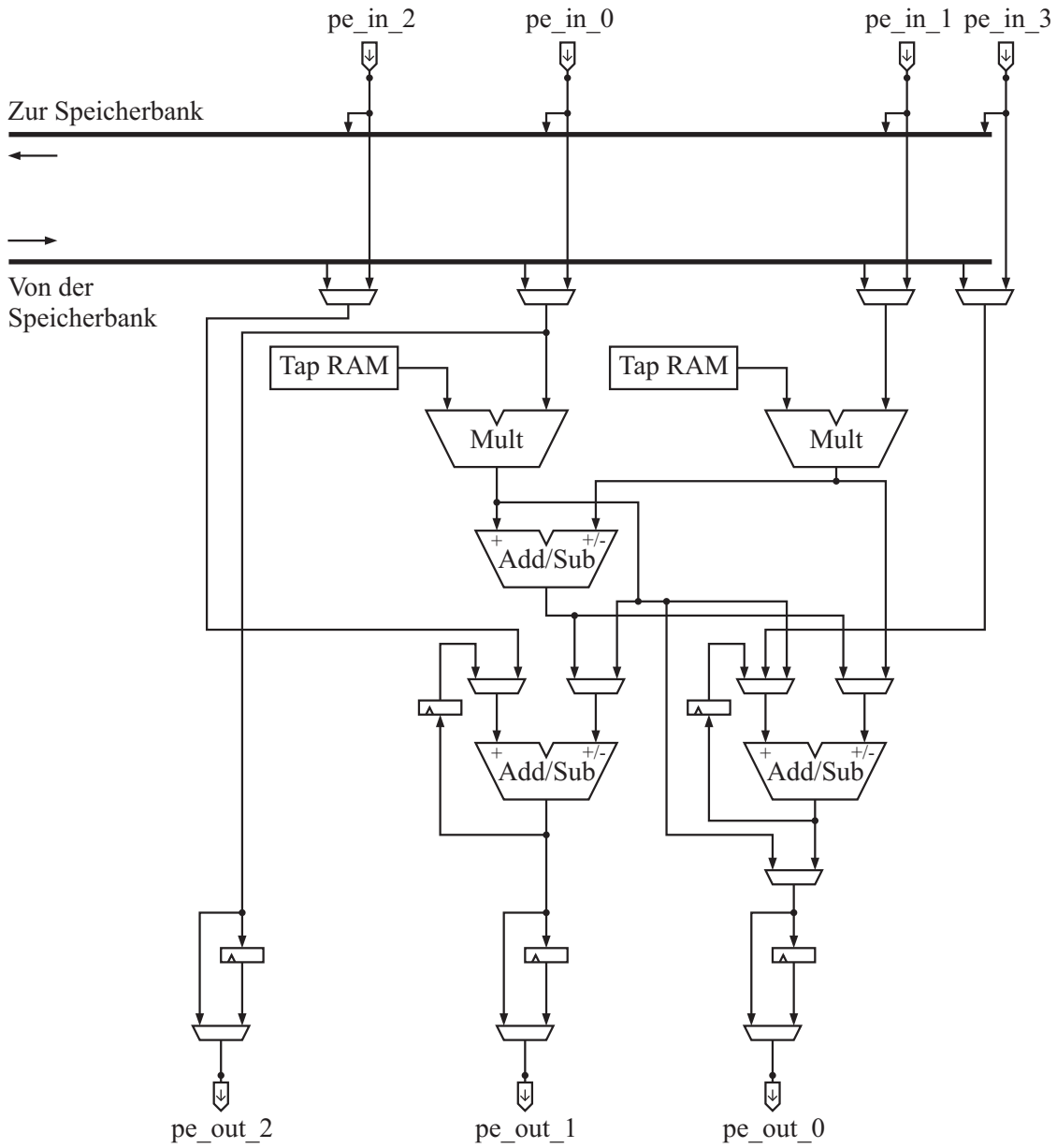


Abbildung 5.8: Datenpfad des Prozessorelementes für lineare Operationen mit zwei Multiplizierern (*PE2M*)

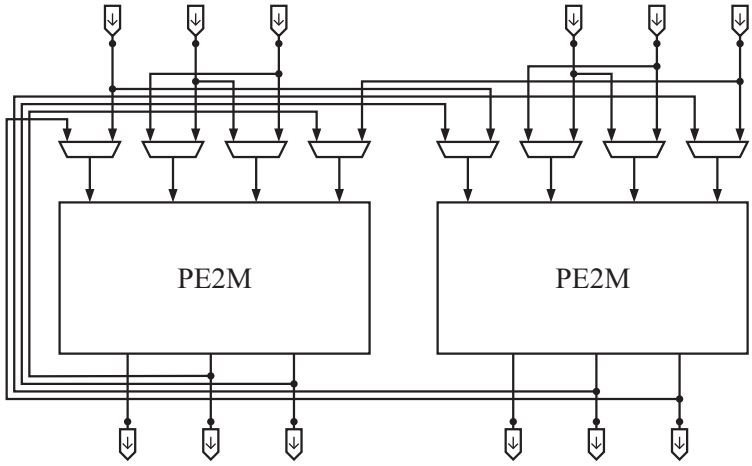


Abbildung 5.9: Datenpfad des *PE4M*, Cluster aus zwei (*PE2M*)

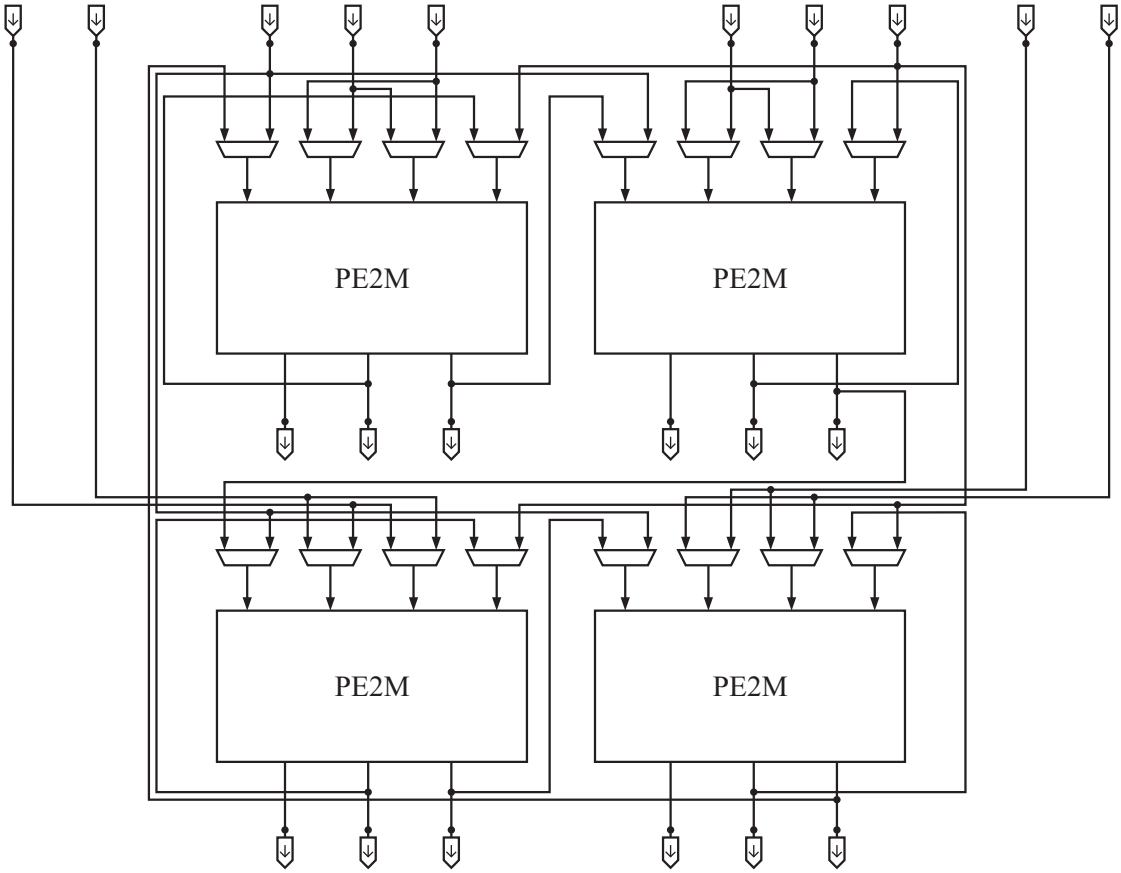


Abbildung 5.10: Datenpfad des *PE8M*, Cluster aus vier (*PE2M*)

Architekturerweiterung vorgestellt, mit dessen Hilfe die Basiszellen für nichtlineare Operationen konfiguriert werden können.

5.3.2 Beispiele für nichtlineare Algorithmen

Median-Filter

Ein Median-Filter ist ein Spezialfall eines Rangordnungsfilters, bei dem allgemein das jeweilige Ergebnis m_n dem n-ten Wert einer aufsteigend sortierten Folge von N Werten aus einem vorgegebenen ein- oder zweidimensionalen Filterfenster entspricht. Der Median ist definiert als der mittlere Wert m_{N+1} einer aufsteigend sortierten Folge von $2N + 1$ Werten (siehe z.B. [SB00]).

Es existieren viele verschiedene Varianten von Implementierungsmöglichkeiten für Medianfilter. In dieser Arbeit soll lediglich die einfache Variante betrachtetet werden, die auf der Sortierung der Wertefolge beruht (Odd-Even-Transposition-Netzwerk). Bei diesem Verfahren werden Werte paarweise miteinander verglichen, und gegebenenfalls vertauscht, bis die Folge vollständig sortiert ist. Eine Zelle, die eine solche Operation ausführt (*Compare&Swap-Element*), ist in Abbildung 5.11a gezeigt.

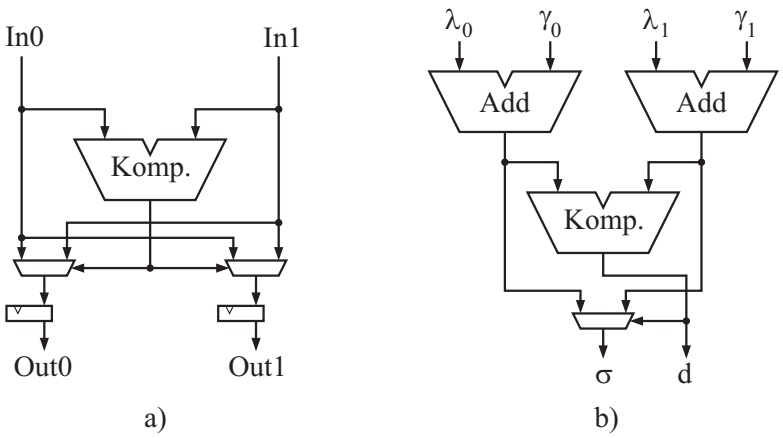


Abbildung 5.11: a) Compare&Swap-Element zur Medianfilterung nach dem Odd-Even-Transposition-Verfahren, b) Add-Compare-Select-Zelle für einen Viterbi-Decoder

Add-Compare-Select-Operation für Viterbi-Decoder

Ein Viterbi-Decoder führt eine Maximum-Likelihood-Schätzung zur Ermittlung der über einen verrauschten Kanal empfangenen, am wahrscheinlichsten gesendeten Wertefolge

durch. Zu diesem Zweck werden empfangene Datenwörter mit allen im Codebuch enthaltenen möglichen Sendesymbolen verglichen, und die Distanz (beispielsweise die Hamming-Distanz oder die Euklid'sche Distanz in Soft-Decision-Decodern) zu diesen Symbolen ermittelt. Eine mögliche gesendete Nachricht entspricht dabei jeweils einem Pfad durch das *Trellis-Diagramm*. In jedem Schritt werden die jeweils besten Symbole ausgewählt (*Survivor*), und zu den Pfadmetriken akkumuliert. Nach einer bestimmten Anzahl von Zyklen wird der Pfad mit der besten Pfadmetrik ausgewählt, und als am wahrscheinlichsten gesendeter Pfad dekodiert.

Ein Viterbi-Decoder setzt sich in üblichen Implementierungen aus mehreren Blöcken zusammen. Die bei Weitem rechenintensivste Operation wird in der *Add-Compare-Select*-Einheit (ACSU) ausgeführt, in der in jedem Symboltakt die Zweigmetriken zu den Pfadmetriken addiert, die Ergebnisse miteinander verglichen, und die jeweils beste Metrik ausgewählt werden muss.

Abbildung 5.11b zeigt einen möglichen Aufbau einer einfachen ACS-Zelle.

5.3.3 Erweiterungsblock für nichtlineare Algorithmen

Der oben angesprochene Block zur Ausführung nichtlinearer Algorithmen, deren Implementierungen auf Vergleichsoperationen beruhen, ist in Abbildung 5.12 dargestellt. Der Block wird hinter die Addierer der in Abschnitt 5.2.2 vorgestellten Prozessorelemente geschaltet. Über Multiplexer kann der Ausgang des korrekten Addierers ausgewählt werden.

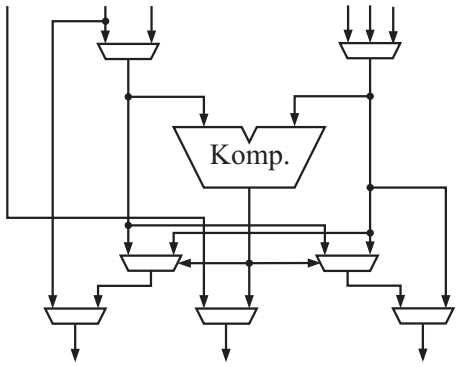


Abbildung 5.12: Erweiterungsblock für nichtlineare Algorithmen

6 Abschätzung der physikalischen Eigenschaften

Das in Kapitel 4 vorgestellte Programm zur Analyse der grobgranularen rekonfigurierbaren Modelle enthält Methoden zur Abschätzung der VLSI-Eigenschaften der Strukturen. Dabei werden sowohl alle Architekturparameter, als auch der jeweils abgebildete Algorithmus berücksichtigt. Die dabei zur Anwendung kommenden physikalischen Modelle werden in diesem Kapitel vorgestellt.

6.1 VLSI-Eigenschaften der Modellarchitekturen

6.1.1 Modellierung des Flächenbedarfs

Ein wesentliches Kriterium bei der Bewertung einer Architektur ist die zur Implementierung erforderliche Chipfläche. Diese bestimmt primär die Ausbeute, also die Anzahl auf einem Wafer integrierbarer Einheiten, und hat daher unmittelbar Einfluss auf die Herstellungskosten. Ein wesentliches Entwurfsziel ist daher stets ein minimaler Flächenbedarf unter Einhaltung der in der Spezifikation vorgegebenen Anforderungen beispielsweise an den Datendurchsatz und den Energieverbrauch. Da feingranulare Strukturen wie FPGAs naturgemäß nur eine geringe Flächeneffizienz aufweisen, ist ein Vergleich mit verschiedenen Alternativen, vor allem mit grobgranularen Architekturen, von besonderem Interesse. In diesem Abschnitt wird das zur Modellierung der für die rekonfigurierbaren Modellarchitekturen erforderlichen Chipfläche angewandte Verfahren vorgestellt.

Für eine exakte Angabe der Siliziumfläche einer Architektur ist prinzipiell ein kompletter Entwurf bis auf Layoutebene erforderlich. Dies ist hier bei der Vielzahl der betrachteten Implementierungsalternativen aus Aufwandsgründen nicht durchführbar. Es wird daher ein Ansatz verfolgt, bei dem die Chipfläche einerseits mit Hilfe experimentell gewonnener Ergebnisse von Syntheseläufen für die Basiszellen, und andererseits durch ein analytisches Modell für die Fläche der Verbindungsstruktur abgeschätzt wird.

Neben dem zu implementierenden Algorithmus bzw. der Anwendung, aus der heraus eine Schaltung entsteht, gibt es eine Vielzahl weiterer Einflussfaktoren auf die Chipfläche einer Architektur. Diese sind bei der Entwicklung der Flächenmodelle zu berücksichtigen, und es ist im Einzelfall anzugeben, unter welchen Randbedingungen die Ergebnisse ermittelt wurden. Die wichtigsten Faktoren sind:

- *Die Entwurfsmethode.* Die Wahl der Implementierungsmethode, z.B. Semi- oder Full-Custom-Entwurf, FPGA-Implementierung oder Software-Programmierung eines DSPs, beeinflusst unmittelbar das Flächenmodell.

- *Die Technologiefeinheit.* Die Strukturfeinheit der zugrunde liegenden Technologielinie hat direkten Einfluss auf die Chipfläche. In dieser Arbeit wird sowohl für die durch einen Semi-Custom-Entwurf erhaltenen Flächen für die Basiszellen, als auch zur Abschätzung der Eigenschaften der Verbindungsstruktur ein $0.18\mu m$-Prozess von Chartered Semiconductors zugrunde gelegt. Alle absoluten Flächenangaben für die Gesamtarchitekturen oder einzelne Funktionsblöcke in dieser Arbeit beziehen sich auf diese Technologie. Um einen aussagefähigen Vergleich zu ermöglichen, werden zudem die Flächen aller in dieser Arbeit betrachteten Implementierungsalternativen auf eine $0.18\mu m$-Technologie normiert.

- *Synthesewerkzeug und Constraints.* Das zur Synthese der Gatternetzlisten aus einer HDL-Beschreibung verwendete Softwaretool beeinflusst den Flächenbedarf. Ebenso kann das Syntheseergebnis durch Setzen von Constraints beispielsweise in Bezug auf Flächenbedarf oder Geschwindigkeit gesteuert werden.

Der Flächenbedarf der in Kapitel 4 vorgestellten rekonfigurierbaren Modellarchitekturen ist eine Funktion mehrerer Parameter, wie beispielsweise der Anzahl und Art der PEs, der Speichergröße, aber auch der Fläche der Bauteile der Verbindungsstruktur. Die hier angewandte Methodik zur Modellierung der Chipfläche wird in den folgenden Abschnitten dargestellt.

Gesamtfläche der Modellarchitekturen

Die gesamte modellierte Fläche der in Abbildung 4.2 in Abschnitt 4.2.2 dargestellten globalen Struktur ergibt sich folgendermaßen aus der Fläche der Einzelkomponenten:

$$A_{gesamt} = A_{PEs,gesamt} + A_{VB,gesamt} + A_{SB,gesamt} + A_{RAM,gesamt} \tag{6.1}$$

Mit:

$A_{PEs,gesamt}$:	Gesamtfläche aller Basiszellen
$A_{VB,gesamt}$:	Gesamtfläche aller Verbindungsblöcke
$A_{SB,gesamt}$:	Gesamtfläche aller Switchblöcke
$A_{RAM,gesamt}$:	Gesamtfläche des Blockspeichers

Fläche der Basiszellen

Basiszellen werden in dieser Arbeit in einem Semi-Custom-Entwurf aus einer VHDL-Beschreibung heraus synthetisiert. Die Fläche A_{PE} einer Zelle lässt sich mit Hilfe des Synthesetools bestimmen, und in Form von Gatteräquivalenten angeben, woraus anschließend durch Multiplikation mit einem Umrechnungsfaktor die absolute Fläche in μm^2 bestimmt werden kann. Der Umrechnungsfaktor für die $0.18\mu m$-Technologie ist durch Messung der absoluten Abmessungen des Layouts einiger Standardzellen bestimmt worden. Die so erhaltenen Flächen für die Basiszellen verstehen sich inklusive einer Abschätzung der Verdrahtungsfläche, die durch das Synthesetool durchgeführt wird.

Aus der Basiszell-Fläche lässt sich die Kantenlänge eines PEs ermitteln, wobei hier ein quadratisches Basiszell-Layout angenommen wird. Die Kenntnis der Kantenlänge ist zur Bestimmung der Länge der Verbindungsleitungen erforderlich, welche unter anderem einen Einfluss auf das Zeitverhalten hat.

Fläche der Verbindungsstruktur

Wie in Kapitel 4 dargestellt, liegt die Verbindungsstruktur nicht in Form einer Gatternetzliste oder gar eines Layouts vor. Aus diesem Grunde ist eine analytische Vorgehensweise bei der Bestimmung der Fläche erforderlich. Ein ähnliches Problem wird bei dem in Abschnitt 3.1.1 vorgestellten Verfahren zur Entwurfsraumexploration bei FPGAs adressiert. Dort wird die Chipfläche für ein FPGA anhand der Anzahl erforderlicher minimaler Transistorgrößen abgeschätzt. Die in der vorliegenden Arbeit angewandte Methodik zur Bestimmung der Flächen für die Funktionsblöcke der Verbindungsstruktur fußt ebenfalls auf einer Schätzung der Transistorzahl. Allerdings ist hier zusätzlich die Angabe eines absoluten Flächenmaßes normiert auf die $0.18\mu m$-Technologie erforderlich, um die Werte mit anderen Implementierungsformen sinnvoll vergleichen zu können. Um eine realitätsnahe Flächenabschätzung zu erhalten, ist es erforderlich, wie in [BRM99] neben der reinen Anzahl auch die Größe der einzelnen Transistoren mit in die Berechnung einzubeziehen. Die Transistorgrößen wiederum können durch Layout einzelner Transistoren und aus den Entwurfsregeln der Zieltechnologie extrahiert werden. Wichtig ist hierbei die Beachtung der in den Entwurfsregeln vorgeschriebenen minimalen Abstandsmaße zwischen einzelnen Transistoren (s.a. [BRM99]). Mit den so erhaltenen Transistorflächen lässt sich nach Abzählen der in den Funktionsblöcken enthaltenen Transistoren ein Flächenmaß für die Verbindungsstruktur angeben.

Die Fläche eines Verbindungsblockes für die vertikalen Routingkanäle lässt sich unter Kenntnis der Bauelementflächen folgendermaßen bestimmen:

$$
\begin{aligned}
A_{VB,VK} &= [N_{PE-Outputs} \cdot N_{VK} \cdot N_d] \cdot A_{TG,PE-Outputs} \\
&\quad + [N_{PE-Outputs} \cdot N_d] \cdot A_{Buf,PE-Outputs} \\
&\quad + [N_{PE-Outputs} \cdot N_{VK}] \cdot A_{SRAM-Zelle} \\
&\quad + [N_{PE-Inputs} \cdot (N_{VK}+1) \cdot N_d] \cdot A_{TG,PE-Inputs} \\
&\quad + [N_{PE-Inputs} \cdot (N_{VK}+1) \cdot N_d] \cdot A_{Buf,PE-Inputs} \\
&\quad + [N_{PE-Inputs} \cdot (N_{VK}+1)] \cdot A_{SRAM-Zelle}
\end{aligned} \tag{6.2}
$$

Mit:

$N_{PE-Outputs}$:	Anzahl der PE-Ausgangsports zum Verbindungsblock
N_{VK}:	Anzahl vertikaler globaler Routingkanäle
$A_{TG,PE-Outputs}$:	Fläche eines Transmission-Gates zur Verbindung mit den Routingkanälen
N_d:	Globale Datenwortbreite
$A_{Buf,PE-Outputs}$:	Fläche eines Leitungstreibers am PE-Ausgang
$A_{SRAM-Zelle}$:	Fläche einer SRAM-Zelle
$N_{PE-Inputs}$:	Anzahl der PE-Eingangsports vom Verbindungsblock
$A_{TG,PE-Inputs}$:	Fläche eines Transmission-Gates am PE-Eingang
$A_{Buf,PE-Inputs}$:	Fläche eines Leitungstreibers am PE-Eingang

Abbildung 6.1 verdeutlicht die Bedeutung der einzelnen Symbole.

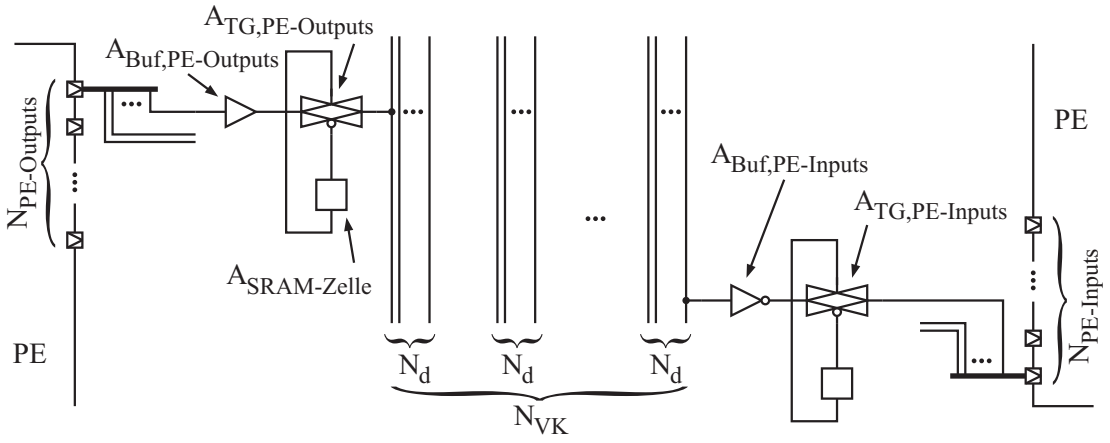


Abbildung 6.1: Bestimmung der Fläche der Verbindungsstruktur

Die Fläche $A_{VB,HK}$ eines Verbindungsblockes für die horizontalen Kanäle ergibt sich äquivalent zu Gleichung 6.2. Damit bestimmt sich die Gesamtfläche aller Verbindungsblöcke in einem $N_{PE,z} \times N_{PE,s}$-PE-Feld zu:

$$A_{VB,gesamt} = N_{PE,z} \cdot (N_{PE,s} + 1) \cdot A_{VB,VK} + (N_{PE,z} + 1) \cdot N_{PE,s} \cdot A_{VB,HK} \tag{6.3}$$

Ein ungebufferter programmierbarer Verbindungspunkt (VP) nach Abbildung 4.13 in einem Switchblock liefert folgenden Beitrag zum Flächenbedarf:

$$A_{SB,VP} = 6 \cdot [A_{TG,Switch} + A_{SRAM-Zelle}] \tag{6.4}$$

Bei einem gebufferten Verbindungspunkt nach Abbildung 4.14 ergibt sich folgender Flächenanteil:

$$\begin{aligned} A_{SB,BufVP} &= 6 \cdot [A_{TG,Switch} + A_{SRAM-Zelle}] \\ &\quad +4 \cdot [A_{TG,Switch} + A_{Buf,Switch} + A_{SRAM-Zelle}] \end{aligned} \tag{6.5}$$

Mit:

$A_{TG,Switch}$:	Fläche eines Transmission-Gates in einem Switch
$A_{Buf,Switch}$:	Fläche eines Leitungstreibers in einem Switch

Unter Annahme von $N_{SB,VP}$ ungebufferten und $N_{SB,BufVP}$ gebufferten Verbindungspunkten in einem Switchblock nach Disjoint-Topologie (siehe Abbildung 4.12a) ergibt sich der gesamte Flächenanteil eines Switchblockes zu

$$A_{SB} = N_{SB,VP} \cdot A_{SB,VP} + N_{SB,BufVP} \cdot A_{SB,BufVP}, \tag{6.6}$$

woraus sich für ein $N_{PE,z} \times N_{PE,s}$-PE-Feld die gesamte anteilige Switchblock-Fläche durch

$$A_{SB,gesamt} = [N_{PE,z} + 1] \cdot [N_{PE,s} + 1] \cdot A_{SB} \tag{6.7}$$

bestimmen lässt.

Fläche des Block-Speichers

Die Fläche aller Speicherblöcke ist den jeweiligen Datenblättern entnommen worden. Die Fläche der Multiplexer (siehe Abbildung 4.16) wird ebenfalls berücksichtigt. Aus Gründen der Geheimhaltung der Technologiedaten können die jeweiligen Flächen hier nicht in absoluten Werten angegeben werden.

6.1.2 Modellierung des Zeitverhaltens

Von besonderem Interesse bei der Bewertung der Eigenschaften einer Architektur ist die Frage, mit welcher Geschwindigkeit Daten verarbeitet werden können. Dies erfordert eine Untersuchung des Zeitverhaltens der jeweiligen Architektur. Das Ergebnis der Timing-Analyse ist die Angabe der maximalen Taktfrequenz, mit der die Schaltung sicher betrieben werden kann. Die maximale Taktfrequenz wiederum wird durch den kritischen Pfad bestimmt, also den Signalpfad mit der längsten Verzögerung zwischen zwei getakteten Schaltelementen.

Die Suche nach dem kritischen Pfad erfordert prinzipiell eine Analyse des Zeitverhaltens aller Signalpfade im Entwurf. Bei einem festverdrahteten Full-Custom-Entwurf betrifft dies alle in der Schaltung vorkommenden Pfade. Dagegen sind bei rekonfigurierbaren Architekturen wie FPGAs oder grobgranularen Strukturen die genauen Verläufe der Signalpfade erst nach dem Mapping bekannt. Dementsprechend kann es vorkommen, dass bestimmte Pfade nicht benutzt werden, und daher auch für die Untersuchung des Timingverhaltens nicht betrachtet werden müssen. Wenn beispielsweise die Abbildung einer Schaltung auf ein FPGA nur einen kleinen Anteil der Logikzellenfläche in Anspruch nimmt, und der Rest unbenutzt bleibt, genügt eine Analyse des Timingverhaltens der durch die abgebildete Schaltung belegten Ressourcen. Die Angabe der maximalen Pfadverzögerung bei rekonfigurierbaren Architekturen ist daher stets bezogen auf den jeweils betrachteten Algorithmus.

Die Timinganalyse der in dieser Arbeit untersuchten grobgranularen Architekturen unterteilt sich in die Untersuchung des Zeitverhaltens der Prozessorelemente und der jeweils zu Grunde gelegten Verbindungsarchitektur. Da die PEs als synthetisierte VHDL-Modelle vorliegen, erfolgt die Timinganalyse mit Hilfe des Simulationstools PrimeTime, welches eine zur Abschätzung hinreichend genaue Untersuchung auf Gatterebene ermöglicht. Das Verzögerungsverhalten der Verbindungsstruktur wird durch ein analytisches Modell nachgebildet, welches in diesem Abschnitt vorgestellt wird. Hierbei wird das Schaltverhalten von Transmission-Gates, Leitungtreibern und Leitungssegmenten durch RC-Glieder modelliert, so dass ein Pfad durch ein verteiltes RC-Netzwerk nachgebildet wird. Auf Basis des häufig verwendeten Verzögerungsmodells nach Elmore lassen sich Abschätzungen der Signallaufzeiten in Abhängigkeit von den Bauteilparametern in geschlossener Form angeben.

In diesem Abschnitt wird nach einer kurzen Einführung in die in dieser Arbeit angewandten Vorgehensweise bei der statischen Timing-Analyse das Verzögerungsmodell nach Elmore vorgestellt. Zur Evaluation der Genauigkeit dieses Modells werden im darauf folgenden Abschnitt die Ergebnisse einiger SPICE-Simulationen typischer Signalpfade der betrachteten Verbindungsstruktur angegeben, und den jeweils berechneten Ergebnissen des Elmore-Modells gegenübergestellt. Daran anschließend wird die in diesem Abschnitt

dargestellte Vorgehensweise bei der Timing-Analyse der Gesamtarchitektur zusammengefasst.

Statische Timing-Analyse für die rekonfigurierbaren Modellarchitekturen

Bei der Untersuchung des Zeitverhaltens einer Schaltung wird grundsätzlich zwischen statischer und dynamischer Timing-Analyse unterschieden. Bei einer statischen Analyse werden die Pfadverzögerungen aller relevanten Signalpfade ermittelt, woraus beispielsweise Aussagen zum kritischen Pfad, der maximalen Taktfrequenz oder potentiellen Timing-Hazards ermittelt werden können. Im Gegensatz dazu wird bei einer dynamischen Analyse die Schaltung für einen gegebenen Satz von Stimuli-Vektoren simuliert, woraus unmittelbar das Zeitverhalten eines bestimmten Algorithmus ableitbar ist [Syn03b]. Da das logische Verhalten bei der statischen Methode nicht modelliert werden muss, und außerdem eine umfassendere Analyse der Schaltung erfolgt, da im Gegensatz zur dynamischen Methode alle Timing-Pfade in die Simulation einbezogen werden, wird bei Performance-Untersuchungen üblicherweise die statische Timing-Analyse angewandt.

Abbildung 6.2 illustriert schematisch die Vorgehensweise bei der Analyse des Timingverhaltens der in dieser Arbeit betrachteten rekonfigurierbaren Modellarchitekturen. Dargestellt ist ein in die Verbindungsstruktur eingebettetes Prozessorelement. Ein solches PE beinhaltet in der Regel sowohl kombinatorische Logik als auch nichtkombinatorische Logik in Form von Registern. Ein Timing-Pfad beginnt und endet stets am Eingang eines getakteten Speicherelementes, wie dies in der Abbildung ersichtlich für Pfad B der Fall ist. Die kombinatorische Logik zwischen den beiden Registern kann mehrere Signalpfade enthalten, von denen jeweils derjenige Pfad mit der längsten Verzögerung vom Timing-Analyzer extrahiert werden muss. Für Pfade, deren Anfangs- und Endpunkt innerhalb des Prozessorelementes liegen, wird dies vollständig durch PrimeTime durchgeführt. Dagegen beginnt Pfad C innerhalb des PEs, durchläuft die Verbindungsstruktur, und endet entweder am Eingang eines anderen PEs, oder am Eingang des Blockspeichers. Zur Berechnung der gesamten Pfadverzögerung ist es daher erforderlich, zusätzlich zur Verzögerung der kombinatorischen Logik C des PEs die Signallaufzeit der Verbindungsstruktur zu bestimmen. Ein ähnliches Prinzip gilt für die Pfade, deren Endpunkt innerhalb des PEs liegen, nicht aber deren Startpunkt. Um einen auf eine Architektur abgebildeten Algorithmus komplett in Bezug auf das Zeitverhalten zu charakterisieren, ist es daher erforderlich, sowohl die PE-internen Verzögerungszeiten zwischen den Registerstufen zu bestimmen, als auch die Verzögerungen vom Eingang des PEs bis zur ersten Registerstufe, und von der letzten Registerstufe bis zum Ausgang des PEs. Weiterhin zu beachten ist, wie erwähnt, die Verzögerungszeit der Verbindungsstruktur. Diese Zeit wird in dieser Arbeit mit einem Verzögerungsmodell abgeschätzt, welches im folgenden Abschnitt vorgestellt wird.

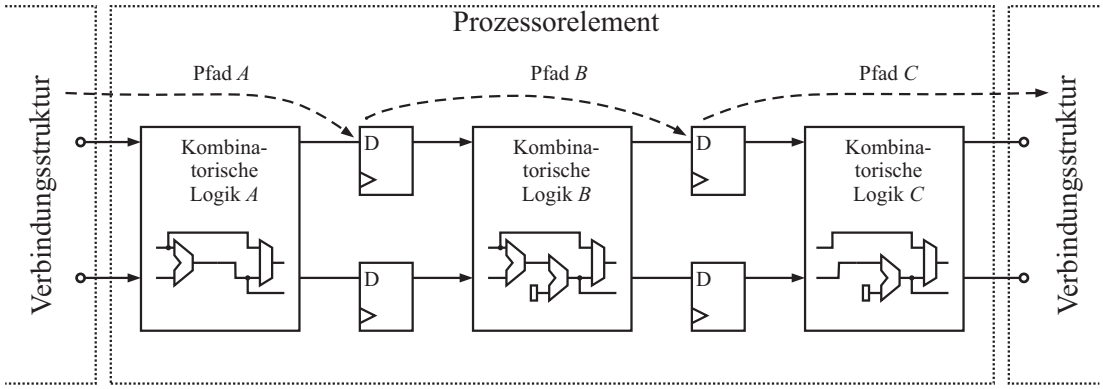


Abbildung 6.2: Timing-Analyse der rekonfigurierbaren Modellarchitekturen

Verzögerungsmodell nach Elmore

Die in Abschnitt 4.2 vorgestellte Verbindungsarchitektur ist aus unterschiedlichen Schalt- und Transferelementen aufgebaut. Zur Illustration zeigt Abbildung 6.3 ein Beispiel für einen Timing-Pfad durch die Verbindungsstruktur. Dieser Pfad beginnt an Ausgang A von $PE1$. Durch entsprechende Konfiguration des Routing-Netzwerkes wird das Signal an $PE2$ vorbei bis zum Eingang B von $PE3$ geleitet. Ein zum Zeitpunkt $t = 0$ an Punkt A gestartetes Signal passiert auf dem Weg zu Punkt B zwei Tristate-Treiber, drei Leitungssegmente, ein Transmission-Gate und einen Treiber zur Pegelregenerierung vor Eingang B von $PE3$. Wenn die Pfadverzögerung für dieses Beispiel berechnet werden soll, muss der Einfluss aller Routingelemente inklusive der Verbindungsleitungen berücksichtigt werden. Zu beachten ist ferner die Auswirkung von auf "sperrend" geschalteten Transmission-Gates und Leitungstreibern, welche mit dem Signalpfad verbunden sind, wie dies beispielsweise in den Switch- und Verbindungsblöcken der Fall ist. Diese Elemente besitzen auch im sperrenden Zustand eine Eingangskapazität, die bei der Berechnung der Pfadverzögerung zu berücksichtigen ist. Auch der Einfluss von Pfadverzweigungen ist in die Berechnung einzubeziehen, da dem betrachteten Pfad hierduch ebenfalls parasitäre Kapazitäten hinzugefügt werden. Derartige parasitäre Effekte haben unmittelbar Einfluss auf das Zeitverhalten, da das Umladen einer Kapazität C umso länger dauert, je größer der Wert von C ist.

