



**UNIVERSITÄT PADERBORN**  
*Die Universität der Informationsgesellschaft*

FAKULTÄT FÜR  
ELEKTROTECHNIK,  
INFORMATIK UND  
MATHEMATIK

**Ressourceneffiziente Schaltungstechnik  
eingebetteter Parallelrechner  
—  
GigaNetIC**

Zur Erlangung des akademischen Grades

**DOKTORINGENIEUR (Dr.-Ing.)**

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

Dipl.-Ing. Jörg-Christian Niemann  
Bad Pyrmont

Referent: Prof. Dr.-Ing. Ulrich Rückert  
Korreferent: Prof. Dr. Klaus Waldschmidt

Tag der mündlichen Prüfung: 17.12.2008

Paderborn, den 19.12.2008

Diss. EIM-E/247

für Andrea und Max & Moritz

# Inhaltsverzeichnis

<b>1 Einleitung</b> .....	<b>1</b>
<b>2 Eingebettete parallele Rechnerarchitekturen</b> .....	<b>7</b>
2.1 Leistungsabschätzungen und Prognosen für CMPs.....	8
2.1.1 AMDAHLs Gesetz – eine asymptotische Barriere für Parallelrechner?.....	8
2.1.2 GUSTAFSONs Gesetz – ein Ausweg für die Parallelwelt?.....	9
2.1.3 Weiterführende Ansätze .....	10
2.1.4 Trends bei parallelen eingebetteten Systemen .....	11
2.2 Kernkomponenten eingebetteter paralleler Rechnerarchitekturen .....	14
2.3 On-Chip-Netzwerke.....	15
2.3.1 NoC-Topologien.....	15
2.3.2 Organisation von On-Chip-Kommunikation.....	21
2.3.3 Beispiele von On-Chip-Netzwerken .....	32
2.3.4 Anforderungen an On-Chip-Netzwerke .....	34
2.4 Eingebettete Verarbeitungseinheiten .....	34
2.4.1 Anforderungen an eingebettete Verarbeitungseinheiten .....	35
2.4.2 Klassen eingebetteter Verarbeitungseinheiten .....	35
2.4.3 Charakteristika von eingebetteten Prozessoren.....	37
2.4.4 Methoden zur Erhöhung der Leistungsfähigkeit von Prozessoren.....	38
2.4.5 Beispiele eingebetteter Prozessorkerne .....	39
2.5 Speicher für eingebettete Systeme.....	41
2.5.1 Wesentliche Charakteristika von Speicherstrukturen .....	42
2.5.2 Anforderungen an eingebettete Speicher .....	44
2.6 Anwendungsgebiete von On-Chip-Parallelrechnern .....	44
2.7 Anforderungen an Chip-Multiprozessoren .....	45
2.8 Varianten eingebetteter paralleler Rechnerarchitekturen .....	47
2.8.1 Beispiele zu Chip-Multiprozessoren .....	49
2.8.2 Ansätze für Chip-Multiprozessoren mit akademischem Ursprung .....	49
2.8.3 Ansätze für Chip-Multiprozessoren aus der Industrie.....	51

2.8.4 Resümierender Vergleich mit dem GigaNetIC-Ansatz.....	55
2.9 Zusammenfassung .....	58
<b>3 Charakterisierung und analytische Modellierung .....</b>	<b>59</b>
3.1 Ressourceneffizienz eingebetteter Systeme.....	59
3.2 Bewertungsmaße für Ressourceneffizienz .....	63
3.2.1 Bewertungsmaße zur Performanz .....	63
3.2.2 Bewertungsmaße zur Leistungsaufnahme.....	65
3.2.3 Bewertungsmaße zur Fläche .....	67
3.2.4 Bewertungsmaße zur Zukunftssicherheit und Flexibilität .....	68
3.2.5 Effizienzmaße zur Bewertung.....	71
3.3 Die vier bestimmenden Kostenmaße der Ressourceneffizienz .....	72
3.4 Zusammenfassung .....	74
<b>4 Die GigaNetIC-Systemarchitektur .....</b>	<b>75</b>
4.1 Neuartiges, ressourceneffizientes und skalierbares CMP-Systemkonzept.....	75
4.2 GigaNoC-On-Chip-Kommunikationsstruktur .....	78
4.2.1 Switch-Boxen als zentrale Kommunikationsknoten auf SoC-Ebene.....	79
4.2.2 On-Chip-Kommunikationsprotokoll .....	86
4.2.3 Performanzanalyse der Kommunikationsinfrastruktur .....	89
4.2.4 Bussysteme auf Cluster-Ebene.....	93
4.3 Verarbeitungseinheiten auf PE-, Cluster- und SoC-Ebene.....	94
4.3.1 Prozessorkern .....	95
4.3.2 Systemerweiterungen und Peripherie – das Prozessorsubsystem .....	97
4.3.3 Hardwarebeschleuniger.....	100
4.3.4 Sonstige IP-Blöcke.....	102
4.4 Speicher .....	103
4.4.1 Lokaler Speicher auf Cluster-Ebene .....	103
4.4.2 Cache-Speicher auf Cluster-Ebene.....	104
4.4.3 Hauptspeicher.....	108
4.5 Programmiermodell .....	109
4.5.1 Programmiermodell auf Clusterebene.....	110



---

4.5.2 Programmiermodell auf SoC-Ebene – <i>Bulk Synchronous Parallel</i> .....	111
4.5.3 Programmiermodell auf SoC-Ebene – Zentraler Kontrollprozessor .....	113
4.6 Diskussion von Topologie und Routingverfahren .....	114
4.7 Skalierung des Systems durch Variation von Systemparametern .....	117
4.8 Zusammenfassung .....	118
<b>5 Analyse und funktionale Verifikation des Chip-Multiprozessorsystems.....</b>	<b>121</b>
5.1 C-basierter Cluster-Simulator zur Simulation und Profilierung .....	123
5.2 Modellierung des GigaNetIC-Chip-Multiprozessors in SystemC .....	126
5.3 HDL-basierte Simulation auf Register-Transfer-Ebene .....	127
5.4 MultiSim – Parametervariation zur gezielten Entwurfsraumexploration .....	132
5.5 Systememulation mit einem Rapid-Prototyping-System.....	133
5.6 Einheitliche Übersetzer-Werkzeugkette .....	134
5.7 Zusammenfassung .....	136
<b>6 Optimierung der Multiprozessorarchitektur .....</b>	<b>137</b>
6.1 Optimierungsmethodik .....	137
6.2 Optimierung auf Prozessorebene .....	140
6.2.1 Compilerbasierter Entwurfsprozess zur Prozessoroptimierung .....	141
6.2.2 Hardwarebasierter Entwurfsprozess zur Prozessoroptimierung .....	143
6.2.3 Optimierungspotential von Befehlssatzerweiterungen – ein Beispiel.....	145
6.2.4 Implementierte anwendungsspezifische Instruktionen .....	148
6.3 Optimierung: Hardwarebeschleuniger auf Cluster- und SoC-Ebene.....	149
6.3.1 Optimierungspotential von Hardwarebeschleunigern – ein Beispiel.....	150
6.4 Kostenanalyse am Beispiel einer Netzwerkanwendung .....	163
6.5 Implementierte anwendungsspezifische Hardwarebeschleuniger .....	169
6.6 Optimierungspotential der Kommunikationsinfrastruktur .....	170
6.7 Optimierung im Hinblick auf die Speicherhierarchie.....	172
6.8 Optimierung auf SoC-Ebene – Einsatz paralleler Prozessorfelder.....	177
6.8.1 Optimierung der System- und Anwendungssoftware .....	177
6.8.2 Optimierung der Aufgabenverteilung und Interprozesskommunikation .....	178
6.9 Zusammenfassung .....	181

<b>7 Performanzanalyse skalierbarer GigaNetIC-Netzwerkprozessoren.....</b>	<b>183</b>
7.1 Einsatzgebiet im Zugangsnetzwerk – DSLAM .....	184
7.2 Definition eines IP-DSLAM-Benchmarks auf Systemebene .....	186
7.2.1 Funktionelle Spezifikation .....	187
7.2.2 Implementierung .....	188
7.2.3 Verkehrsmodell .....	188
7.2.4 Bewertungsmethode zum Vergleich unterschiedlicher Architekturen.....	189
7.2.5 DSLAM-Benchmarkanalysen für skalierbare GigaNetIC-CMPs .....	190
7.3 Instruktionssatzerweiterungen zur optimierten Protokollverarbeitung .....	192
7.4 Modulare, effiziente Modellierung von Netzwerkanwendungen .....	194
7.4.1 Erweiterung des DSLAM-Benchmarks zum Referenzbenchmark .....	196
7.4.2 IP-DSLAM-Referenzbenchmark – Ergebnisse.....	197
7.5 Visualisierungswerkzeug zur Entwurfsraumexploration.....	203
7.5.1 Vergleich eingebetteter Prozessorkerne – DSLAM-System-Explorer I .....	204
7.5.2 Einbeziehung von HW-Erweiterungen – DSLAM-System-Explorer II .....	207
7.6 Einsatz GigaNetIC-basierter Netzwerkprozessoren als Router.....	210
7.7 Analyse der Anschlussarten von Hardwarebeschleunigern im GigaNoC .....	211
7.8 Zusammenfassung .....	215
<b>8 Prototypische Implementierung des Systems .....</b>	<b>217</b>
8.1 FPGA-Realisierung – GigaNetIC-Prototyping-Plattform .....	217
8.1.1 Aufbau und Syntheseergebnisse.....	218
8.1.2 GigaNetIC-Demonstrator – Einsatz in einem realen Netzwerkszenario.....	220
8.2 ASIC-Realisierung in CMOS-Standardzellen .....	223
8.2.1 GigaNetIC-Architektur mit SRAM-basiertem L1-Speicher .....	223
8.2.2 GigaNetIC-Architektur mit integrierten Multiprozessorcaches.....	226
8.2.3 „Floorplan“ – ressourceneffiziente, kachelförmige Flächenaufteilung.....	227
8.3 Bewertung der Ressourceneffizienz .....	230
8.3.1 Einheitliche, werkzeuggestützte Performanzbewertung .....	231
8.3.2 Universalbenchmarks zur Bewertung der GigaNetIC-Architektur .....	232
8.3.3 Netzwerkbenchmark zur Bewertung der GigaNetIC-Architektur .....	233