Die genaueste simulative Methode zur Bestimmung des Timingverhaltens einer Schaltung ist die Analyse mit Hilfe von SPICE, da hierbei alle relevanten analogen Effekte berücksichtigt werden. Die Benutzung von SPICE erweist sich in diesem Fall jedoch als unpraktikabel, da die zu betrachtenden Strukturen nicht als entsprechende Modelle vorliegen, und eine Simulation aller in einem Entwurf vorkommenden Pfade erheblich zu aufwändig wäre. Wünschenswert ist dagegen die Bestimmung von Verzögerungszeiten anhand geschlossener Berechnungsvorschriften. Das häufigste in diesem Zusammenhang ver-

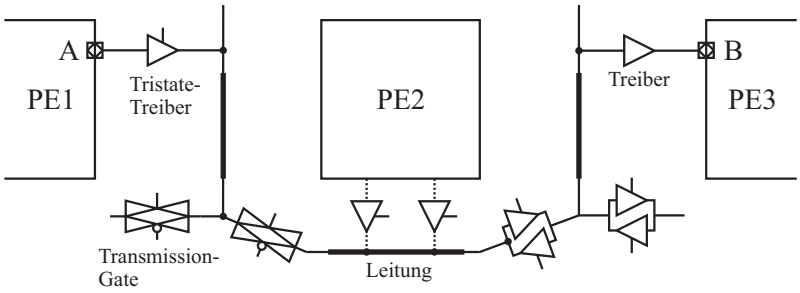


Abbildung 6.3: Beispiel für einen Timing-Pfad innerhalb der Verbindungsstruktur

wendete Verfahren ist das Verzögerungsmodell nach Elmore ([Elm48]). In dessen bereits 1948 veröffentlichtem Aufsatz werden die Signallaufzeiten, sowie die Anstiegs- und Abfallzeiten der Spannungskurven in verteilten RC-Netzwerken modelliert. Später wurde gezeigt, dass sich das Elmore-Modell auch zur Timing-Analyse von MOS-Schaltungen eignet, wenn Routing-Elemente wie Pass-Transistoren, Transmission-Gates und Leitungssegmente durch einfache RC-Glieder modelliert werden. In [KBV93] wird gezeigt, dass sich auf diese Art Pfadverzögerungen in SRAM-basierten FPGAs in guter Näherung berechnen lassen. Das Tool "VPR" [BRM99] zur FPGA-Entwurfsraumexploration basiert ebenfalls auf der Modellierung von Routing-Strukturen durch verteilte RC-Netzwerke, und der Anwendung des Elmore-Modells zur Bestimmung von Verzögerungszeiten.

Da das Elmore-Modell einfache und geschlossene Berechnungsvorschriften zur Abschätzung von Verzögerungszeiten liefert, und zudem in seiner Genauigkeit als hinreichend charakterisiert worden ist, wird es auch in dieser Arbeit zur Timing-Analyse verwendet. Die Anwendung des Modells wird im folgenden dargestellt. Zunächst soll jedoch die von Elmore [Elm48] verwendete Näherung zur Bestimmung der Signalverzögerung aufgezeigt werden, da diese leicht von der sonst üblichen Definition abweicht.

Betrachtet werden soll die Antwort $h_s(t)$ eines Systems auf die Erregung mit einem Spannungssprung am Eingang zur Zeit $t = 0$ (Abbildung 6.4). Die Signalverzögerung wird üblicherweise definiert als Zeitdifferenz zwischen den 50%-Durchgängen von Ein- und Ausgangssignal ($\tau_{50\%}$).

Die Impulsantwort $h_i(t)$ ergibt sich aus der Sprungantwort zu

$$h_i(t) = \frac{d}{dt} h_s(t) \tag{6.8}$$

Die von Elmore [Elm48] definierte Verzögerung ergibt sich als Flächenschwerpunkt der Impulsantwort $h_i(t)$:

$$\tau_{Elmore} = \int_0^\infty t \cdot h_i(t) dt \tag{6.9}$$

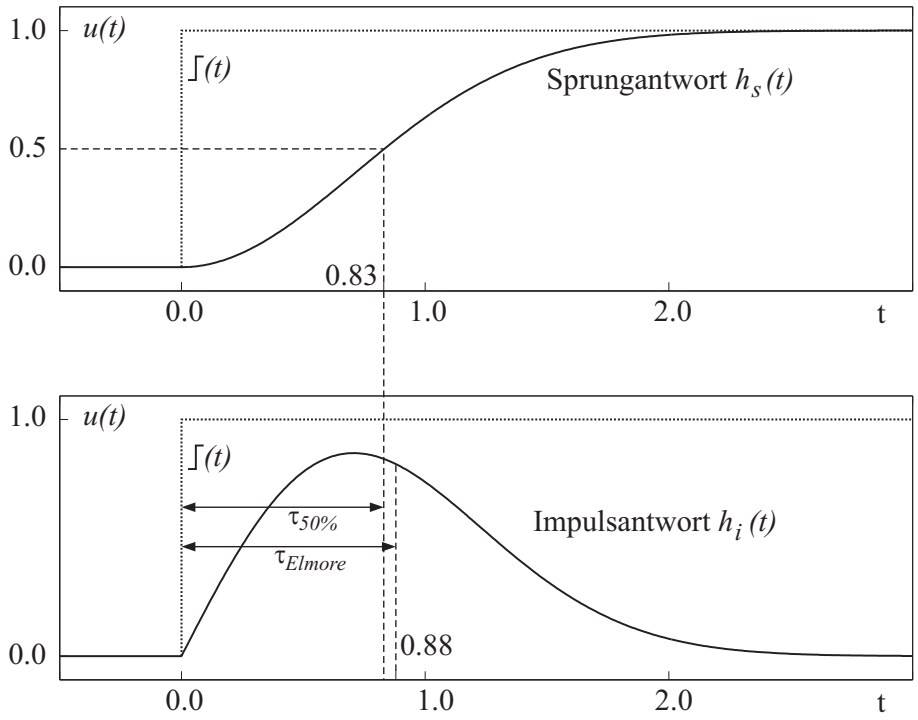


Abbildung 6.4: Zur Darstellung der Elmore-Verzögerung: Typische Sprungantwort eines Systems bei Erregung mit einem Spannungssprung bei $t = 0$ (oben), Impulsantwort (unten). Eingezeichnet sind die 50%-Verzögerung $\tau_{50\%}$ und die Elmore-Verzögerung τ_{Elmore}

In [RP83] wird nachgewiesen, dass die Sprungantwort $h_s(t)$ eines RC-Netzwerkes, welches keinen resistiven Pfad zur Masse enthält, eine monoton steigende Funktion ist. $h_i(t)$ sei die zu diesem Netzwerk gehörende Impulsantwort. Aus Gleichung 6.8 folgt, dass $h_i(t)$ eine nicht-negative Funktion ist. Weiterhin wird in [RP83] gezeigt, dass für $h_i(t)$ gilt: $\int_0^\infty h_i(t)dt = 1$. Dies sind hinreichende Bedingungen dafür, dass die Impulsantwort $h_i(t)$ rein formal als Wahrscheinlichkeitsdichtefunktion interpretiert werden kann. Die Sprungantwort $h_s(t)$ erfüllt demgegenüber die Bedingungen einer Verteilungsfunktion. Diese formale Betrachtungsweise führt zu einer Interpretation der Definitionen für die 50%-Verzögerung $\tau_{50\%}$ und die Elmore-Verzögerung τ_{Elmore} als Median, bzw. Mittelwert einer kontinuierlichen Zufallsgröße. Definitionsgemäß ergibt sich $\tau_{50\%}$ als Median der Impulsantwort $h_i(t)$, also als Wert, für den gilt: $\int_0^{\tau_{50\%}} h_i(t)dt = \frac{1}{2}$. Die in Gleichung 6.9 angegebene Elmore-Verzögerung τ_{Elmore} lässt sich dagegen als Mittelwert der Impulsantwort interpretieren (s.a. [BS95]).

Aus dieser Betrachtungsweise geht hervor, dass die Elmore-Verzögerung nur für symmetrische Impulsantworten mit der 50%-Verzögerung übereinstimmt. Für nicht-symmetrische

Impulsantworten dagegen repräsentiert τ_{Elmore} nicht exakt $\tau_{50\%}$, wie in Abbildung 6.4 illustriert.

Die formale Behandlung von $h_s(t)$ und $h_i(t)$ als Verteilungs-, bzw. Wahrscheinlichkeitsdichtefunktionen erlaubt auch eine andere Interpretation des Elmore-Delays, von der ausgehend in [RP83] eine einfache und geschlossene Formel zur Bestimmung der Signalverzögerung in RC-Netzwerken hergeleitet wird. $H(s)$ sei die zur Impulsantwort $h_i(t)$ gehörende Laplace-Transformierte. Wird $H(s)$ in eine Taylor-Reihe entwickelt, ergeben sich die Momente n-ter Ordnung der Impulsantwort als Koeffizienten der Taylor-Reihe bei $s = 0$ (siehe [ALKD04]. Diese Momente n-ter Ordnung sind in etlichen Veröffentlichungen als Verzögerungsmaß angegeben worden. Das Elmore-Delay als Näherung für die tatsächliche Verzögerung entspricht hier dem Moment erster Ordnung. Für genauere Verzögerungsmodelle werden häufig zusätzlich die Momente höherer Ordnung hinzugezogen (siehe z.B. [CPH+97], [ASB03], [ADK00]).

Eine einfache und geschlossene Formel zur Berechnung der Elmore-Verzögerung als erstes Moment der Impulsantwort von RC-Netzwerken wird in [RP83] angegeben. Demnach lässt sich das Elmore-Delay einer Kaskade von N RC-Gliedern erster Ordnung folgendermaßen bestimmen:

$$\tau_{Elmore,RC-Kette} = \sum_{i=1}^{N} C_i \sum_{j=1}^{i} R_j = \sum_{i=1}^{N} R_i \sum_{j=i}^{N} C_j \quad , \tag{6.10}$$

wobei R_i und C_i den Widerstands-, bzw. Kapazitätswerten des RC-Glieds i innerhalb der Kette entsprechen.

In verzweigten RC-Netzwerken wie der in Abbildung 6.5 gezeigten Anordnung ist der Einfluss der Zweige auf die Gesamtverzögerung des Netzwerks zu berücksichtigen. Soll beispielsweise die Elmore-Verzögerung des Pfades von Knoten A nach Knoten B der Schaltung aus Abbildung 6.5 berechnet werden, ist zu beachten, dass durch den Zweig-Abgriff nach Widerstand R_1 die kapazitive Belastung am Signaleinkopplungspunkt A erhöht wird. Durch eine einfache Erweiterung von Gleichung 6.10 lässt sich die Elmore-Verzögerung eines beliebigen Signalpfades $Pfad(a \rightarrow b)$ innerhalb eines verzweigten RC-Netzwerkes folgendermaßen bestimmen:

$$\tau_{Elmore,Pfad(a \rightarrow b)} = \sum_{i \in Pfad(a \rightarrow b)} R_i \cdot C_{i,gesamt} \tag{6.11}$$

Es wird über alle Knoten des betrachteten Pfades summiert. R_i entspricht wieder dem Widerstandswert an Knoten i, und $C_{i,gesamt}$ entspricht der gesamten kapazitiven Belastung an Knoten i.

Die Elmore-Verzögerung für Pfad $Pfad(A \rightarrow B)$ in Abbildung 6.5 ergibt sich nach Gleichung 6.11 zu:

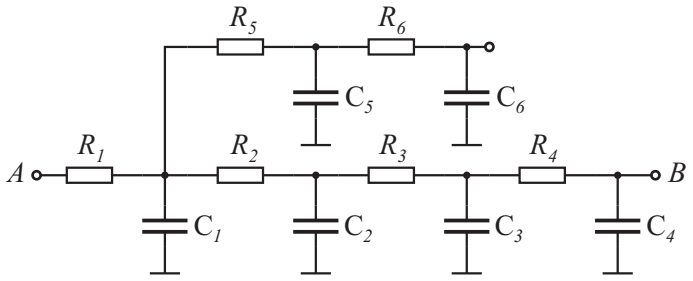


Abbildung 6.5: Beispiel für ein einfaches RC-Netzwerk

$$\begin{aligned}\tau_{Pfad(A\rightarrow B)} &= R_1 \cdot (C_1 + C_2 + C_3 + C_4 + C_5 + C_6) \\ &\quad + R_2 \cdot (C_2 + C_3 + C_4) \\ &\quad + R_3 \cdot (C_3 + C_4) \\ &\quad + R_4 \cdot C_4\end{aligned}$$

Das Verzögerungsmodell nach Elmore lässt sich in guter Näherung zur Timing-Analyse von MOS-Schaltungen verwenden. Hierbei werden die in Abschnitt 4.3.1 beschriebenen Transmission-Gates und Tristate-Treiber, sowie Leitungssegmente zur Verbindung der Strukturen als lineare RC-Glieder modelliert. Der nichtlineare Charakter der Kennlinienfelder von MOS-Transistoren wird dabei vernachlässigt. Im Folgenden werden die in dieser Arbeit angenommenen Ersatzschaltbilder für die Bauelemente der Verbindungsstruktur dargestellt. Im nächsten Unterabschnitt wird anhand von Simulationen nachgewiesen, dass die Näherung für diese Arbeit hinreichend genaue Ergebnisse liefert.

Abbildung 6.6 zeigt die Ersatzschaltbilder für ein Transmission-Gate. Hier muss zwischen leitendem und sperrendem Zustand unterschieden werden. Im leitenden Zustand wird ein einzelnes Transmission-Gate durch einen äquivalenten Widerstand R_{TG} und eine äquivalente Kapazität $C_{TG,leitend}$ modelliert. Die Kapazität $C_{TG,leitend}$ ist bei gleichbleibender Gate-Länge $L = L_{min}$ proportional zum Verhältnis W/W_{min} der Gate-Länge W zur minimalen Gate-Länge W_{min}. Der äquivalente Widerstand R_{TG} dagegen ist umgekehrt proportional zu W/W_{min}. Ein sperrendes Transmission-Gate wird durch zwei voneinander getrennte, identische Kapazitäten $C_{TG,sperrend}$ modelliert.

Für Schaltungen, die Buffer oder Tristate-Treiber enthalten, ist eine leicht modifizierte Form des Verzögerungsmodells erforderlich. Ein Buffer besitzt aufgrund des internen Aufbaus aus mehreren kaskadierten Inverter-Stufen (Abbildung 4.6) eine eingeprägte, intrinsische Laufzeit $\tau_{Buf,int}$. Diese ist gleich der Verzögerungszeit des kapazitiv unbelasteten Buffers, und muss bei der Bestimmung des Verzögerungsverhaltens eines Pfades berücksichtigt werden. In [OC96] wird zu diesem Zweck eine modifizierte Form des Elmore-Modells vorgeschlagen. Die zur Bestimmung der Elmore-Verzögerung eines Pfades zu verwendende

Abbildung 6.6: Ersatzschaltbilder für ein Transmission-Gate im leitenden und im sperrenden Zustand

Formel ergibt sich unter Berücksichtigung der eingeprägten Buffer-Laufzeit $\tau_{Buf,int}$ aus Gleichung 6.11 zu:

$$\tau_{Elmore,Pfad(a\rightarrow b),mod} = \sum_{i \in Pfad(a\rightarrow b)} [R_i \cdot C_{i,gesamt} + \tau_{Buf,int,i}] \tag{6.12}$$

Für den Fall, dass Schaltelement i des Pfades ein Buffer ist, wird die entsprechende eingeprägte Laufzeit $\tau_{Buf,int,i}$ zur gesamten Pfadverzögerung hinzuaddiert. Zu beachten ist zusätzlich, dass ein Timing-Pfad durch einen Buffer vom hinter dem Treiber liegenden Netzwerk kapazitiv getrennt wird. $C_{i,gesamt}$ in Gleichung 6.12 bezeichnet also lediglich die Kapazität zwischen dem Netzwerkknoten i und dem nächsten auf dem Signalpfad liegenden Buffer.

Das korrespondierende Ersatzschaltbild eines Buffers für das Elmore-Modell ist in Abbildung 6.7 gezeigt. Aufgrund des unsymmetrischen Aufbaus des Schaltelementes sind die entsprechenden Ein- und Ausgangsskapazitäten $C_{Buf,in}$ und $C_{Buf,out}$ voneinander verschieden. Die Abhängigkeit der Bufferverzögerung von der ausgangsseitigen kapazitiven Belastung wird durch den Widerstandswert R_{Buf} modelliert ([OC96]).

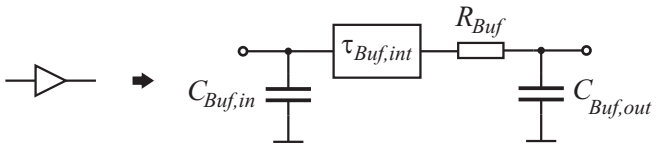


Abbildung 6.7: Ersatzschaltbild für einen Leitungstreiber unter Berücksichtigung der eingeprägten Buffer-Laufzeit $\tau_{Buf,int}$

Für Tristate-Treiber wird in dieser Arbeit die in Abbildung 4.6b gezeigte häufige Implementierungsform verwendet, bei der durch den ausgangsseitigen Pass-Transistor oder das Transmission-Gate der Ausgang mit dem Eingang verbunden oder von ihm hochohmig getrennt werden kann. Bei der Bestimmung der Pfadverzögerung werden Tristate-Treiber daher in dieser Arbeit durch Kaskadierung eines Buffers und eines Transmission-Gates modelliert.

Ein Leitungssegment wird wie in Abbildung 6.8 gezeigt durch einen äquivalenten Widerstand $R_{Leitung}$ und eine Kapazität $C_{Leitung}$ modelliert.

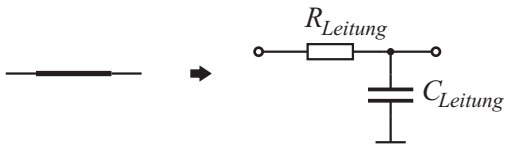


Abbildung 6.8: Ersatzschaltbild für ein Leitungssegment

Die das Verzögerungsverhalten der Verbindungsstruktur bestimmenden charakteristischen Größen R_{TG}, $C_{TG,leitend}$, $C_{TG,sperrend}$, R_{Buf}, $C_{Buf,in}$, $C_{Buf,out}$, $\tau_{Buf,int}$, $R_{Leitung}$ und $C_{Leitung}$ sind durch Schaltungssimulationen bestimmt worden. Die hierbei angewandten Methoden werden kurz in Anhang A dargestellt.

Evaluierung der Genauigkeit des Elmore-Modells

Das im vorigen Abschnitt beschriebene Verzögerungsmodell für die Verbindungsstruktur auf Basis der Elmore-Verzögerung ist durch die geschlossene Berechnungsvorschrift einfach zu implementieren, stellt jedoch eine Näherung dar. Daher existieren eine große Zahl von Vorschlägen für genauere Modelle (z.B. [CPH+97], [ASB03], [ADK00]), von denen einige neben den hier betrachteten Kapazitäten und ohmschen Widerständen auch Induktivitäten berücksichtigen (siehe z.B. [KM97]). Insbesondere ist bekannt, dass das Elmore-Modell für Netzwerke mit resistiv abgeschirmten Kapazitäten sehr unakkurat sein kann ([ASNC04]). Zur Demonstration, dass das Elmore-Modell für die in dieser Arbeit betrachteten Architekturen hinreichend genaue Ergebnisse liefert, sind für einige typische Teile der Verbindungsstruktur SPICE-Simulationen durchgeführt worden. Die gemessenen Verzögerungszeiten wurden mit den durch das Elmore-Modell abgeschätzten Zeiten verglichen. Alle in diesem Abschnitt exemplarisch betrachteten Netzwerke sind am Ende des Signalpfades mit der Eingangsstufe eines Buffers verbunden, wie dies in der Praxis ebenfalls der Fall ist. Die Eingangskapazität dieses Buffers wurde bei der Bestimmung der Signallaufzeiten berücksichtigt.

Die Schaltzeiten der in Abschnitt 4.3.2 dargestellten Multiplexer an den PE-Eingängen werden durch die Kaskadierung einer bestimmten Anzahl von Transmission-Gates bestimmt. Tabelle 6.1 zeigt beispielhaft die Ergebnisse der Untersuchungen für den einfachen Fall einer Kaskade aus N_{TG} Transmission-Gates. Dargestellt sind die absoluten Werte der mit SPICE gemessenen 50%-Verzögerungen τ_{SPICE}, das korrespondierende Elmore-Delay τ_{Elmore}, sowie die jeweilige prozentuale Abweichung. Aus der Tabelle geht hervor, dass die Elmore-Verzögerungen für ein einzelnes Transmission-Gate, und für eine Kaskade aus zwei Transmission-Gates relativ deutlich um 56.1%, bzw. 21.7% von den SPICE-Werten

abweichen. Da der absolute Wert dieser beiden Verzögerungszeiten jedoch nur im Bereich zwischen $10ps$ und $40ps$ liegt, fällt diese recht hohe prozentuale Abweichung bei der Bestimmung des Verzögerungsverhaltens der Gesamtarchitektur, deren Signalverzögerungen im Bereich mehrerer Nanosekunden liegen, kaum ins Gewicht, und kann vernachlässigt werden. Für alle anderen Fälle liegt die Abweichung im akzeptablen Bereich zwischen 8.9% und 13.2%.

Tabelle 6.1: Verzögerungszeiten für Ketten aus N_{TG} kaskadierten Transmission-Gates. τ_{SPICE}: Mit SPICE-Simulation gemessene Verzögerungszeit, τ_{Elmore}: mit dem Elmore-Modell abgeschätzte Verzögerungszeit

N_{TG}	τ_{SPICE}/ps	τ_{Elmore}/ps	Abweichung
1	10.6	16.5	56.1%
2	35.1	42.7	21.7%
3	72.3	81.8	13.2%
4	121.5	134.0	10.3%
6	254.9	277.5	8.9%
8	432.8	473.2	9.3%
10	655.0	721.1	10.1%

Die Verbindungsstruktur selbst besteht aus Transmission-Gates und Buffern, sowie die einzelnen Schaltelemente verbindenden Leitungssegmenten. Die modellierten Verzögerungszeiten sollen ebenfalls mit Simulationen verifiziert werden. Tabelle 6.2 zeigt zunächst die Ergebnisse für den Fall einer alternierenden Kaskade aus N_{TGL} Transmission-Gates und Leitungssegmenten. Die Kapazität eines Leitungssegmentes wird einerseits durch die Leitung selbst, und andererseits durch die parasitären Kapazitäten bestimmt, die durch eventuell mit dem Leitungssegment verbundenen Schaltelemente entstehen und der Gesamtkapazität hinzuaddiert werden müssen (siehe Abbildung 6.3). Wie Tabelle 6.2 zeigt, ergeben sich für die relevanten Fälle der Kaskaden aus ein bis maximal sechs Transmission-Gates und Leitungssegmenten Abweichungen der Elmore-Verzögerungen von höchstens 12.4% von den mit SPICE simulierten Verzögerungszeiten.

Tabelle 6.3 zeigt Simulationsergebisse für den Fall, dass neben Transmission-Gates auch Leitungstreiber in die Verbindungsstruktur integriert sind, wie dies bei den hier betrachteten Architekturen der Fall ist. Gemessen wurden die Verzögerungszeiten von alternierenden Kaskaden aus Transmission-Gates, Leitungssegmenten und Treibern. Durch Anwendung von Gleichung 6.12 wurde der Einfluss der Leitungstreiber auf die Verzögerungszeit berücksichtigt. Zu beachten ist hierbei sowohl die eingeprägte Buffer-Laufzeit $\tau_{Buf,int,i}$, als auch die Tatsache, dass ein Buffer einen Timing-Pfad vom hinter dem Treiber liegenden

Tabelle 6.2: Verzögerungszeiten für Ketten aus N_{TGL} kaskadierten Transmission-Gates und Leitungssegmenten. τ_{SPICE}: Mit SPICE-Simulation gemessene Verzögerungszeit, τ_{Elmore}: mit dem Elmore-Modell abgeschätzte Verzögerungszeit

N_{TGL}	τ_{SPICE}/ps	τ_{Elmore}/ps	Abweichung
1	202.0	177.0	-12.4%
2	590.9	524.0	-11.3%
3	1136.8	1044.5	-8.1%
4	1830.6	1738.5	-5.0%
5	2671.1	2606.0	-2.4%
6	3574.9	3664.0	+2.5%

Netzwerk kapazitiv trennt. Die Ergebnisse der Elmore-Verzögerungen der hier betrachteten Schaltungen liegen innerhalb von 7.6% der Ergebnisse aus den SPICE-Simulationen.

Tabelle 6.3: Verzögerungszeiten für Ketten aus N_{TGLB} kaskadierten Transmission-Gates, Leitungssegmenten und Buffern. τ_{SPICE}: Mit SPICE-Simulation gemessene Verzögerungszeit, τ_{Elmore}: mit dem Elmore-Modell abgeschätzte Verzögerungszeit

N_{TGLB}	τ_{SPICE}/ps	τ_{Elmore}/ps	Abweichung
1	589.2	544.6	-7.6%
2	1391.7	1393.5	0.1%
3	2195.2	2242.4	2.2%
4	2997.8	3091.3	3.1%
5	3800.0	3940.2	3.7%
6	4602.0	4789.0	4.0%
7	5403.8	5638.0	4.3%

Da durch den Aufbau der Verbindungsstruktur die Möglichkeit vorgesehen ist, verzweigte Verbindungen aufzubauen, sind auch die modellierten Verzögerungszeiten von Verzweigungen anhand von Simulationen verifiziert worden. Die Ergebnisse sind in Tabelle 6.4 dargestellt. Simuliert wurden die Verzögerungszeiten einer Kaskade aus drei Transmission-Gates und Leitungssegmenten. Zwischen den einzelnen Elementen wurden Abzweigungen in Form von zwei Transmission-Gates und Leitungssegmenten geschaltet, welche eine Erhöhung der effektiven Kapazität des Pfades, und somit einen Anstieg der Signallaufzeit zur

Folge haben. Da die Positionen der Abzweigungen ebenfalls einen Einfluss auf die Verzögerungszeit haben, wurden verschiedene Fälle betrachtet, bei denen die Abzweige jeweils an unterschiedlichen Positionen erfolgten. Der maximale Fehler durch das Elmore-Modell bei den hier betrachteten Fällen beträgt 9.8%.

Tabelle 6.4: Verzögerungszeiten verzweigter Transmission-Gate-Netzwerke. Der Abzweig besteht aus einer Kaskade von zwei Transmission-Gates und Leitungssegmenten nach den Elementen 1, 2 oder 3 der Transmission-Gate-Kette. τ_{SPICE}: Mit SPICE-Simulation gemessene Verzögerungszeit, τ_{Elmore}: mit dem Elmore-Modell abgeschätzte Verzögerungszeit

Abzweig nach Element	τ_{SPICE}/ps	τ_{Elmore}/ps	Abweichung
1	1554.5	1402.8	-9.8%
2	1784.8	1750.0	-1.9%
3	2037.5	2097.1	2.9%
1,2	2295.3	2097.1	-8.6%
1,3	2575.0	2444.3	-5.1%
2,3	2805.6	2791.4	-0.5%
1,2,3	3351.2	3138.5	-6.3%

Die in den Tabellen 6.1 bis 6.4 dargestellten Ergebnisse zeigen, dass das Verzögerungsmodell nach Elmore, welches zur Bestimmung der Signallaufzeiten der Verbindungsstruktur herangezogen wurde, zwar die Zeiten nicht SPICE-genau voraussagen kann, die sich ergebenden Abweichungen jedoch für eine Abschätzung klein genug sind. Es wurden einige typische Strukturen der in Abschnitt 4.2 vorgestellten Verbindungsstruktur exemplarisch zur Verifikation des Elmore-Modells herangezogen. Die Differenzen zwischen dem Elmore-Modell und der mit SPICE ermittelten 50%-Signallaufzeit liegen in fast allen relevanten Fällen bei maximal $10 - 12\%$.

Methodik bei der Analyse des Zeitverhaltens der Modellarchitekturen

Die statische Timing-Analyse der in dieser Arbeit untersuchten Modellarchitekturen erfolgt mit zwei unterschiedlichen Verfahren, die sich aus den besonderen Randbedingungen des verwendeten Modells ergeben. Da die Prozessorelemente mit einem Semi-Custom-Entwurf auf Basis von Standardzellen implementiert wurden, ist es möglich, das Zeitverhalten eines PEs mit Hilfe eines Tools zur statischen Timing-Analyse zu untersuchen. In dieser Arbeit wird hierzu das Programm PrimeTime von Synopsys verwendet. Mit diesem Werkzeug

können Simulationen zum Zeitverhalten auf Standardzellebene durchgeführt werden. Die Genauigkeit von PrimeTime wird unter der Voraussetzung, dass die zugrunde liegende Zellbibliothek genau in Bezug auf Gatterlaufzeiten charakterisiert ist, von der Herstellerfirma auf $3-5\%$ innerhalb der Genauigkeit von SPICE-Simulationen angegeben.

Die Verbindungsstruktur der untersuchten Modellarchitekturen liegt im Gegensatz zu den Prozessorelementen nicht als synthetisierte Gatternetzliste vor, da die Struktur mit bidirektional arbeitenden Transmission-Gates auch Full-Custom-Elemente enthält. Da ein solcher Full-Custom-Entwurf für jede zu untersuchende Architekturvariante erheblich zu aufwändig wäre, wird das Verzögerungsverhalten der Verbindungsstruktur durch ein analytisches Modell nachgebildet. Hierbei kommt das Verzögerungsmodell nach Elmore zum Einsatz, deren Anwendung auf die Modellarchitekturen im vorigen Abschnitt beschrieben wurde.

Die Ergebnisse der Timing-Simulationen für die Basiszellen und des analytischen Modells für die Verbindungstruktur fließen in die Untersuchung des Zeitverhaltens der Gesamtstruktur ein, bei der die maximale Taktfrequenz ermittelt wird, mit welcher die rekonfigurierbare Architektur für einen bestimmten Algorithmus betrieben werden kann.

6.1.3 Modellierung der Verlustleistung

Seit Beginn der CMOS-Ära waren lange Zeit ein minimaler Flächenbedarf und eine maximale Verarbeitungsgeschwindigkeit die primären Entwurfsziele. Aufgrund ständig steigender Transistorzahlen und Taktfrequenzen, und des großen Marktes für tragbare Geräte ist heute in vielen Fällen neben einem geringen Flächenbedarf eine geringe Verlustleistungsaufnahme das wichtigste Ziel, da diese unmittelbaren Einfluss auf die Akkulaufzeit des Geräts besitzt. Der Datendurchsatz hat demgegenüber meist den Charakter einer Randbedingung, die durch die Spezifikation vorgegeben ist, und die eingehalten werden muss. Darüber hinaus werden auch andere Faktoren wie beispielsweise die Wärmeentwicklung eines ICs, die wiederum die Ausfallsicherheit des Bausteins beeinflusst, durch die aufgenommene Leistung bestimmt. Bezogen auf die vorliegende Arbeit und rekonfigurierbare Architekturen ist daher eine Analyse der Leistungsaufnahme erforderlich.

Für diese Arbeit ist ein bibliotheksbasiertes Verlustleistungsmodell erstellt worden worden. In diesem Abschnitt werden die bei der Entwicklung des Modells angewandten Methoden und die Vorgehensweise bei der Untersuchung der Verlustleistung der Architekturen dargestellt. Nach einer theoretischen Einführung in die Quellen der Leistungsaufnahme von CMOS-Schaltungen wird die angewandte Methodik vorgestellt, dessen Genauigkeit in einem anschließenden Abschnitt evaluiert wird.

Leistung und Energie

Die Augenblicksleistung $p(t)$ ist proportional zum augenblicklichen Strom $i(t)$ und der Spannung $u(t)$, die bei integrierten Schaltungen in der Regel der konstanten Betriebsspannung V_{DD} entspricht (siehe z.B. [WH05],[KSW95]):

$$p(t) = i(t) \cdot V_{DD} \tag{6.13}$$

Von Interesse bei Analysen der Leistungsaufnahme ist in den meisten Fällen weniger die Augenblicksleistung $p(t)$, sondern die über einen längeren Zeitraum T gemessene durchschnittliche Leistung $\bar{P}$:

$$\bar{P} = \frac{1}{T} \int_0^T i(t) V_{DD} dt \tag{6.14}$$

Die Energie ist das zeitliche Integral über die Leistung:

$$E = \int_{t_1}^{t_2} p(t) dt = \int_{t_1}^{t_2} i(t) V_{DD} dt = \bar{P} \cdot (t_2 - t_1) \tag{6.15}$$

Die SI-Einheit der Leistung ist Watt (W), die der Energie Wattsekunden (Ws) oder Joule (J).

Leistungsaufnahme von CMOS-Schaltungen

Die Verlustleistung in komplementären MOS-Schaltungen lässt sich in zwei Anteile aufteilen. Dies sind die statische und die dynamische Verlustleistung:

$$P_{gesamt} = P_{statisch} + P_{dynamisch} \tag{6.16}$$

Die Zusammensetzung der beiden Anteile lässt sich anhand des Schaltbildes eines kapazitiv belasteten Inverters darstellen (Abbildung 6.9). Der Vorteil von CMOS-Schaltungen liegt unter anderem darin, dass im Idealfall immer einer der beiden Transistoren gesperrt ist, und somit kein Strom durch die Schaltung fließt, was sich nach Gleichung 6.13 in einer verschwindenden Verlustleistung äußert. In realen Schaltungen existieren jedoch eine ganze Reihe verschiedener Ströme, die für einen entsprechenden Leistungsverbrauch sorgen. Sekundäre Effekte wie beispielsweise der Unterschwellenleckstrom, oder Tunneleffekte durch die Gate-Oxidschicht sorgen auch im statischen Fall, in denen kein Umschalten des Inverters stattfindet, für einen Stromfluss. Bei zunehmender Verkleinerung der Strukturgrößen moderner Prozesstechnologien auf unter $100nm$ wird die durch statische Ströme hervorgerufene Verlustleistung zunehmend zum Problem. Bei älteren Technologien dagegen wird die Verlustleistung im Wesentlichen durch dynamische Effekte dominiert. Die dynamische Leistungsaufnahme resultiert zum größten Teil daraus, dass während jedes Schaltvorganges

die ausgangsseitigen parasitären Lastkapazitäten C_L umgeladen werden müssen, was über einen dynamischen Strom i_d erfolgt. Desweiteren sind beim Umschaltvorgang kurzzeitig beide Transistoren leitend, was wiederum den Querstrom i_q hervorruft.

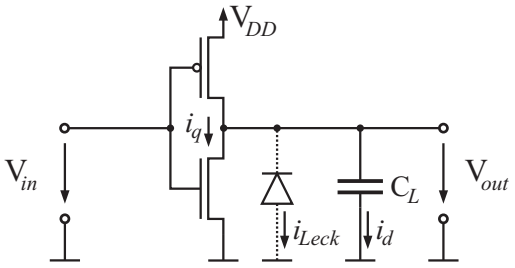


Abbildung 6.9: Quellen des Leistungsverlustes in CMOS-Schaltungen, dargestellt am Beispiel eines kapazitiv belasteten Inverters

Die dynamische Verlustleistungsaufnahme $P_{dynamisch}$ ist offenkundig abhängig von der Betriebspannung V_{DD}, der Größe der Lastkapazität C_L, der Taktfrequenz f, sowie von der mittleren Wahrscheinlichkeit, mit der ein Schaltvorgang stattfindet. Die Beziehung wird in dieser Arbeit zur Bestimmung der Verlustleistung der Verbindungsstruktur verwendet, und kann folgendermaßen hergeleitet werden (siehe z.B. [WH05]). Angenommen wird, dass eine Lastkapazität C_L zwischen V_{DD} und Masse umgeladen werden soll. Die Anzahl der Umladevorgänge in einem Zeitintervall T beträgt $N_U = T \cdot \bar{f}_s$, wobei $\bar{f}_s$ die mittlere Frequenz ist, mit der Umschaltvorgänge stattfinden. Da während des Ladevorgangs ein Strom von der Versorgungsleitung zur Last fließt, und während des Entladevorgangs von der Last zur Masse, wird während eines solchen Vorgangs eine Ladung von

$$Q_L = C_L V_{DD} \tag{6.17}$$

von der Versorgungsleitung zur Masse transportiert. Aus Gleichung 6.14 folgt für die dynamische Verlustleistung aus den Umschaltvorgängen:

$$P_{dynamisch} = \frac{V_{DD}}{T} \int_0^T i_d(t)dt \tag{6.18}$$

Aus $Q_L = \int_0^T i_d(t)dt = C_L V_{DD}$ folgt:

$$P_{dynamisch} = \frac{V_{DD}}{T} \cdot T\bar{f}_s C_L V_{DD} = C_L {V_{DD}}^2 \bar{f}_s \tag{6.19}$$

Gleichung 6.19 bestätigt die intuitive Vermutung einer direkten Proportionalität der Verlustleistung zur Häufigkeit der Schaltvorgänge. Die Verlustleistung ist daher in starkem Maße von der Art des Eingangssignals einer Schaltung abhängig.

Eine Vielzahl von Forschungsarbeiten befasst sich mit dem Thema, die Verlustleistung integrierter Schaltungen zu minimieren. Eine solche Optimierung kann auf allen Ebenen des Entwurfs stattfinden. So ist beispielsweise auf Schaltungsebene auf eine geringe Schaltaktivität zu achten, da die Verlustleistung nach Gleichung 6.19 direkt proportional zur Aktivität ist. Ebenso sind unerwünscht auftretende temporäre Zwischenzustände (Glitches) zu vermeiden, die ebenfalls die Leistung beeinflussen können. Da ein großer Teil der Verlustleistung durch das Taktsignal verursacht wird, werden Methoden wie das Clock-Gating verwendet, um temporär unbenutzte Blöcke abzuschalten. Auf Layout-Ebene ist auf geringe Lastkapazitäten zu achten. Eine niedrige Versorgungsspannung wirkt sich aufgrund der quadratischen Abhängigkeit (Gleichung 6.19) ebenfalls positiv auf die Verlustleistung aus. Für tiefergehende Darstellungen des Low-Power-Design sei auf die einschlägige Literatur verwiesen.

Bibliotheksbasierte Methodik zur Abschätzung der Verlustleistung

Ähnlich wie bei der statischen Timing-Analyse existieren auch zur Abschätzung der Verlustleistung eine Vielzahl von Methoden, die sich in Aufwand und Genauigkeit unterscheiden. In jedem Fall ist aufgrund der im obigen Abschnitt bereits erwähnten Datenabhängigkeit der Leistungsaufnahme die Art des Eingangssignals zu berücksichtigen, weshalb auch von einer dynamischen Leistungsanalyse gesprochen wird. Der genaueste simulative Ansatz ist, wie bei der Timing-Analyse, eine SPICE-Simulation der Schaltung, die jedoch aufgrund des erforderlichen Aufwands, der Simulationsdauer, und der Nichtverfügbarkeit geeigneter SPICE-Modelle für jeden Schaltungsteil in den meisten Fällen ausscheidet. Weniger genau, dafür jedoch universeller einsetzbar, sind Gatterebenen-Simulatoren wie PrimePower von Synopsys, die mit einer synthetisierten Gatternetzliste und detaillierten Power-Bibliotheksmodellen für jede Standardzelle eine Leistungsanalyse unter Berücksichtigung der Eingangsdaten auf Gatterebene durchführen. Darüber hinaus existieren hybride Simulatoren wie Synopsys NanoSim (vormals PowerMill) oder Mentor Graphics Mach PA, die mit SPICE-Bibliotheken größere Schaltungen mit guter Genauigkeit simulieren können.

Simulations-basierte Verfahren erfordern stets mindestens die Verfügbarkeit einer Gatternetzliste der jeweils betrachteten Schaltung, die jedoch speziell bei Entwurfsraumexplorationen häufig nicht zur Verfügung steht. Abhilfe schaffen hier beispielsweise bibliotheksbasierte Verfahren. Der Ansatz liegt darin, einzelne funktionale Einheiten einer Schaltung in Bezug auf den Leistungsverbrauch zu charakterisieren, und die Ergebnisse in einer Bibliothek abzulegen. Dabei ist jeweils auf eine geeignete Parametrisierung zu achten, um beispielsweise unterschiedliche Wortbreiten bei verschiedenen Schaltaktivitäten zu berücksichtigen. In [Lan94] wird durch eine entsprechende Differenzierung zusätzlich der Tatsache Rechnung getragen, dass die MSBs einen anderen Einfluss auf die Leistungsaufnahme besitzen als die LSBs.

In dieser Arbeit wurden für die einzelnen Schaltungsteile jeweils geeignete Methoden ausgewählt, um ein einfach zu handhabendes Modell zur Abschätzung der Verlustleistung der Gesamtarchitektur zu erhalten. Aus Sicht der Verlustleistungsmodellierung besteht die Gesamtarchitektur aus drei verschiedenen Komponenten. Dies sind die Datenpfade der Prozessorzellen, die Verbindungsstruktur und die Speicherblöcke. Die gesamte Verlustleistung P_{gesamt} setzt sich additiv aus den Ergebnissen der einzelnen Komponenten zusammen:

$$P_{gesamt} = P_{Datenpfade} + P_{Verbindungsstruktur} + P_{Speicher} \tag{6.20}$$

Für die drei Einzelkomponenten wurden folgende Methoden zur Modellierung der Verlustleistung angewandt:

- *Leistungsaufnahme der Datenpfade.* Für die Datenpfade wurde ein bibliotheksbasiertes Modell gewählt. Die einzelnen Komponenten eines Datenpfades wie Multiplizierer, Addierer oder Register wurden durch Simulationen in Bezug auf die Leistungsaufnahme charakterisiert, und die Simulationsergebnisse in parametrisierten Bibliotheken abgelegt. Die Bibliotheken wiederum wurden in den Array-Analyzer integriert, so dass mit diesem unter Bereitstellung der Wechselwahrscheinlichkeiten der Eingangssignale und der PE-Struktur die Leistungsanalyse erfolgen kann.

- *Leistungsaufnahme der Verbindungsstruktur.* Die durch die Verbindungsstruktur hervorgerufene Verlustleistung wurde durch eine Abschätzung der Größe der jeweils umzuladenden Kapazitätswerte, der jeweiligen Konfiguration, sowie den an den PE-Ausgängen ermittelten Wechselwahrscheinlichkeiten ermittelt.

- *Leistungsaufnahme der Speicherblöcke.* Die Verlustleistung der Speicherblöcke wurde aus den Tabellen der entsprechenden Datenblätter entnommen.

Das in den Array-Analyzer integrierte Modul zur Bestimmung der Verlustleistung der Datenpfade benötigt eine Reihe von Informationen, die teilweise durch den Benutzer zur Verfügung gestellt werden müssen, und teilweise automatisch berechnet werden. Abbildung 6.10 verdeutlicht das hinter der Methodik stehende Prinzip. Die Einzelkomponenten sind als "Black-Boxes" vom Typ A, B und C dargestellt, deren interner Aufbau zunächst irrelevant ist. Die Boxen könne beispielsweise einen Multiplizierer oder einen Addierer darstellen. Wie oben bereits dargestellt, wird die Verlustleistung in starkem Maße von der Anzahl, bzw. der Wahrscheinlichkeit der Bitwechsel an den Eingängen beeinflusst (*Wechselwahrscheinlichkeit*, häufig auch als *Schaltaktivität* bezeichnet). Theoretisch müssten daher alle $1 \rightarrow 0$ oder $0 \rightarrow 1$ Übergänge jedes einzelnes Bits betrachtet werden, was von simulationsbasierten Tools wie PrimePower an jedem Gattereingang auch so gehandhabt wird. Bei einer Datenwortbreite von 16 Bits wäre ein solches Vorgehen jedoch erheblich zu aufwändig. Stattdessen werden sowohl zeitlich als auch örtlich Vereinfachungen vorgenommen.

Es wird angenommen, dass die Eingangs-Wechselwahrscheinlichkeiten über einen längeren Zeitraum betrachtet einen konstanten Durchschnittswert annehmen. Die Wechselwahrscheinlichkeit σ_b eines einzelnen Bits ist daher die über einen langen Zeitraum gemittelte Wahrscheinlichkeit, mit der dieses Bit seinen Wert ändert. Demnach besitzt beispielsweise das Taktsignal die höchstmögliche Wechselwahrscheinlichkeit von 100%. Weiterhin werden keine einzelnen Bits betrachtet, sondern Gruppen von Bits. Mit diesen vereinfachenden Annahmen wird die Wechselwahrscheinlichkeit σ einer Gruppe von Bits definiert als die über alle Bits gemittelte Langzeit-Wechselwahrscheinlichkeit.

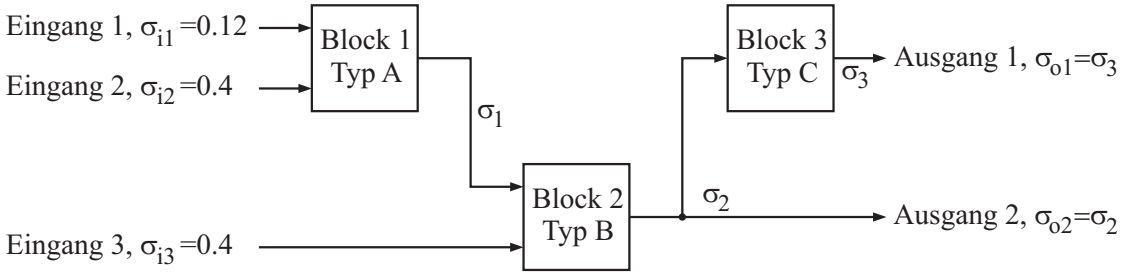


Abbildung 6.10: Illustration zur Verlustleistungsschätzung der Datenpfade

Besitzt ein Architekturblock beispielsweise zwei Dateneingänge mit jeweils 16 Bit Wortbreite, so muss im Bibliotheksmodell für diesen Block für alle möglichen Gruppen-Wechselwahrscheinlichkeiten jedes der beiden Eingänge die entsprechende Verlustleistung abrufbar sein. Beispielsweise besitzt Block 1 in Abbildung 6.10 zwei Eingänge, von denen die Wechselwahrscheinlichkeiten bekannt sind, so dass die Abschätzung für die Verlustleistung aus einer Tabelle ermittelt werden kann. Block 2 besitzt ebenfalls zwei Eingänge, von denen jedoch nur die Wechselwahrscheinlichkeit des zweiten Eingangs bekannt ist, der andere Eingang liest den Ausgang von Block 1. Es ist daher nicht ausreichend, nur die Verlustleistung für verschiedene Wechselwahrscheinlichkeiten in einer Bibliothek abzulegen. Vielmehr müssen zusätzlich für alle Kombinationen der Eingangs-Wechselwahrscheinlichkeiten die entsprechenden Wahrscheinlichkeiten an den Ausgängen in einer zweiten Bibliothek abgelegt werden. Mit Kenntnis der Wechselwahrscheinlichkeiten an den Eingängen einer Schaltung, beispielsweise eines Datenpfades, und der Struktur der Schaltung, kann so die Verlustleistung der gesamten Schaltung bestimmt werden.

In dieser Arbeit sind für alle in den Datenpfaden vorkommenden Architekturblöcke wie Multiplizierer, Addierer, Register und Multiplexer die beiden beschriebenen Bibliotheken durch Synthese des Blockes auf die 0.18μm-Prozesstechnologie (*typical*-Prozess) und anschließende PrimePower-Simulation bestimmt worden. Die Bibliotheken wiederum wurden in den Array-Analyzer integriert. Da jeweils nur bestimmte Kombinationen von Eingangs-Wechselwahrscheinlichkeiten betrachtet werden können, werden entsprechende Zwischenwerte durch Interpolation bestimmt.

In [Lan94] wird empfohlen, bei der Abschätzung nicht die komplette Busbreite zu einer Gruppe von Bits zusammenzufassen, sondern zusätzlich die unterschiedliche Auswirkung der Wechselwahrscheinlichkeiten der MSBs und der LSBs zu berücksichtigen. Begründet wird dies durch Untersuchungsergebnisse, nach denen Bits mit niedrigerer Wertigkeit eine signifikant höhere Wechselwahrscheinlichkeit besitzen als Bits mit höherer Wertigkeit, und demzufolge einen anderen Einfluss auf den Energieverbrauch haben. Bezüglich der Multiplizierer wurde dies in der vorliegenden Arbeit berücksichtigt. Bei den Addierern konnte dagegen kein wesentlicher Unterschied zwischen dem Einfluss der MSBs und der LSBs festgestellt werden, so dass eine solche Unterscheidung für die Addierer nicht erfolgt ist.

Beim Multiplizierer ergibt sich zusätzlich die Besonderheit, dass die Verlustleistung nicht nur von den Wechselwahrscheinlichkeiten, sondern auch von den Eingangsdaten selbst, bzw. vom Koeffizienten abhängig ist. In vielen Fällen, beispielsweise bei der parallelen Implementierung eines FIR-Filters, ändert sich der Wert eines Eingangs eines Multiplizierers nicht. Die Annahme der Wechselwahrscheinlichkeit von Null würde in diesem Fall jedoch falsche, bzw. ungenaue Ergebnisse liefern. So ist die Verlustleistung eines Multiplizierers bei einem stark mit Nullen besetzten Koeffizienten erheblich geringer als bei einem Koeffizienten, der mehr Einsen enthält. Um dieser Tatsache Rechnung zu tragen, wurde eine Gewichtsfunktion eingeführt, mit der das Ergebnis des Leistungswertes korrigiert werden kann. Alle Bibliotheksmodelle für Multiplizierer beziehen sich in dieser Arbeit zunächst auf einen Koeffizienten, der acht Einsen und acht Nullen enthält. Abbildung 6.11 zeigt den durch Simulationen ermittelten Verlauf der Gewichtsfunktion für andere 1-0-Verhältnisse, exemplarisch für einen Carry-Save-Array-Multiplizierer.

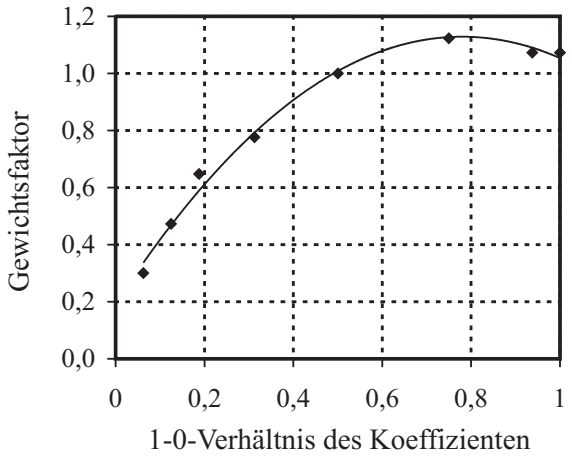


Abbildung 6.11: Berücksichtigung des Koeffizienten beim Bibliotheksmodell des Leistungsverbrauchs für einen Carry-Save-Array-Multiplizierer

Alle Datenpfadelemente der in Kapitel 5 vorgestellten Basiszellen sind durch Simulationen bei unterschiedlichen Aktivitäten der jeweiligen Eingänge auf ihre Leistungsaufnahme hin charakterisiert worden. Die Modelle wurden parametrisiert in Bibliotheken abgelegt. Abbildung 6.12 zeigt die Ergebnisse für einen Wallace-Tree- und einen Carry-Save-Array-Multiplizierer, zwei Varianten eines Carry-Lookahead-Addierers, sowie für einen 2:1-Multiplexer und ein Pipeline-Register. Dargestellt ist jeweils der Leistungsverbrauch in Abhängigkeit von den Schaltaktivitäten σ der Eingänge. Bei den Multiplizierern wurde, wie oben dargestellt, zwischen den Wechselwahrscheinlichkeiten σ_{MSB} und σ_{LSB} der acht jeweils höchst- bzw. niedrigstwertigsten Bits der 16-Bit-Eingänge unterschieden.

Neben den Bibliotheksmodellen steht dem Array-Analyzer zur Leistungsanalyse die Struktur jeder einzelnen Basiszelle in Form einer Netzliste zur Verfügung. Unter Berücksichtigung der durch den Benutzer vorgegebenen Wechselwahrscheinlichkeiten an den Eingängen der Schaltung, sowie der Konfiguration der Verbindungsstruktur, bestimmt das Programm zunächst die Wechselwahrscheinlichkeiten an allen relevanten Punkten, also an allen PE-Eingängen und innerhalb der PE-Datenpfade. Anschließend wird mit Hilfe der Bibliotheken die Verlustleistung aller Datenpfade berechnet.

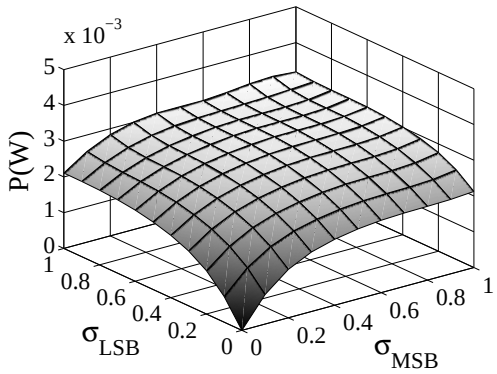
Um bei der Bestimmung der Verlustleistung auch die Verbindungsstruktur zu berücksichtigen, sind alle Verbindungsleitungen, auf denen eine Schaltaktivität stattfindet, in die Berechnung einzubeziehen. Dies wurde durch Berücksichtigung der äquivalenten Kapazitätswerte erreicht, die in jedem Schaltvorgang umgeladen werde müssen. Aus Gleichung 6.19 ergibt sich folgende Vorschrift zur Bestimmung der Verlustleistung eines einzelnen Verbindungspfades:

$$P_V^{(i,b)} = \sigma^{(i,b)} \cdot C_L^{(i,b)} V_{DD}^2 f_T \tag{6.21}$$

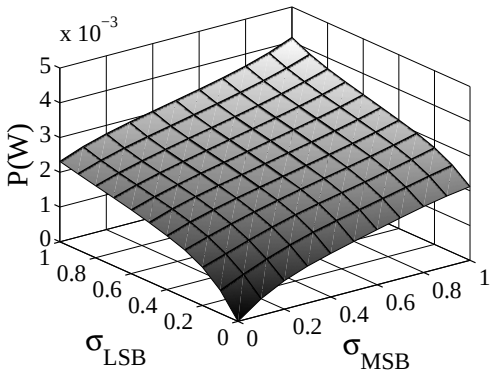
In Gleichung 6.21 bezeichnet $P_V^{(i,b)}$ die Verlustleistung der b-ten Bitleitung des i-ten Verbindungspfades, und $C_L^{(i,b)}$ dessen Kapazität. Die Vorgehensweise zur Bestimmung der Kapazität einer Verbindungsleitung wird in Anhang A in Bezug auf die Elmore-Verzögerung beschrieben. Simulationen haben jedoch gezeigt, dass zur Schätzung der Verlustleistung der gleiche Kapazitätswert verwendet werden kann. Weiterhin bezeichnet in Gleichung 6.21 V_{DD} die Betriebsspannung, und f_T die Taktfrequenz. Der Faktor $\sigma^{(i,b)}$ bezeichnet die durchschnittliche Schaltaktivität auf der Leitung. Die Schaltaktivität ist dabei entweder ein durch den Benutzer vorgegebener Wert an den Eingängen des Arrays, oder wird durch den Array-Analyzer mit den beschriebenen Methoden intern berechnet.

Evaluierung der Genauigkeit des Verlustleistungsmodells

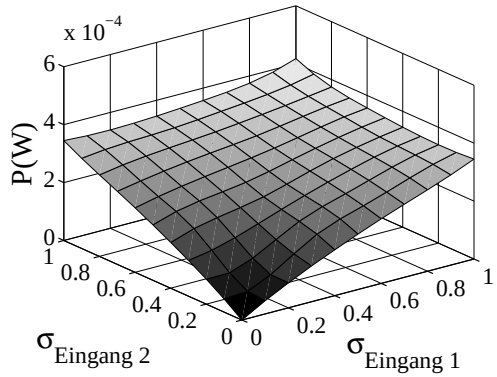
Wie im vorigen Abschnitt zur Evaluierung der Genauigkeit des Elmore-Modells soll auch für das Verlustleistungsmodell eine Untersuchung der Genauigkeit erfolgen. Zu diesem Zweck wurde ein einfaches MAC-basiertes Prozessorelement betrachtet, welches einen Mul-



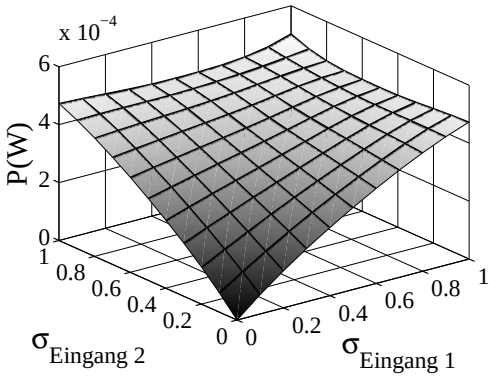
Wallace-Tree-Multiplizierer mit Booth-Codierung



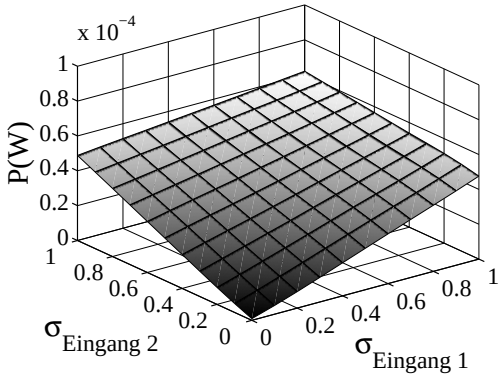
Carry-Save-Array Multiplizierer



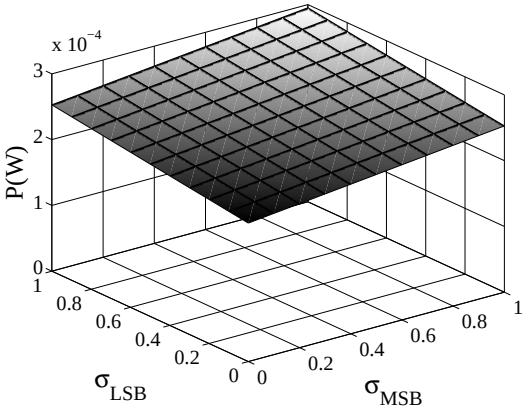
Carry-Lookahead-Addierer



Fast Carry-Lookahead-Addierer



2:1-Multiplexer



16-Bit Pipeline-Register

Abbildung 6.12: Bibliotheksmodelle für den Leistungsverbrauch der Datenpfadelemente

tiplizierer, einen Addierer, einen Multiplexer und ein Register enthält. Es wurden elf unterschiedliche Graustufen-Eingangsbilder ausgewählt, dessen Grauwerte mit Hilfe der Zelle mit einem konstanten Wert multipliziert, und das Ergebnis anschließend zu einer weiteren Konstanten addiert wurde (s.a. [Sch05]). Die Verlustleistung dieser Struktur wurde einerseits mit PrimePower simulativ auf Gatterebene ermittelt, und andererseits mit dem oben beschriebenen Bibliotheksmodell abgeschätzt. Zu diesem Zweck wurde zunächst die mittlere Schaltaktivität jedes einzelnen Bildes durch Simulation bestimmt, und der Wert als Eingangsparameter für das Modell verwendet.

Die Ergebnisse der Simulationen sind in Tabelle 6.5 aufgelistet. Alle Simulationen wurden mit zwei verschiedenen Multiplikationskoeffizienten simuliert, jeweils mit einem 1-0-Verhältnis von $\alpha_{(1,0)} = 0.5$ und $\alpha_{(1,0)} = 0.125$. Zu erwähnen ist, dass es sich bei den ersten acht Bildern ("Lena" bis "Duerer") um natürliche Bilder mit recht heterogenem Charakter handelt, deren Eingangsdaten eine recht hohe mittlere Schaltaktivität $\bar{\sigma}$besitzen. Die Bilder "Simpsons1", "Simpsons2" und "Snoopy" sind dagegen synthetische Bilder von Zeichentrickfiguren, die eine etwa um den Faktor 3 niedrigere Schaltaktivität besitzen.

Grundsätzlich fällt auf, dass die Abweichung des Bibliotheksmodells von der PrimePower-Simulation sich bei den natürlichen Bildern für $\alpha_{(1,0)} = 0.5$ durchschnittlich im Bereich zwischen 20% und 25% bewegen. Ausnahmen sind die Bilder "Clown" und "Teen" mit höheren Abweichungen. Für $\alpha_{(1,0)} = 0.125$ liegt die prozentuale Abweichung in allen Fällen deutlich niedriger. Bei den drei synthetischen Bildern mit geringerer Schaltaktivität sind die Schätzfehler jedoch mit 29.3%, 39.9% und 114.5% erheblich höher. Offenbar liefert das in dieser Arbeit verwendete einfache Verlustleistungsmodell erst ab einer bestimmten unteren Grenze für die Schaltaktivität halbwegs verlässliche Werte. Im Folgenden werden daher für Leistungsabschätzungen stets natürliche Bilder als Eingangsdaten angenommen.

Insgesamt ist eine durchschnittliche Abweichung von 20% bis 25% vom tatsächlichen Wert für eine erste Abschätzung der Verlustleistung hier als hinreichend genau anzusehen. Auffällig ist zudem, dass alle mit dem Bibliotheksmodell geschätzten Werte für die Verlustleistung höher als die mit PrimePower simulierten Ergebnisse liegen. Die in dieser Arbeit angegebenen Werte für Leistung und Energie der Datenpfade sind daher als eher etwas zu pessimistische Schätzwerte anzusehen.

Neben der Verlustleistungsschätzung der Datenpfade auf Basis des Bibliotheksmodells wurde auch das Modell für die Verbindungsstruktur auf die Genauigkeit hin untersucht. Zu diesem Zweck wurden vier unterschiedliche Beispielpfade angenommen, die hier mit Pfad A bis D bezeichnet werden. Pfad A ist ein nicht verzweigter Pfad bestehend aus zwei Buffern, einem Leitungssegment und einem Transmission-Gate. Die anderen Pfade enthalten Verzweigungen mit entsprechend mehr Elementen. Pfad D besteht aus einem Baum mit acht Buffern, vier Leitungssegmenten und vier Transmission-Gates. Alle Test-Pfade sind typisch für viele bei den abgebildeten Algorithmen auftretenden Verbindungen zwischen den Basiszellen. Die Verlustleistung dieser Schaltungen wurde für unterschiedliche Werte

Tabelle 6.5: Evaluierung des bibliotheksbasierten Modells zur Schätzung der Verlustleistung (s.a. [Sch05])

Bild	$\bar{\sigma}$	$\alpha_{(1,0)}$	$P_{PrimePower}$ (mW)	P_{ARA} (mW)	Abweichung
Lena	0.316	0.5	1.2060	1.4986	24.3%
		0.125	0.7696	0.8674	12.7%
Teen	0.292	0.5	1.1430	1.4429	26.2%
		0.125	0.7238	0.8424	16.4%
Kewgarden	0.282	0.5	1.1500	1.4196	23.4%
		0.125	0.7296	0.8319	14.0%
Goldhill	0.330	0.5	1.2500	1.5312	22.5%
		0.125	0.8166	0.8820	8.0%
Mandrill	0.387	0.5	1.3310	1.6637	25.0%
		0.125	0.9154	0.9415	2.9%
Clown	0.319	0.5	1.1220	1.5056	34.2%
		0.125	0.8159	0.8705	6.7%
Detail	0.292	0.5	1.1670	1.4429	23.6%
		0.125	0.7885	0.8424	6.8%
Duerer	0.336	0.5	1.3030	1.5451	18.6%
		0.125	0.8337	0.8883	6.6%
Simpsons1	0.111	0.5	0.6085	0.7868	29.3%
		0.125	0.3975	0.5189	30.5%
Simpsons2	0.120	0.5	0.5945	0.8315	39.9%
		0.125	0.4285	0.5419	26.5%
Snoopy	0.096	0.5	0.3316	0.7113	114.5%
		0.125	0.2789	0.4796	72.0%

der Schaltaktivität $\sigma^{(i,b)}$ (siehe Gleichung 6.21) zunächst mit SPICE, und anschließend mit dem oben vorgestellten Verlustleistungsmodell ermittelt.

Die Ergebnisse dieser Untersuchungen zeigt Tabelle 6.6. Der maximale Fehler aller Simulationen liegt bei 18.4%. Die anderen Fehlerwerte liegen teilweise deutlich darunter. Eine signifikante Abhängigkeit des Fehlers von der angenommenen Schaltaktivität ist

zudem nicht zu erkennen. Nach dieser Untersuchung bietet somit auch das analytische Modell zur Abschätzung der Verlustleistung der Verbindungsstruktur hinreichend genaue Ergebnisse. Desweiteren gilt auch hier, dass alle mit dem Modell geschätzten Werte höher sind als die simulativ ermittelten Werte, so dass auch die ermittelte Verlustleistung für die Verbindungsstruktur als eher pessimistisch anzusehende obere Schranke zu betrachten ist.

Tabelle 6.6: Evaluierung des analytischen Modells zur Verlustleistungsschätzung der Verbindungsstruktur

Pfad	$\sigma^{(i,b)}$	$P_{SPICE}/\mu W$	$P_{ARA}/\mu W$	Abweichung
A	1	15.037	16.841	12.0%
	0.66	9.3898	11.115	18.4%
	0.5	7.1889	8.4203	17.1%
	0.33	4.9007	5.5574	13.2%
B	1	30.954	31.585	2.0%
	0.66	20.207	20.846	3.2%
	0.5	15.119	15.793	4.5%
	0.33	10.107	10.423	3.1%
C	1	43.200	46.330	7.2%
	0.66	28.213	30.578	8.4%
	0.5	20.951	23.165	10.6%
	0.33	14.141	15.289	8.1%
D	1	55.579	61.075	9.9%
	0.66	36.219	40.310	11.3%
	0.5	26.782	30.538	14.0%
	0.33	18.174	20.155	10.9%

6.2 VLSI-Eigenschaften von Implementierungsalternativen

Da diese Arbeit nicht nur das Ziel verfolgt, rekonfigurierbare Architekturen unterschiedlicher Granularität vergleichend zu bewerten, sondern auch die Eigenschaften grobgranularer Architekturen mit denjenigen anderer Implementierungsformen zu vergleichen, ist es erforderlich, diese alternativen Architekturen in Bezug auf ihre VLSI-Eigenschaften zu

evaluieren. Von besonderem Interesse sind hier vor allem FPGAs, da diese den feingranularen Gegenpol zu den betrachteten grobgranularen Architekturen darstellen. Neben FPGAs sind auch Semi-Custom-Entwürfe für bestimmte Anwendungen, sowie digitale Signalprozessoren (DSPs) zum Vergleich herangezogen worden. Die Vorgehensweise bei der Bestimmung der VLSI-Eigenschaften dieser Architekturen wird in diesem Abschnitt kurz dargestellt.

6.2.1 Semi-Custom-Entwürfe

Die Eigenschaften der Semi-Custom-Entwürfe sind auf Basis von Standardzellimplementierungen abgeschätzt worden. Dazu ist jeder Algorithmus in VHDL implementiert, verifiziert, und anschließend mit der $0.18\mu m$-Technologiebibliothek von Chartered synthetisiert worden.

Bei der Implementierung wurde darauf geachtet, dass nur die zur Ausführung des Algorithmus erforderlichen Architekturblöcke verwendet wurden, so dass keinerlei rekonfigurierbare Anteile in der Schaltung enthalten waren. Um einen fairen Vergleich zu ermöglichen, wurde der Parallelitätsgrad identisch zu den rekonfigurierbaren Modellarchitekturen gewählt, so dass beispielsweise bei der Implementierung eines linearen Filters die gleiche Anzahl Multiply-Add-Stufen implementiert wurden. Sollte beispielsweise die Implementierung eines 128-Tap FIR-Filters auf einem 4×4-PE-Feld aus *PE1M*-Prozessorzellen mit einer Standardzellimplementierung verglichen werden, dann wurden für den Semi-Custom-Entwurf 16 parallele Multiply-Add-Stufen implementiert, mit denen die Filterung entsprechend serialisiert ausgeführt werden kann. Die Größe des zur Speicherung der Zwischenwerte erforderlichen Speichers wurde von Fall zu Fall an den jeweiligen Algorithmus angepasst. Die Größe der Speicherbänke für die Ein- und Ausgangswerte des Algorithmus wurde jeweils identisch zur Speichergröße der grobgranularen Modellarchitekturen gewählt.

Nach der Implementierung sind die bereits in Abschnitt 4.1 angesprochenen EDA-Tools zur Analyse der VLSI-Eigenschaften verwendet worden. Die Fläche wurde mit Hilfe der vom Design-Compiler gelieferten Anzahl von Gatteräquivalenten und anschließender Umrechnung in Quadratmillimeter ermittelt. Das Timing-Verhalten und der Energieverbrauch wurde mit PrimeTime, bzw. PrimePower ermittelt.

6.2.2 FPGAs

Um einen sinnvollen und aussagefähigen Vergleich grobgranularer Architekturen mit feingranularen zu ermöglichen, sind die zum Vergleich herangezogenen feingranularen Strukturen sorgfältig auszuwählen. Wenn eine ausschließlich feingranulare Struktur betrachtet werden soll, verbietet sich hierzu aus Gründen der Vergleichbarkeit die Auswahl eines

FPGAs, der neben dem Logikzellen-Feld auch dedizierte Funktionsblöcke wie Multiplizierer enthält, wie dies in vielen modernen FPGAs heutzutage der Fall ist (s.a. Abschnitt 2.3.2). Als feingranulare Vergleichsstruktur wurde daher in dieser Arbeit ein FPGA der Firma Xilinx vom Typ VirtexE XCV300E herangezogen ([Xil02]).

Abbildung 6.13a zeigt die Struktur des XCV300E-Logikzellenfeldes. Das Feld besteht aus konfigurierbaren Logikzellen (Configurable Logic Blocks, CLBs), die in einem Array aus 32 Zeilen und 48 Spalten angeordnet sind. Jeder CLB setzt sich weiterhin aus zwei Slices zusammen, von denen jede wiederum zwei Lookup-Tabellen mit jeweils vier Eingängen beinhaltet. Abbildung 6.13b zeigt schematisch den Aufbau einer Slice. Zusätzlich zu den Lookuptabellen sind eine Carry-Logik, sowie zwei Flipflops in einer Slice integriert. Desweiteren sind in das Logikzellenfeld insgesamt 32 Dual-Port SRAM-Speicherbänke eingebettet, welche, wie in Abbildung 6.13a gezeigt, in vier senkrechten Spalten auf der Chipfläche verteilt sind. Darüber hinaus enthält dieser Baustein keine zusätzliche dedizierte Logik, so dass es sich um eine rein feingranulare FPGA-Architektur handelt.

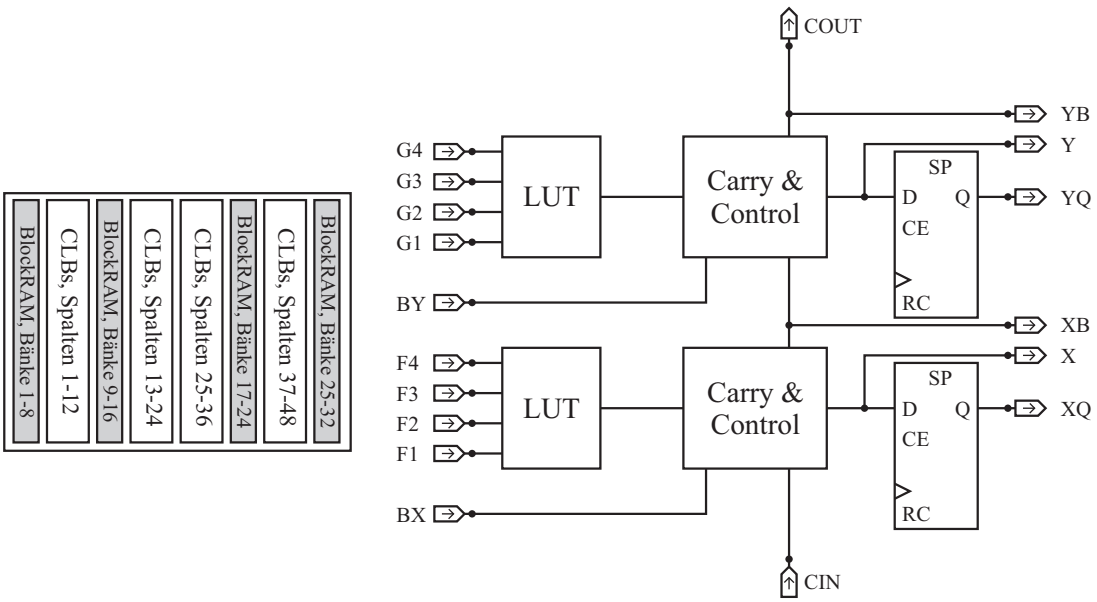


Abbildung 6.13: a) Struktur des Logikzellenfeldes eines Xilinx VirtexE XCV300E mit BlockRAMs (die Größenverhältnisse sind nicht maßstabsgetreu), b) VirtexE-Slice. Siehe hierzu [Xil02]

Der VirtexE XCV300E ist in einer $0.18\mu m$-Technologie gefertigt, und arbeitet bei $1.8V$ Betriebsspannung. Diese Technologiedaten entsprechen dem in dieser Arbeit zur Imple-

mentierung der Modellarchitekturen verwendeten Prozess, so dass eine gute Vergleichbarkeit der Ergebnisse ohne eine Skalierung und Umrechnung gegeben ist.

Die zur Evaluation verwendeten Algorithmen wurden auf dem FPGA implementiert. Dabei wurden soweit wie möglich die in der von Xilinx angebotenen IP-Bibliothek *Corelib* enthaltenen fertigen IP-Blöcke verwendet, um eine effiziente Implementierung zu gewährleisten. In dieser Bibliothek sind neben Architektur-Blöcken wie Multiplizierern oder ALUs auch fertige, und in weiten Bereichen parametrisierbare Komponenten wie FIR-Filter oder DCTs enthalten. Algorithmen, die nicht in der Bibliothek enthalten, bzw. nicht lizenziert waren, sind entsprechend neu in VHDL codiert, und mit Hilfe der Xilinx-Tools auf den FPGA-Baustein abgebildet worden.

Die unter dem Xilinx Project Navigator zusammengefassten Tools bieten die Möglichkeit, abgebildete Architekturen simulativ unter anderem in Bezug auf den Flächenbedarf, die maximal mögliche Taktfrequenz, und die Leistungsaufnahme zu untersuchen:

Der *Flächenbedarf* wird in Form der Anzahl beim Mapping verwendeter Slices angegeben, und ist für den Vergleich in einen absoluten Flächenbedarf umzurechnen, der neben der reinen Logikzellenfläche auch die Fläche der verwendeten Routing-Ressourcen mit einbezieht.

Die maximal mögliche *Samplerate* eines Entwurfs ergibt sich aus der maximalen Taktfrequenz, die mit dem Timing Analyzer bestimmt werden kann.

Die *Leistungsaufnahme* wurde mit dem Xilinx-Tool *XPower* ermittelt. Ähnlich wie bei der Berechnung des Flächenbedarfs der FPGA-Implementierung sind auch bei der Bestimmung der Verlustleistung nur die Komponenten des FPGA-*Cores* zu betrachten, die zur Abbildung des jeweiligen Algorithmus verwendet werden. Dies sind die FPGA-Slices mit den Logikzellen, die Verbindungsstruktur, sowie eventuell eingebundene Speicherblöcke. Die Verlustleistung periphärer Komponenten wie I/O-Blöcke soll ebenso wie die Verlustleistung der nicht verwendeten Logik- und Routingressourcen nicht in die Gesamt-Verlustleistung einfließen, um auch bei der Leistungsaufnahme die Vergleichbarkeit zu gewährleisten.

XPower gibt nach der Simulation sowohl die durch den gesamten Baustein aufgenommene Verlustleistung an, als auch die Leistungsaufnahme jeder einzelnen Komponente. Insbesondere wird zusammenfassend die Verlustleistung des Taktsignal-Netzwerkes, der Logikzellen, der Routing-Struktur, der Ein- und Ausgabeblöcke, sowie die durch den Baustein verbrauchte statische Verlustleistung angegeben. Die Leistung der Ein- und Ausgabeblöcke ist für diese Arbeit aus o.g. Gründen nicht relevant. Besondere Aufmerksamkeit verdient die Bestimmung der statischen Verlustleistung. Da diese nur für den gesamten Baustein angegeben wird, ist zur Aufrechterhaltung der Vergleichbarkeit eine anteilsmäßige Umrechnung auf die verwendeten Logik-Ressourcen erforderlich. Der Anteil der in die Berechnung eingehenden statischen Verlustleistung wird aus der Anzahl der benutzten Logikzellen im Verhältnis zur Gesamtzahl der Logikzellen gebildet.