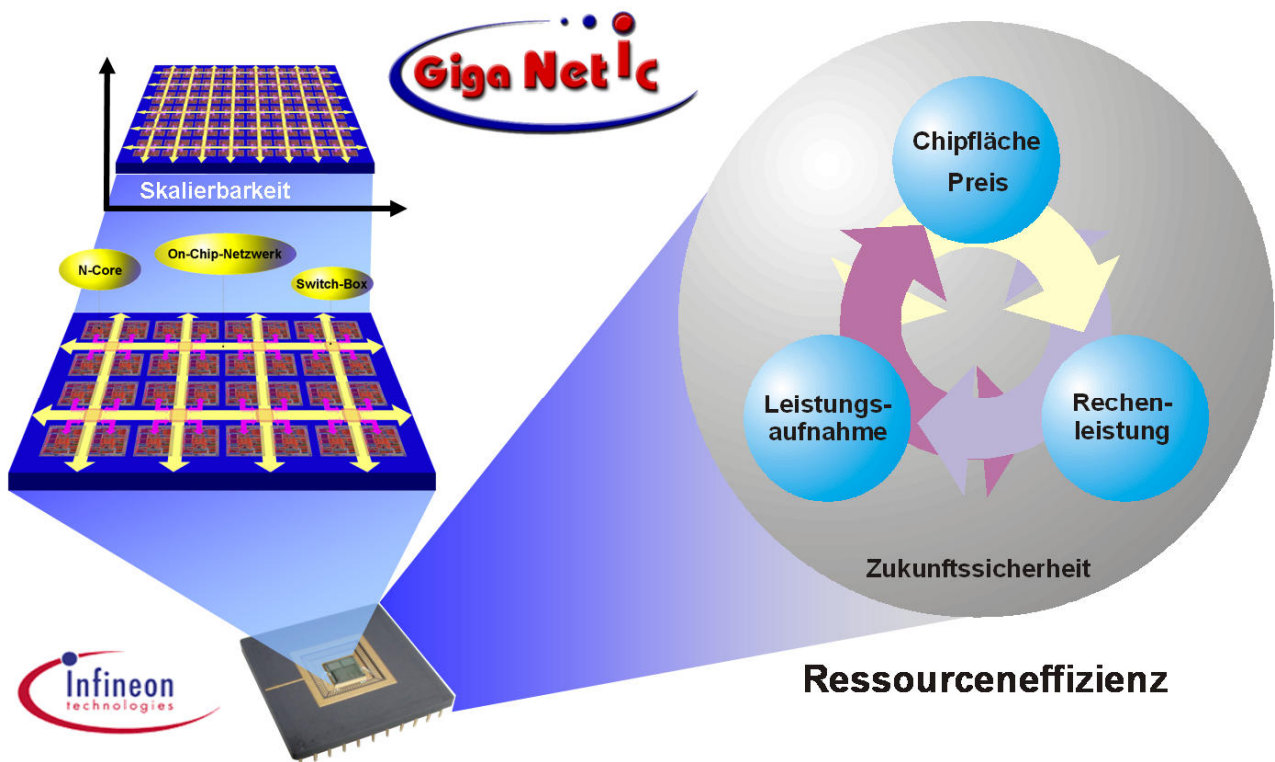
---

8.4 Zukünftige Architekturen .....	236
8.5 Zusammenfassung .....	237
<b>9 Zusammenfassung und Ausblick .....</b>	<b>239</b>
<b>Verzeichnis verwendeter Formelzeichen und Abkürzungen .....</b>	<b>245</b>
<b>Literaturverzeichnis.....</b>	<b>251</b>
<b>Eigene Veröffentlichungen .....</b>	<b>263</b>
<b>Anhang A (GigaNetIC-C-Bibliotheksfunktionen).....</b>	<b>267</b>
<b>Anhang B (Parametrisierbarkeit der GigaNetIC-Architektur).....</b>	<b>269</b>
<b>Anhang C (Ablauf der Kommunikation auf Switch-Box-Ebene) .....</b>	<b>273</b>
<b>Anhang D (Instruktionssatz des N-Cores) .....</b>	<b>275</b>
<b>Anhang E (Details zum IP-Headercheck-Hardwarebeschleuniger).....</b>	<b>279</b>
<b>Anhang F (IP-DSLAM-Referenzbenchmark) .....</b>	<b>281</b>



# 1 Einleitung

**Inhaltlicher Überblick.** Diese Arbeit dokumentiert den Entwurf und die Analyse einer leistungsfähigen und zugleich skalierbaren, ressourceneffizienten Chip-Multiprozessor(CMP)-Architektur, der GigaNetIC-Architektur, siehe Abbildung 1-1. Das GigaNoC ist die zu Grunde liegende hierarchische On-Chip-Kommunikationsinfrastruktur, die es durch ihre Modularität und Skalierbarkeit erlaubt, die Entwurfsproduktivitätslücke stärker zu schließen, als es bisher möglich war. Die Entwurfsproduktivitätslücke bezeichnet die Problematik, dass aufgrund der immer größer werdenden Integrationsdichte auf zukünftigen Halbleiterbausteinen mehr Funktionalität untergebracht werden kann, als in der zur Verfügung stehenden Zeit durch Entwicklerteams konstruktiv neu erzeugt werden kann.



**Abbildung 1-1: Netzwerktechnik der nächsten Generation - Architektur des massiv parallelen Netzwerkprozessors aus Paderborn, untergebracht auf dem 20stel der Fläche eines Cents**

**Motivation.** Das Konzept des vor nunmehr fast 50 Jahren entworfenen integrierten Schaltkreises sowie die fortwährenden, dem vor mehr als 40 Jahren aufgestellten MOORESchen Gesetz gehorchenden Verbesserungen in der Halbleitertechnologie erlauben immer komplexere und leistungsfähigere Schaltungen. Die von MOORE aufgestellte Regel besagt, dass sich die Anzahl der zu integrierenden Transistoren und damit die Komplexität integrierter Schaltkreise alle 18 bis 24 Monate verdoppeln. Abbildung 1-2 zeigt die Entwicklung von Speicher- und Prozessor-Modulen (MPU) in GBits bzw. in Logiktransistoren pro Chip auf und überlagert die prognostizierte Entwicklung nach MOORE [1].

Glaubt man den Prognosen der *International Semiconductor Roadmap (ITRS)* [2], die von führenden Halbleiterherstellern aus der gesamten Welt verfasst wird, so wird sich die prognostizierte Entwicklung nach MOORE auch in den nächsten Jahren fortsetzen (vgl. Abbildung 1-2). Dies schafft die

Voraussetzungen für höchst komplexe Systeme, die auf einen Chip integriert werden können (*System-on-Chip / SoC*). Nunmehr gilt es, diese technologischen Möglichkeiten sinnvoll einzusetzen und weiterhin beherrschbar zu halten, denn längst übersteigt die Anzahl der integrierbaren Transistoren die Leistungsfähigkeit vieler Schaltungsentwurfswerkzeuge. In diesem Zusammenhang spricht man auch, wie eingangs erwähnt, von einer Entwurfsproduktivitätslücke (*Design Productivity Gap*). Auch wenn die Beständigkeit des MOORESchen Gesetzes noch für die nahe Zukunft proklamiert wird, stößt man in einigen Gebieten schon jetzt an Grenzen. So wird die bis heute stetige Erhöhung der Taktfrequenz nicht mehr allein ausreichen, um die Leistungsfähigkeit von Prozessoren und Systemen angemessen zu erhöhen [3]. Vielmehr gehen die etablierten Hersteller, wie z. B. Intel, AMD, Sun oder IBM, bereits von allgemein verwendbaren Prozessoren (*General Purpose CPUs*) zu einer Integration mehrerer Rechenkerne auf einem Chip über [4][5]. Die ITRS prognostiziert nach einem Modell der *Japan Semiconductor Technology Roadmap Design Working Group* einen 1000mal größeren Bedarf an Rechenleistung für „Consumer“-SoCs als auch „High Performance“-SoCs in zehn Jahren, bei nahezu gleichen Anforderungen an die Leistungsaufnahme [2]. Auch hier geht man davon aus, dass die Lösung in parallelen Architekturen mit einer Vielzahl von integrierten Verarbeitungseinheiten liegt.

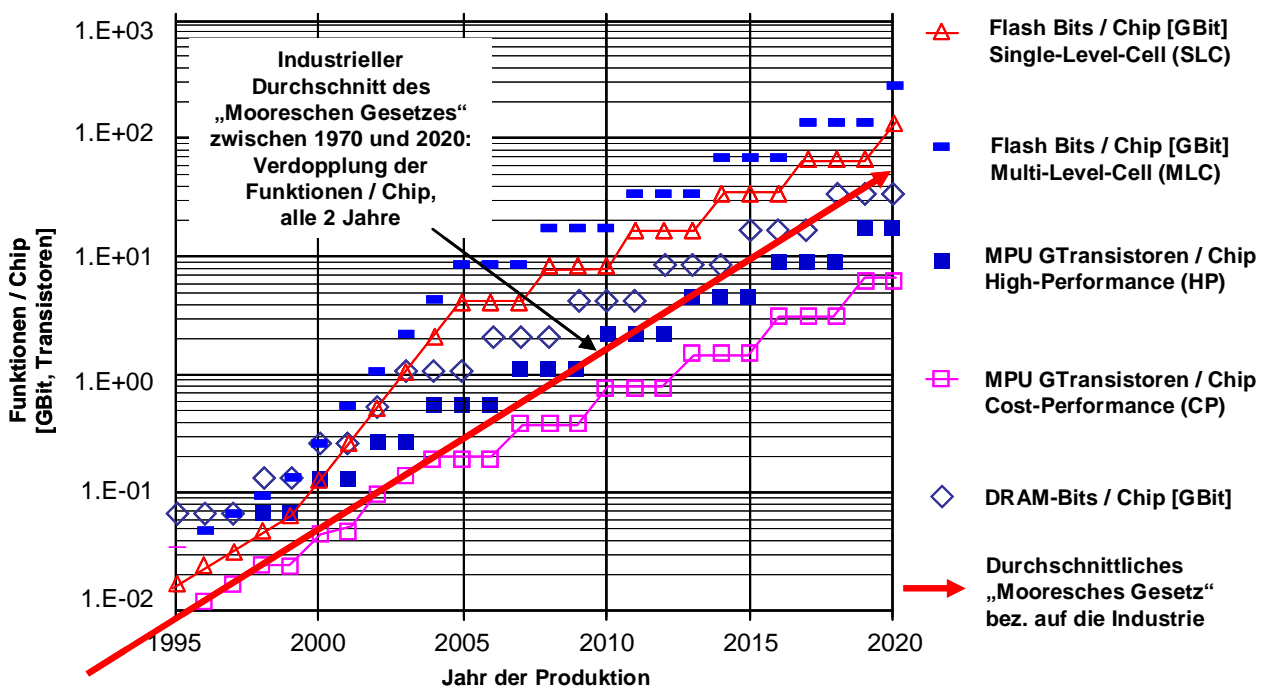
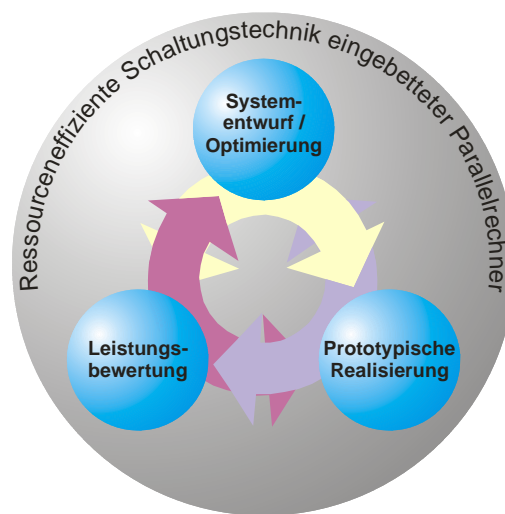


Abbildung 1-2: Produkt-Technologie-Trends – Funktionen pro Chip [2]

Durch Parallelität können in Abhängigkeit von der Anwendung die derzeitigen technischen Grenzen der Leistungsfähigkeit erweitert werden. Ein weiterer wesentlicher Aspekt der Verwendung von parallelen Einheiten ist die Möglichkeit der Reduktion bzw. Begrenzung der Leistungsaufnahme. Dies ist speziell für zukünftige eingebettete Systeme von großer Bedeutung, da diese mehr und mehr Einsatz in mobilen Bereichen finden, bei denen die Energieressourcen beschränkt sind. In diesem Zusammenhang steht auch die besondere Bedeutung der Ressourceneffizienz für diese Arbeit. Das heißt, die zu entwerfende Architektur soll, je nach Schwerpunkt der Systemspezifikation, unter Berücksichtigung einer Kostenfunktion eine bestmögliche Lösung bezüglich der betrachteten Kostenmaße Fläche, Leistungsaufnahme, Rechenleistung und Flexibilität darstellen (vgl. Abschnitt 3.1).

**Einbettung.** Durch die thematische Einbettung dieser Dissertation in das BMBF-Projekt GigaNetIC [6][7][8], profitiert das hier vorgeschlagene Architekturkonzept bereits im frühen Entwurfsstadium von der engen interdisziplinären Arbeitsweise der drei beteiligten Projektpartner der Universität Paderborn und Infineon Technologies, München. Die Expertise der drei Paderborner Fachgebiete liegt u. a. in der Entwicklung hochintegrierter Schaltkreise (Fachgebiet Prof. Dr.-Ing. Ulrich Rückert, Schaltungstechnik), in der Konzeption von Programmiersprachen und Übersetzern (Fachgebiet Prof. Dr. Uwe Kastens, Programmiersprachen und Übersetzer) sowie in der Entwicklung, Analyse und Implementierung von Kommunikationsalgorithmen (Fachgebiet Prof. Dr. math. Friedhelm Meyer auf der Heide, Algorithmen und Komplexität). Auf diese Weise kann bei der Konzeption der Systemarchitektur ein ganzheitlicher Ansatz verfolgt werden, der sowohl schaltungstechnische Entwurfskriterien berücksichtigt als auch compiler- und algorithmenbedingte Entwurfsentscheidungen im Wechselspiel miteinander vereint.



**Abbildung 1-3: Die Kernpunkte dieser Arbeit**

Die wesentlichen Beiträge dieser Arbeit (vgl. Abbildung 1-3) zur ressourceneffizienten Schaltungstechnik eingebetteter Parallelrechner sind:

- der Systementwurf eines massiv parallelen, skalierbaren SoC, basierend auf einem hierarchischen On-Chip-Netzwerk
- eine Werkzeugkette, die einen hierarchisch gerichteten Optimierungsansatz für SoCs unterstützt
- die prototypische Realisierung des Gesamtsystems als Simulationsmodell sowie als FPGA (*Field Programmable Gate Array*)- und Standardzellenimplementierung in 130-nm- und 90-nm-CMOS-Technologie
- Analyse und Definition von Benchmarks sowie die Leistungsbewertung des entworfenen Systems für ausgewählte Anwendungsszenarien.