Diese Vorgehensweise soll durch ein Beispiel verdeutlicht werden. Bei der Simulation einer FFT mit 8-Bit-Datenpfaden gibt XPower für das Taktnetzwerk eine Verlustleistung von 1.22 mW/MHz an, weiterhin für die Logikzellen 16.28 mW/MHz, für das Routing-Netzwerk 4.08 mW/MHz, und für die gesamte statische Verlustleistung 17.48 mW/MHz. Die Abbildung der Logik erfordert 848 Slices, das sind 27% der insgesamt 3072 auf dem Baustein vorhandenen Slices. Die Summe der Leistungswerte für das Taktsignal, die Logikzellen und das Routingnetzwerk, sowie 27% der statischen Verlustleistung des Bausteins ergibt eine Leistungsaufnahme von 26.3 mW/MHz für die Implementierung des Algorithmus.

6.2.3 DSPs

Eine weitere in dieser Arbeit zum Vergleich mit grobgranularen Architekturen herangezogene Implementierungsalternative ist die Abbildung der betrachteten Algorithmen auf einem digitalen Signalprozessor (DSP). Bei der Auswahl des zum Vergleich herangezogenen Bausteins gelten ähnliche Kriterien wie die bereits in Abschnitt 6.2.2 erwähnten. Es soll sich um ein eingebettetes System handeln, Flächen- und Verlustleistungsangaben sollen sich demzufolge auf den reinen DSP-Core ohne IO-Blöcke beziehen. Ein diese Bedingungen erfüllender Baustein, zu welchem gleichzeitig entsprechende Daten bezüglich Fläche, Timing und Verlustleistung zur Verfügung stehen, ist der Adelante RD16025-DSP-Core der Firma Philips (siehe hierzu [Phi05], [Moe02a], [Moe02b]). Dieser DSP ist als vollständig synthetisierbarer VHDL- oder Verilog-Code lizenzierbar.

Die Architektur des RD16025-Cores ist in Abbildung 6.14 schematisch dargestellt. Es handelt sich um eine 16-Bit Dual-Harvard VLIW Architektur mit zwei unabhängigen parallelen Datenbussen und separatem Programmspeicher. Die Ausführungseinheit beinhaltet unter anderem zwei parallele Multiplizierer, vier ALUs, sowie weitere mögliche parallele Komponenten, die auch anwendungsabhängig in Form von IP-Blöcken hinzugeschaltet werden können.

Bei den Instruktionen wird unterschieden zwischen Standard-Instruktionen und applikationsspezifischen Befehlen. Letztere können in Abhängigkeit vom auszuführenden Algorithmus in der Application Specific Instruction Table (ASI Table) abgelegt werden. Die ASI Table dient als Lookup-Tabelle für 8-Bit-Befehle, aus welchen zusammen mit den Standard-Instruktionen ein 96-Bit VLIW-Wort generiert wird. Auf diese Art können maximal zwölf Operation pro Taktzyklus ausgeführt werden.

Der RD16025-Core belegt in einer $0.18\mu m$-Technologie eine Chipfläche von $0.5mm^2$. Die maximale Taktfrequenz beträgt 210MHz, und die Leistungsaufnahme des Core beträgt für typische Anwendungen 0.25mW/MHz.

Die absoluten Performanceangaben sind für eine Abschätzung der Samplerate und der pro Ausgangswert verbrauchten Energie nicht ausreichend. Vielmehr ist in jedem Einzelfall der auf dem DSP implementierte Algorithmus zu betrachten. Zu diesem Zweck ist

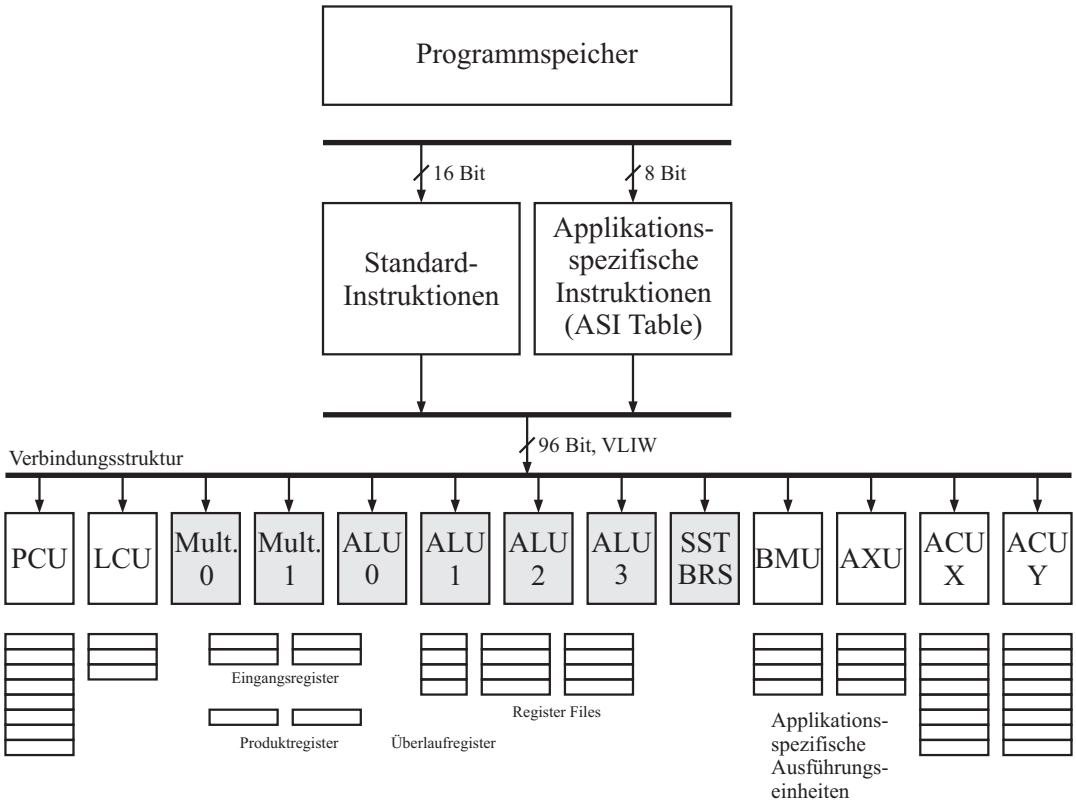


Abbildung 6.14: Architektur des RD16025-DSP Core von Philips (siehe [Phi05], [Moe02a], [Moe02b])

für jeden der Algorithmen ein Pseudo-Assemblercode erzeugt worden, anhand dessen eine Abschätzung der zur Berechnung erforderlichen Taktzahl durchgeführt wurde. Dieses Vorgehen wird im Folgenden anhand eines einfachen Beispiels dargestellt.

Es soll die bei einer DSP-Implementierung erforderliche Anzahl von Takten zur Berechnung einer 8-Punkte Radix-2 FFT bestimmt werden. Abbildung 6.15 zeigt die hier zugrude liegende Butterfly-Struktur, welche bereits in Abschnitt 5.2.1 dargestellt wurde. Abbildung 6.15 zeigt zusätzlich die jeweils bei einer DSP-Implementierung verwendeten Register, hier mit R0 bis R13 gekennzeichnet.

Der folgende Pseudo-Assemblercode repräsentiert die bei der Implementierung auf dem RD16025-DSP auszuführenden Maschinenbefehle als Kern der inneren Schleifen des Programms:

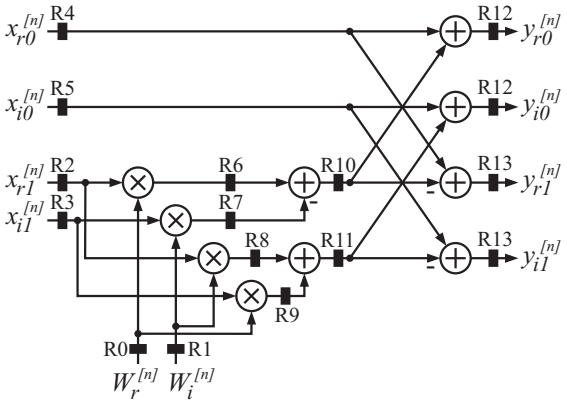


Abbildung 6.15: Radix-2 FFT Butterfly zur Illustration der DSP-Programmierung

Takt	**Instruktion**	**Operanden**	**Kommentare**
1	`Laden`	$\mathrm{R0} = w_r^{[0]}$	Beginn erster Butterfly
	`Laden`	$\mathrm{R1} = w_i^{[0]}$	
2	`Laden`	$\mathrm{R2} = x_{r1}^{[0]}$	
	`Laden`	$\mathrm{R3} = x_{i1}^{[0]}$	
3	`Laden`	$\mathrm{R4} = x_{r0}^{[0]}$	
	`Laden`	$\mathrm{R5} = x_{i0}^{[0]}$	
	`Mult`	$\mathrm{R6} = \mathrm{R2} \cdot \mathrm{R0}$	
	`Mult`	$\mathrm{R7} = \mathrm{R3} \cdot \mathrm{R1}$	
4	`Mult`	$\mathrm{R8} = \mathrm{R2} \cdot \mathrm{R1}$	
	`Mult`	$\mathrm{R9} = \mathrm{R3} \cdot \mathrm{R0}$	
	`Sub`	$\mathrm{R10} = \mathrm{R6} - \mathrm{R7}$	
	`Laden`	$\mathrm{R0} = w_r^{[1]}$	
	`Laden`	$\mathrm{R1} = w_i^{[1]}$	
5	`Add`	$\mathrm{R11} = \mathrm{R8} + \mathrm{R9}$	
	`Add`	$\mathrm{R12} = \mathrm{R4} + \mathrm{R10}$	
	`Sub`	$\mathrm{R13} = \mathrm{R4} - \mathrm{R10}$	
	`Laden`	$\mathrm{R2} = x_{r1}^{[1]}$	
	`Laden`	$\mathrm{R3} = x_{i1}^{[1]}$	
6	`Add`	$\mathrm{R12} = \mathrm{R5} + \mathrm{R11}$	Ende der Berechnung
	`Sub`	$\mathrm{R13} = \mathrm{R5} - \mathrm{R11}$	des ersten Butterflies
	`Laden`	$\mathrm{R4} = x_{r0}^{[1]}$	Beginn zweiter Butterfly

	`Laden`	$\mathrm{R5} = x_{i0}^{[1]}$	
	`Mult`	$\mathrm{R6} = \mathrm{R2} \cdot \mathrm{R0}$	
	`Mult`	$\mathrm{R7} = \mathrm{R3} \cdot \mathrm{R1}$	
7	`Mult`	$\mathrm{R8} = \mathrm{R2} \cdot \mathrm{R1}$	
	`Mult`	$\mathrm{R9} = \mathrm{R3} \cdot \mathrm{R0}$	
	`Sub`	$\mathrm{R10} = \mathrm{R6} - \mathrm{R7}$	
	`Laden`	$\mathrm{R0} = w_r^{[2]}$	
	`Laden`	$\mathrm{R1} = w_i^{[2]}$	
8	`Add`	$\mathrm{R11} = \mathrm{R8} + \mathrm{R9}$	
	`Add`	$\mathrm{R12} = \mathrm{R4} + \mathrm{R10}$	
	`Sub`	$\mathrm{R13} = \mathrm{R4} - \mathrm{R10}$	
	`Laden`	$\mathrm{R2} = x_{r1}^{[2]}$	
	`Laden`	$\mathrm{R3} = x_{i1}^{[2]}$	
9	`Add`	$\mathrm{R12} = \mathrm{R5} + \mathrm{R11}$	Ende der Berechnung
	`Sub`	$\mathrm{R13} = \mathrm{R5} - \mathrm{R11}$	des zweiten Butterflies
	`Laden`	$\mathrm{R4} = x_{r0}^{[2]}$	Beginn dritter Butterfly
	`Laden`	$\mathrm{R5} = x_{i0}^{[2]}$	
	`Mult`	$\mathrm{R6} = \mathrm{R2} \cdot \mathrm{R0}$	
	`Mult`	$\mathrm{R7} = \mathrm{R3} \cdot \mathrm{R1}$	

Der Code ist unter Berücksichtigung der Hardwareeigenschaften des DSPs erstellt worden. Die Dual-Harvard-Struktur erlaubt zwei parallele Speicherzugriffe in einem Takt. Die hochstellten Indizes $^{[n]}$ kennzeichnen die Zugehörigkeit zum n-ten berechneten Butterfly. Da der DSP maximal 12 Instruktionen in einem Takt parallel verarbeiten kann, ist bei der Erstellung des Codes auf eine größtmögliche Ausnutzung dieser Parallelität unter Berücksichtigung der zur Verfügung stehenden Hardwareressourcen mit zwei Multiplizierern und vier ALUs geachtet worden.

Aus dem Beispiel geht hervor, dass die Berechnung des ersten Butterflies insgesamt sechs Takte in Anspruch nimmt, während alle nachfolgenden nur noch drei Takte benötigen, da bereits während der Berechnung des vorhergehenden Butterflies erforderliche Werte aus dem Speicher geladen werden können (beispielsweise $w_r^{[1]}$, $w_i^{[1]}$, $x_{r1}^{[1]}$ und $x_{i1}^{[1]}$ in den Takten 4 und 5). Die Takte 6, 7 und 8 werden schließlich zur Berechnung der nachfolgenden Butterflies zyklisch wiederholt. Die Berechnung einer einzelnen 8-Punkte Radix-2 FFT erfordert demzufolge bei zwölf Butterflies $6+11 \cdot 3 = 39$ Takte. Bei einer Taktfrequenz von 210MHz ergibt sich die Blockrate f_B für die 8-Punkte-FFT zu

$$f_B = \frac{1}{39} \cdot 210 \cdot 10^6 \cdot s^{-1} = 5.38 \cdot 10^6 s^{-1} \tag{6.22}$$

Für alle in dieser Arbeit in Bezug auf eine DSP-Implementierung untersuchten Algorithmen ist anhand eines wie oben dargestellten Maschinencodes eine Abschätzung der erforderlichen Taktzahl durchgeführt worden.

7 Modellbasierte Kostenanalyse und Schlussfolgerungen

Die im Rahmen dieser Arbeit untersuchten Modelle für grobgranulare rekonfigurierbare Architekturen sollen in diesem Kapitel untereinander und bezüglich der in Abschnitt 6.2 erwähnten alternativen Implementierungsformen verglichen und bewertet werden. Grundlage für die Vergleiche sind die mit den in Kapitel 6 beschriebenen Verfahren erhaltenen Simulationsergebnisse bezüglich der VLSI-Eigenschaften der Architekturen. Diese Eigenschaften sind stets sowohl von den Hardwarestrukturen selbst, als auch von den darauf implementierten Algorithmen abhängig. Aus diesem Grunde sind jeweils in einem ersten Schritt auf alle Architekturvarianten verschiedene Algorithmen und Benchmark-Anwendungen mit teilweise unterschiedlichen Implementierungsparametern abgebildet worden, bevor schließlich die Eigenschaften wie Flächenbedarf, Datendurchsatz und Energieverbrauch ermittelt wurden. Weiterhin ist die Effizienz untersucht worden, mit der ein FFT-Butterfly als einfache Basis-Operation auf grob- und feingranulare Architekturen abgebildet werden kann.

Um unterschiedliche Architekturvarianten fair und aussagefähig miteinander vergleichen zu können, ist auf gleiche Randbedingungen bei den Simulationen zu achten. Dies bezieht sich vor allem auf die Wahl der Speichergröße der Bausteine. So können beispielsweise die rekonfigurierbaren Modellarchitekturen und der DSP nur dann sinnvoll miteinander verglichen werden, wenn für beide die gleiche Speichergröße für Instruktions- und Datenspeicher angenommen wird. Weiterhin ist eine Normierung der zu Grunde liegenden Prozesstechnologie erforderlich. In Abschnitt 7.2.1 werden die den hier präsentierten Simulationsergebnissen zu Grunde gelegten Annahmen dargestellt.

Ein aussagefähiger Vergleich von Rechnerarchitekturen erfordert zudem die Verfügbarkeit geeigneter Kosten- bzw. Gütewerte. Beispielsweise ist die Angabe der absoluten Datendurchsatzrate oder der Samplerate allein ein unzureichendes Kriterium, da keine Aussage bezüglich der verwendeten Hardwareressourcen getroffen wird. So kann ein massiv parallel arbeitendes Rechenfeld geeignete Algorithmen schneller ausführen als ein DSP mit stärker serialisierter Programmausführung, belegt dafür jedoch auch entsprechend mehr Chipfläche. Demzufolge ist auch die Angabe der Chipfläche allein ohne Erwähnung der Rechenleistung ein wenig aussagefähiges Kriterium. Ein fairer Vergleich gelingt erst dann, wenn beide Maße miteinander kombiniert werden, was auf das *AT-Produkt* [1] führt.

[1] Hinweis: In der englischsprachigen Literatur ist der Begriff *Area-Delay-Product* üblich. In deutschsprachigen Texten wird dagegen häufig vom AT-Produkt gesprochen.

Daher werden in Abschnitt 7.1 zunächst Kosten- und Gütemaße für Rechenarchitekturen eingeführt, mit dessen Hilfe ein aussagefähiger Vergleich der in dieser Arbeit untersuchten Architekturvarianten möglich ist, bevor in Abschnitt 7.2 die Ergebnisse präsentiert werden.

7.1 Kosten- und Gütemaße für Rechnerarchitekturen

Ein Kosten- bzw. Gütemaß wird in Form eines dimensionsbehafteten oder dimensionslosen Wertes angegeben. Wenn ein höherer Wert objektiv oder subjektiv ein auf eine bestimmte Eigenschaft bezogen schlechteres System bezeichnet als ein niedrigerer Wert, ist das Kriterium ein Kostenmaß, andernfalls ein Gütemaß. Der Flächenbedarf, der Energieverbrauch und das AT-Produkt sind Beispiele für Kostenmaße, während die Angabe der maximalen Takt- oder Samplerate ein Gütemaß ist.

In [DeH98] werden Richtlinien zum Vergleich von Rechensystemen angeführt, beispielsweise bezüglich der Normierung von Prozesstechnologien und der Bewertung der Fläche, Geschwindigkeit und Energieverbrauches einer Architektur. Im Folgenden werden die in dieser Arbeit verwendeten Vergleichskriterien definiert. Diese sind konform mit den in der Literatur häufig verwendeten Kosten- und Gütemaßen. So verwenden die meisten der in Kapitel 2 und 3 zitierten Veröffentlichungen die gleichen Bewertungskriterien. Desweiteren werden im Hinblick auf rekonfigurierbare Architekturen die Begriffe der relativen Flächen-, Zeit- und Energieeffizienz eingeführt.

7.1.1 Flächenbedarf und relative Flächeneffizienz

Wie in der Einleitung zu diesem Kapitel erwähnt, ist die Angabe des Flächenbedarfs für sich allein in den meisten Fällen kein geeignetes Kriterium zur Bewertung einer Architektur, da sich erst in Kombination mit anderen Maßen aussagekräftige Vergleiche treffen lassen. Dennoch ist die Angabe des absoluten Flächenbedarfs sinnvoll, sofern dies unter Beachtung der anderen Kriterien erfolgt.

In dieser Arbeit wird zwischen dem *absoluten Flächenbedarf* und der *relativen Flächeneffizienz* unterschieden.

Unter dem *absolutem Flächenbedarf* wird hier die Gesamtfläche aller an der Berechnung beteiligten Hardwareressourcen verstanden. Wenn beispielsweise auf einer rekonfigurierbaren Architektur, die aus einem 4×4-PE-Array besteht, ein Algorithmus ausgeführt wird, der lediglich 5 der insgesamt 16 Basiszellen belegt, gehen auch nur die 5 Basiszellen in den angegebenen absoluten Flächenbedarf ein, da die anderen Zellen theoretisch für andere Operationen verwendet werden könnten. Ebenfalls gehen die verwendeten Routing-Ressourcen, sowie alle weiteren Elemente inklusive Speicherbänke ein, die nicht zur Ausführung anderer Algorithmen verwendet werden können. Bei FPGAs gehen in den absoluten

Flächenbedarf ebenfalls nur die Fläche der verwendeten Logikzellen und Routingressourcen ein. Der absolute Flächenbedarf ist daher nicht zu verwechseln mit der Gesamtfläche des Bausteins.

Unter der *relativen Flächeneffizienz* einer rekonfigurierbaren Architektur wird das Verhältnis der Fläche der auch in dedizierten Komponenten zur Ausführung des Algorithmus erforderlichen Rechenressourcen zur Gesamtfläche verstanden:

$$\text{Relative Flächeneffizienz} = \frac{\text{Nutzfläche}}{\text{Gesamtfläche}} \cdot 100\% \tag{7.1}$$

Unter der *Nutzfläche* wird in diesem Zusammenhang die Fläche der Multiplizierer, Addierer, ALUs oder Lookup-Tabellen verstanden. Ebenfalls inbegriffen sind Pipeline-Register, da diese auch in dedizierten Realisierungen zur Erhöhung der Taktrate eingesetzt werden. Die Gesamtfläche ist bei rekonfigurierbaren Architekturen größer als die Nutzfläche, da zusätzlich der Flächenbedarf der Rekonfigurations-Ressourcen eingeht. Dies sind beispielsweise Multiplexer zum Umschalten der Konfiguration, Register und Speicherzellen für Konfigurationsdaten, und ein konfigurierbares Verbindungsnetzwerk. Die Fläche der Speicherblöcke für die Ein- und Ausgangswerte, sowie zur internen Zwischenspeicherung von Daten, werden hier von der Berechnung der relativen Flächeneffizienz ausgenommen, und gehen weder in die Nutzfläche noch in die Gesamtfläche ein. Dies begründet sich dadurch, dass die relative Flächeneffizienz ein Maß für die Effizienz der Datenpfadimplementierung und der Verbindungsstruktur rekonfigurierbarer Architekturen sein soll, und zudem nicht in allen Fällen exakt definierbar ist, ob ein Speicherblock zur Nutzfläche oder zur Overhead-Fläche gerechnet werden kann. Das Maß ist jedoch stets mit dem Hinweis auf die zusätzliche Existenz der Speicherblöcke anzugeben.

Dedizierte Realisierungen in Form von Semi- oder Full-Custom-Entwürfen besitzen daher definitionsgemäß eine relative Flächeneffizienz von 100%. Bei rekonfigurierbaren Architekturen ist der Wert niedriger, da zusätzlicher Flächenaufwand durch Rekonfigurations-Komponenten entsteht. Die relative Flächeneffizienz ist ein Gütemaß. Demgegenüber lässt sich leicht ein Maß zur Bewertung der *Rekonfigurations-Kosten*, bzw. des *Rekonfigurations-Overhead* bezogen auf den Flächenbedarf definieren:

$$\text{Rekonfigurations-Overhead (Fläche)} = 100\% - \text{Relative Flächeneffizienz} \tag{7.2}$$

7.1.2 Taktfrequenz, Samplerate und zeitliche Effizienz

Neben dem Flächenbedarf ist die Frage nach der Geschwindigkeit von Interesse, mit der Daten verarbeitet werden können. Dies ist insbesondere in Echtzeitsystemen von Bedeutung, da hier bestimmte Mindestanforderungen an die Performanz durch die Randbedingungen des Systems vorgegeben sind. Aus diesem Grunde reicht die Angabe der maximalen

Taktfrequenz $f_{max} = T_{min}^{-1}$, mit der eine Schaltung betrieben werden kann, als Kriterium nicht aus, da hierdurch noch keine Aussage über den Datendurchsatz getroffen wird. Besser ist die Angabe der maximal erreichbaren Samplerate als Maß für die Anzahl der Ausgangswerte, die in einer bestimmten Zeit maximal berechnet werden können:

$$f_{Sample} = \frac{1}{T_{Sample}} = \frac{f_{max}}{N_T} \tag{7.3}$$

Die Anzahl N_T der zur Berechnung eines Ausgangswertes erforderlichen Takte, und somit auch die Samplerate, ist demzufolge sowohl von der Hardware, als auch von der jeweiligen Anwendung abhängig, sowie in gewissem Maße von der Effizienz, mit der ein Programmierer den Algorithmus auf die Architektur abzubilden vermag.

Theoretisch ließe sich analog zur Definition der relativen Flächeneffizienz, bzw. der auf die Fläche bezogenen Rekonfigurations-Kosten, eine *zeitliche Effizienz* definieren als Verhältnis der Summe der Verzögerungszeiten der Nutz-Blöcke zur Gesamt-Verzögerungszeit eines Pfades. Dies ist jedoch insbesondere bei rekonfigurierbaren Architekturen nicht sinnvoll möglich, da die Länge der Pfade bedingt durch die konfigurierbare Verbindungsstruktur vor allem bei FPGAs zum Teil erheblich voneinander abweichen kann. Es lassen sich aber Pfad-Profile für die Prozessor- und Logikzellen angeben, anhand derer die Durchlaufzeiten für Nutz- und Overheadblöcke von fein- und grobgranularen Basiszellen verglichen werden können.

7.1.3 Energieverbrauch und Energieeffizienz

Ähnlich wie bei der Bewertung der Verarbeitungsgeschwindigkeit einer Architektur ist auch beim Vergleich der Leistungsaufnahme und des Energieverbrauches auf eine geeignete Normierung zu achten. Die Angabe der Leistungsaufnahme allein ist unzureichend, da seriell arbeitende Architekturen im Regelfall weniger Leistung verbrauchen als massiv parallele Strukturen, dafür jedoch auch eine längere Rechenzeit benötigen, bzw. höher getaktet werden müssen, um eventuelle Performanzanforderungen zu erfüllen. Ein besseres Maß ist die Angabe der Energie E_{Sample}, die erforderlich ist, um einen Ausgangswert (Sample) zu berechnen:

$$E_{Sample} = \frac{E_{gesamt}}{N_{Samples}} = \frac{\bar{P}}{f_{Sample}} = \bar{P} \cdot T_{Sample} \tag{7.4}$$

Analog zur Definition der relativen Flächeneffizienz (Gleichung 7.1) lässt sich die relative Energieeffizienz folgendermaßen definieren:

$$\text{Relative Energieeffizienz} = \frac{E_{Sample,Nutz}}{E_{Sample}} \cdot 100\% \tag{7.5}$$

$E_{Sample,Nutz}$ entspricht dabei dem Energieverbrauch der Nutzblöcke pro Sample. Die auf den Energieverbrauch bezogenen Rekonfigurationskosten, bzw. der Energie-Rekonfigurationsoverhead bestimmen sich analog zu Gleichung 7.2 zu:

$$\text{Rekonfigurations-Overhead (Energie)} = 100\% - \text{Relative Energieeffizienz} \tag{7.6}$$

7.1.4 AT-Produkt

Wie in den obigen Abschnitten bereits erwähnt, ist die alleinige Angabe des Flächenbedarfs bei einer Implementierung der Architektur zwar von großem Interesse, als Kriterium zum Vergleich verschiedener Prozessorstrukturen jedoch nur begrenzt aussagefähig, da massiv parallele Architekturen mit relativ hohem Flächenbedarf auch eine höhere Verarbeitungsgeschwindigkeit bieten. Wünschenswert ist ein Maß, bei dem der Flächenbedarf auf die Datendurchsatzrate normiert ist. Ein solches Maß ist das AT-Produkt, das folgendermaßen definiert ist:

$$\text{AT-Produkt} = A_{ges} \cdot T_{Sample} = \frac{A_{ges}}{f_{Sample}} \tag{7.7}$$

In Zusammenhang mit dem AT-Produkt wird häufig auch von *Rechenleistungsdichte* oder *Computational Density* gesprochen.

7.1.5 ATE-Produkt

Wenn Architekturen mit einem einzigen Kostenmaß bewertet werden sollen, kann das *ATE-Produkt* verwendet werden, bei dem im Vergleich zum Area-Delay-Product zusätzlich die Energie pro Sample hinzugezogen wird:

$$ATE = A_{ges} \cdot T_{Sample} \cdot E_{proSample} \tag{7.8}$$

Dieses Kostenmaß wird beispielsweise in dem in [BHFN02] dargestellten Verfahren zur modellbasierten Entwurfsraumexploration für SoCs zur Bewertung unterschiedlicher Architekturblöcke herangezogen. In dieser Arbeit wird das ATE-Produkt nicht verwendet, da hier ein Vergleich der Architekturen anhand der Einzelkriterien, bzw. des AT-Produkts sinnvoller erscheint.

7.1.6 Flexibilität

Von besonderer Bedeutung ist neben den oben aufgeführten physikalischen Kriterien zusätzlich die Bewertung der Flexibilität einer Architektur. Rein subjektiv wird unter Flexibilität die Möglichkeit verstanden, eine Architektur, bzw. die darauf implementierte Soft-

ware, an unterschiedliche Algorithmen anpassen zu können. Demnach ist die Flexibilität umso höher, je größer die Menge der abbildbaren Algorithmen ist.

Bekannt ist, dass ein vollkundenspezifischer Entwurf die besten Eigenschaften in Bezug auf den Flächenbedarf, die Datendurchsatzrate und den Energieverbrauch besitzt, demgegenüber jedoch keinerlei Flexibilität aufweist. Demzufolge lässt sich der zusätzlich zu betreibende Aufwand zur Bereitstellung einer Programmierbarkeit oder Rekonfigurierbarkeit verglichen mit dedizierten Komponenten durch eine höhere Flexibilität motivieren. So ergeben sich beispielsweise aus dem primären Vorteil der Anpassbarkeit an verschiedene Randbedingungen unter Umständen erheblich geringere *einmalige Entwurfskosten* (Non-Recurring Engineering Costs, NRE-Kosten), die beim Erstellen eines Projektplanes eine wesentliche Rolle spielen.

Im Gegensatz zu den physikalischen Eigenschaften lässt sich bezüglich der Flexibilität kein quantitatives Bewertungmaß bilden. Zwar könnte beispielsweise die Anzahl der implementierbaren Algorithmen als Kriterium herangezogen werden, dies wird jedoch dem subjektiven Verständnis des Begriffs der Flexibilität nicht gerecht. So können beispielsweise sowohl auf einem FPGA, als auch auf einem DSP theoretisch beliebig viele unterschiedliche Algorithmen implementiert werden. Dennoch wird die Flexibilität eines DSP aufgrund der im Vergleich zum FPGA relativ einfachen Software-Programmierbarkeit im Allgemeinen subjektiv höher bewertet.

Eine weiter gefasste Definition der Flexibilität schließt neben einer rein subjektiven Bewertung der Architektur zusätzlich deren physikalische Eigenschaften mit ein, bzw. die Eigenschaften, welche die Struktur bei Abbildung unterschiedlicher Algorithmen bietet. Nach dieser Definition bietet diejenige Implementierung die größte Flexibilität, deren Eigenschaften wie Implementierungsfläche, Datenrate und Energieverbrauch sich bei Abbildung unterschiedlicher Anwendungen aus unterschiedlichen Klassen von Algorithmen am wenigsten ändert. Den Extremfall stellt also auch nach dieser Definition ein kundenspezifischer Entwurf einer dedizierten Architektur dar, da dieser für eine ganz bestimmte Anwendung optimal, für andere Anwendungen jedoch völlig ungeeignet, und die physikalischen Eigenschaften für diese Anwendung gewissermaßen unendlich schlecht sind. In Abschnitt 7.2.7 wird diese Definition zur Bewertung der in der vorliegenden Arbeit betrachteten Architekturen angewendet.

7.2 Simulationsergebnisse

7.2.1 Hinweise zu den Simulationen

Annahmen

Mit Hilfe des in Kapitel 4 dargestellten Analyseablaufs sind Simulationen zur Abschätzung der VLSI-Eigenschaften der Architekturmodelle durchgeführt worden. Um einen fairen

Vergleich der Modelle untereinander und mit DSPs, FPGAs und Standardzellentwürfen zu ermöglichen, sind eine Reihe von Annahmen zu treffen, die im Folgenden dargestellt werden.

Jede Variante der grobgranularen Architekturen ist unter den folgenden Prämissen entworfen worden:

- Mit jeder Architektur sollen zumindest die in Kapitel 5 dargestellten linearen Benchmark-Algorithmen, also FFT, DCT, FIR-Filter und Matrix-Vektor-Multiplikation implementierbar sein.
- Alle grobgranularen Modelle sollen gleich viele Rechenressourcen aufweisen. Da die Datendurchsatzrate bei linearen Algorithmen in den meisten Fällen von der Anzahl der Multiplizierer abhängig ist, sollen alle Varianten gleich viele Multiplizierer-Blöcke aufweisen.

Weiterhin wird, wie bereits in Abschnitt 5.2.2 erläutert, der Granularitätsgrad einer Architektur als umso höher angenommen, je mehr Multiplizierer die Basiszellen enthalten. Zusätzlich zu den in Abschnitt 5.2.2 dargestellten grundlegenden Basiszell-Strukturen sind Implementierungsvarianten mit jeweils einer, bzw. zwei Pipeline-Stufen pro PE untersucht worden.

Zum Vergleich grob- und feingranularer Architekturen

Zum Vergleich der Eigenschaften grob- und feingranularer Architekturen sind jeweils vollständige Algorithmenkerne implementiert worden. Die Simulationsergebnisse beinhalten demzufolge auch den Einfluss der Kontrolllogik und des Speichers. Lediglich die Fläche des FPGA-BlockRAMs geht nicht in die hier angegebene FPGA-Fläche ein, da für die Block-RAM-Fläche keine verlässlichen Daten zur Verfügung standen.

Es wird daher angenommen, dass die rekonfigurierbare Einheit als eigenständiger Coprozessor agiert, welcher selbständig den Algorithmus abarbeitet. Zwischenwerte werden lokal gespeichert, so dass sich die Kommunikation zwischen der rekonfigurierbaren Architektur und einem Host-Prozessor auf das Lesen und Schreiben der Ein- und Ausgangswerte beschränkt. Dies kann nach den in Abschnitt 3.1.2 vorgestellten Definitionen mit Hilfe der *losen Kopplung* erfolgen.

Interessant ist zudem die Frage nach der Effizienz einer grob- oder feingranularen Architektur, wenn die Anbindung an den Host-Prozessor nach *enger Kopplung* erfolgt. In diesem Fall wird die Architektur als rekonfigurierbare Befehlssatzerweiterung in die Prozessorpipeline eingebunden, während die Kontrolllogik auf dem Host-Prozessor implementiert ist. Da die externen Schnittstellen beider Architekturvarianten identisch sein können, beschränkt sich die Frage nach der Effizienz auf eine Untersuchung der Implementierung des Datenpfades ohne Kontrolllogik und ohne Speicher. In [vNBN06] wird die Effizienz einer

anwendungsspezifisch optimierten eFPGA-Architektur untersucht, indem einzelne Operatoren auf die Struktur abgebildet werden, ohne Kontrolllogik und Speicher zu betrachten (siehe auch Abschnitt 3.1.2). In dieser Arbeit wird nach ähnlichem Schema zur Effizienzanalyse nach enger Kopplung die Implementierung eines Radix-2-Butterfly-Operators auf die grob- und feingranularen Architekturen herangezogen.

Wortbreite der betrachteten Architekturvarianten

Die Datenwortbreite in der Bild- und Videosignalverarbeitung beträgt häufig 8 Bit. In anderen anderen Einsatzbereichen, beispielsweise in der Audiosignalverarbeitung oder der digitalen Kommunikationstechnik, kommen dagegen höhere Wortbreiten zum Einsatz. Eine Architektur, die den Anspruch erhebt, Algorithmen aus mehreren Anwendungsgebieten ausführen zu können, muss daher auf die höhere Wortbreite ausgelegt sein. Aus diesem Grunde werden für alle hier betrachteten grobgranularen Architekturmodelle 16-Bit-Datenpfade angenommen. Die Flexibilität von FPGAs erlaubt dagegen eine algorithmenspezifische Anpassung der Wortbreiten. Aus diesem Grunde sind alle auf das FPGA abgebildeten Algorithmen sowohl mit 8, als auch mit 16 Bit Wortbreite implementiert worden. Die 8-Bit-Implementierungen tragen der Anpassbarkeit des FPGAs an Algorithmenkerne mit niedrigerer Wortbreite Rechnung. Die 16-Bit-Implementierungen bieten dagegen die Möglichkeit, die grob- und feingranularen Architekturen unter den gleichen Randbedingungen bezüglich der Wortbreite miteinander zu vergleichen. In beiden Fällen wird bei den Simulationen jeweils die volle Wortbreite der Datenpfade genutzt. Für den Standardzellentwurf werden in allen Fällen 16-Bit-Datenpfade betrachtet, die Wortbreite des DSP ist ebenfalls auf 16 Bit festgelegt.

Übersicht

Eine Übersicht zu den in dieser Arbeit untersuchten Varianten grobgranularer Architekturen bietet Tabelle 7.1

Alle PE-Felder besitzen das in Abschnitt 4.4 beschriebene Block-RAM mit 16 Blöcken für jeweils 128 Werte. Die Wortbreite beträgt in allen Fällen 16 Bit. Alle Modell-Architekturen enthalten zudem einen globalen Instruktionsspeicher für insgesamt jeweils 1024 Instruktionen. Für den RD16025-DSP ist mit zwei unabhängigen 1024×16-Bit SRAM-Blöcken die identische globale Speichergröße wie für die grobgranularen Modelle angenommen worden.