**Gliederung.** Kapitel 2 gibt einen Überblick über eingebettete parallele Rechnerarchitekturen und die wesentlichen Komponenten dieser SoCs. Typische Anwendungsgebiete von On-Chip-Parallelrechnern werden aufgezeigt und vermitteln einen ersten Einblick in die Anforderungen, die an diese Systeme gestellt werden.

Definitionen, Bewertungsmaße, Kostenfunktionen und analytische Modelle werden in Kapitel 3 zur Bewertung der Architektur eingeführt und im weiteren Verlauf dieser Arbeit mit den Syntheser-

gebnissen der Hardwareblöcke korreliert. Diese Modelle können herangezogen werden, um für zukünftige Technologien und Architekturvarianten im Vorfeld der Realisierung erste Bewertungen treffen zu können. Anhand von Kostenfunktionen können dann besonders geeignete Realisierungen dedizierter Anwendungsszenarien leichter ermittelt werden. Der Begriff der Ressourceneffizienz wird hier diskutiert und im schaltungstechnischen Kontext definiert.

Kapitel 4 zeigt den Aufbau und die Besonderheiten des eigenen Ansatzes für ein ressourceneffizientes eingebettetes System. Es soll eine flexible, skalierbare Architektur, die auf dem Konzept massiver Parallelverarbeitung basiert, entwickelt werden, also ein leistungsstarker Parallelrechner auf einem Chip. Kernkomponenten werden eine Vielzahl homogener Verarbeitungseinheiten sein, die über ein hoch performantes On-Chip-Netzwerk (*Network-on-Chip / NoC*) miteinander verbunden sind. Durch die besondere Skalierbarkeit der Systemarchitektur kann eine Verwendbarkeit für viele Einsatzgebiete erreicht werden. Das modulare, leicht erweiterbare Konzept erlaubt die einfache Integration zusätzlicher Hardwarebeschleuniger und anderer anwendungsspezifischer Funktionseinheiten an verschiedenen, unterschiedlich leistungsfähigen Schnittstellen im SoC. Besonders geeignete Systemkonfigurationen für dedizierte Anwendungen können durch die in Kapitel 6 vorgestellte Werkzeugkette komfortabel bestimmt und im Sinne der Ressourceneffizienz optimiert werden. Der *IP(Intellectual Property)*-basierte Ansatz hilft, die Entwurfsproduktivitätslücke, die bei modernen Chipentwürfen zunehmend eine Rolle spielt, zu schließen. So kann in vielen Fällen auf bereits bestehende Hardwareblöcke zurückgegriffen werden, welche dann leicht mit Hilfe definierter Kapselungen (*Wrapper*) in das SoC integriert werden können. Die Wiederverwendbarkeit wird erhöht und der Entwurfsaufwand reduziert sich. Weitere Vorteile dieser homogenen, skalierbaren Systemarchitektur liegen in dem einheitlichen Programmiermodell und der vereinfachten Testbarkeit.

Kapitel 5 zeigt die verschiedenen Abstraktionsebenen, die für die Simulation bzw. Emulation des Systems entworfen werden, auf. Eine zyklenakkurate C-basierte Simulation auf *Prozessor-Cluster*-Ebene ermöglicht ein schnelles und komfortables Ausmessen der Laufzeiten einzelner Programmabschnitte. Die abstraktere Modellierung in SystemC liefert hingegen Aussagen zur Leistung des Gesamtsystems und ermöglicht eine frühe Verifizierung der Funktionsfähigkeit der Systemsoftware. Durch Variation wesentlicher Systemparameter können aufgrund der hohen Simulationsgeschwindigkeit schnell Rückschlüsse auf die zu erwartende Performanz der späteren Hardware gezogen werden. Die rechenintensivere *RTL(Register Transfer Level / Register-Transfer-Ebene)*-Simulation erlaubt letztendlich detaillierte Aussagen über das Verhalten der einzelnen Hardwarekomponenten. Im Anschluss an die Verifikation auf dieser Ebene erfolgt der Test einzelner Blöcke und der Gesamtschaltung auf dem FPGA-basierten Rapid-Prototyping-System RAPTOR2000 [9][10], das, verglichen mit der RTL- und SystemC-Simulation, eine um Größenordnungen schnellere Emulation des Multiprozessorsystems ermöglicht und zusätzlich die Anbindung realer Netzwerkkomponenten gestattet.

In Kapitel 6 wird eine Methode vorgestellt, die dem Ziel dient, eine besonders effiziente Architekturvariante – im Sinne der Ressourceneffizienz – für ein gegebenes Anwendungsszenario zu erzielen. Hierbei kann die homogene Ausgangsarchitektur durch eine Reihe von Mechanismen optimiert werden. Der Optimierungsansatz ist hierarchisch gerichtet und sieht u. a. folgende Maßnahmen vor: Anpassung und Optimierung der Software, Erweiterung des Instruktionssatzes des Prozessors, Hinzufügen von Hardwarebeschleunigern sowie Abstimmung der On-Chip-Kommunikationsinfrastruktur auf die zu erwartenden Datenraten und Einsatz von parallel arbeitenden Rechenclustern auf



einem Chip sowie die Optimierung der Speicherhierarchie im Hinblick auf die spätere Anwendung. Exemplarisch wird die in Kapitel 3 vorgestellte Methode anhand von 16 Realisierungsvarianten eines einfachen selbst entworfenen Netzwerkprozessors demonstriert. Mit Hilfe der Kostenfunktions-basierten Analyse wird für unterschiedliche Anwendungsszenarien die eine möglichst pareto-optimale und damit ressourceneffiziente Lösung ermittelt.

Kapitel 7 greift die bisherigen Ergebnisse auf und wendet sie auf konkrete Kommunikations- und Netzwerkanwendungen an. Hierzu werden sowohl etablierte Benchmarks verwendet als auch neue Benchmarks definiert. Besondere Berücksichtigung finden hier Funktionen aus dem stark wachsenden *DSL(Digital Subscriber Line)*-Segment. *DSL-Access-Multiplexer (DSLAMs)* realisieren die schnelle Datennetzanbindung der DSL-Endkunden. Für diese Netzwerkknoten, die eine Vielzahl von Datenströmen aggregieren und zum Internetdiensteanbieter (*Internet-Service-Provider / ISP*) weiterleiten und umgekehrt, wird ein spezifischer Benchmark entworfen und auf der GigaNetIC-Architektur evaluiert. Netzwerksimulationen sollen Aufschluss über die zu wählende Topologie und den internen Aufbau des On-Chip-Netzwerks liefern und so einen maximalen Durchsatz und damit einhergehend eine möglichst optimale Lastverteilung auf die einzelnen Verarbeitungseinheiten garantieren. Es werden Instruktionssatzerweiterungen und spezifische Hardwarebeschleuniger vorgestellt, die in dieser Arbeit entstanden sind. Exemplarisch werden die Funktionsweise und die Integration eines Hardwarebeschleunigers zur Paketverarbeitung detaillierter beschrieben.

Gegenstand von Kapitel 8 sind prototypische Realisierungen der GigaNetIC-Architektur. Das in Kapitel 7 untersuchte Netzwerkszenario wird als FPGA-Realisierung in einer realen Netzwerkkumgebung in Betrieb genommen und dient zur Veranschaulichung und Verifikation des Systemkonzepts. Auf Basis der erfolgreichen Realisierung werden dann Implementierungen für komplexere Systeme in 130-nm- und 90-nm-CMOS-Standardzellentechnologie vorgestellt. Abbildung 1-1 verdeutlicht das in dieser Arbeit vorgestellte Konzept und die Anforderungen an die Realisierung. Weitere Bewertungen der Architektur im Hinblick auf die Ressourceneffizienz schließen die praktischen Betrachtungen der GigaNetIC-Chip-Multiprozessorarchitektur ab.

Zusammenfassend soll die hier zu entwerfende GigaNetIC-Architektur als Basis für weitere CMP-Varianten dienen und ein neues Paradigma der Prozessorarchitektur aufzeigen, das besonders durch Modularität, Skalierbarkeit und Ressourceneffizienz sowie einen ganzheitlichen Ansatz hervorsteicht und prototypisch verifiziert wird.



## 2 Eingebettete parallele Rechnerarchitekturen

Bereits 2004 zeichnete sich deutlich ein Umdenken in der Prozessorindustrie hinsichtlich paralleler Strukturen ab. Der Ende 2004 veröffentlichte Microprocessor-Report-Artikel „Intel Cancels 4 GHz P4“ [3] sah die Abkündigung der Desktop-CPU mit der bis dahin höchsten Taktfrequenz als Überraschung an. Tatsächlich ist ein serienmäßiger 4-GHz-Pentium bis heute nicht zu erwerben. Letztendlich erkannte man das kontinuierliche Steigern der Taktfrequenz als unzureichende Maßnahme, und nicht nur Intel wurde gezwungen die Prozessor-Roadmap neu zu überdenken, was nicht zuletzt in der Entwicklung der Dual-Cores, der heutigen Core-Architektur resultierte.

Trotz aller Hindernisse bei manchen Anwendungsszenarien ist selbst bei den namhaften Desktop- und Server-CPU-Herstellern wie Intel und AMD der Trend zu Dual-, Quad- oder gar Multi-Cores zu verzeichnen. Für Prozessoren mit noch mehr Kernen wurde bei Intel der Name „Many-Cores“ etabliert, um sich von den heutigen „Multi-Cores“ mit einigen wenigen Prozessorkernen noch stärker abgrenzen zu können (vgl. Abschnitt 2.8.1). Nicht nur Intel plant, Architekturen zu bauen, die weit mehr als nur eine CPU auf einem Siliziumchip vereinen. Eines der jüngsten Beispiele für solche „Mehrkern-Prozessoren“ ist der Cell-Prozessor von IBM, Sony und Toshiba, der u. a. in der Playstation 3 eingesetzt wird. Er wurde ab März 2001 mit einem Budget von über 400 Mio. US\$ von einem 400 Personen umfassenden Team entwickelt [11]. Der Cell-Prozessor integriert acht mit 3,2 GHz getaktete Recheneinheiten auf einem *Die*<sup>1</sup>. Die ungefähre Leistungsaufnahme dieses Systems liegt deshalb auch deutlich über 100 W [12]. Die permanente Erhöhung der Taktfrequenz, die eine Verringerung der Betriebsspannung nur schwerlich ermöglicht, führt wie in Abschnitt 3.1 beschrieben, zu einer immensen Leistungsaufnahme der CPUs, die u. a. zu hohen Kosten für Gehäuse und Kühlung führt. Anders kann sich dies bei parallelen Architekturen gestalten. Hier lässt sich die Versorgungsspannung aufgrund der geringeren Frequenz deutlich reduzieren. Aufgrund des quadratischen Anteils der Versorgungsspannung an der dynamischen Verlustleistung führt dies zu einer drastischen Reduktion der Leistungsaufnahme des Systems, vgl. Gleichung (3.18). Weiterhin lässt sich die Performanz einer Prozessorarchitektur durch Maßnahmen wie Architekturoptimierung und Integration von Hardwarebeschleunigern deutlich erhöhen. Gleichzeitig kann eine Reduktion der Leistungsaufnahme verzeichnet werden (vgl. Kapitel 6). 2007 betrug in den USA der Anteil von Computersystemen und Peripherie am Gesamtenergiebedarf mehr als 15% [13]. Derzeit benötigen die Betreiber der fünf größten Suchmaschinen ca. zwei Millionen Server, die ungefähr 2,4 GW Leistung aufnehmen [13]. Mit permanent wachsender Serverzahl wird in Zukunft der Bedarf steigen, große Serversysteme auf energieeffizienten Parallelsystemen zu virtualisieren.

Auch AMD hat eine solche Architekturerweiterung für die zukünftigen Chip-Generationen angekündigt, bei der nicht nur homogene Mehrkernprozessoren, sondern auch anwendungsspezifische Hardwarebeschleuniger integriert werden sollen. AMD spricht hier von der „Accelerated Processing Era“, die ab 2009 mehr und mehr an Bedeutung gewinnen werde. Diese Konstellation von sowohl homogenen, massiv-parallelen Rechenkernen als auch beliebig integrierbaren Hardwarebeschleunigern für verschiedenste Anwendungsgebiete ist bei der GigaNetIC-Architektur bereits rea-

---

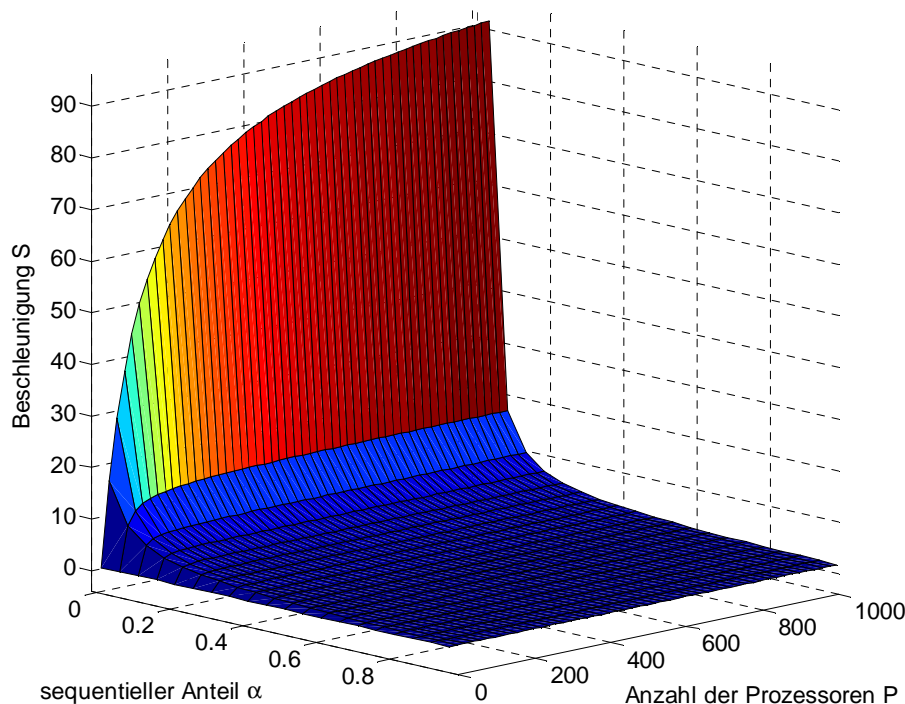
<sup>1</sup> Der *Die* bezeichnet den zumeist rechteckigen Halbleiterblock einer integrierten Schaltung.

lisiert und erfolgreich getestet. Dieses neuartige Systemkonzept wurde bereits 2002 strukturell [14] und 2003 detailliert [6] von mir vorgestellt. Ähnliche Ansätze halten nun zunehmend weltweit Einzug in Prozessorarchitekturen der nächsten Generation. Dabei ist außerdem zu berücksichtigen, dass zu der Architektur eines Systems nicht nur der reine Aufbau der Hardware zählt, sondern auch die Einbettung in ein funktionales Programmiermodell, das zusammen mit den Fachgebieten „Programmiersprachen und Übersetzer“, Prof. Uwe Kastens und „Algorithmen und Komplexität“, Prof. Friedhelm Meyer auf der Heide, der Universität Paderborn für die GigaNetIC-Architektur erfolgreich entworfen wurde.

## 2.1 Leistungsabschätzungen und Prognosen für CMPs

Der folgende Abschnitt dokumentiert einige etablierte Abschätzungen bzw. postulierte Gesetzmäßigkeiten für Parallelrechner. Abschließend wird auf Prognosen zur Entwicklung von Mehrprozessorsystemen und die damit verbundenen Herausforderungen eingegangen.

### 2.1.1 AMDAHLs Gesetz – eine asymptotische Barriere für Parallelrechner?



**Abbildung 2-1: Anwendungsbeschleunigung durch Ausnutzung inhärenter Parallelität nach dem Gesetz von AMDAHL**

Eine etablierte, wenn auch sehr vereinfachte Abschätzung zur Leistungssteigerung durch Parallelität liefert das AMDAHLsche Gesetz [15]. Es besagt, dass Anwendungen durch parallele Ausführung nur zu dem Grad beschleunigt werden können, wie es die enthaltene Parallelität des sequentiellen Anwendungsprogramms zulässt:

$$S(P) = \frac{1}{\alpha + \frac{1-\alpha}{P}} \Rightarrow \lim_{P \rightarrow \infty} S(P) \leq \frac{1}{\alpha} \quad (2.1)$$

$S$  kennzeichnet hierbei den *Speedup*, also die Beschleunigung der Anwendung bei der Verwendung von  $P$  parallel arbeitenden Verarbeitungseinheiten. Dabei ist  $\alpha$  mit  $0 \leq \alpha \leq 1$  als sequentieller Anteil des Programms zu sehen.

Abbildung 2-1 zeigt die mögliche Beschleunigung  $S$  eines Programms durch parallele Verarbeitung in Abhängigkeit von der Anzahl der Prozessoren  $P$  und dem sequentiellen Anteil  $\alpha$  des Programms. Es ist deutlich zu sehen, dass bereits ab  $\alpha > 0,1$  eine drastische Reduktion der Beschleunigung einsetzt. Die maximal erreichbare Beschleunigung bei einem sequentiellen Anteil von 1 % nähert sich asymptotisch dem Faktor 100. Bei 1024 Prozessoren liegt sie bei 91,18.

Gleichung (2.1) ergibt eine begrenzte, asymptotisch verlaufende Beschleunigung durch Parallelität für Anwendungen mit einem üblicherweise nicht vollkommen vernachlässigbaren sequentiellen Anteil. Anwendungsklassen, deren Problemgröße hingegen skalierbar ist, wie z. B. bei den in Kapitel 7 betrachteten Netzwerkszenarien, erfordern andere Modelle.

### 2.1.2 GUSTAFSONS Gesetz – ein Ausweg für die Parallelwelt?

Das bereits 1967 von AMDAHL postulierte Gesetz wurde 1988 von GUSTAFSON aufgrund praktischer Beobachtungen an einem Parallelrechnersystem mit 1024 Prozessoren modifiziert und ging als GUSTAFSONS Gesetz in die Literatur [16] ein.

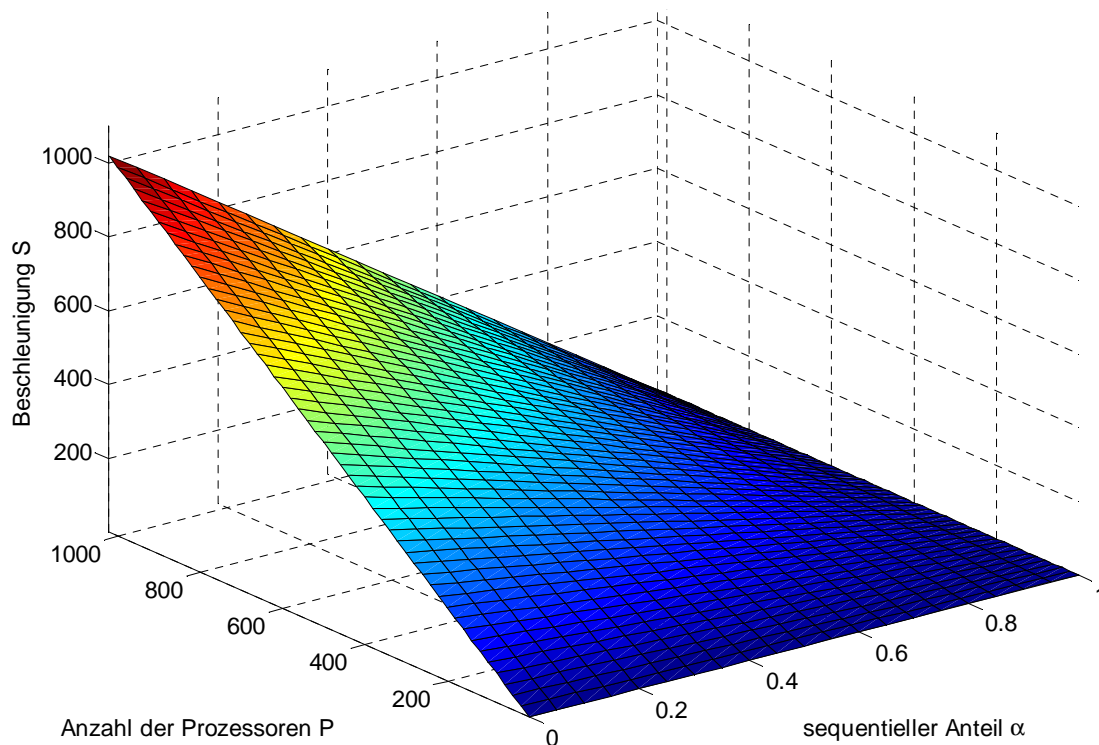


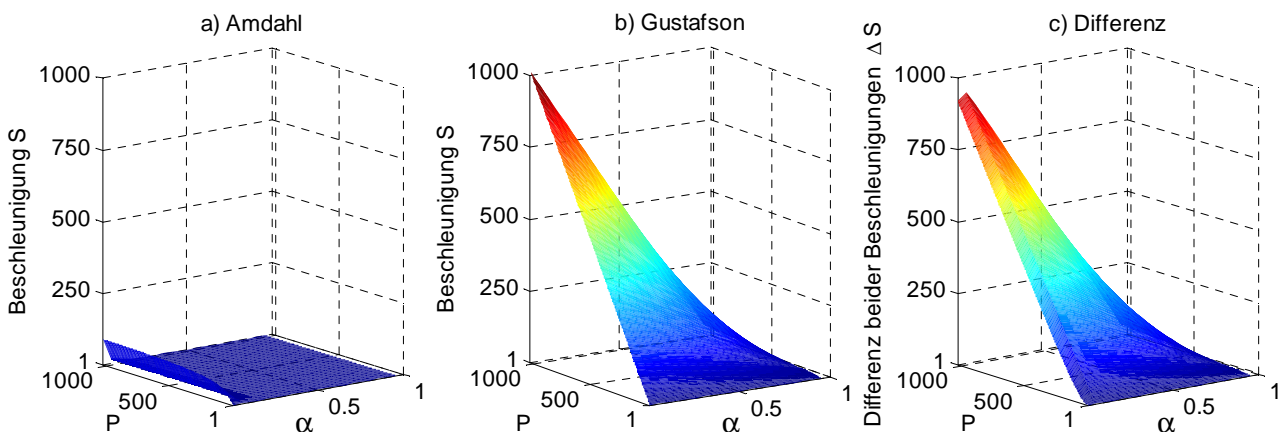
Abbildung 2-2: Beschleunigung durch Parallelität nach dem Gesetz von GUSTAFSON

GUSTAFSON beobachtete für Anwendungsklassen, deren Problemgröße skalierbar war eine deutliche Beschleunigung durch den Einsatz von zusätzlichen Prozessoren. Dies bedeutete eine Modifikation der Randbedingungen von AMDAHLs Gesetz. Er wies darauf hin, dass AMDAHLs Gesetzmäßigkeit diese wesentliche Komponente vieler Anwendungsszenarien außer Acht lässt: „*in practice the problem size scales with the number of processors*“. Das bedeutet, dass ein möglicher Zusammenhang zwischen dem parallelen Anteil der Anwendung und der Anzahl zur Verfügung stehender Prozessoren in AMDAHLs Formel (2.1) ignoriert wird, da Amdahl diese Art Anwendungsklasse

nicht untersucht hat. GUSTAFSON macht deutlich, dass man ein Problem mit bekannter Komplexität nicht auf einen beliebig großen Rechnerverbund auslagert, sondern versucht wird, komplexere Probleme durch mehr Parallelität in endlicher Zeit zu lösen. Dies impliziert, dass eher die Laufzeit als konstant anzusehen ist als die Problemgröße [16]. Die modifizierte Formel hat deshalb die folgende Form:

$$\begin{aligned}
 S_{\text{skaliert}}(P) &= \frac{\text{Mehrprozessorsystem}}{\text{Einzelprozessor}} \\
 &= \frac{\text{sequentieller Anteil} + \text{paralleler Anteil} \cdot \text{Prozessoren}}{\text{sequentieller Anteil} + \text{paralleler Anteil}} \\
 &= \frac{\alpha + (1 - \alpha) \cdot P}{\alpha + (1 - \alpha)} \\
 &= \alpha + (1 - \alpha) \cdot P \Rightarrow \lim_{P \rightarrow \infty, \alpha < 1} S(P) = \infty
 \end{aligned}
 \tag{2.2}$$

Gleichung (2.2) zeigt für Anwendungsklassen skalierbarer Problemgrößen deutlich realistischere Perspektiven für parallele Systeme und ihre Leistungsfähigkeit auf. Abbildung 2-2 stellt, in Abhängigkeit von der Anzahl der Prozessoren  $P$  und dem sequentiellen Anteil  $\alpha$  des Programms, den linearen Verlauf der Beschleunigung nach GUSTAFSONS Gesetz dar.