7.2.2 Flächenanalyse

Tabelle 7.2 gibt zunächst einen Überblick über den absoluten Flächenbedarf aller untersuchten Architekturen. Die Flächen der grobgranularen Modellarchitekturen betragen

Tabelle 7.1: Übersicht über die untersuchten grobgranularen Modellarchitekturen

Referenz-Name	Granularität (Anzahl Mult.)	Array-Dimensionen	Kommentare
PE1M	1	4×4	Prozessorzellen mit einer Pipeline-Stufe
PE2M	2	4×2	
PE4M	4	2×2	
PE8M	8	2×1	
PE1Mp	1	4×4	Prozessorzellen mit zwei Pipeline-Stufen
PE2Mp	2	4×2	
PE4Mp	4	2×2	
PE8Mp	8	2×1	

jeweils etwas mehr als $3mm^2$, lediglich das PE1M-Array belegt $0.4mm^2$ bis $0.5mm^2$ mehr Fläche. Der DSP ist etwa um den Faktor 3 kleiner als die grobgranularen Architekturen, besitzt jedoch auch nur 2 Multiplizierer gegenüber 16 bei den grobgranularen Strukturen. Der Standardzellentwurf ist für eine 64-Punkte-FFT aufgrund der dedizierten Architektur um dem Faktor 5 kleiner. Demgegenüber belegt eine 64-Punkte FFT auf dem feingranularen FPGA bei gleicher Parallelität (16 Multiplizierer) und Wortbreite (16 Bit) fast das zwanzigfache der Fläche der grobgranularen Modelle. Bei 8 Bit Wortbreite beträgt die FPGA-Implementierungsfläche das sechsfache der Fläche der grobgranularen Architekturen.

Zusätzlich angegeben ist die Fläche ohne globalen Speicher, sowie die relative Flächeneffizienz nach Gleichung 7.1. Demnach liegt das Verhältnis zwischen Nutzfläche und Gesamtfläche zwischen etwa 60% bei den PE1M-Architekturen und maximal 68% beim PE4Mp-Array. Die Flächeneffizienz der Semi-Custom-Entwürfe liegt definitionsgemäß bei 100%, während der Wert für den DSP und das FPGA aufgrund fehlender detaillierter Flächeninformationen hier nicht angegeben werden kann.

Dennoch ist es möglich, in begrenztem Umfang einen auf andere Art erhaltenen Detailvergleich zwischen grob- und feingranularen Architekturen anzugeben, wenn statt des Xilinx VirtexE die analytischen FPGA-Strukturen herangezogen werden, die in [BRM99] vorgestellt werden (siehe auch Abschnitt 3.1.1). In [BRM99] wird der Flächenbedarf der einzelnen Bauelemente eines FPGA-Clusters mit 10 Lookup-Tabellen abgeschätzt (s.a. Abbildung 2.8). Es ergibt sich die in Abbildung 7.1 angegebene Verteilung. Der Nutzflächenanteil nach der in Abschnitt 7.1.1 angegebenen Definition beträgt 41% innerhalb des Clusters, wenn die Lookup-Tabelle und die Register zur Nutzfläche, und die anderen

Tabelle 7.2: Gesamt-Flächenbedarf der Architekturen

Name	Gesamtfläche (mm^2)	Gesamtfläche ohne globalen Speicher (mm^2)	Relative Flächeneffizienz
PE1M	3.38	1.65	59.78%
PE2M	3.01	1.28	64.03%
PE4M	3.00	1.27	66.08%
PE8M	3.05	1.32	59.61%
PE1Mp	3.51	1.78	63.69%
PE2Mp	3.07	1.34	66.83%
PE4Mp	3.06	1.32	68.80%
PE8Mp	3.10	1.38	62.57%
RD16025 DSP Core	1.08	0.5	k.a.
Xilinx XCV300E FPGA, 64-Punkt FFT, 16 Mult., 8 Bit	18.41	k.a.	k.a.
Xilinx XCV300E FPGA, 64-Punkt FFT, 16 Mult., 16 Bit	59.76	k.a.	k.a.
Semi-Custom-Entwurf, 64-Punkt FFT, 16 Mult.	0.66	0.50	100%

Bauelemente zur Kostenfläche gerechnet werden. Demgegenüber besitzt der reine Datenpfad (ohne Speicherblöcke, siehe Abschnitt 7.1.1) eines *PE2M*-Prozessorelementes einen Nutzflächenanteil von 86%.

Ein aussagefähiger Vergleich der Flächeneffizienz erfordert zusätzlich die Berücksichtigung der globalen Routing-Fläche. In [BRM99] werden Flächenabschätzungen für eine Vielzahl unterschiedlicher Verbindungsstrukturen angegeben. Den geringsten Flächenbedarf der dort untersuchten Routingstrukturen (die jedoch nicht die höchste Performance bietet) besitzt die Routing-Architektur, die an das Xilinx-4000X-FPGA angelehnt ist. Diese Werte werden in dieser Arbeit zum Vergleich herangezogen, bei allen anderen in [BRM99] untersuchten Varianten ist die Flächeneffizienz entsprechend schlechter. Werden die Ergebnisse in ein Diagramm eingetragen, ergibt sich das in Abbildung 7.2 gezeigte Bild.

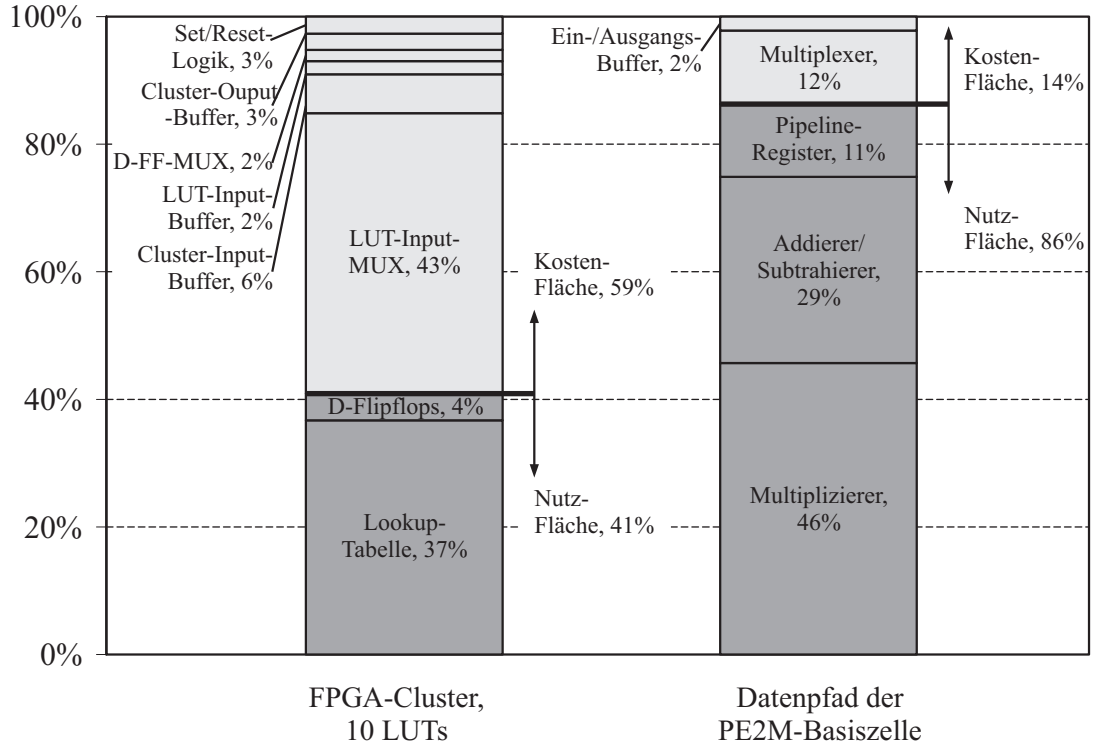


Abbildung 7.1: Vergleich der Flächeneffizienz eines FPGA-Clusters mit 10 LUTs und des Datenpfades eines PE2M-Prozessorelementes

Als FPGA-Logikzelle ist wie in [BRM99] wieder ein Cluster mit zehn Lookup-Tabellen angenommen. Der Flächenoverhead, der sich allein durch das globale Routing-Netzwerk ergibt, liegt bei 71%. Weitere 17% der Gesamtfläche belegt die bereits oben angesprochene Kostenfläche innerhalb der Cluster, so dass lediglich 12% der Gesamtfläche als Nutzfläche bezeichnet werden kann. Demgegenüber beträgt der Nutzflächenanteil der grobgranularen Modellarchitektur mit *PE2M*-Basiszelle 64%.

Nach der bisherigen allgemeinen Analyse des Flächenaufwandes sollen im Folgenden die Flächen betrachtet werden, die für bestimmte Anwendungen bei den verschiedenen Implementierungsvarianten belegt werden.

Abbildung 7.3 zeigt die Ergebnisse bezüglich des Flächenaufwandes bei einer FFT und bei der FIR-Filterung. Anhand der Darstellung lässt sich der Einfluss eines algorithmenspezifischen Parameters auf die jeweils erforderliche Fläche beobachten. Es gelten die in Abschnitt 7.2.1 erläuterten Annahmen. Alle grobgranularen Modelle besitzen insgesamt 16 Multiplizierer. Bei der FPGA-Implementierung der Algorithmen wurde aus Gründen der Vergleichbarkeit darauf geachtet, dass ebenfalls maximal 16 Multiplizierer auf das Feld abgebildet wurden. Einzige Ausnahme bildet der DSP, der systembedingt lediglich zwei

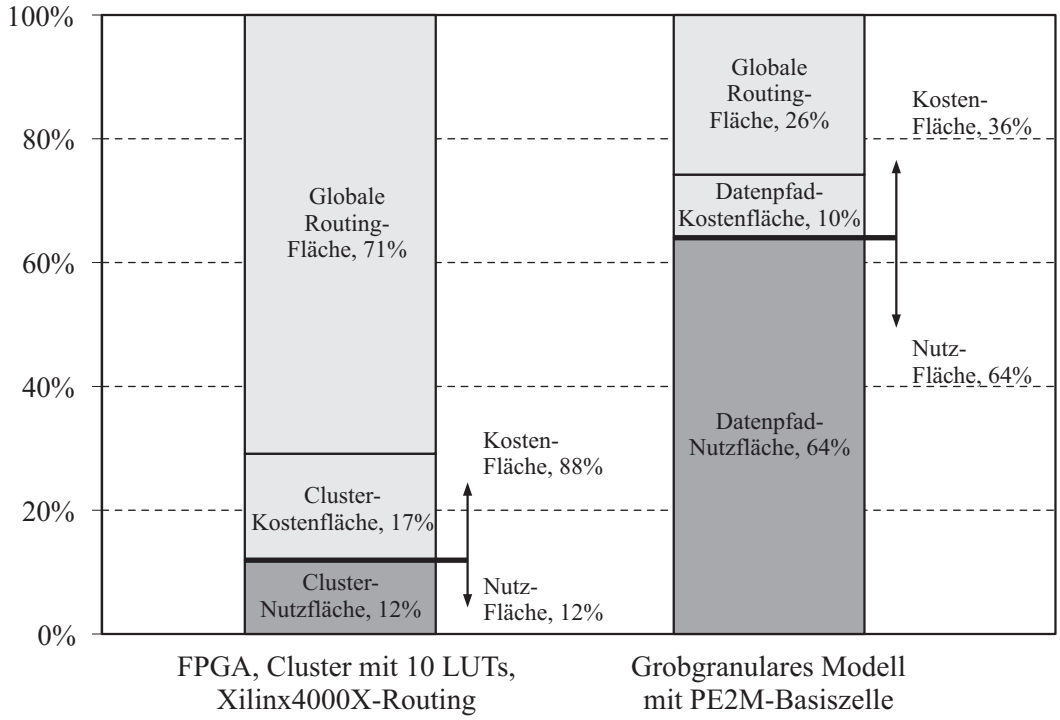


Abbildung 7.2: Vergleich der Flächeneffizienz zwischen FPGAs und den grobgranularen Modellen inklusive globaler Routing-Struktur

Multiplizierer beinhaltet, was bei der Einordnung der Flächenergebnisse entsprechend berücksichtigt werden muss.

Es zeigen sich ähnliche Flächenverhältnisse wie bereits oben angesprochen (s.a. Tabelle 7.2). Aufgrund der maximalen Multipliziererzahl von 16 sind die Flächen ab einer FFT-Länge von 8, bzw. einer FIR-Filterlänge von 32 Taps für alle Architekturen nahezu konstant. FIR-Filter mit 4 und 8 Taps benötigen entsprechend weniger Fläche, da nur 4, bzw. 8 Multiplizierer eingebunden sind. Auffällig ist, dass bei einer FPGA-Implementierung eines FIR-Filters der Flächenbedarf eines 32-Tap-Filters gegenüber einem 16-Tap-Filter nochmals ansteigt, obwohl der Parallelitätsgrad der gleiche ist. Die Ursache liegt darin, dass bei einer Filter-Länge bis zu 16 Taps Konstanten-Multiplizierer auf das Logikzellenfeld synthetisiert werden, während bei mehr Taps aufgrund der serialisierten Ausführung Multiplizierer mit zwei variablen Koeffizienten implementiert werden müssen, die entsprechend mehr Fläche belegen. Ein ähnlicher Effekt zeigt sich bei der FFT beim Übergang der FFT-Länge von 4 auf 8, sowie bei den Semi-Custom-Entwürfen, bei denen ebenfalls die Möglichkeit besteht, flächengünstige Multiplizierer mit einem festen Koeffizienten zu synthetisieren.

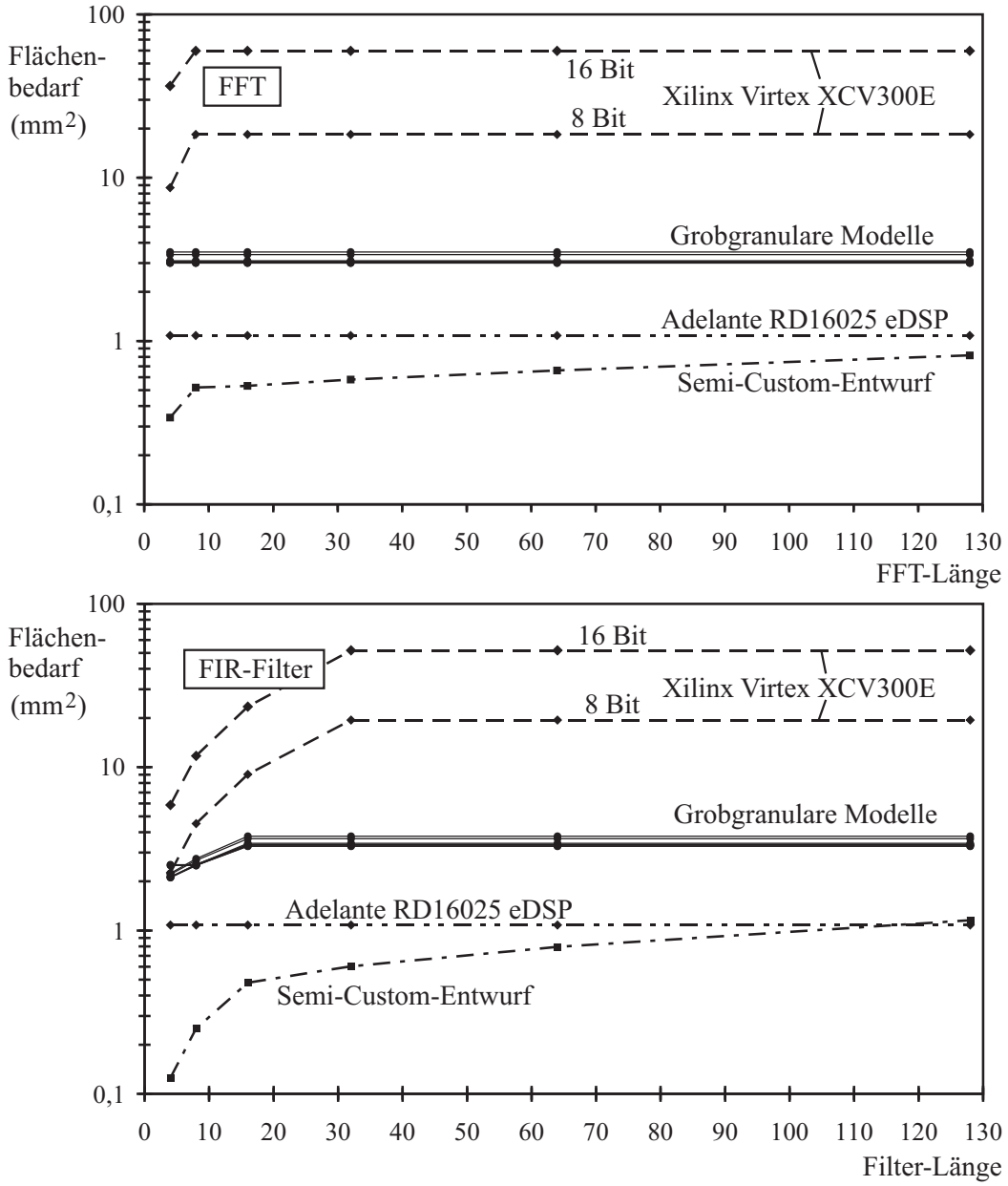


Abbildung 7.3: Flächenbedarf zur FFT-Berechnung und zur FIR-Filterung in Abhängigkeit von der FFT-Länge, bzw. der Anzahl der Filter-Taps

Abbildung 7.4 zeigt den Flächenbedarf der untersuchten Architekturvarianten bei Abbildung einiger ausgewählter Anwendungen, deren zu Grunde liegenden Algorithmen sich aus linearen Operationen zusammensetzen. Die oben bereits angesprochenen Flächenverhältnisse lassen sich auch in diesem Diagramm erkennen. Bei der DCT belegen die 8-Bit FPGA-Implementierungen etwa die achtfache Fläche der jeweils besten grobgranularen Modelle PE2M, PE4M und PE8M, und bei der 16-Bit Implementierung etwa die sechzehnfache Fläche. Bei den anderen Anwendungen verbessert sich dieses Verhältnis bezüglich des FPGAs nur geringfügig. Erkennbar ist auch der bereits oben angesprochene Effekt einer trotz gleichen Parallelitätsgrades geringeren Implementierungsfläche beim FPGA für das 16-Tap-FIR-Filter gegenüber dem 32-Tap-Filter, wenn Konstanten-Multiplizierer synthetisiert werden. Das Diagramm lässt zudem die Unterschiede zwischen den einzelnen grobgranularen Modellen deutlicher werden. Demzufolge besitzen die PE1M-Architekturen aufgrund der höheren Routingflächen eine geringfügig höhere Gesamtfläche, während alle anderen Architekturen in etwa auf einem Level liegen.

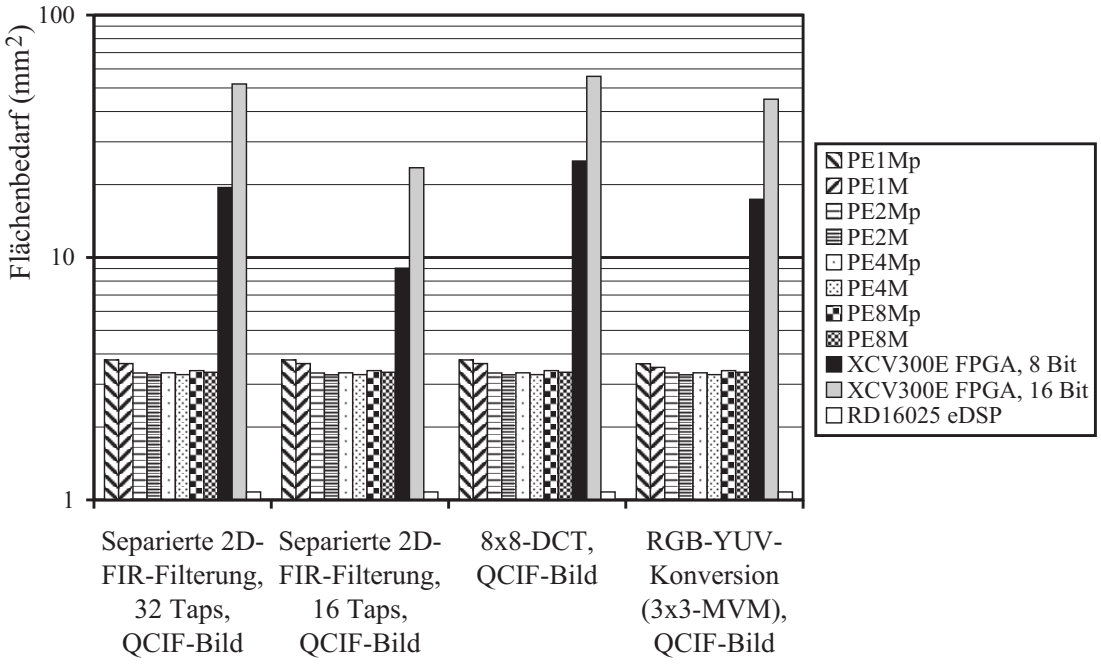


Abbildung 7.4: Flächenbedarf ausgewählter Anwendungen mit linearen Operationen

Bislang wurden Algorithmen und Anwendungen untersucht, für welche die grobgranularen Modellarchitekturen entwickelt und optimiert wurden, woraus sich die offensichtlichen Vorteile bezüglich des Flächenaufwandes ergeben. Es soll jedoch ebenfalls der Frage nachgegangen werden, welche Verhältnisse sich bei Anwendungen ergeben, für die die Modelle

nicht primär entwickelt worden sind. Aus diesem Grunde sind zwei nichtlineare Algorithmen implementiert worden, die mit Hilfe des im Abschnitt 5.3 dargestellten zusätzlichen Architekturblocks ausgeführt werden können.

Die Resultate sind in Abbildung 7.5 dargestellt, jeweils bezüglich einer nichtlinearen Filterung mit einem Medianfilter und einer Add-Compare-Select-Operation für einen Viterbi-Decoder mit der Constraint-Länge $K = 3$. Die sich ergebenden Verhältnisse bezogen auf die grobgranularen Modelle sind deutlich schlechter im Vergleich mit einer 8-Bit FPGA-Implementierung als bei den linearen Algorithmen. So liegen die Flächen der PE2M-, PE4M- und PE8M-Varianten nunmehr gleichauf mit dem Flächenbedarf einer FPGA-Implementierung, die PE1M-Modelle sind noch unwesentlich besser. Der Flächenbedarf zur Abbildung von Addierern und Komparatoren auf einem FPGA skaliert sich etwa linear mit der Wortbreite, daher belegen die mit 16 Bit implementierten nichtlinearen Algorithmen eine etwa um den Faktor zwei größere Fläche als die 8-Bit-Varianten.

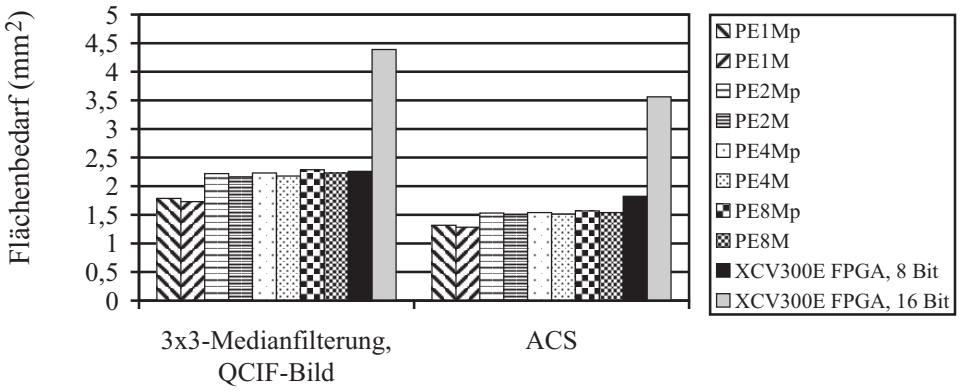


Abbildung 7.5: Flächenbedarf bei Abbildung von Anwendungen mit nichtlinearen Operationen

Die Resultate zeigen, dass die hier betrachteten grobgranularen Modellarchitekturen feingranularen Architekturen bezüglich des absoluten Flächenbedarfs bei gleicher Wortbreite um bis zu 1500% überlegen sind. Bei Reduktion der Wortbreite auf 8 Bit bei der FPGA-Implementierung beispielsweise für Anwendungen aus der Bild- oder Videosignalverarbeitung beträgt das Verhältnis immer noch etwa 700% zugunsten der grobgranularen Strukturen. Bei Anwendungen jedoch, für welche die Strukturen nicht optimiert sind, ergeben sich keine Flächenvorteile mehr. Stattdessen wirkt sich in diesen Fällen der große "Ballast"-Anteil im Form nicht nutzbarer Ressourcen stark negativ auf die Implementierungsfläche aus. Ebenfalls sollte bereits hier erwähnt werden, dass eine große Zahl von Anwendungen auf die hier betrachteten Architekturvarianten überhaupt nicht abgebildet

werden kann. Dies liegt in der Natur grobgranularer Architekturen begründet, die stets mit einer anwendungsspezifischen Ausrichtung entworfen werden.

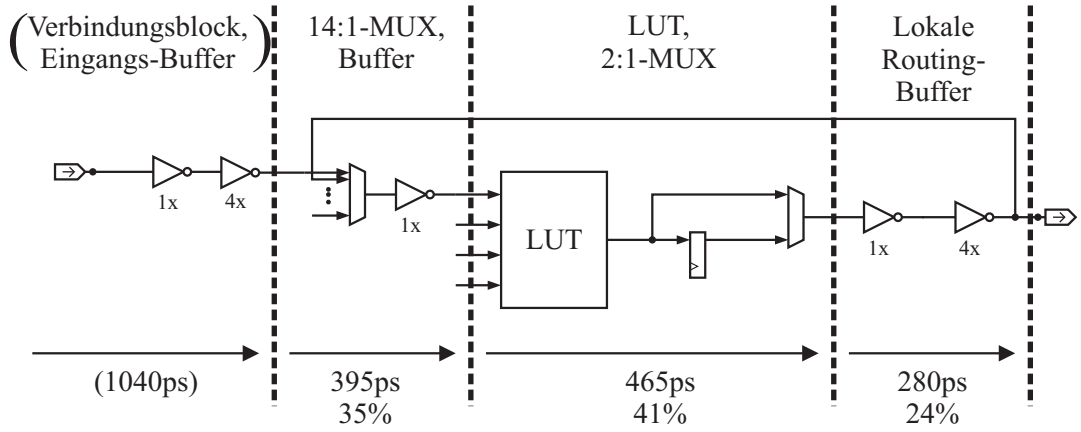
7.2.3 Zeitverhalten

Analog zur Betrachtung der Flächeneffizienz rekonfigurierbarer Bausteine ist auch bezogen auf das Zeitverhalten die Frage interessant, welchen Einfluss die Nutz- und Kostenblöcke einer Architektur auf die jeweils erzielbare Verarbeitungsgeschwindigkeit besitzen. Allerdings ist ein zeitliches Effizienzmaß schwieriger zu definieren, da beispielsweise bei feingranularen Architekturen die Verzögerungszeit jedes Pfades unmittelbar vom jeweiligen Routing abhängig ist, welches sich von Algorithmus zu Algorithmus unterscheidet. Aus diesem Grunde wird hier auf ein analog zur Flächeneffizienz definiertes algorithmenunabhängiges Maß für die zeitliche Effizienz verzichtet. Es lassen sich allerdings die Zusammensetzungen der Pfadverzögerungen innerhalb des Datenpfades einer fein- bzw. grobgranularen Basiszelle für einen Vergleich der zeitlichen Eigenschaften der beiden Architekturvarianten heranziehen.

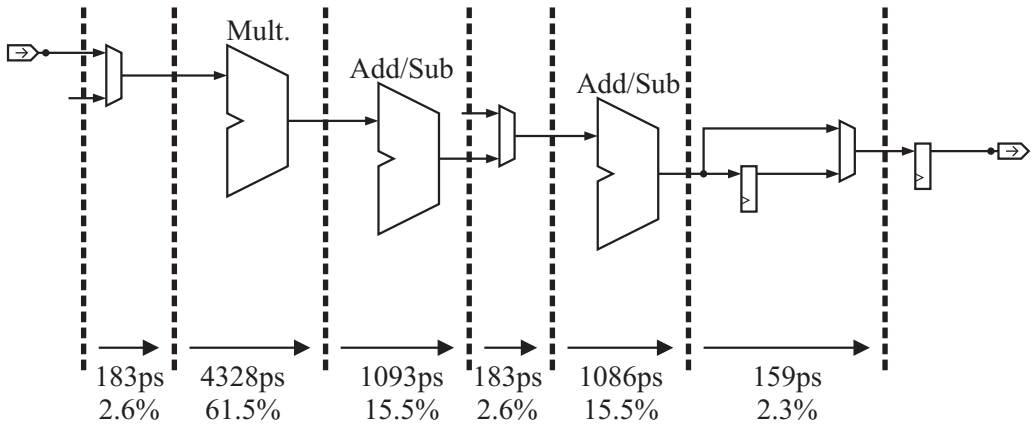
In Abbildung 7.6 sind die Timing-Profile eines feingranularen Logikzellen-Clusters mit zehn Lookup-Tabellen, sowie einer grobgranularen PE1M-Basiszelle inklusive der Verzögerungszeiten der einzelnen Elemente dargestellt. Die absoluten Zeitangaben des FPGA-Clusters beziehen sich auf eine 0.35μm-Technologie, und sind [BRM99] entnommen, während die Laufzeiten der grobgranularen Zelle auf eine 0.18μm-Technologie bezogen sind. Interessanter als die absoluten Zeiten sind die relativen Verzögerungszeiten der Bauelemente im Vergleich zur Gesamtverzögerung. Die Verzögerungszeit der Lookup-Tabelle ist in [BRM99] nur in Kombination mit dem nachgeschalteten Multiplexer angegeben, so dass diese Zeit hier als Nutzblock-Verzögerung angenommen wird.

Es zeigt sich, dass die Verzögerungszeit der Lookup-Tabelle inklusive des Multiplexers nur 41% der gesamten Verzögerungszeit des Cluster-Datenpfades ausmacht. Dies steht in ungefährer Übereinstimmung mit der Flächeneffizienz des betrachteten FPGA-Clusters, bei dem die Lookup-Tabelle 37% der Gesamtfläche einnimmt (siehe Abschnitt 7.2.2). Die Overhead-Blöcke des Clusters, hier die Multiplexer und lokalen Routing-Buffer, sind dagegen zu 59% an der Pfadverzögerung beteiligt. Demgegenüber wird die Durchlaufzeit der PE1M-Basiszelle zu 92.5% von den Nutzblöcken, also dem Multiplizierer und den Addierern, bestimmt, so dass der Overhead-Anteil einer derartigen Zelle bezogen auf das Zeitverhalten lediglich 7.5% beträgt.

Tabelle 7.3 listet die maximal möglichen Taktfrequenzen der untersuchten Architekturvarianten für verschiedene Algorithmen auf. Wie in Abschnitt 7.1.2 bereits dargestellt, ist die alleinige Angabe der Taktrate zur Bewertung der zeitlichen Performanz eines Rechnersystems nicht ausreichend, da keine Aussage bezüglich der Parallelität getroffen wird. Dennoch ist eine Übersicht über die absoluten maximalen Taktfrequenzen sinnvoll, da hier bereits erste Unterschiede deutlich werden, und die Ergebnisse der nachfolgenden Diagramme besser eingeordnet werden können.



a) Timing-Profil FPGA-Cluster (0.35µm Technologie)



b) Timing-Profil grobgranulare Basiszelle (PE1M, 0.18µm Technologie)

Abbildung 7.6: Timing-Profile eines FPGA-Clusters (Verzögerungszeiten bezogen auf eine $0.35\mu m$-Technologie aus [BRM99]) und einer grobgranularen Basiszelle ($0.18\mu m$)

Die Taktfrequenzen der Architekturmodelle mit Prozessorzellen mit zwei Pipelinestufen sind erwartungsgemäß höher als bei nur einer Pipelinestufe pro PE. Der genaue Betrag ist vom Algorithmus und der jeweiligen Zelle abhängig. Die PE8M-Strukturen bieten aufgrund des lokalen Interconnects der Basiszelle und der damit verbundenen kurzen Kommunikationswege in fast allen Fällen die höchsten Taktraten. Bei den anderen PEs ist die Frequenz teilweise signifikant niedriger, da mehr Verbindungen über die langsamere globa-

le Interconnect-Struktur geschaltet werden müssen. Dies gilt nicht für die FIR-Filterung, da hier ausschließlich benachbarte Zellen miteinander kommunizieren, was sich in einer höheren Taktfrequenz äußert. Bei den bekanntermaßen sehr kommunikationsintensiven Algorithmen Medianfilterung und ACS-Operation liegt die Taktrate aufgrund der aufwändigen Konfiguration der Verdrahtung mit langen Signalwegen signifikant niedriger.

Bei den linearen Algorithmen liegen die erreichbaren Taktfrequenzen des FPGA bei 8-Bit-Implementierung im Bereich der grobgranularen Modelle mit einer Pipeline-Stufe. Bei den PEs mit zwei Pipeline-Stufen lassen sich entsprechend höhere Taktraten erzielen. Die 16-Bit-Implementierungen auf dem FPGA sind langsamer aufgrund der längeren Signalwege. Bei den nichtlinearen Algorithmen profitiert das FPGA von der feingranularen Anpassbarkeit, was sich in entsprechend höheren Taktraten äußert.

Tabelle 7.3: Maximale Taktfrequenzen der betrachteten Architekturen für verschiedene Algorithmen

Architektur	128-PT. FFT (MHz)	128-Tap FIR (MHz)	3×3-Medianfilter (MHz)	ACS-Operation (MHz)
PE1M	91	102	75	73
PE2M	89	103	74	71
PE4M	83	101	70	77
PE8M	101	101	76	83
PE1Mp	113	134	115	110
PE2Mp	109	131	110	105
PE4Mp	100	128	101	116
PE8Mp	128	128	115	131
RD16025 DSP Core	210	210	210	210
Xilinx XCV300E FPGA, 8 Bit	103	98	162	136
Xilinx XCV300E FPGA, 16 Bit	93	86	131	113
Semi-Custom-Entwurf	135	153	261	240

Bei Performanzuntersuchungen ist letztendlich die Frage entscheident, wie schnell eine Anwendung ausgeführt werden kann. Im Folgenden werden daher die Datendurchsatzraten und Rechenzeiten bei unterschiedlichen Algorithmen und Anwendungen betrachtet.

Abbildung 7.7 zeigt die Block- und Sampleraten der untersuchten Architekturen für die FFT und die FIR-Filterung. Bei steigender FFT- bzw. Filterlänge sinken die Sampleraten bei allen Varianten aufgrund des gleichbleibenden Parallelitätsgrades von maximal 16 Multiplizierern. Grundsätzlich ist zu beobachten, dass sich die Block- bzw. Sampleraten einer FPGA-Implementierung auf dem Niveau der grobgranularen Modelle bewegen. Die Sampleraten der Semi-Custom-Entwürfe dagegen liegen aufgrund der höheren Taktraten (siehe Tabelle 7.3) höher als die der rekonfigurierbaren Architekturen. Bei den Ergebnissen der DSP-Implementierung ist zu beachten, dass die Programmausführung serialisiert mit nur zwei MAC-Stufen erfolgt, so dass die Sampleraten entsprechend niedriger liegen.

Einen Vergleich der Verarbeitungsgeschwindigkeit für einige lineare Anwendungen aus der Bildsignalverarbeitung zeigt Abbildung 7.8. Aufgetragen ist jeweils die Zeit, die erforderlich ist, um den Algorithmus auf ein QCIF-Bild (176×144 Pixel) anzuwenden. Erkennbar ist, dass die Rechenzeiten aller grobgranularen Modellarchitekturen in etwa auf dem Niveau der 8-Bit FPGA-Implementierung liegen. Für das 32-Tap-Filter ist die FPGA-Implementierung sogar um ein Drittel schneller, da ab 32 Taps ein zusätzlicher Wartezyklus bei den grobgranularen Modellen eingelegt werden muss. Der Einfluss dieses Wartezyklus verringert sich allerdings mit steigender Filterlänge, was auch aus Abbildung 7.7 hervorgeht. Die Rechenzeit des DSP ist je nach Anwendung um den Faktor 2.5 bis 9 langsamer.

Analog zur Betrachtung des Flächenaufwandes ist auch bezüglich der Rechenzeit die Untersuchung von Anwendungen interessant, für welche die Architekturen nicht explizit entworfen worden sind. Abbildung 7.9 zeigt die Ergebnisse bezüglich der Median-Filterung und der ACS-Operation. In beiden Fällen ist die 8-Bit FPGA-Implementierung aufgrund des optimierten Mappings bis zu doppelt so schnell wie die langsamste grobgranulare Modellstruktur. Selbst die 16-Bit-Implementierung ist noch um bis zu ein Drittel schneller.

Aus den obigen Betrachtungen bezüglich der Rechengeschwindigkeit ergibt sich, dass feingranulare Strukturen wie FPGAs durchaus in der Lage sind, mit den hier betrachteten grobgranularen Modellen mitzuhalten. In Fällen, in denen pure Rechenleistung das ausschlaggebende Kriterium ist, sind FPGAs daher aufgrund der gleichzeitig hohen Flexibilität grobgranularen Strukturen vorzuziehen. Allerdings ist maximale Rechenleistung häufig nicht das primäre Entwurfsziel, beispielsweise wenn die Performanzanforderungen einer Spezifikation sich auch mit langsameren Architekturen erfüllen lassen. In diesen Fällen sind insbesondere die Fläche und der Energieverbrauch kritisch, während die Geschwindigkeit den Status einer Randbedingung besitzt.