**Abbildung 2-3: AMDAHLs und GUSTAFSONS Gesetz in Bezug auf Anwendungen mit skalierbarer Problemgröße**

Die Prognosen beider Ansätze in Bezug auf Anwendungen mit skalierbarer Problemgröße und die sich ergebende Differenz ist in Abbildung 2-3 dargestellt. Eine Missinterpretation des AMDAHLschen Gesetzes für diese Anwendungsklassen würde zu dramatischen Fehleinschätzungen für die Zukunft paralleler Systeme führen. In der Vergangenheit zeigte sich deshalb, eine häufig unbegründete, Skepsis gegenüber der Leistungsfähigkeit massiv paralleler Architekturen.

Ein Manko der Ansätze von sowohl AMDAHL, als auch von GUSTAFSON ist die unzureichende Modellierung des Datenaustauschs zwischen den parallelen Prozessorelementen, der lediglich mit in den sequentiellen Anteil der Anwendung eingehen kann.

### 2.1.3 Weiterführende Ansätze

Weitere, tiefer gehende Ansätze zur Bestimmung bzw. Bewertung von massiv-parallelen Rechnersystemen liefern CULLER [17] mit dem *LogP*-Modell und VALIANT mit dem *BSP* (*Bulk Synchronous Parallel*)-Ansatz [18] (vgl. Abschnitt 4.5.2). Beiden Ansätzen ist gemein, dass sie versuchen, die

Lücke zwischen theoretischen Erkenntnissen und realen Maschinen zu schließen. In beiden Modellen wird außerdem die Kommunikation zwischen den einzelnen Knoten berücksichtigt.

Das LogP-Modell von CULLER wurde speziell für den praktischen Einsatz auf realen Multiprozessor-Topologien entworfen. Es geht dabei von einem Multiprozessorsystem mit verteiltem Speicher und einer beliebigen Kommunikationsinfrastruktur mit Punkt-zu-Punkt-Verbindungen beliebiger Topologie aus. Die wesentlichen Parameter des Modells sind die Latenz  $L$  als obere Grenze für die Zeitspanne der Übertragung einer Nachricht weniger Wörter vom Sender zum Empfänger und  $o$  als *Overhead* oder Verwaltungsaufwand, mit dem ein Prozessor aktiv mit der Übertragung beschäftigt ist, während der er keine anderen Aufgaben erledigen kann.  $g$  definiert die Lücke *Gap* zwischen zwei aufeinander folgenden Übertragungen bzw. Empfangsvorgängen eines Prozessors. Der reziproke Wert von  $g$  entspricht der Übertragungsbandbreite  $B$ , die den einzelnen Prozessoren zur Verfügung steht.  $P$  beziffert die Anzahl der Prozessor-Speicher-Kombinationen. Aufgrund der als endlich angenommenen Kapazität der Übertragungskanäle ergibt sich die Anzahl der zeitgleich übertragbaren Nachrichten zu  $\left\lceil \frac{L}{g} \right\rceil$ .  $L$ ,  $o$  und  $g$  werden in Vielfachen eines Prozessortaktes angegeben.

Das Modell ist auch auf „Shared Memory“-Architekturen anwendbar, wobei dann für die Kommunikation zum Speicher hin und zurück ein Wert von  $2L+4o$  angesetzt wird. Für eine gegebene Zeit  $T$  und feste Werte für  $L$ ,  $o$ ,  $g$  und  $P$  kann dann eine effiziente Verteilung der Aufgaben auf die Prozessoren, „Computation Schedule“, und ein Zeitplan für die Nachrichtenübertragung, „Communication Schedule“, aufgestellt werden. BSP als auch LogP sind geeignete Modelle, um sowohl Computernetzwerke im Allgemeinen als auch On-Chip-Netzwerke bezüglich ihrer Leistungsfähigkeit zu charakterisieren.

#### 2.1.4 Trends bei parallelen eingebetteten Systemen

Aufgrund der Anforderungen der von mir betrachteten Anwendungsszenarien, die eine von GUSTAFSON beschriebene Parallelität enthalten und nicht, wie das AMDAHLSche Gesetz (2.1) vermuten ließe, nur wenige Verarbeitungseinheiten ausnutzen können, werden mehr und mehr Prozessorelemente (PE) auf heutigen und zukünftigen Halbleiterbausteinen integriert. Diese stetige Zunahme wird ebenfalls in der *International Technology Roadmap for Semiconductors* der *Semiconductor Industry Association* [2] prognostiziert (vgl. Abbildung 2-4). Demzufolge ist ein rapider Anstieg der realisierbaren Anzahl von Verarbeitungseinheiten auf einem Chip bis zum Jahre 2020 zu erwarten: von gegenwärtig um die 20 Verarbeitungseinheiten bis zu über 870 bei einer konstant bleibenden Chipgröße von 64 mm<sup>2</sup> in weiteren 15 Jahren. Dies entspricht einem mehr als 40-fachen Zuwachs bis zum Ende der nächsten Dekade.

Außerdem zeichnet sich der Trend ab, dass die Größe des integrierten Speichers eines SoCs stärker zunimmt als die Größe für Logikblöcke. Dies ist wiederum mit der bereits erwähnten Entwurfsproduktivitätslücke und der Wiederverwendbarkeit von Hardwareblöcken zu erklären. So lässt sich Speicher leichter wiederverwenden als Logikblöcke, die bei einer Wiederverwendung immer noch einen gewissen Entwurfsaufwand zur Integration benötigen. Bis zu 50 % des normalen Entwurfsaufwandes für Logikblöcke entstehen u. a. durch funktionale Erweiterung und den Aufwand für die physikalische Implementierung. Um die Möglichkeiten, die neue Technologien bieten, ausschöpfen zu können, wird aufgrund der effizienten Nutzungsmöglichkeiten mehr und mehr der zur Verfügung stehenden Fläche mit Speicher ausgefüllt. Nach [2] sind allerdings insbesondere im Bereich

der Logikblöcke zusätzliche Anstrengungen erforderlich, um die Entwurfsproduktivität in diesem Bereich zu steigern. Das kann durch Instanzieren von bekannten Verarbeitungseinheiten erfolgen, die im Sinne von Dokumentation, Testbarkeit und Verifizierung gut erfasst sind. Dies erfordert dann jedoch eine leistungsfähige Kommunikationsinfrastruktur auf dem Chip, um die wiederverwendeten Einheiten effizient einsetzen zu können (siehe Abschnitt 4.2). Außerdem werden „High-Level“-Modellierungsansätze auf abstrakterer Ebene, wie sie durch SystemC-Beschreibungen möglich sind, unabdingbar, denn sie erlauben eine deutliche Steigerung der Entwurfsproduktivität [2]. Für das gesamte GigaNetIC-System wurde bereits eine solche Modellierung erstellt, von der bereits viele Bereiche der Soft- und Hardwareverifizierung sowie -planung profitieren (vgl. Abschnitt 5.2).

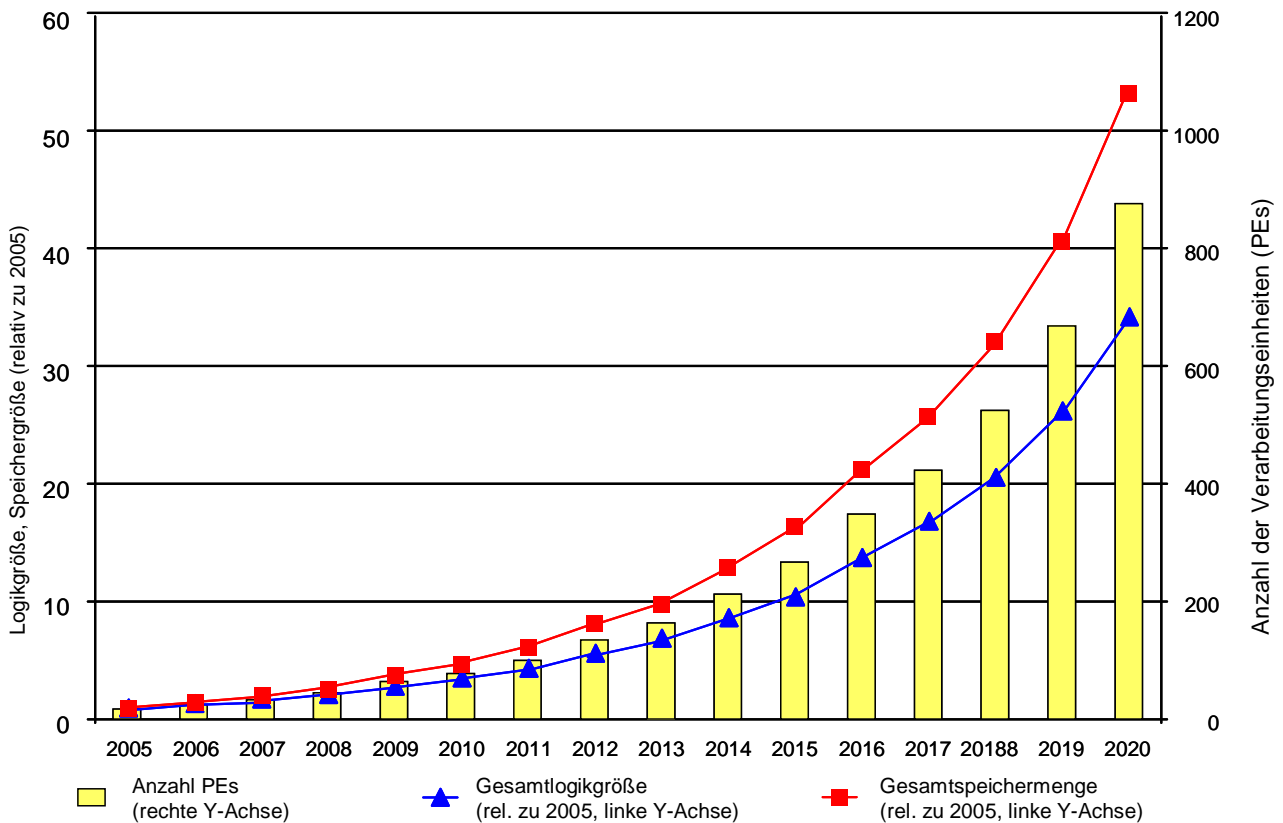


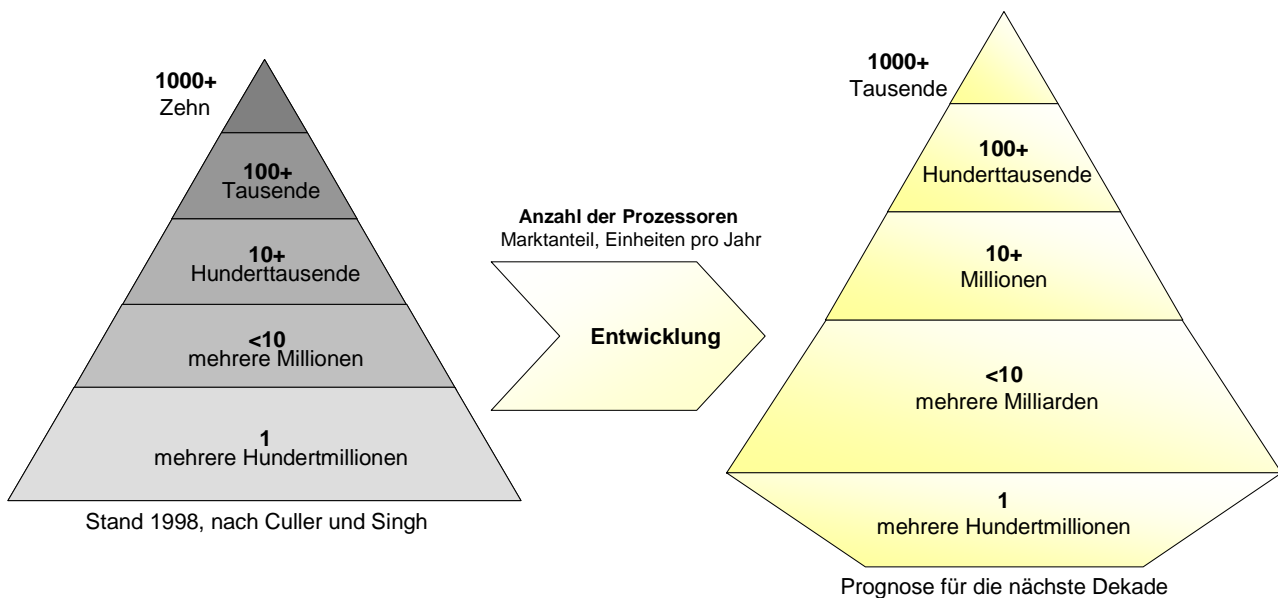
Abbildung 2-4: System-on-Chip-Entwicklungs-trends [2]

Ein weiterer wesentlicher Ansatz zur Steigerung der Entwurfsproduktivität liegt nach [2] u. a. in der Verbesserung der Entwurfsautomatismen und der damit verbundenen Entwurfswerkzeuge, die für die GigaNetIC-Architektur in den Kapiteln 5 und 6 ausführlich beschrieben werden. Die ITRS prognostiziert eine immense Erhöhung der Rechenleistungsanforderungen an „Consumer“-Produkte, und zwar um den Faktor 200 in den nächsten 15 Jahren [2]. Die potentiellen Anwendungsgebiete, die diesen Zuwachs fordern, werden in Abschnitt 2.6 kurz vorgestellt.