7.2.4 AT-Produkt

Bislang wurden die Kriterien Fläche und Datenrate einzeln betrachtet. Wie in Abschnitt 7.1 bereits erläutert, lässt sich anhand der alleinigen Auswertung eines der beiden Maße noch keine Aussage bezüglich der Effizienz treffen, die eine bestimmte Architektur für

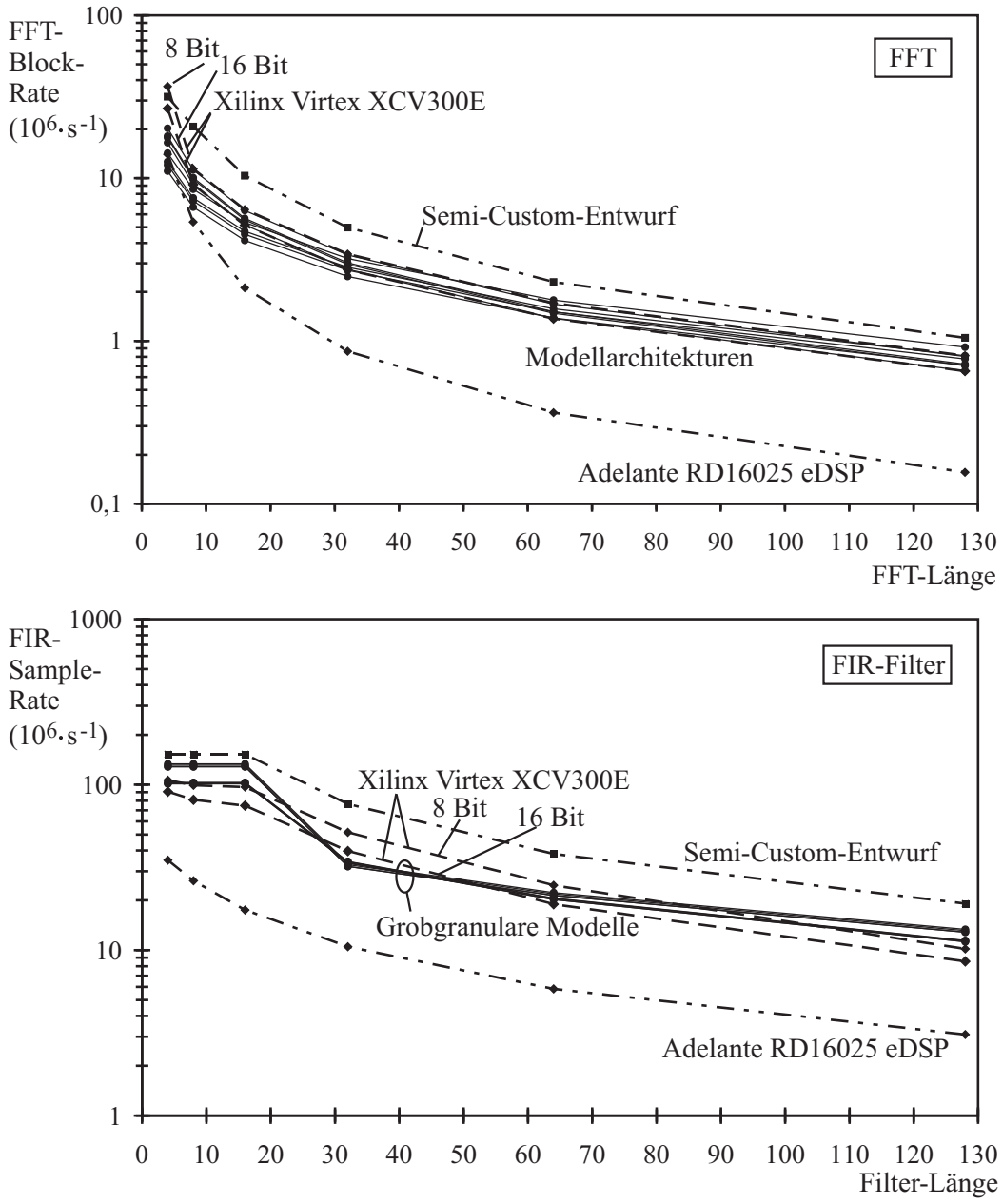


Abbildung 7.7: Block- und Sampleraten bei der FFT, bzw. beim FIR-Filter in Abhängigkeit von der FFT-Länge, bzw. von der Anzahl der Filter-Taps

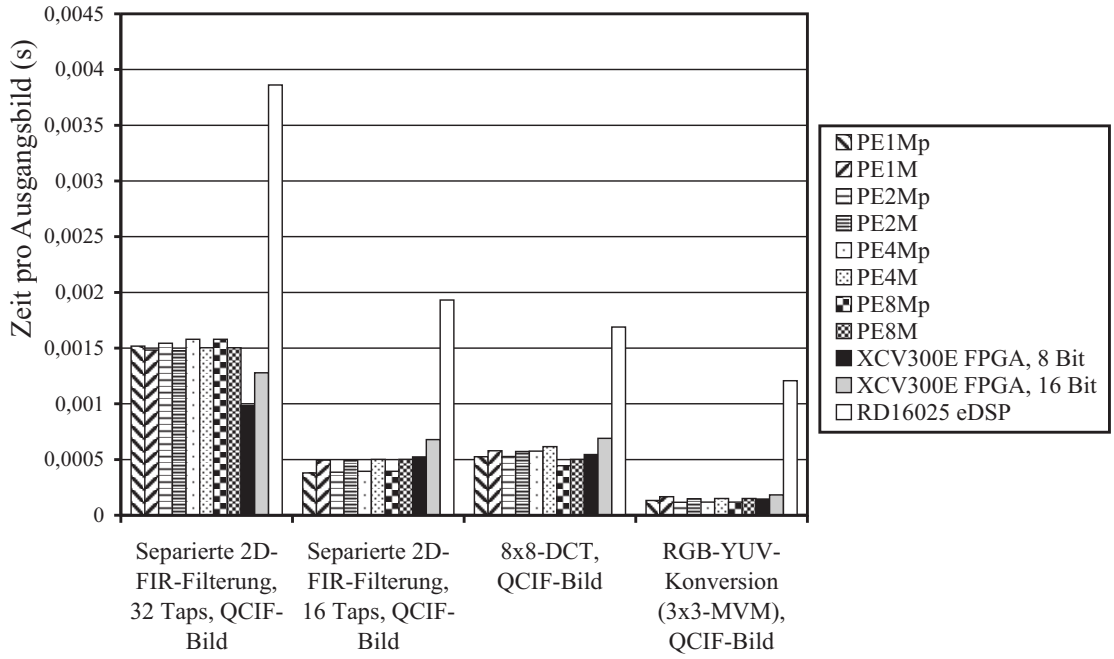


Abbildung 7.8: Rechenzeit pro Ausgangsbild bei ausgewählten Anwendungen mit linearen Operationen

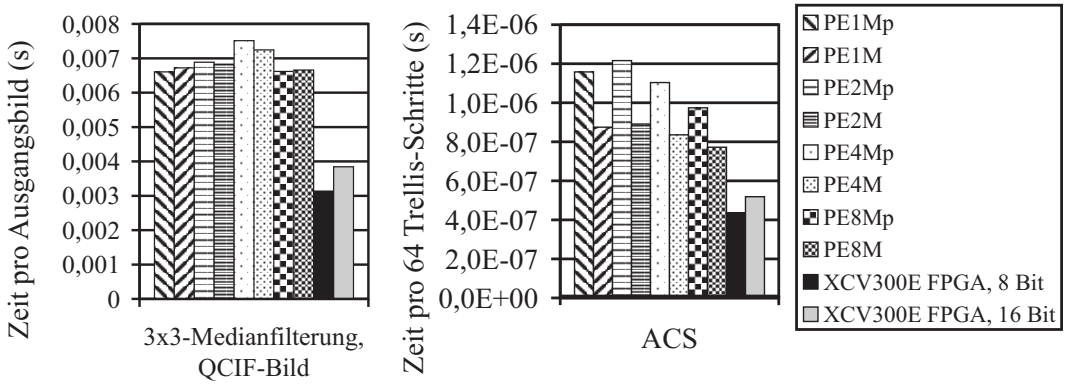


Abbildung 7.9: Rechenzeit pro Ausgangsbild/Ausgangsblock bei Abbildung von Anwendungen mit nichtlinearen Operationen

eine ausgewählte Anwendung bietet. Dagegen ist das Produkt aus Implementierungsfläche A und Samplerate T ein Kriterium zur Bewertung der "Rechenleistungsdichte", also der

auf die Fläche normierten Datenrate. Auf diese Weise lassen sich auch Architekturen mit unterschiedlichen Parallelitätsgraden fair miteinander vergleichen.

In Abbildung 7.10 ist das AT-Produkt bei Implementierung einer FFT und einer FIR-Filterung auf die untersuchten Architekturen aufgetragen, jeweils für unterschiedliche FFT- und Filter-Längen. Zunächst fällt auf, dass die Rechenleistungsdichte des hier untersuchten Signalprozessors mit derjenigen der grobgranularen Architekturmodelle bezüglich der beiden Algorithmen fast identisch ist. Dabei ist zu beachten, dass die absolute Datenrate der grobgranularen Modelle aufgrund der höheren Parallelität je nach Anwendung um den Faktor 2.5 bis 9 höher ist.

Um den gleichen Datendurchsatz mit DSPs zu erzielen, müssten mehrere Kerne parallel betrieben werden. Der bei einem solchen Parallelbetrieb von DSPs zu treibende zusätzliche Aufwand für die Kommunikationsstruktur würde sich dabei wiederum negativ auf die Flächeneffizienz auswirken.

Dieser zusätzliche Flächenaufwand für die Verbindungsstruktur der Datenpfade ist in den Werten für das AT-Produkt der grobgranularen Architekturmodelle dagegen bereits enthalten, so dass sich die Verhältnisse bezüglich des AT-Produkts bei gleicher absoluter Rechenleistung verglichen mit dem DSP wieder zugunsten der grobgranularen Modelle verschieben. Die Rechenleistungsdichte des FPGAs ist bei beiden Anwendungen für die 8-Bit-Implementierung etwa um den Faktor 4 bis 5 schlechter als bei den grobgranularen Strukturen. Bei 16 Bit, also gleicher Wortbreite wie bei den grobgranularen Modellen, weist das FPGA eine mehr als zwanzigfach schlechtere Rechenleistungsdichte auf.

Im Folgenden sollen wieder, wie in den Abschnitten zum Flächenbedarf und zum Zeitverhalten, die grobgranularen Architekturen und die FPGA-Implementierung für verschiedene konkrete Anwendungen miteinander verglichen werden, hier bezogen auf das AT-Produkt. Die Ergebnisse sind in den Abbildungen 7.11 und 7.12 dargestellt.

Es lassen sich ähnliche Effekte beobachten wie bei der Fläche und beim Zeitverhalten. Bei Anwendungen mit linearen Operationen sind die hier untersuchten Modellarchitekturen der rein feingranularen FPGA-Struktur bezüglich der Rechenleistungsdichte bei 8-Bit-Implementierung um den Faktor 4 bis 5 überlegen (Abbildung 7.11). Bei den 16-Bit-Implementierungen ergibt sich bezüglich der Rechenleistungsdichte auf dem FPGA eine Verschlechterung um mehr als das Zwanzigfache. Bezogen auf die nichtlinearen Algorithmen Medianfilterung und ACS-Operation ergibt sich auch für das AT-Produkt die aus den vorangegangen Abschnitten bekannte Verschiebung der Ergebnisse zugunsten der feingranularen Struktur (Abbildung 7.12).

Bei alleiniger Betrachtung der grobgranularen Architekturen lassen sich bezogen auf die linearen Operationen keine signifikanten Unterschiede erkennen, abgesehen von der bekannten Differenz zwischen den Strukturen mit einer und denjenigen mit zwei Pipelinestufen. Bei der Medianfilterung bieten die PE1M-Basiszellen dagegen ein durchaus signifikant besseres AT-Produkt als die anderen Varianten.

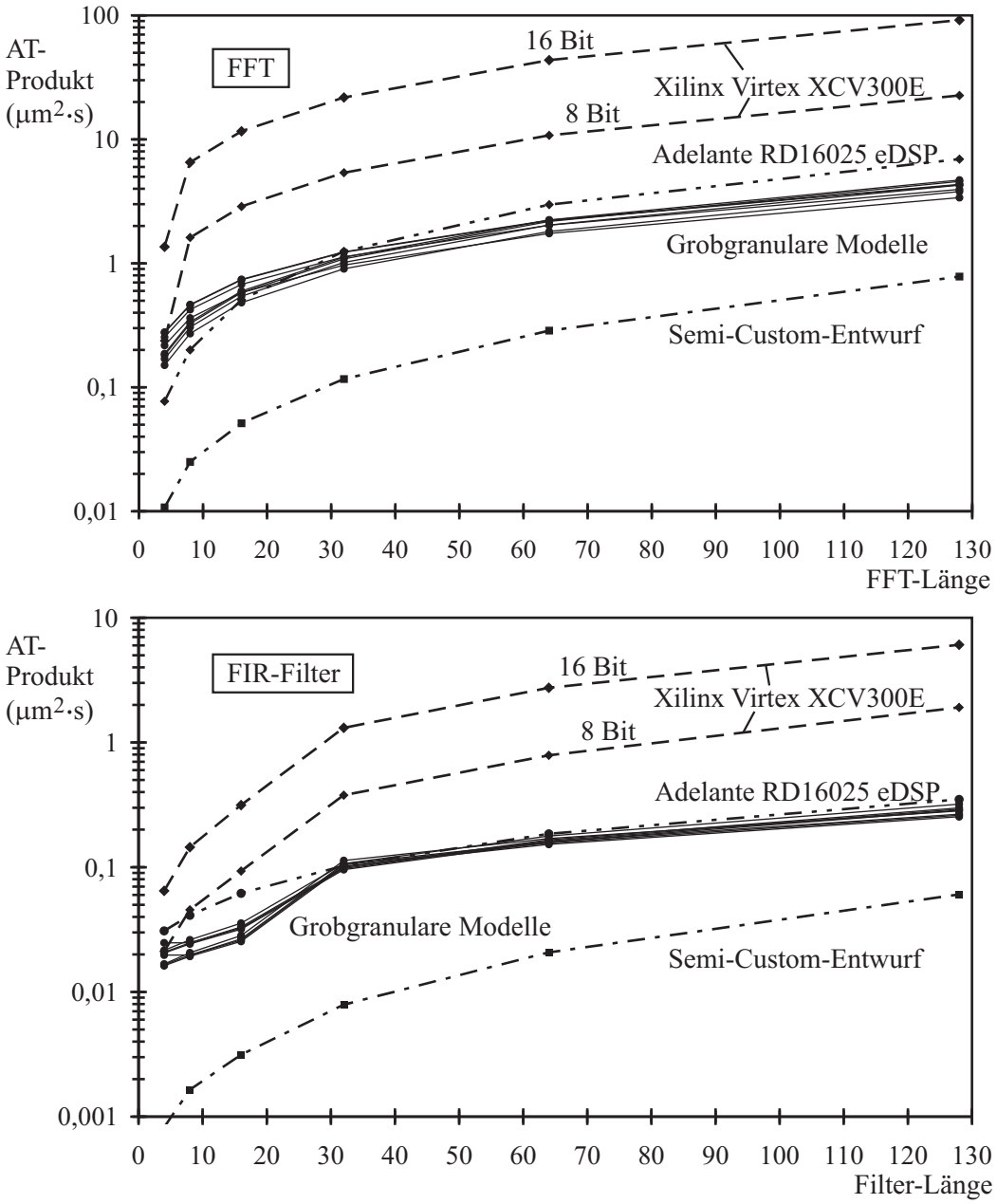


Abbildung 7.10: AT-Produkt bei der FFT, bzw. beim FIR-Filter in Abhängigkeit von der FFT-Länge, bzw. von der Anzahl der Filter-Taps

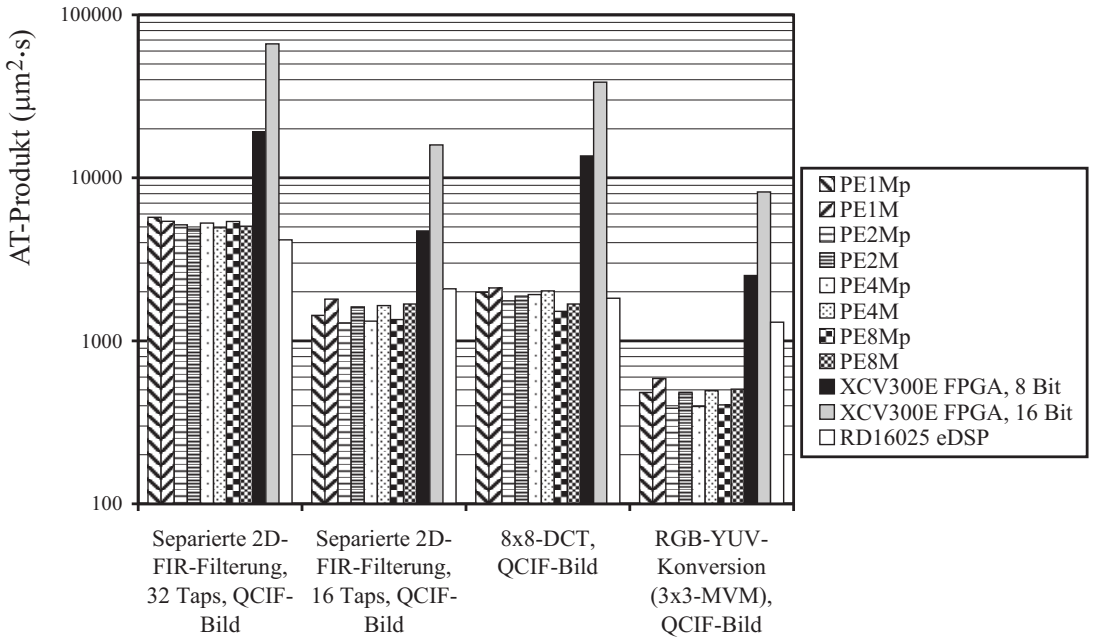


Abbildung 7.11: AT-Produkt bei ausgewählten Anwendungen mit linearen Operationen

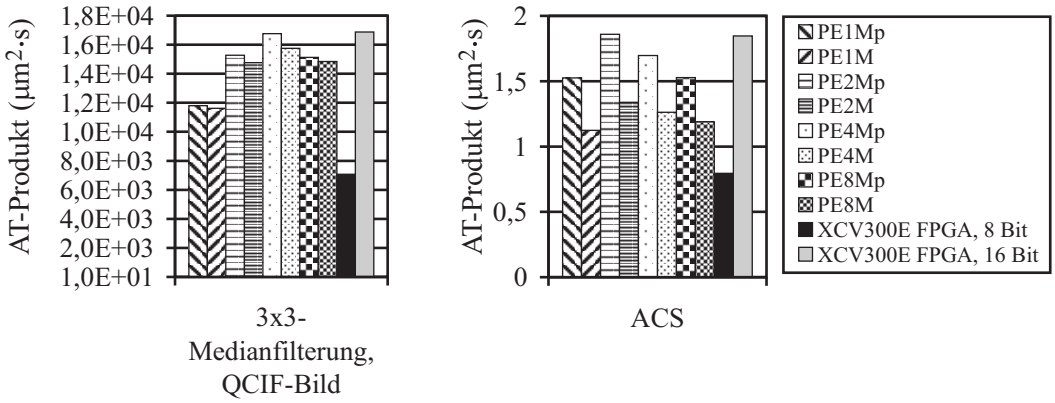


Abbildung 7.12: AT-Produkt bei ausgewählten Anwendungen mit nichtlinearen Operationen

7.2.5 Energieverbrauch

Die Bedeutung eines geringen Energieverbrauchs moderner integrierter Schaltungen ist bereits in den vorangegangenen Kapiteln angesprochen worden. In dieser Arbeit sind da-

her neben den bereits besprochenen Eigenschaften Fläche und Datenrate Untersuchungen bezüglich der Leistungsaufnahme und des Energieverbrauchs der hier betrachteten Architekturen untersucht worden. Für die dabei angewandten Methoden sei auf Kapitel 6 verwiesen.

Abbildung 7.13 zeigt zunächst einen Vergleich der Energiebilanz der grobgranularen Architekturmodelle für ein 16- und ein 32-Tap-FIR-Filter. Die dunkel schattierten Teile der Balken stehen für den Energieverbrauch der Speicherbänke, während die hell dargestellten Flächen den Energieverbrauch der Basiszellen und der Verbindungsstruktur repräsentieren. Bei beiden Anwendungen fällt der hohe prozentuale Anteil der Speicherstruktur am Gesamtverbrauch auf. Zwischen zwei Drittel und drei Viertel der Gesamtenergie geht durch Speicherzugriffe verloren. Der Anteil ist bei Abbildung eines 32-Tap-Filters höher als beim 16-Tap-Filter, da durch die erforderliche Zwischenspeicherung der Daten in den PE-internen Speicherbänken und damit verbundenen zusätzlichen Speicherzugriffen mehr Energie verbraucht wird.

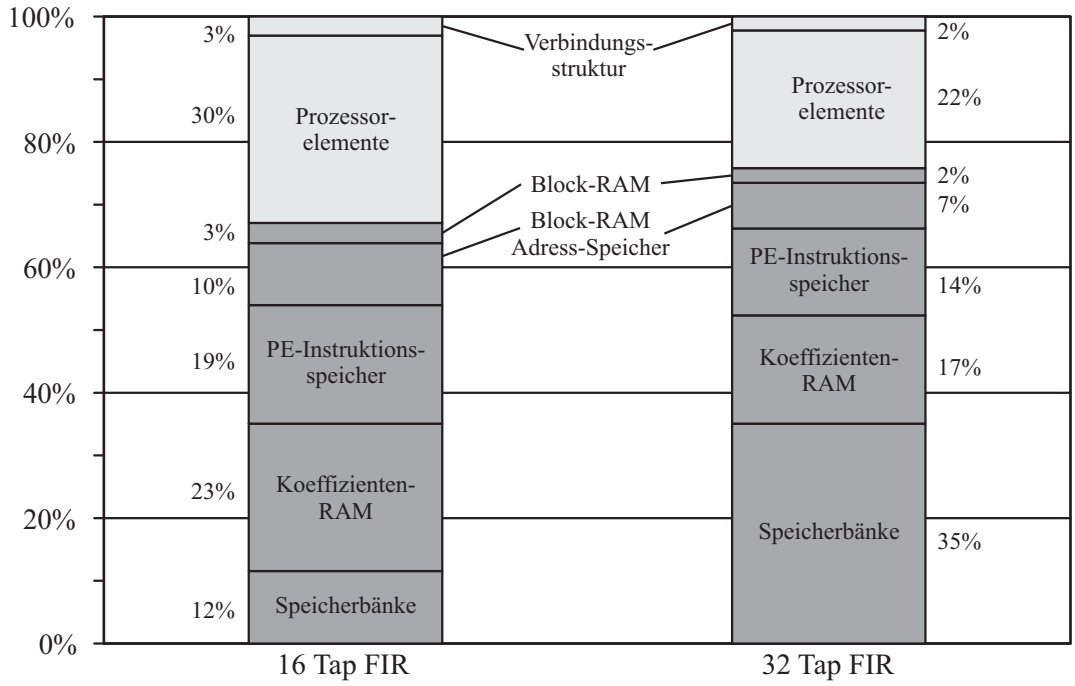


Abbildung 7.13: Prozentualer Vergleich der Energiebilanz der grobgranularen Architekturmodell bei Abbildung eines 16- und eines 32-Tap-FIR-Filters

Wie in den obigen Abschnitten soll auch die Auswertung bezüglich des Energieverbrauchs zunächst anhand der Ergebnisse bei der Abbildung zweier linearer Algorithmen erfolgen, bevor auf konkrete Anwendungen mit linearen und nichtlinearen Algorithmen

eingegangen wird. Abbildung 7.14 zeigt die Energie zur Berechnung eines Ausgangswertes der untersuchten Architekturen für die FFT und die FIR-Filterung in Abhängigkeit von der FFT-, bzw der Filter-Länge. Zunächst fällt auf, dass der Energieverbrauch der grobgranularen Modelle um bis zu eine Größenordnung, also um den Faktor 10, niedriger liegt als bei der 8-Bit-Implementierung des gleichen Algorithmus auf der feingranularen FPGA-Struktur. Bei Implementierung mit gleicher Wortbreite (16 Bit) auf dem FPGA verdoppelt bis verdreifacht sich der Energieverbrauch nochmals. Lediglich bei den parallelen Implementierungen mit bis zu 16 Taps beim FIR-Filter und der 4-Punkt-FFT, bei denen auf dem FPGA Konstanten-Multiplizierer synthetisiert werden können, fällt die Differenz mit dem Faktor 2 bis 5 geringer aus, jedoch noch immer mit signifikanten Vorteilen für die grobgranularen Strukturen. Die Messung des Energieverbrauchs des DSP-Cores ergab bei der FFT und der FIR-Filterung eine Verschlechterung etwa um den Faktor 2 bezogen auf die grobgranularen Modelle. Lediglich bei den kürzeren FFT-Längen liegen die beiden Architekturvarianten gleichauf, was sich dadurch erklärt, dass sich die bei den hier betrachteten Modellen einzufügenden Wartezyklen nach der Berechnung einer Stufe bei kurzen FFTs negativ auf den Energieverbrauch pro Abtastwert auswirken. Der Semi-Custom-Entwurf verbraucht erwartungsgemäß von allen Architekturen am wenigsten Energie.

Die in Abbildung 7.14 zu beobachtenden Verhältnisse zwischen den Architekturen bestätigen sich bei Betrachtung konkreter Anwendungen mit linearen Operationen. Abbildung 7.15 zeigt die jeweils verbrauchte Energie bei FIR-Filterung, DCT und RGB-YUV-Konversion eines Bildes im QCIF-Format. Eine 16-Tap-FIR-Filterung lässt sich von den grobgranularen Modellen etwa um den Faktor acht energiesparender ausführen als mit dem FPGA (8 Bit), bei 32 Taps liegt die Differenz noch etwas höher. Bei der DCT ist das FPGA um den Faktor 5 schlechter, und bei der Farbraumkonversion um den Faktor 12. Letzteres ergibt sich aus der Tatsache, dass bei diesem Algorithmus bei den grobgranularen Architekturen im Gegensatz zur DCT keine Wartezyklen erforderlich sind, welche die Berechnung verzögern, und sich somit negativ auf den Energieverbrauch auswirken. Wie bereits angesprochen, verschlechtert sich die Energiebilanz des FPGAs bei 16 Bit nochmals um den Faktor 2 bis 3. Die Ergebnisse bei der DSP-Implementierung der Anwendungen ergibt in allen Fällen eine etwa um den Faktor 2 schlechtere Energiebilanz im Vergleich zu den grobgranularen Architekturen.

Bei Betrachtung der nichtlinearen Anwendungen ergaben sich bezüglich des Flächenbedarfs und der Sampleraten deutlich veränderte Verhältnisse zugunsten des FPGAs. Die Rechenzeiten und das AT-Produkt fallen bei der FPGA-Implementierung sogar günstiger aus, was sich durch die optimierte Synthese auf die feingranularen Strukturen erklären lässt (vgl. Abschnitte 7.2.2, 7.2.3 und 7.2.4). Auch bei der Energiebilanz ergeben sich deutliche Unterschiede bei den nichtlinearen Anwendungen im Vergleich zu den linearen Algorithmen, jedoch verbraucht die FPGA-Implementierung immer noch mehr Energie als bei Abbildung auf die grobgranularen Architekturen. Abbildung 7.16 zeigt die gemessenen Verhältnisse. Erkennbar ergeben sich für die grobgranularen Strukturen um den Faktor 2

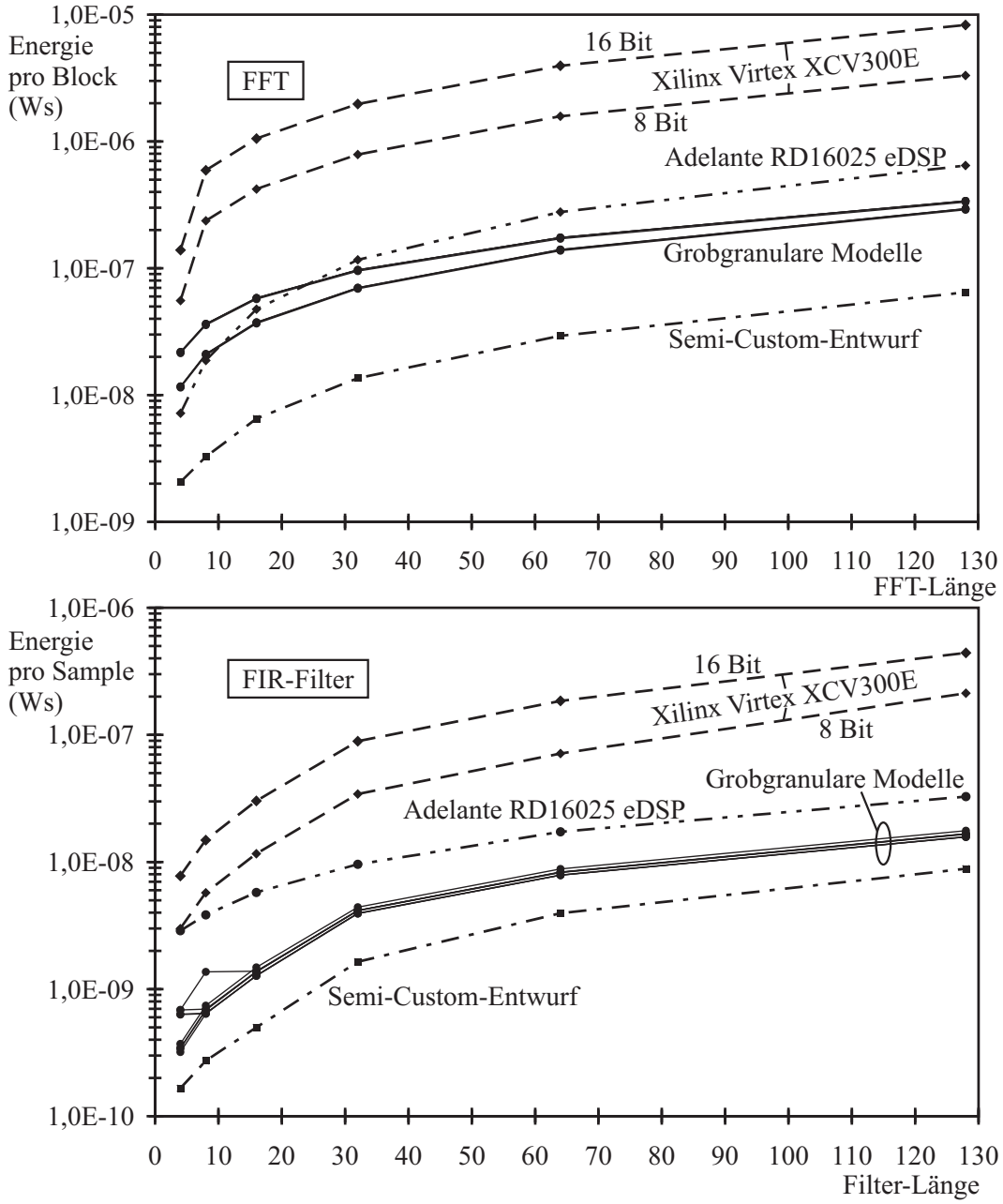


Abbildung 7.14: Energieverbrauch bei der FFT, bzw. beim FIR-Filter in Abhängigkeit von der FFT-Länge, bzw. von der Anzahl der Filter-Taps

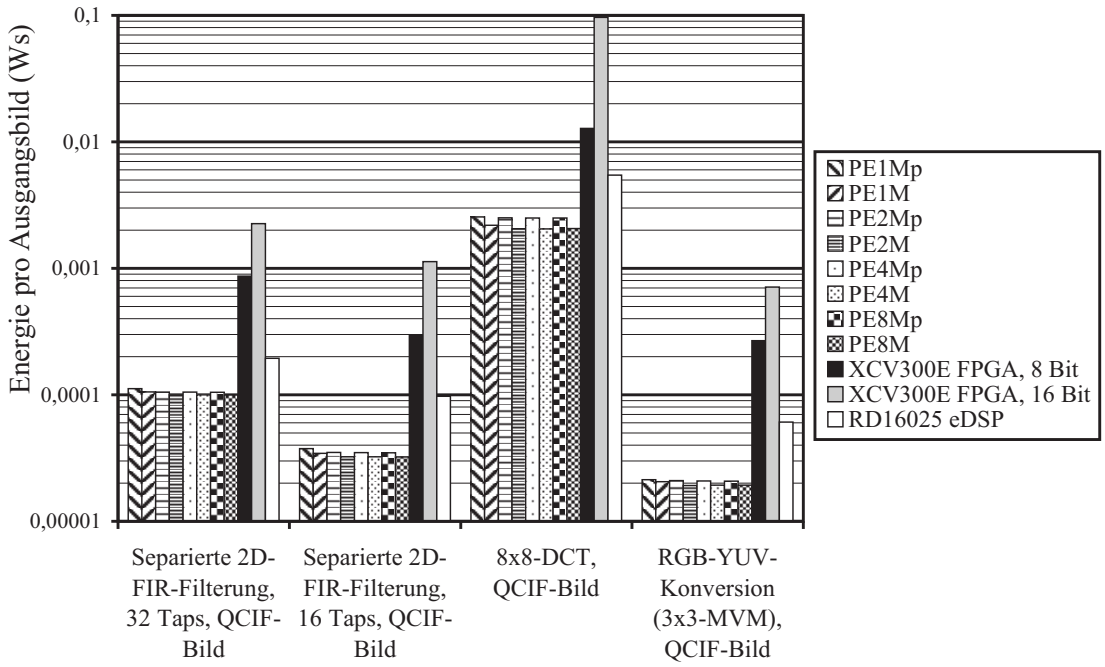


Abbildung 7.15: Energieverbrauch bei ausgewählten Anwendungen mit linearen Operationen

bis 3 energieeffizientere Implementierungen einer Medianfilterung, je nachdem ob die Architekturen mit einer oder zwei Pipelinestufen betrachtet werden. Bei der ACS-Operation sind die grobgranularen Modelle etwa um den Faktor 2.5 energiesparender als das FPGA.

7.2.6 Untersuchungen zur Effizienz rekonfigurierbarer Datenpfade

Bislang wurden die Eigenschaften der Architekturvarianten anhand von abgebildeten Anwendungen untersucht. Wie bereits in 7.2.1 dargestellt, ist zudem zumindest für den Vergleich grob- und feingranularer Architekturen die Frage von Interesse, mit welcher Effizienz einzelne Datenpfade implementiert werden können, ohne den Einfluss von Speicherbänken und Kontrollblöcken zu betrachten. Die Kenntnis der Effizienz kann beispielsweise bei der Auswahl einer Architektur zur Einbindung in die Instruktionspipeline eines Prozessors eine Rolle spielen, wenn der Prozessor selbst auch die Kontrollfunktionen übernimmt und die Speicherzugriffe regelt. Zur Analyse dieses Sachverhaltes ist eine Radix-2 Butterfly-Operation zur Berechnung der FFT (siehe Abbildung 6.15) als recht komplexe Basisoperation auf die grobgranularen Modelle und das FPGA abgebildet worden.

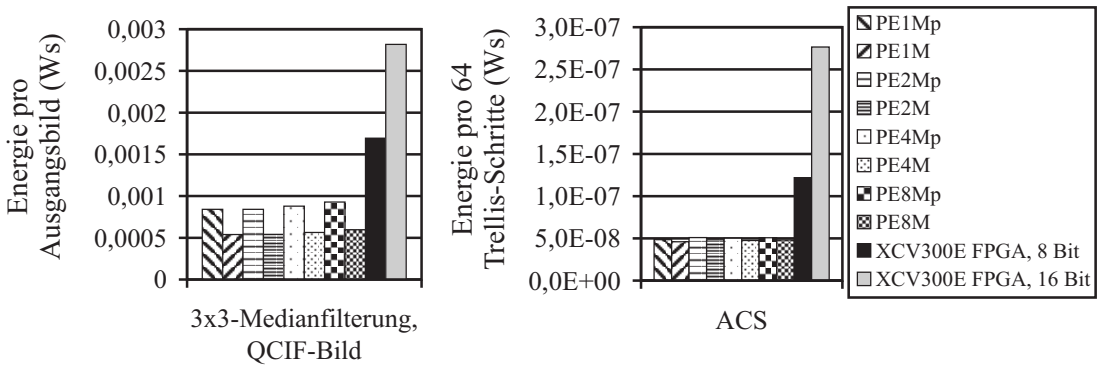


Abbildung 7.16: Energieverbrauch bei ausgewählten Anwendungen mit nichtlinearen Operationen

Grobgranulare Datenpfade und FPGAs

Die Ergebnisse des Vergleichs zwischen grob- und feingranular implementierten Datenpfaden sind in Abbildung 7.17 zusammengefasst. Demnach lässt sich eine Butterfly-Operation im Vergleich zum FPGA bis zu 20 mal flächeneffizienter implementieren, wenn auf eine anwendungsspezifische grobgranulare Architektur zurückgegriffen wird. Bei halber Wortbreite für die FPGA-Implementierung ist die grobgranulare Architektur noch um das Zehnfache kleiner. Ähnliche Verhältnisse ergeben sich für den Energieverbrauch und das AT-Produkt. Lediglich bezüglich der Datendurchsatzrate fällt der Vorteil kleiner aus. Die grobgranulare Architektur mit zwei Pipelinestufen ist etwa 35% schneller als die ebenfalls mit zwei Pipelinestufen und 16 Bit implementierte FPGA-Struktur, bei einer eventuell anwendungsspezifisch optimierten 8-Bit-Implementierung gar um etwa 30% langsamer.

Grobgranulare Datenpfade und anwendungsspezifisch optimierte eFPGAs

Die in Abbildung 7.17 dargestellten Ergebnisse beziehen sich auf einen Vergleich grobgranularer Datenpfade mit der Implementierung auf einem Standard-FPGA. Die Vorteile, die sich durch die grobgranulare, anwendungsspezifische Optimierung bezüglich Rechenleistungsdichte und Energieeffizienz ergeben, sind offensichtlich. Aus den Ergebnissen resultiert die Frage, wie sich die Verhältnisse verschieben, wenn die FPGA-Architektur ebenfalls anwendungsspezifisch ausgerichtet wird.

In [vNBN06] wird die Effizienz einer anwendungsspezifisch optimierten eFPGA-Architektur anhand einer Gegenüberstellung mit einem Cyclone-I-FPGA von Altera untersucht. Dieser Baustein besteht aus einem Logikzellenfeld mit eingebetteten Speicherblöcken, jedoch ohne dedizierte Komponenten wie Prozessorkerne oder Multiplizierer, und

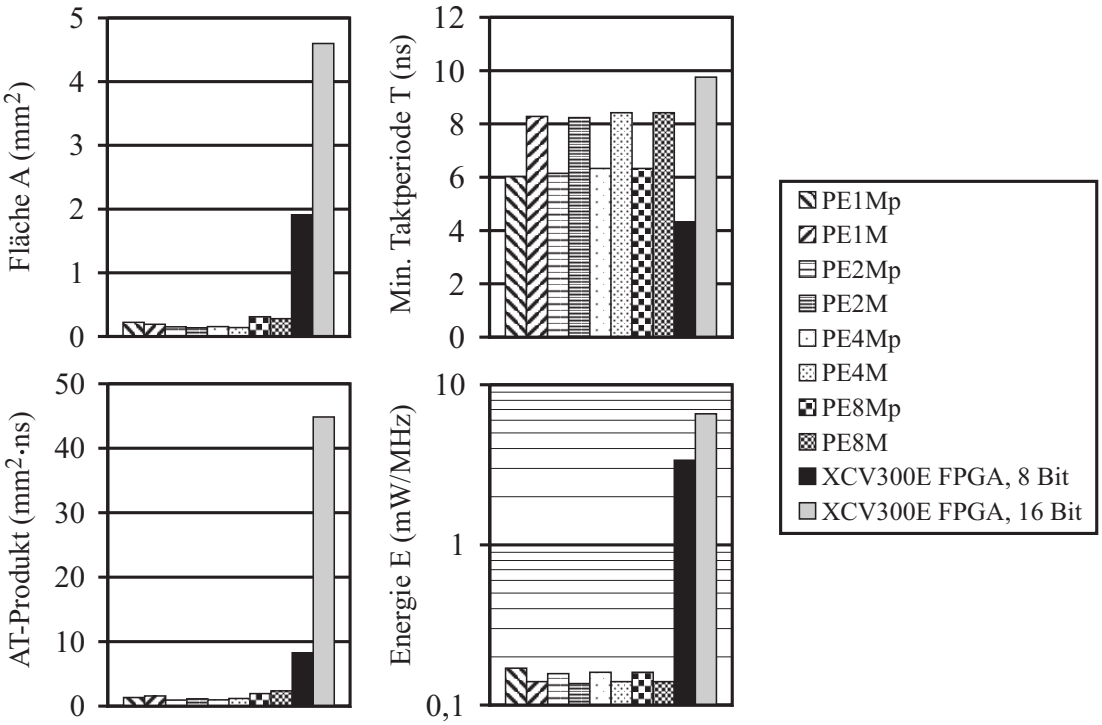


Abbildung 7.17: Auswertung der Eigenschaften fein- und grobgranularer rekonfigurierbarer Datenpfade

weist damit strukturelle Ähnlichkeiten mit dem in dieser Arbeit verwendeten Vergleichsbaustein VirtexE von Xilinx auf. Die Effizienz der Architektur ist in [vNBN06] unter anderem durch Abbildung einer 4-Bit Butterfly-Operation untersucht worden. Flächenbedarf und Energieverbrauch der anwendungsspezifisch optimierten eFPGA-Architektur konnten etwa um den Faktor 5 gegenüber einer Implementierung auf dem Cyclone-I verbessert werden, bei einer gleichzeitigen Halbierung der Verzögerungszeit. Demgegenüber sind die grobgranularen Modellarchitekturen im Vergleich zum VirtexE etwa um den Faktor 20 flächen- und energieeffizienter, bei einem Geschwindigkeitsvorteil von 35%.

7.2.7 Zusammenfassende Bewertung unter Berücksichtigung der Flexibilität

Bislang sind die VLSI-Eigenschaften Fläche, Datenrate und Energieverbrauch getrennt voneinander betrachtet worden. Lediglich die Auswertung des AT-Produkts kombiniert den Flächenbedarf und die Geschwindigkeit zu einem gemeinsamen normierten Kostenmaß, in welches jedoch der Energieverbrauch nicht eingeht. Es sind verschiedene Darstel-

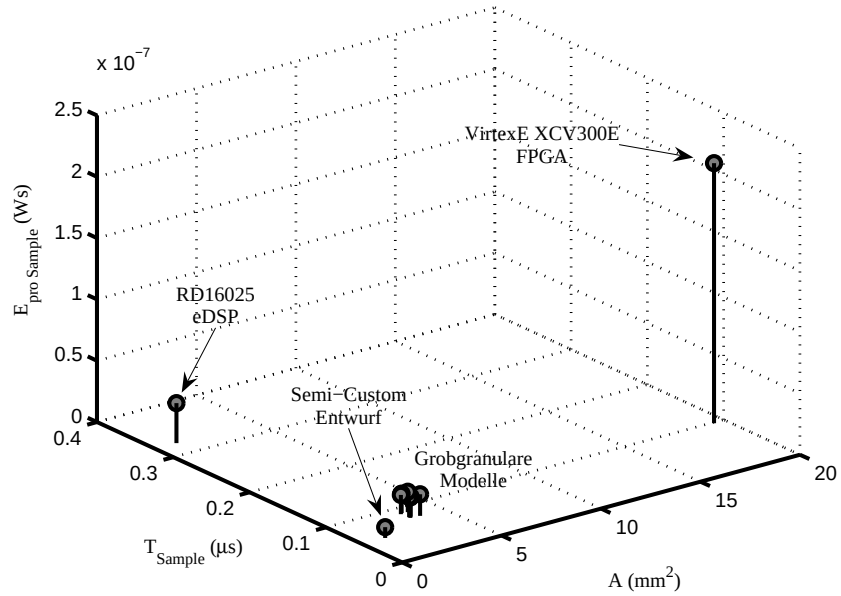
lungsformen gewählt worden, um die gemessenen und abgeschätzten Eigenschaften aller untersuchten Architekturvarianten möglichst gut aufzulösen. In diesem Abschnitt sollen die Architekturen unter Berücksichtigung aller Eigenschaften gemeinsam bewertet werden.

Physikalische Eigenschaften

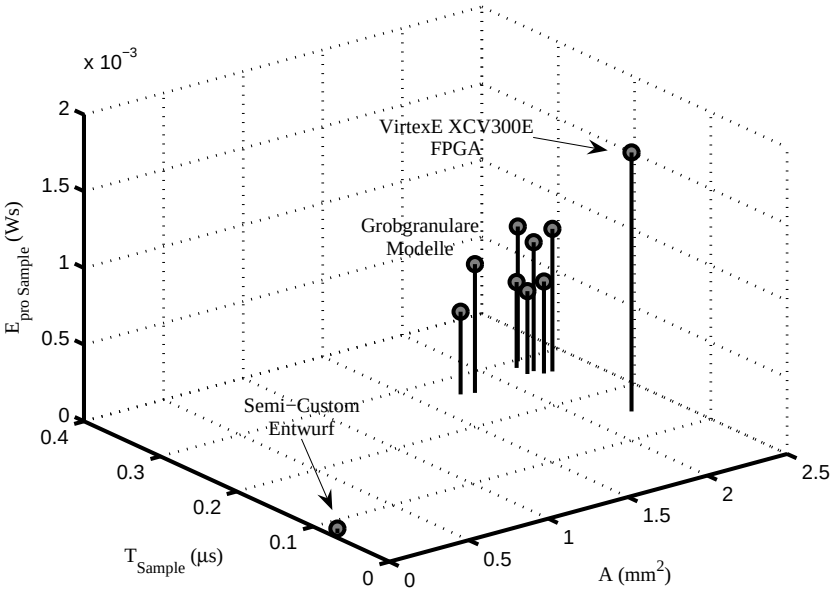
Fläche, Datenrate und Energie spannen gemeinsam einen dreidimensionalen Eigenschaftsraum auf, in den sich die betrachteten Architekturen auf Basis der gewonnenen Messergebnisse einordnen lassen. In Abbildung 7.18 sind die drei Eigenschaften für alle Strukturen bezogen auf die Abbildung eines FIR-Filters (oben) und eines Median-Filters (unten) dargestellt. Anstelle der Datenrate als Gütemaß ist auf der y-Achse jeweils die minimale Sampleperiode aufgetragen. Auf diese Weise liegen alle drei VLSI-Maße in Form von Kostenmaßen vor. Das theoretische Optimum bezüglich aller drei Eigenschaften liegt demnach im Koordinatenursprung.

Der Semi-Custom-Entwurf stellt für diese Arbeit das erreichbare Optimum dar (Full-Custom-Implementierungen wurden nicht betrachtet), und liegt daher mit seinen Eigenschaften für beide Algorithmen in der Nähe des Koordinatenursprungs. Besonders deutlich werden die Vor- und Nachteile einer FPGA-Implementierung und dem DSP bei Betrachtung des FIR-Filters. Beide Architekturen können jeweils bezüglich einer Eigenschaft mit dem Optimum mithalten, offenbaren jedoch Schwächen bezüglich der anderen Eigenschaften. Der DSP-Kern bietet in der hier betrachteten Variante mit zwei parallelen MAC-Einheiten eine durchaus gute Flächeneffizienz (s.a. Abschnitt 7.2.4), aus der serialisierten Programmausführung ergibt sich jedoch naturgemäß eine niedrigere Datenrate, bzw. eine höhere Sampleperiode. Bezüglich des Energieverbrauchs liegt der DSP etwas schlechter als die grobgranularen Architekturen. Fast reziprok dazu ergeben sich die Verhältnisse des FPGA. Die durchaus guten erreichbaren Sampleraten gehen mit massiven Einbußen bei der Flächeneffizienz und dem Energieverbrauch einher. Dies gilt für alle in dieser Arbeit betrachteten Algorithmen. Anzumerken sei, dass bezüglich einer Medianfilterung keine verlässlichen Daten für den DSP verfügbar waren, da der in [Phi05] angegebene Energieverbrauch ein Durchschnittswert bei Verwendung beider MAC-Stufen ist. Diese sind bei einer nichtlinearen Filterung jedoch nicht einbezogen.

Bezüglich der grobgranularen Modellarchitekturen wird in Abbildung 7.18 vor allem die Abhängigkeit der physikalischen Eigenschaften von dem jeweils abgebildeten Algorithmus deutlich. Die in dieser Arbeit betrachteten Varianten wurden vor allem für die Ausführung Multiply-Add- und Multiply-Accumulate-basierter DSP-Algorithmen entworfen und optimiert. Diese anwendungsspezifische Ausrichtung führt zu guten Eigenschaften bezüglich aller drei Kostenmaße bei Abbildung eines FIR-Filters. Verglichen mit dem DSP und der FPGA-Implementierung liegen die grobgranularen Modelle in der Nähe des Semi-Custom-Entwurfs beim Koordinatenursprung. Bei Abbildung eines Algorithmus, für den die Architekturen nicht spezifisch entwickelt wurden, ergeben sich jedoch teilweise dras-



FIR-Filter



Median-Filter

Abbildung 7.18: Übersichtsdarstellungen zu den VLSI-Eigenschaften Fläche, Datenrate und Energie der betrachteten Architekturvarianten bei Abbildung eines FIR- und eines Median-Filters

tische Verschlechterungen, zu sehen im unteren Diagramm von Abbildung 7.18. Der Flächenbedarf liegt nunmehr im Bereich der FPGA-Implementierung, während die Datenrate sogar schlechter ist. Lediglich bezüglich des Energieverbrauchs bleiben die grobgranularen Modelle für diesen Algorithmus besser als das FPGA, haben aber gegenüber der Semi-Custom-Implementierung verglichen mit dem linearen Algorithmus deutlich an Effizienz verloren.

Flexibilität der untersuchten Implementierungsformen

Aus der massiven Schwankung der Eigenschaften der grobgranularen Modellarchitekturen für unterschiedliche Algorithmen lässt sich ein wesentliches Charakteristikum dieser Implementierungsform ableiten, welche neben den VLSI-Eigenschaften auch die Flexibilität betrifft. In Abbildung 7.18 ist erkennbar, dass sich die physikalischen Eigenschaften des FPGA verglichen mit der Semi-Custom-Implementierung für beide Algorithmen im Verhältnis im gleichen Bereich des Eigenschaftsraumes befinden. Daraus lässt sich ableiten, dass das FPGA Algorithmen unterschiedlichster Natur in etwa gleichermaßen effizient ausführen kann. Eine derartige konstante Effizienz ist ein wesentliches Kennzeichen einer flexiblen Architektur (s.a. die Definition der Flexibilität in Abschnitt 7.1.6). Demgegenüber ist die Effizienz der grobgranularen Modelle, wie oben erwähnt und in den Darstellungen der Eigenschaftsräume erkennbar, in Abhängigkeit vom Algorithmus deutlichen Schwankungen unterworfen. Derartige Schwankungen sind wiederum ein typisches Kennzeichnen weniger flexibler Strukturen. Abbildung 7.18 ist daher neben einer Darstellung der physikalischen Eigenschaften auch ein Indikator für die unterschiedliche Flexibilität der fein- und grobgranularen Architekturen.

Aus den obigen Betrachtungen ergibt sich die Frage nach der Abhängigkeit der Eigenschaften verschiedener Implementierungsformen von der Flexibilität der Architektur. Wie bereits in Abschnitt 7.1.6 dargestellt, existiert für die Flexibilität kein Maß, das in Form einer absoluten Zahl angegeben werden kann. Wenn die VLSI-Eigenschaften mit der Flexibilität in einem Diagramm dargestellt werden sollen, sind daher die Architekturen zunächst subjektiv nach der Reihenfolge ihrer Flexibilitäten zu ordnen. Der Semi-Custom-Entwurf ist naturgemäß am wenigsten flexibel, da jeweils nur ein bestimmter Algorithmus implementiert wird. Demgegenüber besitzt der DSP aufgrund der Software-Programmierbarkeit unter den hier betrachteten Implementierungsformen subjektiv die größte Anpassbarkeit. Die grobgranularen Architekturen lassen sich zwischen dem Semi-Custom-Entwurf und dem DSP einordnen. Das FPGA ist ebenfalls flexibel in dem Sinne, dass sich, wie oben beschrieben, Anwendungen verschiedener Klassen mit etwa konstanter Effizienz auf das Logikzellenfeld abbilden lassen. Die Frage, ob ein DSP flexibler als ein FPGA ist, lässt sich daher nicht ohne Weiteres sinnvoll beantworten. Zwar wird allgemein, wie bereits in Abschnitt 7.1.6 dargestellt, ein DSP aufgrund der im Gegensatz zum FPGA mög-

lichen Software-Programmierbarkeit als flexibler eingestuft. Dennoch soll im Folgenden nicht mehr zwischen der Flexibilität eines DSP und eines FPGA unterschieden werden.

Die beiden Diagramme in Abbildung 7.19 zeigen die Abhängigkeit der VLSI-Eigenschaften der Implementierungsvarianten von der subjektiv abgeschätzten Flexibilität. Im linken Diagramm ist der DSP, und im rechten Diagramm das FPGA zusammen mit dem Semi-Custom-Entwurf und den grobgranularen Modellen aufgetragen. Die physikalischen Eigenschaften sind jeweils auf den größten Wert normiert. Die Werte beziehen sich auf die Abbildung eines FIR-Filters auf die Architekturen.

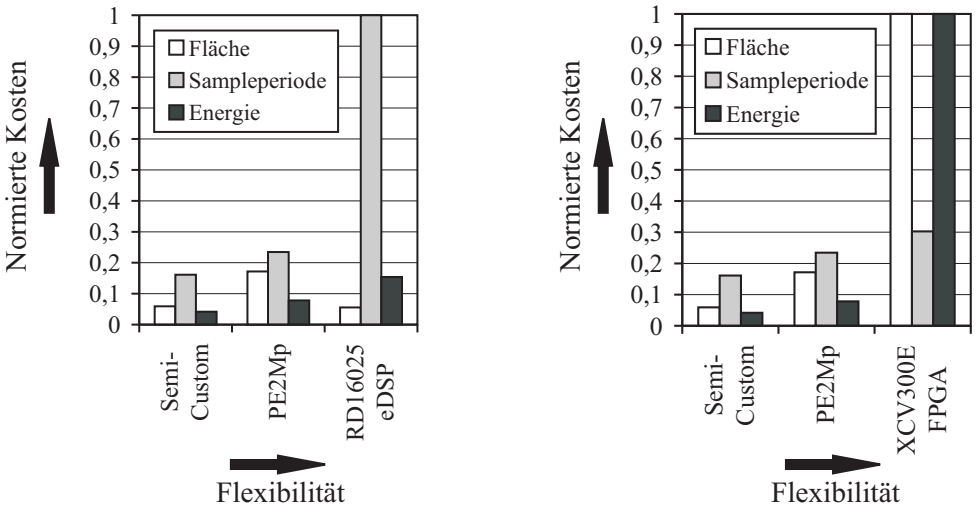


Abbildung 7.19: Abhängigkeit der physikalischen Eigenschaften in Abhängigkeit von der subjektiven Bewertung der Flexibilität der Architekturen

Die Diagramme geben Aufschluss über das Maß der mit einer höheren Flexibilität verbundenen Einbußen bei den anderen Eigenschaften einer Architektur. Abgesehen vom Flächenbedarf bei der DSP-Implementierung werden Fläche, Datendurchsatz und Energieverbrauch der Architekturen erwartungsgemäß mit zunehmender Flexibilität schlechter. Der niedrigere Flächenbedarf des DSP ergibt sich als Konsequenz aus dem um den Faktor acht geringereren Parallelitätsgrad. Dies wirkt sich im umgekehrter Gewichtung auch auf die Datenrate des DSP aus, was bei der Bewertung der Verarbeitungsgeschwindigkeit zu berücksichtigen ist. Deutlich wird in dem Diagramm nochmals der hohe Flächen- und Energiebedarf bei der FPGA-Implementierung.

Erweiterung rekonfigurierbarer Architekturen zur Steigerung der Flexibilität

Durch eine anwendungsspezifische Einschränkung der Flexibilität einer Architektur lässt sich, wie oben dargestellt, die Effizienz der Struktur bezüglich der jeweiligen Algorithmenklasse steigern. Die in dieser Arbeit untersuchten rekonfigurierbaren Beispielarchitekturen ziehen aus dieser Tatsache Nutzen, indem die zugrunde liegenden Basiszellen für lineare Algorithmen, bzw. mit dem Erweiterungsblock für lineare und nichtlineare Algorithmen optimiert wurden. Auch die Organisation der Kontrolllogik und der Speicherstruktur wurde, wie in den Kapiteln 4 und 5 erwähnt, anwendungsspezifisch angepasst.

Wenn, bei geringerer Effizienz, eine hohe Flexibilität auf dem Niveau von DSPs oder Universalprozessoren erzielt werden soll, sind Änderungen an der Datenpfadstruktur, der Kontrolllogik, sowie der Speicherorganisation erforderlich. Die Datenpfade müssen mindestens neben Multiplizierern und Addierern die Möglichkeit zur bitweisen Manipulation von Eingangsdaten durch eine allgemeine ALU vorsehen. Wenn eine prozessorartige Programmierbarkeit erzielt werden soll, muss zudem die Kontrolllogik eine maschinenprogrammierbare Einheit enthalten, mit der allgemeine Steuerungsoperationen wie beispielsweise bedingte und unbedingte Verzweigungen realisiert werden können. In letzter Konsequenz führt dies auf die Theorie massiv parallel arbeitender Rechenfelder und Multikernprozessoren, die hohe Parallelität und Flexibilität miteinander vereinen.

7.3 Schlussfolgerungen

In den obigen Abschnitten sind die Ergebnisse dargestellt, die sich aus einer modellbasierten Untersuchung einer grobgranularen rekonfigurierbaren Architektur ergeben. Die physikalischen Eigenschaften besitzen zur Bewertung von Architekturen zwar eine wichtige Rolle, eine vollständige Einordnung erfordert jedoch die Betrachtung weiterer Kriterien. Im Folgenden soll die Bedeutung der in dieser Arbeit vorgestellten Ergebnisse für praktische Entscheidungen für oder wider eine bestimmte Implementierungsform anhand verschiedener Kriterien diskutiert werden.

Physikalische Eigenschaften

Das hauptsächliche Einsatzfeld für grobgranulare rekonfigurierbare Architekturen ist in der flexiblen Erweiterung der Rechenressourcen eines heterogenen System-on-a-Chip zu sehen. Derartige Architekturen stellen eine Alternative dar zu anderen Implementierungsformen wie Universalprozessoren, DSPs, eingebetteten FPGAs oder dedizierten Entwürfen. Bereits beim Systementwurf für einen neu zu entwerfenden Chip ist die Entscheidung zu treffen, welche der Alternativen zur Realisierung bestimmter Anwendungen herangezogen wird. Da in vielen Fällen die Architektur bestimmte physikalische Randbedingungen (”Constraints”)

erfüllen muss, ist zunächst die Frage nach den VLSI-Eigenschaften von grundlegendem Interesse. Weitere Eigenschaften der Architektur wie Flexibilität, Rekonfigurationszeit oder die Anwendungsentwicklung spielen im darauf folgenden Entscheidungsprozess eine Rolle, wenn unter den Architekturen ausgewählt wird, welche die physikalischen Randbedingungen erfüllen.

In Abbildung 7.20 sind die physikalischen Eigenschaften der untersuchten Architekturen bezüglich einer 64-Punkt-FFT zusammenfassend dargestellt. Die Ergebnisse beziehen sich auf eine Implementierung des vollständigen Algorithmen-Kernes inklusive Kontrollogik und Speicher.

Auffällig ist vor allem die hohe Differenz zwischen der grobgranularen Architektur und dem FPGA bezüglich der Implementierungsfläche und des Energieverbrauches. Mit den in dieser Arbeit vorgeschlagenen grobgranularen Architekturmodellen lassen sich geeignete Algorithmen bei etwa gleicher Datenrate bis zu 90% flächen- und energieeffizienter implementieren als mit einem FPGA. Dies gilt unter der Annahme, dass der Algorithmus auf dem FPGA mit 8 Bit implementiert wird, während die Wortbreite der grobgranularen Architektur 16 Bit beträgt. Bei gleicher Wortbreite ist die Effizienz der grobgranularen Struktur bezüglich Fläche und Energie nochmals etwa 2 bis 3 mal höher. Der DSP belegt aufgrund der geringeren Parallelität weniger Fläche, ist demgegenüber jedoch entsprechend langsamer. Die Semi-Custom-Implementieung stellt für diese Arbeit das jeweils erreichbare Optimum dar. Mit handoptimierten Full-Custom-Entwürfen lässt sich eine nochmalige Verbesserung erreichen, diese wurden jedoch hier nicht betrachtet.

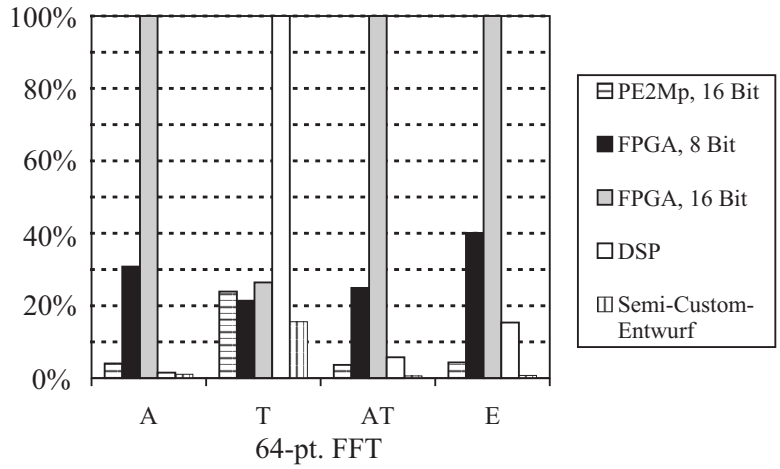


Abbildung 7.20: Zusammenfassende Übersicht zu den physikalischen Eigenschaften der untersuchten Architekturen am Beispiel einer 64-Punkt-FFT

Es ist ebenso der Frage nachgegangen worden, welche Effizienz die grobgranularen Modelle im Vergleich zu feinganularen Strukturen bieten, wenn lediglich die reine Datenpfad-

Implementierung betrachtet wird. Dies ist von Interesse, wenn die Architektur beispielsweise direkt in die Instruktionspipeline eines Host-Prozessors eingebunden werden soll, welcher die Kontrolllogik und die Regelung der Speicherzugriffe übernimmt. Bei dieser flexiblen Befehlssatzerweiterung wird auf den Speicherbereich des Host-Prozessors zugegriffen. Abbildung 7.21 zeigt die Ergebnisse bezüglich der Abbildung einer Radix-2-Butterfly-Operation. Demnach ist die Operation auf der grobgranularen Modellarchitektur im Vergleich zum FPGA bei gleicher Wortbreite bis zu 20 mal flächen- und energieeffizienter implementierbar.

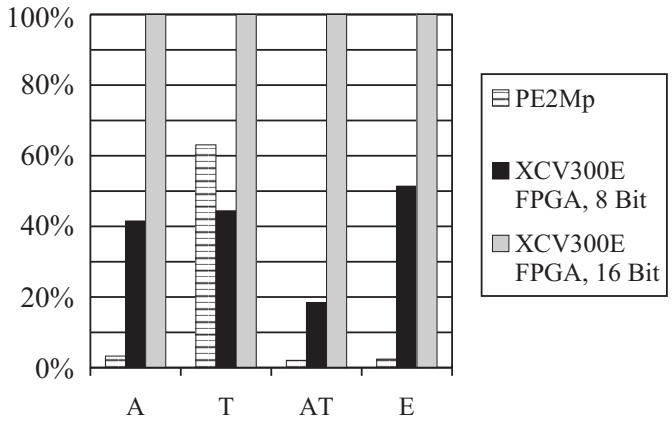


Abbildung 7.21: Zusammenfassende Übersicht zu den physikalischen Eigenschaften grob- und feingranularer rekonfigurierbarer Datenpfade am Beispiel einer Radix-2 Butterfly-Operation

In [vNBN06] konnte gezeigt werden, dass durch anwendungsspezifische Optimierung eines eFPGA dessen Flächen- und Energieeffizienz um bis zu 80% gegenüber einem Standard-FPGA (Altera Cyclone-I) verbessert werden können. Ein direkter Vergleich zwischen optimiertem eFPGA und anwendungsspezifisch ausgerichteter grobgranularer Architektur erscheint aufgrund des unterschiedlichen Vergleichsmaßstabes (in dieser Arbeit dient ein VirtexE-FPGA zum Vergleich mit feingranularen Architekturen) nicht exakt möglich. Dennoch findet sich aufgrund der Ergebnisse die intuitive Vermutung bestätigt, dass grobgranulare Architekturen auch gegenüber anwendungsspezifisch optimierten FPGAs Vorteile bezüglich der Flächen- und Energieeffizienz besitzen. Eine Quantifizierung der Verhältnisse kann als Thema für zukünftige Arbeiten auf diesem Gebiet dienen.

Die hier zusammengefasten Simulationsergebnisse vermitteln eine Übersicht, welche Eigenschaften von einer der in dieser Arbeit vorgeschlagenen grobgranularen Modellarchitekturen im Vergleich zu den bekannten Alternativen bei Einsatz in einem SoC zu erwarten sind. Es sei nochmals erwähnt, dass sich alle Ergebnisse auf die hier zu Grunde gelegten

Modelle beziehen. Für anders aufgebaute Architekturen werden sich demzufolge andere Verhältnisse ergeben. Die Ergebnisse sollen als Anhaltspunkt dienen für die Eigenschaften grobgranularer Architekturen im Vergleich zu alternativen Realisierungsformen.

Flexibilität

Die physikalischen Eigenschaften allein sind in den seltensten Fällen das einzige Kriterium für Entscheidungen in der Systementwurfsphase eines SoC. Häufig wird gefordert, dass Teile des Systems sich nachträglich an veränderte Anwendungen oder Randbedingungen anpassen lassen.

Es ist bekannt, dass die Vorteile grobgranularer Architekturen gegenüber FPGAs mit einer geringeren Flexibilität erkauft werden. Die in dieser Arbeit präsentierten Ergebnisse beziehen sich auf Architekturen, die für die Klasse der linearen Algorithmen optimiert sind. Andere Algorithmen lassen sich entweder mit deutlich schlechterer Effizienz oder gar nicht implementieren. Beim Systementwurf eines SoC ist abzuwägen, wieviel Flexibilität erforderlich ist. Die Ergebnisse dieser Arbeit zeigen, dass der Gewinn bei Beschränkung auf lineare Algorithmen bezüglich Fläche und Energie eine volle Größenordnung umfassen kann. Somit kann die vorgeschlagene Modellarchitektur beispielsweise als flächen- und energieeffiziente Beschleunigungseinheit in einem Signalprozessor zur Bild- und Videosignalverarbeitung eingesetzt werden.

Ein mögliches Thema für weiterführende Forschungen ist die Erarbeitung von Effizienzmaßen für andere als lineare Algorithmen und Anwendungen.

Rekonfigurationszeit

Die erforderliche Zeit zur Umkonfiguration einer fein- oder grobgranularen Architektur ist insbesondere in Fällen von Interesse, in denen eine dynamische Anpassung zur Laufzeit erfolgen soll. Wird beispielsweise eine rekonfigurierbare Architektur als Beschleunigungseinheit zur Videosignalverarbeitung eingesetzt, ist der Fall denkbar, dass neben einer linearen Filterung auch eine DCT und eine Farbraumkonversion mit der Struktur durchgeführt werden soll. Da diese Algorithmen auf jedes einzelne Bild angewendet werden sollen, muss die Architektur in schnellen Abständen umkonfiguriert werden. Die dazu erforderliche Zeit kann unter Umständen kritisch für die Performance des Systems werden.

Die zur Rekonfiguration erforderliche Anzahl von Taktzyklen ist im Wesentlichen abhängig von der Menge der Konfigurationsdaten. Hier sind grobgranulare Architekturen eindeutig im Vorteil gegenüber feingranularen, da die Zahl der zu konfigurierenden Elemente naturgemäß erheblich geringer ist. So sind beispielsweise für das 4×4-PE-Feld mit PE1M-Basiszelle inklusive Routing etwa 1.5 Kilobyte Konfigurationsdaten erforderlich. Demgegenüber kann die generierte Bitdatei zur Konfiguration kleiner FPGAs bereits mehrere 100 Kilobyte groß sein.

Entwurfssoftware für rekonfigurierbare Architekturen

In dieser Arbeit wird ein neues Verfahren zur modellbasierten Analyse grobgranularer rekonfigurierbarer Architekturen vorgeschlagen. Die Ergebnisse bezüglich der physikalischen Eigenschaften zeigen durchaus signifikante Vorteile gegenüber feingranularen Architekturen. Falls die Flexibilität als ausreichend angesehen wird, stellen grobgranulare Architekturen bei alleiniger Betrachtung der Hardware eine flächen- und energieeffiziente Entwurfsalternative dar.

Wenn ein Entwicklungsteam zur Systementwurfsphase unter mehreren alternativen Hardwarestrukturen wählen kann, spielt jedoch nicht nur die Hardware selbst, sondern auch die dazugehörige Software eine wichtige Rolle. Bei der Beurteilung der kommerziellen Erfolgschancen einer Architektur ist daher im Sinne des "Hardware-Software-Codesign" auch die Entwicklungsumgebung für die Software einzubeziehen.

DSPs und FPGAs besitzen gegenüber den heute kommerziell erhältlichen grobgranularen Architekturen den Vorteil, mit mehr oder weniger konventionellen Softwaretools programmierbar, bzw. konfigurierbar zu sein. Während DSPs üblicherweise mit C/C++ oder Assembler programmiert werden, dienen bei FPGAs in den meisten Fällen VHDL oder Verilog zur Konfiguration. Mit diesen Standard-Sprachen lassen sich alle kommerziell relevanten DSPs, bzw. FPGAs programmieren. Eine Einarbeitung erfordert lediglich die integrierte Entwicklungsumgebung unterschiedlicher Hersteller.

Für grobgranulare Architekturen hat sich derzeit noch kein einheitlicher Standard für eine Entwicklungssprache herausgestellt. Alle Hersteller bieten einen eigenen Software-Flow an. Das XPP-Array von PACT wird beispielsweise mit der proprietären "Native Mapping Language" (NML) programmiert, und mit einer eigenen grafischen Benutzeroberfläche konfiguriert. Dagegen verwendet picoChip einen komplexen Tool-Flow unter Zuhilfenahme von VHDL, ANSI-C und Assembler (siehe Abschnitt 2.4.2).

Die Frage, ob sich in Zukunft ein einheitlicher Entwicklungsstandard für grobgranulare Architekturen bilden wird, lässt sich derzeit nicht beantworten. Für den praktischen Nutzen, und damit für den kommerziellen Erfolg grobgranularer Architekturen spielt ein solcher vereinheitlichter Softwareentwurf jedoch eine herausragende Rolle.

Zusammenfassung der Schlussfolgerungen

Die hier präsentierten Ergebnisse zeigen anhand einer exemplarischen Implementierung eines Architekturmodells für lineare Algorithmen, dass grobgranulare rekonfigurierbare Architekturen deutliche Vorteile vor allem bezüglich der Flächen- und Energieeffizienz gegenüber feingranularen FPGAs aufweisen. Für den Fall, dass die mit der anwendungsspezifischen Ausrichtung einhergehenden Einbußen an Flexibilität als akzeptabel eingestuft werden, können bezüglich Fläche und Energie um bis zu eine Größenordnung effizientere Implementierungen realisiert werden. Die Tatsache, dass aufgrund der groben Granularität

im Vergleich zu FPGAs erheblich weniger Konfigurationsdaten erzeugt werden müssen, erleichtert weiterhin bekanntermaßen die Entwicklung dynamischer Architekturen. Als problematisch ist demgegenüber derzeit das Fehlen eines einheitlichen Software-Entwurfs für grobgranulare Architekturen anzusehen.

8 Zusammenfassung

Bei der Implementierung von Algorithmen der digitalen Signalverarbeitung auf heterogenen Systems-on-a-Chip kann zwischen mehreren grundsätzlich verschiedenen Architekturen gewählt werden. Universalprozessoren und digitale Signalprozessoren bieten eine hohe Flexibilität, jedoch nur eine geringe Geschwindigkeit. Hohe Performance verbunden mit einem niedrigen Energieverbrauch bieten demgegenüber dedizierte Full- oder Semi-Custom-Implementierungen. Diese haben jedoch den Nachteil, nach der Fertigung nicht mehr an veränderte Randbedingungen anpassbar zu sein. FPGAs wiederum stellen eine hohe Geschwindigkeit bei gleichzeitig guter Flexibilität zur Verfügung. Nachteilig bei diesen Bausteinen ist jedoch der hohe Flächenaufwand und die schlechte Energieeffizienz, die aus der feingranularen Rekonfigurierbarkeit auf Bit-Ebene resultieren. Einen Ausweg bieten grobgranulare rekonfigurierbare Architekturen, die unter Verzicht auf einen Teil der Flexibilität eine wesentlich verbesserte Flächen- und Energieeffizienz bieten, bei gleichzeitig hoher Verarbeitungsgeschwindigkeit. Erreicht wird dies durch die Verwendung von Datenpfaden mit höherer Wortbreite, welche dedizierte Elemente wie Multiplizierer, Addierer oder Komparatoren enthalten können. Hier stellt sich die Frage, wie effizient eine solche anwendungsspezifisch optimierte Architektur im Vergleich zu anderen Implementierungsformen arbeitet.

Grobgranulare rekonfigurierbare Architekturen sind durch eine Vielzahl von Parametern charakterisierbar, die gemeinsam einen multidimensionalen Entwurfsraum aufspannen. Aus Aufwandsgründen ist es unmöglich, jede einzelne Variante zu implementieren und zu simulieren. Eine Quantifizierung der Eigenschaften grobgranularer Strukturen erfordert daher ein flexibles Analyseverfahren, das mit geringem Aufwand eine Untersuchung verschiedener Architekturvarianten gestattet.

In dieser Arbeit wird eine neue Methodik vorgestellt, mit dessen Hilfe grobgranulare Architekturen in Bezug auf Flächenbedarf, Datenrate und Energieverbrauch charakterisiert werden können. Es ist ein Matlab-basiertes Software-Werkzeug erstellt worden, welches die einfache Analyse einer durch eine Vielzahl von Parametern bestimmbaren Architektur gestattet. Die VLSI-Eigenschaften werden aufgrund von physikalischen Modellen berechnet, die in das Werkzeug integriert sind.

Mit dem entwickelten Analyseverfahren sind Untersuchungen grobgranularer Modellarchitekturen durchgeführt worden. In Anlehnung an bereits existierende grobgranulare Strukturen ist für das Modell eine zweidimensionale Feldstruktur gewählt worden, in dem jeweils die unmittelbar benachbart liegenden Prozessorzellen direkt miteinander kommunizieren können. Weiter entfernt liegende Zellen können über globale Busleitungen miteinan-

der verbunden werden. Der Speicher ist hierarchisch in lokale und globale Blöcke aufgeteilt. Die 16-Bit-Prozessorzellen sind für die effiziente Ausführung linearer Algorithmen wie FIR- und IIR-Filterungen, lineare Transformationen und Matrix-Operationen optimiert worden. Darüber hinaus besteht die Möglichkeit, nichtlineare Operationen durch einen integrierten Komparatorbaustein auszuführen. Im Rahmen der Arbeit wurden unterschiedlich komplexe Prozessorzellen mit einem, zwei, vier, bzw. acht Multiplizierern entwickelt.

Auf alle Architekturvarianten sind mit Hilfe des Software-Toolflows unter anderem lineare Algorithmen, sowie Medianfilter und Add-Compare-Select-Operationen für Viterbi-Decoder abgebildet worden. Die Ergebnisse der Simulationen gestatten einen Vergleich der grobgranularen Architekturvarianten sowohl untereinander, als auch mit Implementierungen auf FPGAs und DSPs, sowie mit Standardzellentwürfen. Auf dem FPGA wurden die Algorithmen sowohl mit 8, als auch mit 16 Bit Wortbreite implementiert. Dies gestattet einen Vergleich der grobgranularen Modelle mit FPGAs, wenn beispielsweise Algorithmen der Bildsignalverarbeitung betrachtet werden, die häufig mit 8 Bit breiten Datenbussen arbeiten.