CULLER und SINGH [19] veranschaulichen in einer pyramidenförmigen Aufstellung die Marktanteile für Prozessorsysteme (vgl. Abbildung 2-5). Bereits 1998 gab es weltweit einen Markt für mehr als hunderttausend Systeme mit mehr als zehn Prozessoren, die zumeist in Serveranwendungen Einsatz gefunden haben. Der Bedarf für Rechner mit mehr als 100 Prozessorkernen beschränkte sich auf wenige Tausend. Für Systeme mit mehreren 1000 CPUs war der Markt auf einige Dutzend limitiert. Der größte Marktanteil jedoch wurde von Uniprozessor-Systemen mit einigen Hundertmillionen eingenommen.



Die Leistung von Prozessoren verhundertfacht sich innerhalb einer Dekade bzw. steigt sogar um den Faktor 200, wenn LINPACK- bzw. SpecFP-Benchmarks herangezogen werden, wobei die Taktfrequenz von Prozessoren um den Faktor 10 bis 15 pro Dekade gesteigert wird [19]. Dies geht einher mit einer Verhundertfachtung der Speicherkapazität von DRAM innerhalb dieser Zeitspanne [19]. All diese Verbesserungen werden nicht zuletzt durch die Erhöhung der Transistorzahl pro Chip um den Faktor 30 innerhalb von zehn Jahren ermöglicht. Ich halte es für sehr wahrscheinlich, dass Multicore-Systeme schon bald den Markt beherrschen werden. Sie halten schon jetzt Einzug in Desktop PCs und Notebooks, so dass eine Verschiebung in Richtung einer kegelförmigen Pyramidenstruktur zu erwarten ist, und Mehrkernprozessorsysteme den größten Marktanteil der Prozessoren einnehmen werden. Unberücksichtigt sind hierbei noch die zahllosen eingebetteten Systeme z. B. in Mobiltelefonen oder PDAs (*Personal Digital Assistants*), die größtenteils ebenfalls schon heute über mehrere Prozessoren verfügen und deren Bedeutung in Zukunft deutlich zunehmen wird (vgl. Abbildung 2-5).



**Abbildung 2-5: Marktanteile für Parallelcomputer nach [19] für 1998 und eigene Prognose für die kommende Dekade**

Der Weiterentwicklung dieser komplexen Systeme stehen ab einem gewissen Zeitpunkt mehrere potentielle Barrieren im Weg:

- Die Ausbreitungsgeschwindigkeit von elektrischen Signalen setzt eine untere Grenze für Latenzzeiten. Aufgrund der Begrenzung durch die Lichtgeschwindigkeit können Signale innerhalb einer Zeit von 250 ps, dies entspricht einer Taktperiode von 4 GHz, maximal eine Distanz von 7,5 cm (im Vakuum) überwinden. Dieser Wert ist für On-Chip-Verbindungsleitungen mit ca.  $2/3 c$  anzusetzen, also können im besten Falle 5 cm von einem Signal auf einem Chip innerhalb dieser Zeit zurückgelegt werden.
- Eine weitere Barriere besteht in der immensen Leistungsaufnahme komplexer, hochgetakter CMOS-Systeme, die bereits heute schon mehr als  $100\text{W}/\text{cm}^2$  – im Vergleich zu einer Herdplatte mit  $10\text{W}/\text{cm}^2$  – aufnehmen.
- Die Kosten steigen mit jedem neuen Technologieschritt, so dass evtl. die Herstellung von Systemen mit einer Fläche des maximal technisch Machbaren nicht rentabel ist, und so mas-

siv parallele Systeme auf einen Chip aus Kostengründen nicht in der realisierbaren Ausprägung gefertigt werden.

- Bei einer zu erwartenden Komplexität von mehreren Milliarden Transistoren auf einem Chip werden Fehler sowohl beim Design als auch durch die Fertigung immer wahrscheinlicher. Werden nicht geeignete Konzepte zur Fehlertoleranz (vgl. Definition 33) erarbeitet, und integriert so dürfte die geringe Ausbeute (*Yield*) aus kommerzieller Sicht dem Erfolg dieser Systeme im Wege stehen.
- Ungenügende Leistungsreserven der Kommunikationsinfrastruktur (vgl. LITTLES Gesetz [20]) führen dazu, dass bei konkurrierenden Zugriffen und Transfers die systemimmanente Latenz, die durch die maximale Ausbreitungsgeschwindigkeit nach unten begrenzt ist, zusätzlich erhöht wird.
- Die Weiterentwicklung und Optimierung passender Programmiermodelle für massiv parallele Systeme und die gezielte Ausbildung der Softwareentwickler für künftige parallelverarbeitungstaugliche Anwendungen muss noch stärker vorangetrieben werden. Dies gilt für alle Klassen von Software: Programmiersprachen, Compiler, Betriebssysteme und Anwendungen.

Abschließend sei hier auf den bis 2007 schnellsten Supercomputer der Welt, BlueGene/L von IBM verwiesen, der auf massiv paralleler Verarbeitung beruht und 2005 in Betrieb genommen wurde [21]. Er umfasst 65.536 700-MHz-Dual-PowerPC-440-Kerne mit insgesamt 32,8 TByte Hauptspeicher. Die einzelnen Knoten sind in einem dreidimensionalen Torus miteinander verbunden. Seine maximale Systemleistung beträgt 365 TFlops. Die Hauptanwendungsgebiete liegen im Bereich der Erforschung biomolekularer Phänomene, hydrodynamischer Vorgänge und der Genforschung sowie der Simulation nuklearer Waffen. Das gesamte System ist in 64 Schaltschränken mit je 128 Systemplatinen auf über 230 m<sup>2</sup> untergebracht und somit noch weit von einer Ein-Chip-Lösung entfernt. Seine Leistungsaufnahme beträgt ca. 1,2 MW, und die durchschnittliche Betriebszeit bis zu einem Ausfall wird mit etwas mehr als sechs Tagen beziffert. In den folgenden Abschnitten werden Techniken und Ansätze aufgezeigt, die es ermöglichen werden, derart leistungsfähige Rechensysteme in Zukunft auf deutlich weniger Volumen zu integrieren, als es bisher möglich war.

## 2.2 Kernkomponenten eingebetteter paralleler Rechnerarchitekturen

In den folgenden Abschnitten werden wesentliche Bestandteile von eingebetteten Parallelrechnern bzw. Chip-Multiprozessoren (CMPs) erläutert (vgl. Abbildung 2-6). Hierzu zählen die **Verarbeitungseinheiten**, also *Prozessoren* sowie *Hardwarebeschleuniger* jeder Art (vgl. Abschnitt 2.4). Die Art und Anzahl sowie die Verschaltung dieser Einheiten ist stark von der späteren Anwendung abhängig und sollte im Vorfeld durch Simulation und gründliche Analyse eruiert werden. Weiterhin ist der **Speicher** (vgl. Abschnitt 2.5) eine bedeutende Komponente von Chip-Multiprozessoren, der je nach System zusätzlich noch hierarchisch strukturiert werden kann bzw. werden sollte. Die Kommunikationsinfrastruktur des Systems wird durch ein **On-Chip-Netzwerk** (NoC) gebildet. Die eigentliche Funktion des Gesamtsystems wird erst durch das Zusammenspiel aller Einzelkomponenten realisiert, wobei zusätzlich spezifische Software bzw. Algorithmen benötigt werden. Zusammen definieren diese vier Komponenten das System, den eingebetteten Parallelrechner.

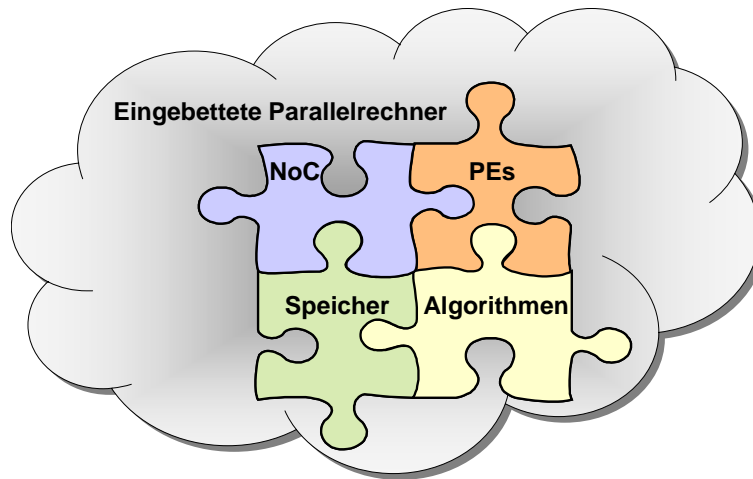


Abbildung 2-6: Kernkomponenten eingebetteter Parallelrechner

Im folgenden Abschnitt wird zunächst auf das Rückgrat eines Chip-Multiprozessors, das On-Chip-Netzwerk, eingegangen.

## 2.3 On-Chip-Netzwerke

Mit dem Fortschreiten der Strukturgrößenminimierung in der Halbleitertechnologie ist es möglich, Milliarden von Transistoren [2] auf einem Chip zu integrieren. Dies erlaubt mehr und mehr Funktionalität auf einem Siliziumträger (*Die*). Neue Systemansätze werden ermöglicht, die ganze Systeme auf einem *Die* realisieren. Man spricht hier von Systems-on-Chips (*SoCs*). Auf einem Baustein können theoretisch Tausende von Modulen, wie Speicher, Verarbeitungseinheiten, Kommunikationsschnittstellen und Mixed-Signal-Elemente, untergebracht werden und parallel arbeiten. Solche SoCs werden schon heute mit einer „Handvoll“ integrierter Module gefertigt und eingesetzt. Besonders Multimedia- und Netzwerkanwendungen profitieren von den neuen Möglichkeiten, die sich aus der Parallelität solcher Bausteine ergeben.

Die Auswahl der richtigen NoC-Topologie hängt entscheidend von den Anforderungen der Anwendung aber auch von den Randbedingungen des Chipdesigns (Preis / Fläche, Leistungsaufnahme, erwartete Leistungsfähigkeit und Flexibilität) ab, vgl. Kapitel 3. Dabei ist zu beachten, dass maßgeschneiderte Netzwerke, die speziell auf den erwarteten Durchsatz und spezifizierte Lastverteilung ausgelegt sind, keine erstrebenswerte Lösung sind, wenn sie keine Flexibilität gegenüber unerwarteten Veränderungen des Datenaufkommens vorhalten. Um eine SoC-Architektur erfolgreich zu realisieren empfiehlt sich ein flexibles und skalierfähiges (vgl. Definition 35) Netzwerk [22].

### 2.3.1 NoC-Topologien

Eine Hauptanforderung für eine ressourceneffiziente Nutzung dieser neuen Möglichkeiten ist eine angemessene On-Chip-Kommunikationsinfrastruktur, über die die einzelnen Module effizient miteinander kommunizieren können. Heutige Systeme verwenden häufig eine „flache“ Bus-, Multiplexer- oder Kreuzschienenverteiler(*Crossbar*)-Topologie (vgl. Abbildung 2-7 a). Solche Verbindungsstrukturen eignen sich jedoch nicht für Strukturen mit Dutzenden oder gar Tausenden von Modulen. Sie benötigen sehr viel Fläche oder weisen eine sehr hohe Latenz auf und skalieren schlecht [23][24]. ZHANG et al. haben in [25] bereits erste hierarchische Verbindungsstrukturen vorgestellt, die die Kopplung heterogener Elemente in SoC-Strukturen unterstützen.