Die Ergebnisse zeigen, dass die in dieser Arbeit modellhaft untersuchten grobgranularen Architekturen bei Abbildung linearer Algorithmen mit 16 Bit Wortbreite bis zu 7 mal kleiner sind, und 10 mal weniger Energie verbrauchen als feingranulare FPGA-Strukturen mit 8 Bit Wortbreite. Wird der Algorithmus auf dem FPGA mit 16 Bit, also gleicher Wortbreite wie bei den grobgranularen Modellen implementiert, verbessert sich die Effizienz bezüglich der Fläche und der Energie nochmals etwa um den Faktor 2 zugunsten der grobgranularen Architekturen. Die Datenraten der fein- und grobgranularen Architekturen liegen in der gleichen Größenodnung, mit einem leichten Vorteil für die grobgranularen Modelle bei 16-Bit-Implementierung.

Ein anderes Bild ergibt sich, wenn Algorithmen implementiert werden, für welche die grobgranularen Architekturen nicht nativ entwickelt und optimiert wurden. Beispielsweise liegt bei Abbildung eines Medianfilters der Flächenbedarf fein- und grobgranularer Architekturen in etwa gleichauf, während die absolute Datenrate der FPGA-Implementierung etwa um den Faktor 2 besser ist. Dies liegt einerseits daran, dass bei der Medianfilterung lediglich der Komparator-Block innerhalb der grobgranularen Zellen aktiv ist, während die anderen Blöcke innerhalb des Datenpfades, beispielsweise Multiplizierer und Addierer, als inaktive Ballast-Komponenten dennoch in den Flächenbedarf eingehen. Die FPGA-Implementierung dagegen erlaubt durch die feingranulare Rekonfigurierbarkeit in diesem Fall wesentlich bessere Möglichkeiten zur Optimierung. Es konnte demnach gezeigt und quantifiziert werden, welche Eigenschaften in Bezug auf Fläche, Datenrate und Energieverbrauch von Architekturen mit jeweils unterschiedlicher Flexibilität zu erwarten sind.

Die Standardzellimplementierung liefert wie erwartet für alle Algorithmen die besten Ergebnisse bezüglich der VLSI-Eigenschaften, bietet demgegenüber jedoch keinerlei Flexibilität. Die Datenrate des DSP ist aufgrund des geringeren Parallelitätsgrades je nach Anwendung um den Faktor 2.5 bis 9 niedriger als die der grobgranularen Modelle, während

das AT-Produkt beider Varianten auf vergleichbarem Niveau liegt. In Fällen, in denen eine sehr hohe Datenrate gefordert wird, ist die grobgranulare Struktur dennoch im Vorteil, da bei mehreren parallel betriebenen DSP-Kernen zusätzlich der Aufwand für Kommunikationsleitungen und Busse berücksichtigt werden muss.

Es bleibt die Erkenntnis, dass grobgranulare rekonfigurierbare Architekturen feingranularen Strukturen vor allem in Bezug auf die Flächeneffizienz und den Energieverbrauch teilweise deutlich überlegen sind, so lange sie in ihrer vorgesehen Anwendungsklasse betrieben werden. Das Ausmaß der Überlegenheit für verschiedene Anwendungen wird in dieser Arbeit quantifiziert. Es soll erwähnt werden, dass sich die Ergebnisse auf das einfache Modell einer rekonfigurierbaren Architektur beziehen, welches nicht den Anspruch erhebt, eine optimale Lösung darzustellen. Die Ergebnisse sind daher als untere Schranke der Möglichkeiten grobgranularer Architekturen anzusehen, die weiterhin ein großes Optimierungspotential besitzen.

Es soll hervorgehoben werden, dass sich die Ergebnisse dieser Arbeit ausschließlich auf die Hardware-Architekturen bezieht, während die Effizienz möglicher Software-Entwicklungstools nicht betrachtet wurde. Die zu einer Architektur gehörende Software ist jedoch ein wesentlicher Faktor in Bezug auf die kommerziellen Chancen eines IP-Cores. Bei Betrachtung aller in dieser Ausarbeitung zitierten Beispiele für grobgranulare Architekturen, einschließlich der kommerziellen Cores, fällt auf, dass jeweils eine eigene proprietäre Entwicklungssoftware bereit gestellt wird, in die ein Entwurfsteam nach einer Lizensierung des Cores eingearbeitet werden muss. Das Fehlen eines einheitlichen Software-Designflows ist derzeit als klarer Nachteil grobgranularer Architekturen gegenüber FPGAs anzusehen, die beispielsweise mit einer Hardwarebeschreibungssprache wie VHDL oder Verilog konfiguriert werden können. Realistische Marktchancen sind grobgranularen Architekturen daher erst nach einer weitgehenden Vereinheitlichung des Software-Entwurfsablaufes einzuräumen.

Die Untersuchung grobgranularer Prozessorarchitekturen, die für andere als lineare Algorithmen optimiert wurden, sind ein möglicher Ansatzpunkt für zukünftige Forschungsarbeiten. Darüber hinaus ist eine Erweiterung des in dieser Arbeit vorgeschlagenen Software-Toolflows zur Analyse hybrider Strukturen denkbar, bei denen versucht wird, die Vorteile optimierter fein- und grobgranularer Komponenten in einer Architektur zu vereinen. Weitere Forschungsfelder bieten beispielsweise dynamisch rekonfigurierbare Strukturen. Neben den hier besprochenen Kriterien Fläche, Datenrate, Energie und Flexibilität ist bei zukünftigen Untersuchungen auch eine Bewertung des zugehörigen Software-Entwurfs von Interesse.

Literaturverzeichnis

[3DS03] 3DSP Corp.: *SP-5 Flex Data Brief.* www.3dsp.com, 2003.

[ADK00] Alpert, C.J., A. Devgan und C. Kashyap: *A two Moment RC Delay Metric for Performance Optimization.* In: *Proceedings of the 2000 International Symposium on Physical Design, San Diego, CA*, 69 – 74, 2000.

[ALKD04] Alpert, C.J., F. Liu, C.V. Kashyap und A. Devgan: *Closed-Form Delay and Slew Metrics Made Easy.* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 23, Nr. 12, S. 1661 – 1669, December 2004.

[Alt05] Altera Corp.: *Stratix II Device Handbook.* www.altera.com, 2005.

[ARM03] ARM Ltd.: *ARM920T Product Overview.* www.arm.com, 2003.

[ASB03] Agarwal, K., D. Sylvester und D. Blaauw: *Simple Metrics for Slew Rate of RC Circuits based on two Circuit Moments.* In: *Proceedings of the 40th Design Automation Conference (DAC03), Anaheim, CA*, 950 – 953, 2003.

[ASNC04] Abou-Seido, A.I., B. Nowak und C. Chu: *Fitted Elmore Delay: A simple and accurate Interconnect Delay Model.* IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 12, Nr. 7, S. 691 – 696, July 2004.

[BBB+06] Blume, H., D. Becker, M. Botteck, J. Brakensiek und T. G. Noll: *Hybrid Functional and Instruction Level Power Modeling for Embedded Processors.* In: *Proceedings of the SAMOS VI Workshop, Samos, Greece*, S. 216 – 226, 17.–20. Juli 2006.

[BGDN03] Bansal, N., S. Gupta, N Dutt und A. Nicolau: *Analysis of the Performance of Coarse-Grained Reconfigurable Architectures with Different Processing Element Configurations.* In: *Workshop on Architecture Specific Processors (WASP)*, December 2003.

[BGK98] Bisdounis, L., D. Gouvetas und O. Koufopavlou: *A comparative study of CMOS circuit design styles for low-power high-speed VLSI circuits.* International Journal of Electronics, Vol. 84, Nr. 6, S. 641 – 653, Juni 1998.

[BGP03] Bossuet, L., G. Gogniat und J. Philippe: *Fast Design Space Exploration Method for Reconfigurable Architectures*. Engineering of Reconfigurable Systems and Algorithms, 2003.

[BHFN02] Blume, H., H. Huebert, H. Feldkämper und T.G. Noll: *Model Based Exploration of the Design Space for Heterogeneous Systems on Chip*. In: *Proceedings of the IEEE ASAP Conference, San Jose, USA*, S. 29–40, 17.–19. Juli 2002.

[BLFC03] Borgatti, M., F. Lertora, B. Foret und L. Cali: *A reconfigurable system featuring dynamically extensible embedded microprocessor, FPGA and customizable I/O*. IEEE Journal of Solid State Circuits, S. 521–529, März 2003.

[BR99] Betz, V. und J. Rose: *FPGA routing architecture: segmentation and buffering to optimize speed and density*. In: *FPGA '99: Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field Programmable Gate Arrays*, S. 59 – 68, New York, NY, USA, 1999. ACM Press.

[BRM99] Betz, V., J. Rose und S. Marquardt: *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, 1999. ISBN 0-7923-8460-1.

[BS95] Bronstein, I.N. und K.A. Semendjajew: *Taschenbuch der Mathematik*. Verlag Harri Deutsch, 1995.

[CH01] Compton, K. und S. Hauck: *Totem: Custom Reconfigurable Array generation*. In: *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, 2001.

[CH02] Compton, K. und S. Hauck: *Reconfigurable Computing: A Survey of Systems and Software*. ACM Computing Surveys, Vol. 34, Nr. 2, S. 171 – 210, June 2002.

[Cha03a] Chartered Semiconductor Manufacturing: *Chartered 0.18 micron, 1.8V High Density Synchronous RAM Datasheet*, 2003.

[Cha03b] Chartered Semiconductor Manufacturing: *Chartered 0.18 micron, 1.8V Synchronous Dual Port RAM Compiler*, 2003.

[Cha04] Chartered Semiconductor Manufacturing: *0.18μm Process Technology Datasheet*. www.charteredsemi.com/technology/pdf/018_datasheet_04072004.pdf, 2004.

[CJT03] Critzer, T. J., A. Jantsch und H. Tenhunen: *Networks on Chip*. Kluwer Academic Publishers, 2003.

[CP95] Chang, J.-M. und M. Pedram: *Register Allocation and Binding for Low-Power*. In: *Proceedings of the 32nd ACM/IEEE Conference on Design Automation*, 29–35, 1995.

[CPH+97] Cong, J., Z. Pan, L. He, C.-K. Koh und K.-Y. Khoo: *Interconnect Design for Deep Submicron ICs*. In: *Proceedings of the 1997 IEEE/ACM International Conference on Computer-Aided Design*, S. 478 – 485, 1997.

[CR92] Chen, D. C. und J. M. Rabaey: *A Reconfigurable Multiprocessor IC for Rapid Prototyping of Algorithmic Specific High-Speed DSP Datapaths*. IEEE Journal of Solid-State Circuits, Vol. 27, No. 12, S. 1895–1904, December 1992.

[CRS+99] Chow, P., J. Rose, S. O. Seo, K. Chung, G. Páez-Monzón und I. Rahardja: *The Design of an SRAM-based Field-Programmable Gate Array - Part I: Architecture*. IEEE Transactions on Very Large Scale Integration Systems, Vol. 7, Nr. 2, S. 191 – 197, 1999.

[CSF77] Chen, W. H., C. H. Smith und S. Fralick: *A Fast Computational Algorithm for the Discrete Cosine Transform*. IEEE Transactions on Communications, Vol. COM-25, S. 1004 – 1009, Sept. 1977.

[CSR+99] Chow, P., S. O. Seo, J. Rose, K. Chung, G. Páez-Monzón und I. Rahardja: *The Design of an SRAM-based Field-Programmable Gate Array - Part II: Circuit Design and Layout*. IEEE Transactions on Very Large Scale Integration Systems, Vol 7, Nr. 3, S. 321 – 330, 1999.

[DeH98] DeHon, A.: *Comparing Computing Machines*. Configurable Computing Technology and Applications, Proceedings of SPIE, Vol. 3526, 1998.

[DPT03] Duller, A., G. Panesar und D. Towner: *Parallel Processing - The picoChip Way!* Communicating Process Architectures, S. 299 – 312, 2003.

[DW99] DeHon, A. und J. Wawrzynek: *Reconfigurable Computing: What, Why, and Implications for Design Automation*. In: *DAC '99: Proceedings of the 36th ACM/IEEE conference on Design automation*, 610–615, New York, NY, USA, 1999. ACM Press.

[ECF96] Ebeling, C., D. C. Cronquist und P. Franklin: *RaPiD - Reconfigurable Pipelined Datapath*. In: *Proceedings of the 6th International Workshop on Field-Programmable Logic and Applications, Darmstadt, Germany*, 1996.

[EET05] EETimes: *NEC Electronics to deploy its embedded DRAM in Microsofts next XBox.* Electronic Engineering Times, http://www.eetimes.com/news/design/technology/showArticle.jhtml?articleID=161600192, April 2005.

[Eli01] Elixent Ltd.: *Applications on the D-Fabrix Array.* www.elixent.com, 2001.

[Elm48] Elmore, W. C.: *The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers.* Journal of Applied Physics, S. 55 – 63, 1948.

[FLW01] Fan, H., J. Liu und Y.-L. Wu: *Combinatorial Routing Analysis and Design of Universal Switch Blocks.* In: *Proceedings of the 2001 Asia and South Pacific Conference on Design Automation*, S. 641 – 644, New York, NY, USA, 2001. ACM Press.

[GKT+95] Gupta, R., B. Krauter, B. Tutuianu, J. Willis und L.T. Pileggi: *The Elmore Delay as a Bound for RC Trees with Generalized Input Signals.* In: *Proceedings of 32nd ACM/IEEE Design Automation Conference*, S. 364 – 369, 1995.

[GSL+99] Guangming, L., H. Singh, M. Lee, N. Bagherzadeh und E. M. C. Filho: *The MorphoSys Parallel Reconfigurable System.* In: *Proceedings of the European Conference on Parallel Computing, Toulouse, France*, September 1999.

[GSM+99] Goldstein, S. C., H. Schmit, M. Moe, M. Budiu, S. Cadambi, R. R. Taylor und R. Laufer: *PipeRench: A Coprocessor for Streaming multimedia Acceleration.* In: *Proceedings of the 26th International Symposium on Computer Architecture, Atlanta, USA*, S. 28–39, 1999.

[Hei05] Heise: *Hintergrund: Zur Leistung der XBox 360.* Heise Online News, www.heise.de/newsticker/meldung/59548, 2005.

[Hey04] Heysters, P. M.: *Coarse-Grained Reconfigurable Processors.* Dissertation, University of Twente, The Netherlands, 2004.

[HHHN00] Hartenstein, R., M. Herz, T. Hoffmann und U. Nageldinger: *KressArray Xplorer: A New CAD Environment to Optimize Reconfigurable Datapath Array Architectures.* In: *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC00), Yokohama, Japan*, Januar 2000.

[HN99] Henning, C. und T.G. Noll: *Scalable Architectures and VLSI Implementations of High Performance Image Processing Algorithms*. In: *Proceedings of the International Conference on Imaging Science, Systems and Technology, Las Vegas, USA*, 28. Juni–1. Juli 1999.

[HSC83] Hitchcock, R., G. Smith und D. Cheng: *Timing Analysis of Computer Hardware*. IBM Journal of Research and Development, S. 100 – 105, Jan. 1983.

[HSM03] Heysters, P. M., G. J. M. Smit und E. Molenkamp: *A Flexible and Energy-Efficient Coarse-Grained Reconfigurable Architecture for Mobile Systems*. The Journal of Supercomputing, Vol. 26, No. 3, S. 283 – 308, 2003.

[Inf02] Infineon Technologies AG: *TriCore TC1MP-S Product Brief*. www.infineon.com, 2002.

[KBV93] Khellah, M., S. Brown und Z. Vranesic: *Modelling Routing Delays in SRAM-based FPGAs*. In: *Proceedings of the Canadian Conference on VLSI*, S. 6B.13 – 6B.18, 1993.

[KM97] Kahng, A.B. und S. Muddu: *An analytical Delay Model for RLC Interconnects*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 16, Nr. 12, S. 1507 – 1514, December 1997.

[KR98] Kusse, E. und J. M. Rabaey: *Low-Energy embedded FPGA Structures*. In: *Proceedings of the 1998 International Symposium on Low Power Electronics and Design*, S. 155 – 160, 1998.

[KSW95] Kories, H. und H. Schmidt-Walter: *Taschenbuch der Elekrotechnik*. Verlag Harri Deutsch, 1995. ISBN: 3-8171-1412-5.

[Lan94] Landman, P. E.: *Low-Power Architectural Design Methodologies*. Dissertation, University of California at Berkeley, 1994.

[LFS+02] Lange, H., O. Franzen, H. Schröder, M. Bücker und B. Oelkrug: *Reconfigurable Multiply-Accumulate based Processing Element*. In: *Proc. of the IEEE Workshop on heterogeneous Systems on a Chip, Hamburg, Germany*, 2002.

[LL02] Lemieux, G. und D. Lewis: *Circuit Design of Routing Switches*. In: *FPGA '02: Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field Programmable Gate Arrays*, S. 19 – 28, New York, NY, USA, 2002. ACM Press.

[Lod06] Lodi, A. et al.: *XiSystem: A XiRisc-Based SoC With Reconfigurable IO Module*. IEEE Journal of Solid-State Circuits, Vol. 41, No. 1, S. 85 – 96, January 2006.

[LOS04] Lange, H., B. Oelkrug und H. Schröder: *Viterbi Algorithm Processing on an ASIP Hardware Accelerator*. In: *Proceedings of the 3rd International Workshop on Software Radios (WSR04), Karlsruhe, Germany*, 2004.

[LS05] Lange, H. und H. Schröder: *Evaluation Strategies for Coarse Grained Reconfigurable Architectures*. In: *Proceedings of the 15th International Conference on Field Programmable Logic and Applications (FPL05), Tampere, Finland*, 2005.

[LTC03a] Lodi, A., M. Toma und F. Campi: *A Pipelined Configurable Gate Aray for Embedded Processors*. In: *Proceedings of the FPGA 03, Monterey, California, USA*, 2003.

[LTC+03b] Lodi, A., M. Toma, F. Campi, A. Capelli, R. Canegallo und R. Guerrieri: *A VLIW Processor With Reconfigurable Instruction Set for Embedded Applications*. IEEE Journal of Solid-State Circuits, Vol. 38, No. 11, S. 1876 – 1886, November 2003.

[MD96] Mirsky., E. und A. DeHon: *MATRIX: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources*. In: *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, April 1996.

[MIP05] MIPS Technologies: *MIPS32 24Kc Processor Core Datasheet*. www.mips.com, 2005.

[Moe02a] Moerman, K.: *Extending the Performance of a DSP: JPEG2000 using C-to-Silicon Technology*. www.adelantetech.com, 2002.

[Moe02b] Moerman, K.: *JPEG2000 on Saturn DSP with C-to-Silicon generated Coprocessor*. www.adelantetech.com, 2002.

[Mor04] Morpho Technologies: *MS1-16 v002: Morpho Reconfigurable DSP (rDSP) IP Core*. www.morphotech.com, 2004.

[MSK+99] Marshall, A., T. Stansfield, I. Kostarnov, J. Vuillemin und B. Hutchings: *A Reconfigurable Arithmetic Array for Multimedia Applications*. In: *Proc. of the 7th International Symposium an Field Programmable Gate Arrays, Monterey, USA*, 1999.

[Nag01] Nageldinger, U.: *Coarse-Grained Reconfigurable Architectures Design Space Exploration.* Dissertation, Universität Kaiserslautern, 2001.

[NBFN03] Neumann, B., H. Blume, H. Feldkämper und T.G. Noll: *Embedded FPGA-Architekturen für Multimedia-Applikationen.* In: *Tagungsband der ITG-Fachtagung Elekronische Medien, Dortmund, Germany*, September/Oktober 2003.

[ND98] Nassif, N. und M.P. Desai: *Robust Elmore Delay Models suitable for Full Chip Timing Verification of a 600MHz CMOS Microprocessor.* In: *Proceedings of the annual ACM/IEEE Design Automation Conference*, S. 230 – 235, 1998.

[NPC02] Nikolaidis, S., H. Pournara und A. Chatzigeorgiou: *Output Waveform Evaluation of Basic Pass Transistor Structures.* In: *Proceedings of the 12th International Workshop on Power and Timing Modeling, Optimization, and Simulation (PATMOS)*, S. 229 – 238, 2002.

[OBU+02] Oelkrug, B., M. Bücker, D. Uffmann, A. Dröge, J. Brakensiek und M. Darianian: *Programmable Hardware Accelerator for Universal Telecommunication Applications.* In: *Proc. of the 2nd International Workshop on Software Radios, Karlsruhe, Germany*, 2002.

[OC96] Okamotoa, T. und J. Cong: *Buffered Steiner Tree Construction with Wire Sizing for Interconnect Layout Optimization.* In: *Proceedings of the IEEE International Conference on Computer Aided Design (ICCAD)*, S. 44–49, 1996.

[OSK+05] Oppold, T., T. Schweizer, T. Kuhn, W. Rosenstiel, U. Kanus und W. StraÃŸer: *Evaluation of Ray-Casting on Processor-Like Reconfigurable Architectures.* In: *Proceedings of the 15 International Conference on Field Programmable Logic and Applications (FPL05)*, 2005.

[OSKR04] Oppold, T., T. Schweizer, T. Kuhn und W. Rosenstiel: *Cost Functions for the Design of Dynamically Reconfigurable Processor Architectures.* In: *Proceedings of the 2004 Workshop on Synthesis and System Integration of Mixed Information Technologies (SASIMI), Kanazawa, Japan*, 2004.

[Ott04] Otte, M.: *Matrixbasierte Signalverarbeitung auf rekonfigurierbaren CORDIC-Architekturen.* Dissertation, Universität Dortmund, 2004.

[PAC05] PACT Informationstechnologie GmbH: *XPP-IIb Core Overview.* www.pactcorp.com, September 2005.

[Phi05] PHILIPS SEMICONDUCTORS: *Philips 16-Bit DSP Core based on Adelante Technology RD16025*. www.philips.com, 2005.

[pic04] PICOCHIP: *PC102 Datasheet*. www.picochip.com, 2004.

[Pil98] PILEGGI, L.: *Timing Metrics for Physical Design of Deep Submicron Technologies*. In: *Proceedings of the 1998 International Symposium on Physical Design*, 28 – 33, 1998.

[PR90] PILLAGE, L.T. und R.A. ROHRER: *Asymptotic Waveform Evaluation for Timing Analysis*. IEEE Transactions on Computer-Aided Design, Vol. 9, Nr. 4, S. 352 – 366, April 1990.

[PWY05] POON, K. K. W., S. J. E. WILTON und A. YAN: *A detailed Power Model for Field Programmable Gate Arrays*. ACM Transactions on Design Automation of Electrical Systems, Vol. 10, No. 2, S. 279 – 302, 2005.

[Rab93] RABAEY, J. M.: *Digital Integrated Circuits, A Design Perspective*. Prentice Hall, 1993.

[Rao80] RAO, K. RAMAMOHAN: *Fast Transforms*. Kluwer Academic Publishers, 1980.

[RBWP02] ROY, S., M. BÜCKER, W. WILHELM und B. S. PANWAR: *Reconfigurable Hardware-Accelerator for a Universal Reed-Solomon Codec*. In: *Proceedings of the 1st IEEE Intl. Conference on Circuits and Systems for Communications, St. Petersburg*, Juni 2002.

[Rei04] REICHENBERG, D.: *Parametrisierbare Routing-Strukturen für rekonfigurierbare Prozessorfelder*. Studienarbeit S3–2004 am Lehrstuhl für Kommunikationstechnik der Universität Dortmund, 2004.

[RFLC90] ROSE, J., R. FRANCIS, D. LEWIS und P. CHOW: *Architecture of Field Programmable Gate Arrays: The Effect of Logic Block Functionality on Area Efficiency*. IEEE Journal of Solid State Circuits, Vol. 25, Nr. 5, Oktober 1990.

[RP83] RUBINSTEIN, J. und P. PENFIELD: *Signal Delay in RC Tree Networks*. IEEE Transactions on Computer-Aided Design, Vol. CAD-2, Nr. 3, S. 202 – 211, July 1983.

[SB00] SCHRÖDER, H. und H. BLUME: *Mehrdimensionale Signalverarbeitung – Band 2: Architekturen und Anwendungen für Bilder und Bildsequenzen*. B.G. Teubner, 2000.

[SC05] Schmitt, H. und V. Chandra: *Layout Techniques for FPGA Switch Blocks.* IEEE Transactions on Very Large Scale Integration Systems, Vol 13, Nr. 1, S. 96 – 105, 2005.

[Sch98] Schröder, H.: *Mehrdimensionale Signalverarbeitung – Band 1: Algorithmische Grundlagen für Bilder und Bildsequenzen.* B.G. Teubner, 1998.

[Sch05] Schuchert, T.: *Modellbasierte Analyse der Energieeffizienz rekonfigurierbarer Hardware.* Diplomarbeit D06-2005 am Lehrstuhl für Kommunikationstechnik der Universität Dortmund, 2005.

[Sil05] Silicon Hive: *Silcon Hive Introduction.* www.siliconhive.com, 2005.

[Syn03a] Synopsys Inc.: *PrimeTime User Guide: Advanced Timing Analysis, Version U-2003.03.* www.synopsys.com, 2003.

[Syn03b] Synopsys Inc.: *PrimeTime User Guide: Fundamentals.* www.synopsys.com, 2003.

[Tay02] Taylor, M. et al.: *The RAW Microprocessor: A Computational fabric for Software Circuits and General Purpose Programs.* IEEE Micro, S. 25 – 35, March/April 2002.

[Tay04] Taylor, M. B.: *The RAW Prototype Design Document.* Masachusetts Institue of Technology, Technical Report, September 2004.

[TB01] Tessier, R. und W. Burleson: *Reconfigurable Computing for Digital Signal Processing: A Survey.* Journal of VLSI Signal Processing, Vol. 28, S. 7–27, 2001.

[Ten04] Tensilica Inc.: *Xtensa LX Product Brief.* www.tensilica.com, 2004.

[vKN+06] von Sydow, T., M. Korb, B. Neumann, H. Blume und T. G. Noll: *Modelling and Quantitative Analysis of Coupling Mechanisms of Programmable Processor Cores and Arithmetic Oriented eFPGA Macros.* In: *Proceedings of the 2006 ReConFig Conference, San Luis Potosi, Mexico*, 20.– 22. September 2006.

[vNBN06] von Sydow, T., B. Neumann, H. Blume und T. G. Noll: *Quantitative Analysis of embedded FPGA-Architectures for Arithmetic.* In: *Proceedings of the 2006 IEEE international conference on Application specific Systems, Architectures and Processors (ASAP 2006), Steamboat Springs, Colorado*, 11.– 13. September 2006.

[Wai97] WAINGOLD, E. ET AL.: *Baring it all to Software: RAW Machines.* IEEE Computer, Vol. 30, No. 9, S. 86 – 93, Sept. 1997.

[Wan99] WANHAMMAR, L.: *DSP Integrated Circuits.* Academic Press, 1999.

[Wei04] WEIKERT, L.: *Connecting memory mapped, circuit switched Interface Components with packet switched Networks.* Diplomarbeit D05-2004 am Lehrstuhl für Kommunikationstechnik der Universität Dortmund, 2004.

[WH05] WESTE, N. H. E. und D. HARRIS: *CMOS VLSI Design, A Circuits and Systems Perspective.* Addison Wesley, 2005. ISBN: 0-321-1491-7.

[WKMV04] WILTON, S. J. E., N. KAFAHI, B. MEI und S. VERNALDE: *Interconnect Architectures for Modulo-Scheduled Coarse-Grained Reconfigurable Arrays.* In: *Proceedings of the International Conference on Field-Programmable Technology,* S. 33 – 40, 2004.

[Xil02] XILINX INC.: *Xilinx VirtexE Datasheet.* www.xilinx.com, 2002.

[Xil05] XILINX INC.: *Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Datasheet.* www.xilinx.com, 2005.

[ZPG+00] ZHANG, H., V. PRABHU, V. GEORGE, M. WAN, M. BENES, A. ABNOUS und J. M. RABAEY: *A 1V Heterogeneous Reconfigurable DSP IC for Wireless Baseband Digital Signal Processing.* IEEE Journal of Solid-State Circuits, Vol. 35, No. 11, Nov. 2000.

A Bestimmung der Technologieparameter

Bei dem zur Bestimmung des Zeitverhaltens einer Architektur in dieser Arbeit verwendeten Verzögerungsmodell (siehe Abschnitt 6.1.2) werden die Elemente der Verbindungsstruktur durch RC-Glieder modelliert. Die anschließende Bestimmung der Signallaufzeit mit dem Elmore-Modell erfordert die Kenntnis der zur Modellierung zugrunde liegenden äquivalenten Bauteilparameter. Dies sind die äquivalenten Widerstände R_{TG} und R_{Buf} von Transmission-Gates und Treibern, die Kapazitäten $C_{TG,leitend}$, $C_{TG,sperrend}$ für ein leitendes und sperrendes Transmission-Gate, die Ein- und Ausgangskapazitäten $C_{Buf,in}$ und $C_{Buf,out}$ der Treiber, die eingeprägte Buffer-Laufzeit $\tau_{Buf,int}$, sowie Widerstand $R_{Leitung}$ und Kapazität $C_{Leitung}$ eines Leitungssegmentes. Diese Bauteilparameter wurden mit Hilfe von Simulationen bestimmt, oder aus den Datenblättern der verwendeten Technologiebibliothek ermittelt. Die Geheimhaltungsvereinbarung mit Chartered Semiconductors verbietet die Angabe der dabei erhaltenen absoluten Zahlenwerte, jedoch werden die angewandten Methoden zur Bestimmung der Parameter im Folgenden kurz vorgestellt.

Äquivalente Widerstandswerte von Transferelementen

Abbildung A.1 verdeutlicht die Vorgehensweise bei der Bestimmung des äquivalenten Widerstandes $R_?$ eines Transferelementes. Ein solches Element, beispielsweise ein Transmission-Gate oder ein Leitungstreiber, wird bei der Bestimmung der Elmore-Verzögerung durch eine Kapazität C_T und den gesuchten äquivalenten Widerstand $R_?$ modelliert. Bei dem zur Bestimmung von $R_?$ verwendeten Simulationsaufbau (Abbildung A.1, unten) wird eine Kapazität C_L am Ausgang des Elementes gegen Masse geschaltet. Für $C_L \gg C_T$ lässt sich die Kapazität C_T vernachlässigen, so dass durch die Schaltung in guter Näherung ein RC-Glied aus dem gesuchten Widerstand $R_?$ und C_L gebildet wird. Die Zeitkonstante $\tau = R_? C_L$ dieses RC-Gliedes soll für das Elmore-Modell möglichst genau die 50%-Verzögerungszeit einer Flanke des Eingangssignales nachbilden. Diese 50%-Verzögerungszeit wiederum kann mit Hilfe einer Simulation bestimmt werden. Unter Kenntnis von τ und C_L ergibt sich $R_?$ zu:

$$R_? = \frac{\tau}{C_L} \tag{A.1}$$

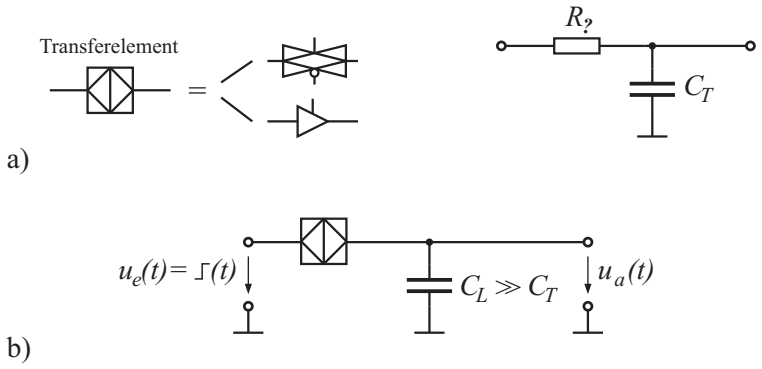


Abbildung A.1: Bestimmung des äquivalenten Widerstandswertes von Transmission-Gates und Treibern. a) Modellierung eines Transferelementes durch äquivalenten Widerstand $R_?$ und Kapazität C_T, b) Schaltung zur Bestimmung des äquivalenten Widerstandes $R_?$

Eingangskapazitätsmessung

Die Kapazität des Eingangspins eines Transferelementes lässt sich mit der in Abbildung A.2 skizzierten Vorgehensweise ermitteln. Ein Eingangspin kann durch eine Kapazität $C_?$ modelliert werden, die über einen Widerstand R_T geladen wird (Abbildung A.2a). Abbildung A.2b zeigt eine Schaltung, mit welcher die 50%-Verzögerungszeit eines mit C_{var} belasteten RC-Gliedes aus R_L und C_L bestimmt werden kann. Wird für verschiedene Werte von C_{var} die Verzögerungszeit τ gemessen, ergibt sich leicht $\tau = \tau(C_{var})$ als lineare Funktion in Abhängigkeit von C_{var} bei festen Werten von R_L und C_L. Wird weiterhin $R_L \gg R_T$ gewählt, lässt sich durch Auflösen dieser Funktion nach C_{var} bei Kenntnis von τ die gesuchte Kapazität $C_?$ des Eingangspins eines Transferelementes bestimmen (Abbildung A.2c).

Eingeprägte Laufzeit eines Leitungstreibers

Die eingeprägte Laufzeit eines Leitungstreibers ist die 50%-Verzögerung eines ausgangsseitig unbelasteten Buffers. Diese Zeit wurde durch Simulation ermittelt. Zu beachten ist hierbei, dass die Verzögerungszeit eines Buffers von der Form des Eingangssignales abhängig ist. Im Allgemeinen ergeben sich für steilere Flanken des Eingangssignales kürzere Laufzeiten. In dieser Arbeit wird zur Bestimmung der eingeprägten Laufzeit das Ausgangssignal einer Treiberstufe als Eingangssignal des betrachteten Buffers angnommen. Die sich dadurch ergebende Laufzeit ist etwa 15% größer als bei Messung mit einer Sprungfunktion als Eingangssignal.

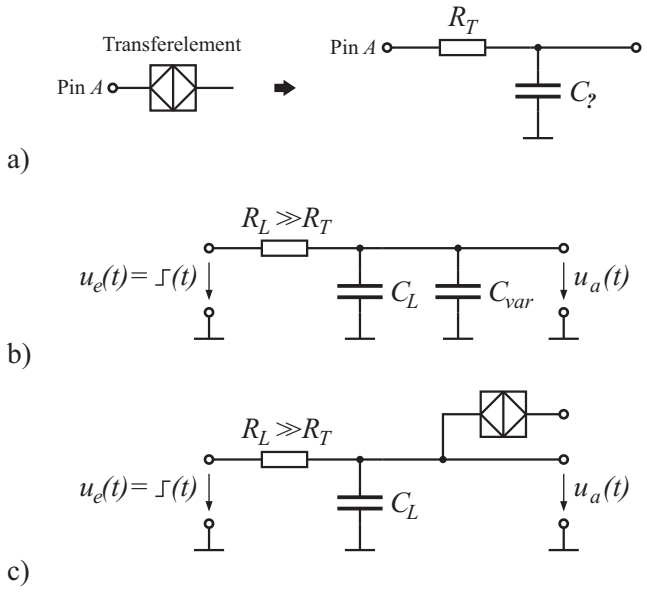


Abbildung A.2: Bestimmung der Kapazität des Eingangspins eines Transferelementes. a) Transferelement und Ersatzschaltbild für Eingangspin A, b) Ermittlung des Einflusses von C_{var} auf die Laufzeit, c) Schaltung zur Bestimmung der Kapazität eines Eingangspins

SRAM-Speicher

Flächenbedarf, Zugriffszeiten und Leistungsaufnahme des SRAM-Blockspeichers wurden jeweils den entsprechenden Datenblättern entnommen.

Widerstand und Kapazität eines Leitungssegmentes

Widerstand $R_{Leitung}$ und Kapazität $C_{Leitung}$ eines Leitungssegmentes sind unmittelbar abhängig von Länge L und Breite W der Leitung. Die Länge eines Segmentes wiederum ergibt sich aus der Kantenlänge der Prozessorelemente. PEs werden in dieser Arbeit als quadratisch angenommen, so dass sich die Länge L als Quadratwurzel aus der abgeschätzten PE-Fläche ergibt. Wie in [BRM99] wird auch in dieser Arbeit für die Leitungsbreite W der minimal zulässige Wert der verwendeten Technologie angenommen. Der ohmsche Widerstand $R_{Leitung}$ lässt sich in Abhängigkeit von L und W aus den Datenblättern der Technologiebibliothek ermitteln. Die Kapazität $C_{Leitung}$ ist von den Dimensionen der Leitung selbst, und zusätzlich vom Abstand zwischen zwei benachbarten Leitungssegmenten abhängig. Aufgrund der Kopplungskapazität zwischen benachbarten Leitungen wird die Gesamtkapazität eines Segmentes umso größer, je enger die Leitungen zusammenliegen. In dieser Arbeit wird angenommen, dass der Leitungsabstand doppelt so groß ist wie der

minimal zulässige Abstand, da dies eine Reduktion der Gesamtkapazität $C_{Leitung}$ eines Segmentes um mehr als 40% zur Folge hat. Die Kapazitätswerte ergeben sich wiederum aus den Datenblättern der Technologiebibliothek.

Lebenslauf

Persönliche Daten

Name	Lange
Vorname	Hendrik
Geburtsdatum	25.03.1975
Geburtsort	Hamm/Westf.

Ausbildung

1994	Abitur am Galilei-Gymnasium Hamm
1994-1995	Wehrdienst in Lippstadt
1995-2001	Studium der Elektrotechnik an der Universität Dortmund
01/2001	Abschluss des Studiums mit Diplom
25.10.2007	Promotion an der Universität Dortmund

Beruflicher Werdegang

01/2001 - 10/2005	Wissenschaftlicher Mitarbeiter am Arbeitsgebiet Schaltungen der Informationsverarbeitung der Universität Dortmund
seit 12/2005	Entwicklungsingenieur im Bereich digitaler IC-Entwurf, TES Electronic Solutions GmbH, Düsseldorf