Für SoC-Kommunikationsstrukturen ergeben sich zunächst die gleichen Modelle wie für verteilte Multiprozessorsysteme, die im *Grid-* oder *Clustercomputing* eingesetzt werden. Die Struktur wird von der verwendeten Topologie und dem zugrunde liegenden Graphen geprägt. Beim SoC ist allerdings derzeit noch zu berücksichtigen, dass Modelle bzw. Graphen, die sich gut auf zweidimensionale Strukturen abbilden lassen, von den Gegebenheiten der heutigen Halbleitertechnologie profitieren. Dreidimensionale Ansätze, wie sie häufig in verteilten Computernetzen eingesetzt werden, stellen momentan noch nicht befriedigend lösbare Anforderungen an den Aufbau von Siliziumchips.

**Nomenklatur.** Im Folgenden wird die Notation für NoC-Begrifflichkeiten, in Anlehnung an DALLY [22], eingeführt:

Ein Netzwerk setzt sich aus **Knoten**  $N^*$  und den **Kanälen**  $C$ , die die einzelnen Knoten miteinander verbinden, zusammen. Datenpakete werden durch **Endknoten**  $N$  ins Netzwerk injiziert und von diesen letztendlich auch terminiert. Hierbei ist  $N \subseteq N^*$ . Viele Netzwerke besitzen nur Endknoten und keine Weiterleitungselemente bzw. **Routingknoten**  $N^+ = N^* \setminus N$ . In diesem Fall werden die Endknoten ebenfalls als Knoten referenziert. Jeder Kanal  $c = (x, y) \in C$  des Netzwerks verbindet einen Quellknoten  $x$  mit einem Zielknoten  $y$  mit  $x, y \in N^{*2}$ . Ein **Quellknoten** wird mit  $s_c$  und ein **Zielknoten** mit  $d_c$  bezeichnet. Eine **Kante**  $k$  des Graphen setzt sich somit zusammen aus dem Kanal  $c_{x,y}$  und dem Kanal  $c_{y,x}$  mit  $x, y \in N^*$ .

Ein Kanal  $c = (x, y)$  wiederum lässt sich durch seine **Weite**  $w_c$ , also die Anzahl paralleler Leitungen, seine **Betriebsfrequenz**  $f_c$ , also die Rate, mit der Datenbits auf jeder Leitung transportiert werden können, und seine **Latenz**  $L_c$  bzw.  $t_c$  charakterisieren. Die Latenz kennzeichnet die Zeitspanne, die benötigt wird, um ein Bit von Knoten  $x$  zu  $y$  zu transportieren (vgl. Definition 16). Sie resultiert im Allgemeinen aus der **Kanallänge**  $l_c$  und der charakteristischen **Ausbreitungsgeschwindigkeit**  $v$  zu  $t_c = \frac{l_c}{v}$ . Die **Bandbreite**  $b_c$  eines Kanals resultiert aus der Weite und der Betriebsfrequenz zu  $b_c = w_c \cdot f_c$ .

Jeder Kommunikationsknoten  $x$  hat einen Satz von Kanälen  $C_x = C_{I_x} \cup C_{O_x}$ . Mit  $C_{I_x} = \{c \in C \mid d_c = x\}$  als **Menge der Eingangskanäle** und mit  $C_{O_x} = \{c \in C \mid s_c = x\}$  als **Menge der Ausgangskanäle**. Der **Grad** eines Knotens  $x$  ist  $\delta_x = |C_x|$ , also die Summe aus allen Eingangskanälen  $\delta_{I_x} = |C_{I_x}|$  und Ausgangskanälen  $\delta_{O_x} = |C_{O_x}|$  eines Knotens. Wenn der Grad aller  $x \in N^*$  gleich ist, wird im Folgenden nur von  $\delta$  als Grad gesprochen.

Unter einem **Schnitt** bzw. einer **Teilung**  $C(N_1, N_2)$  wird die Aufteilung eines Netzwerks  $N^*$  in zwei disjunkte Subnetzwerke  $N_1$  und  $N_2$  verstanden. Jeder Kanal  $c \in C(N_1, N_2)$  hat seine Quelle in  $N_1$  und Senke in  $N_2$  und umgekehrt. Eine **Bisektion** oder auch **Halbierung** ist wiederum eine Teilung des gesamten Netzwerks in zwei, wenn möglich, gleich große – im Sinne der Anzahl der Knoten – Subnetze:  $|N_2| \leq |N_1| \leq |N_2| + 1$ . Dies gilt sowohl für die Endknoten als auch für die Routingknoten. Die **Kanalbisektion**  $B_C$  eines Netzwerks bezeichnet die minimal aufzufindende Kanalanzahl aller möglichen Bisektionen des gesamten Netzwerks  $B_C = \min_{\text{Bisektionen}} |C(N_1, N_2)|$ . Somit ergibt sich die **Bi-**

<sup>2</sup> Beim rückläufigen Kanal kehrt sich die Notation entsprechend um, der Quellknoten ist  $y$  und  $x$  wird der Zielknoten.

**sektionsbandbreite** [26] als minimale Bandbreite aller Bisektionen des Netzwerks zu  $B_B = \min_{\text{Bisektionen}} |B(N_1, N_2)|$ . Für Netzwerke mit gleichgearteter Kanalbandbreite  $b$  ergibt sich  $B_B = b \cdot B_C$ . Sie kennzeichnet die minimal verfügbare Bandbreite zwischen den resultierenden zwei Subnetzen, unter Berücksichtigung aller möglichen Teilungen. Die Bisektionsbandbreite ist ein nützliches Maß, um einerseits die benötigten Ressourcen der globalen Verdrahtung eines Netzwerks zu bestimmen und andererseits eine Aussage über die Ausfallsicherheit bzw. Leistungsfähigkeit des NoCs treffen zu können. Sie gibt sozusagen den Flaschenhals der verfügbaren Bandbreite zwischen den Knoten zweier Subnetze an.

Als **Pfad**  $P$  mit  $P = \{c_1, c_2, \dots, c_n\}$  wird eine geordnete Menge von Kanälen  $c$  bezeichnet, die einen Quellknoten  $s_p = s_{c_1}$  mit dem zugehörigen Zielknoten  $d_p = d_{c_n}$  verbindet. Die Länge eines Pfades, gleichbedeutend mit der Anzahl an **Hops**, ist definiert als  $|P|$ . Existiert für alle Quell-Ziel-Verbindungsmöglichkeiten in einem gegebenen Netzwerk mindestens ein Pfad, unter Berücksichtigung des verwendeten Routingalgorithmus, so bezeichnet man das Netzwerk als **verbunden**. Ein **minimaler Pfad** ist gekennzeichnet durch die geringste Anzahl an benötigten Hops bei der Verbindung zweier beliebiger Knoten im Netz. Die Gesamtheit aller minimalen Pfade wird zu  $R_{xy}$  gesetzt.  $H(x, y)$  ist die Anzahl an Hops eines minimalen Pfades zwischen den Knoten  $x$  und  $y$ . Der **Durchmesser**  $D$  des Netzwerks ist somit bestimmt durch die maximale Anzahl an Hops über alle minimalen Pfade zwischen allen Endknoten  $H_{\max} = \max_{x, y \in N} H(x, y)$  (vgl. [22]). Für ein komplett verbundenes Netzwerk mit  $N$  Endknoten, die über Kopplungselemente mit dem Ausgangsgrad  $\delta_o$  verbunden sind, ergibt sich somit eine untere Grenze für die **maximale Hopanzahl** von  $H_{\max} \geq \log_{\delta_o} N$ . Netzwerke, die sich dieser unteren Schranke nähern, wie z. B. Butterfly-Netzwerke, offerieren keine alternativen Wege, bei ihnen sind die Routen festgelegt. Dies ist bezüglich Ausfallsicherheit und dynamischer Anpassung an Lastverteilungsvariationen ein großer Nachteil (vgl. [22]). Die **durchschnittliche minimale Anzahl an Hops** eines Netzwerks zwischen allen Quell- und Zielknoten ist definiert zu:  $H_{\min} = \frac{1}{N^2} \sum_{x, y \in N} H(x, y)$ .

Viele NoC-Implementierungen sehen außer den minimalen Pfaden zusätzlich noch alternative Routen vor. Es sind aufgrund von dynamischen Wegewahlverfahren und aufgrund spezifischer Realisierungen Szenarien denkbar, in denen es von Vorteil ist, minimale Pfade nicht zu wählen. Für diese Netzwerke definiert sich die **durchschnittliche Hopanzahl**  $H_{\emptyset}$  nicht über die minimalen Pfade bzw. die minimale Anzahl von Hops  $H_{\min}$ , sondern über die verwendeten Pfade im Netzwerk  $H_{\emptyset} \geq H_{\min}$ .

Die physikalische **Distanz** eines Pfades wird beschrieben durch:  $D(P) = \sum_{c \in P} l_c$ , wobei die Verzögerung eines Pfades mit  $t(P) = \frac{D(P)}{v}$  angegeben wird. Die **Konstanz der Kantenlänge** ist ein Parameter, der speziell für die ASIC-Implementierung von Bedeutung ist, da sich unterschiedliche Signallaufzeiten aufgrund variierender Leitungslängen negativ auf die Performanz eines On-Chip-Netzwerks und somit auf das gesamte System auswirken.

Betrachtet man nun die gesamte Übertragungsstrecke, also sowohl den Pfad mit den betreffenden Kanten als auch die involvierten Knoten, so ergibt sich eine Gesamtlatenz von:  $t_{gesamt} = \sum_{i \in P} t_{c_i} + t_{k_i}$

(2.3), wobei  $\sum_{i \in P} t_{c_i}$  die Latenz der benutzten Kanäle beziffert und  $\sum_{i \in P} t_{k_i}$  die Latenz innerhalb der Routing- bzw. Endknoten darstellt.

**Symmetrie** ist für ein Netzwerk ein entscheidendes Kriterium. So ermöglicht eine symmetrische Struktur alternative Routen für universelle Lastverteilungen. Der Wegewahlalgorithmus muss nicht auf spezielle Eigenarten der Struktur angepasst werden, sondern ist in allen Knoten des Netzwerks, je nach Symmetriefform, relativ identisch anzuwenden.

In der Literatur wird zwischen **direkten** und **indirekten Netzwerken** unterschieden. Ein Netzwerk-knoten  $x \in N^*$  kann, wie bereits erwähnt ein Endknoten  $x \in N$  sein, der sowohl Daten versendet als auch empfängt, oder aber nur, wie im Falle eines Routingknotens  $x \in N^+$ , die Daten weiterleitet. Bei einem direkten Netzwerk vereint jeder Knoten beide Eigenschaften in sich: Er fungiert sowohl als Endknoten zur Datenverarbeitung als auch als Routingknoten zur Datenweiterleitung. Bei einem indirekten Netzwerk hingegen wird zwischen den End- und den Weiterleitungsknoten strikt getrennt. Dies ist z. B. beim Butterfly-Netzwerk der Fall (vgl. Abbildung 2-8). Im direkten Netzwerk findet der Datenverkehr zwischen den Endknoten „direkt“ statt, wohingegen beim indirekten Netzwerk der Datentransport immer über zwischengeschaltete Routingknoten laufen muss. Es gibt auch hybride Formen, die sowohl direkte als auch indirekte Subnetzwerke beinhalten. Jedes direkte Netzwerk lässt sich zudem in ein indirektes transformieren, indem der Endknoten in Verarbeitungs- und Weiterleitungseinheit aufgeteilt wird.

**NoC-Charakteristika.** Tabelle 2-1 gibt einen Überblick über die zuvor definierten Eigenschaften verschiedener Netztopologien bzw. Graphen für Computernetze im Allgemeinen ebenso wie für On-Chip-Netzwerke. Die Konstanz der Kantenlänge berücksichtigt die Möglichkeit einer effizienten Realisierung in Halbleiterbausteinen und die damit einhergehende Länge der Verbindungsleitungen zwischen zwei Knoten. Skalierfähige dreidimensionale Topologien sind heutzutage für eingebettete parallele Rechnerarchitekturen nur äußerst eingeschränkt geeignet.  $m$  stellt eine Hilfsgröße dar und dient als Berechnungsgrundlage zur Bestimmung der Knotenanzahl  $N^*$ , des Durchmessers  $D$ , der Bisektionsweite  $B_C$  sowie des Grades  $\delta$  der jeweiligen Netztopologie.

Tabelle 2-1: Eigenschaften verschiedener Netztopologien

Topologie	Anzahl Knoten $N^*$	Durchmesser $D$	Bisektionsweite $B_C$	Grad der Verbindung $\delta$	Kantenlänge konstant
2-d Gitter	$m^2$	$2(m-1)$	$m$	4	Ja
2-d Torus	$m^2$	$m-1$	$2m$	4	Nein
3-d Gitter	$m^3$	$3(m-1)$	$m^2$	6	Ja
$n$ -d Gitter	$m^n$	$n(m-1)$	$m^{n-1}$	$2n$	für kleine $n$
Binärer Baum	$2^m-1$	$2(m-1)$	1	3	Nein
Hyperbaum	$2^m(2^{m+1}-1)$	$2m$	$2^{m+1}$	6	Nein
Pyramide	$(4m^2-1)/3$	$2 \log m$	$2m$	9	Nein
Butterfly	$(m+1) 2^m$	$2m$	$2m$	4	Nein
Hypercube	$m^2$	$m$	$2^{m-1}$	$m$	Nein

Für eine 2-d-Realisierung auf einem Siliziumchip qualifizieren sich nach Tabelle 2-1 zunächst das 2-d-Gitter, der Torus, ein binärer Baum oder die Butterfly-Topologie. Ließe man die Forderung

nach identischen Signallaufzeiten auf den *Long Lines*<sup>3</sup> außer Acht, so wäre die Bisektionsweite des Baumes von 1 ein Hinderungsgrund für die Realisierung einer solchen Topologie. Entweder würde zum Hauptknoten ein Kommunikationsflaschenhals entstehen, oder es müssten, wie z. B. beim *Fat Tree*, eine uneinheitliche Anzahl von Leitungen verwendet werden. Dies würde wiederum zu Lasten der Homogenität des Schaltungsentwurfs gehen. Butterfly-Strukturen bieten zunächst keine Möglichkeit der Wegewahl. Es gibt jeweils einen festen Weg zum Ziel. Würde diese Option zusätzlich implementiert, müssten zusätzliche Stufen zu Lasten des Durchmessers eingebaut werden. Ein weiterer Nachteil bei Butterfly-Netzwerken ist, dass diese nicht realisiert werden können, ohne Leitungen zu integrieren, die die Hälfte des Durchmessers des Netzwerks ausmachen [22]. Deshalb eignen sich solche Strukturen weniger gut für mittlere und große SoCs. Bei kleineren NoCs wird diese Topologie hingegen nicht zuletzt aufgrund des einfachen Routings häufig eingesetzt. Berücksichtigt man den oben erwähnten Punkt gleichlanger Verbindungsleitungen zwischen den Knoten, so reduziert sich die Auswahl auf das 2-d Gitter.

Abbildung 2-7 zeigt Varianten einfacher On-Chip-Netzwerktopologien, die für sich allein genommen nicht oder nur sehr eingeschränkt skalieren (vgl. Definition 35) und so für eine größere Anzahl Teilnehmer nicht effizient einsetzbar sind.

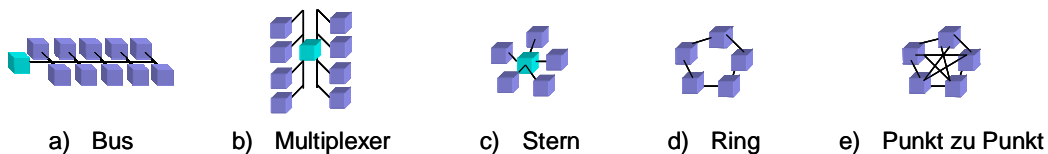


Abbildung 2-7: Varianten einfacher On-Chip-Netzwerke für System-On-Chips

Die in Abbildung 2-8 dargestellten Netzstrukturen stellen bekannte Topologien implementierter On-Chip-Netzwerke dar. Abbildung 2-8 a) und b) zeigen die für die Hardwarerealisierung aufgrund ihrer vorwiegend gleich langen Verbindungen und ihrer relativ einfachen Abbildung auf zweidimensionale Strukturen gut geeigneten Topologien Gitter und Torus. Würfel und Hyperwürfel Abbildung 2-8 c) und d) werden häufiger in konventionellen Computernetzen eingesetzt, ebenso wie Baum- und Butterfly-Topologie.

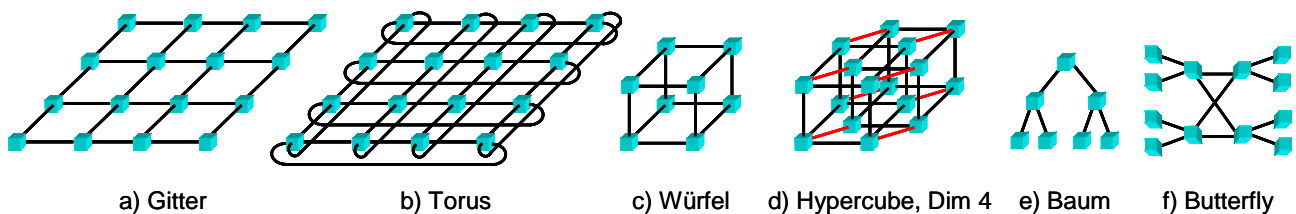


Abbildung 2-8: Topologien etablierter On-Chip-Netzwerke für System-On-Chips

In Abbildung 2-9 werden hierarchische On-Chip-Netzwerke dargestellt. Abbildung 2-9 a) zeigt ein zweistufiges hierarchisches On-Chip-Netzwerk, dessen übergeordnete Hierarchieebene ein regelmäßiges Gitter bildet. Die an den Gitterknoten angeschlossenen Cluster bestehen aus einer variablen Anzahl von Modulen, die über einen lokalen Bus kommunizieren. Abbildung 2-9 b) hingegen stellt ein weniger streng strukturiertes, mehrere Hierarchiestufen umfassendes On-Chip-Netzwerk dar, das weitaus komplexer geartet ist als das aus Abbildung 2-9 a). Für besondere Anwendungsge-

<sup>3</sup> *Long Lines* sind die langen, zumeist intermodularen Verbindungsleitungen eines Chips und werden vorwiegend auf den höheren Metalllagen realisiert.

biete können solche Speziallösungen ggf. sehr effizient sein, allerdings fehlt ihnen evtl. die Flexibilität, um sich auch für andere Anwendungsklassen zu eignen. Ferner sind derartige Strukturen derzeit mit Standard-Technologien noch nicht effizient realisierbar.

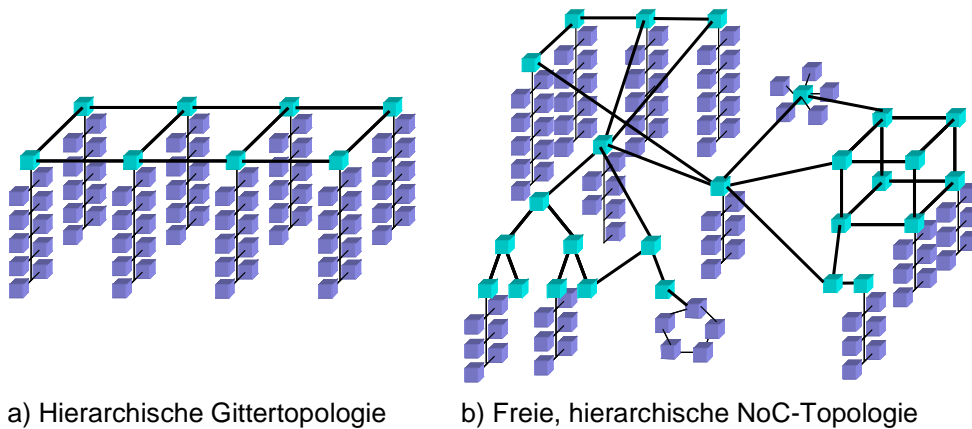


Abbildung 2-9: Topologien von hierarchischen On-Chip-Netzwerken für System-On-Chips

**Vorteile hierarchischer On-Chip-Netzwerke.** Im Folgenden werden die Vorteile von hierarchischen On-Chip-Netzwerken für System-on-Chips vorgestellt und diskutiert. Hierarchische On-Chip-Netzwerke helfen, die Performanz des Gesamtsystems zu erhöhen. Aufgrund der strukturierten hierarchischen Verbindung der einzelnen Module sind verteilte Routingentscheidungen möglich. Aufgrund dieser skalierbaren Dezentralität wird massiv parallele Hardware besser unterstützt. Daraus resultieren geringere zeitliche Arbitrierungsverluste verglichen mit zentralen Kommunikationsstrukturen wie z. B. Bussen oder Multiplexern. Es ergeben sich geringere Latenzen, und je nach Realisierung werden weniger Pufferspeicher benötigt.

NoCs können eine stärkere Entkopplung der einzelnen Verarbeitungseinheiten ermöglichen und erlauben so z. B. unterschiedliche Taktomänen mit lokal höheren Taktraten. In diesem Zusammenhang spricht man auch häufig von global asynchronen, lokal synchronen (*GALS*) Ansätzen [27] [28], die einen der Forschungsschwerpunkte im NoC-Bereich bilden. Aufgrund der begrenzten Ausbreitungsgeschwindigkeit und der immer höheren Taktraten bei gleichzeitiger Verringerung der Strukturgrößen und Reduktion der Versorgungsspannung ist es mittlerweile bei großen SoCs nicht mehr möglich, Signale innerhalb eines Taktes über den gesamten Durchmesser des Chips zu transportieren. Gatter-Verzögerungszeiten skalieren mit der Technologie, Leitungsverzögerungszeiten hingegen steigen nahezu exponentiell oder zumindest linear unter Zuhilfenahme von *Repeatern* [29] an.

Nach [30][31] und [32] werden über 80 % der kritischen Pfade in *ULSI* (*Ultra Large Scale Integration*)-Schaltungen durch Verbindungsleitungen bestimmt. Deshalb müssen Systementwürfe auch Kommunikationsnetze [33] und verteiltes Rechnen mit berücksichtigen. Hierarchische NoCs helfen bei der Minimierung der Anzahl von *global wires / long lines*, also der globalen und damit relativ langsamen Verbindungsleitungen des Chips. Hierdurch wird u. a. auch der *Clock Skew*, die Takt-Varianz, verringert und eine höhere Signalintegrität erreicht. Dies wiederum erhöht die Ausfallsicherheit. Zusätzlich können weniger leistungsstarke Treiberstufen eingesetzt werden, wodurch die Leistungsaufnahme deutlich reduziert wird.

Durch definierte Schnittstellen des NoCs und die Möglichkeit der Kapselung von IP-Blöcken durch so genannte „Wrapper“ (vgl. Abbildung 4-19) lassen sich Hardwarekomponenten einheitlicher integrieren. Aufgrund dieser Integrationsmöglichkeit wird eine Erhöhung der Wiederverwendbarkeit



des Systemkonzepts basierend auf Hardware-/Software-Bibliothekselementen für nachfolgende Projekte erhöht. Parametrisierbare NoCs können leicht auf neue Anforderungen bezüglich Bandbreite oder Energie-/Flächenbedarf angepasst werden. Durch die Wiederverwendbarkeit erreicht man eine kürzere „Time-to-Market-Spanne“ sowie geringere *NRE(Non-recurring Engineering)*-Kosten, d.h. die einmalig entstehenden Kosten für neue Designs werden aufgrund wiederverwertbarer Bestandteile reduziert. Ebenso verringert sich das Entwurfsrisiko, da auf bestehende Strukturen zurückgegriffen werden kann; zudem sinkt der Testaufwand. Ferner kann so die „Time-to-Volume-Spanne“ aufgrund des geringeren Entwurfsaufwandes reduziert werden.

Hierarchische NoCs skalieren im Gegensatz zu Bussen, Multiplexern oder Kreuzschienenverteilern und ermöglichen so eine deutlich größere Anzahl von Teilnehmern. Bei größeren Systemen erzielt man eine deutliche Flächensparnis im Vergleich zu Bussystemen oder Kreuzschienenverteilern. Die Vielseitigkeit hierarchischer NoCs ermöglicht einen größtmöglichen Einsatz in unterschiedlichsten Systemimplementierungen. Für die System- bzw. Anwendungssoftware ergibt sich eine deutlich bessere Portierbarkeit auf zukünftige Systeme, da sie im Normalfall entkoppelt von der System-Topologie implementiert werden kann.

Für Systeme mit wenigen Teilnehmern eignen sich hierarchische NoCs in der Regel weniger, da im Allgemeinen ein deutlich größerer Flächenaufwand durch zusätzliche, meist unbenötigte Leistungsmerkmale, zu verzeichnen ist, es sei denn, das NoC offeriert zusätzlich eine angepasste Variante für Systeme geringerer Komplexität (vgl. Abschnitt 4.2).

### 2.3.2 Organisation von On-Chip-Kommunikation

In diesem Abschnitt wird die Organisation von On-Chip-Kommunikationsprotokollen im Allgemeinen und im Hinblick auf das entworfene GigaNoC (vgl. Abschnitt 4.2) vorgestellt.

**Schichtenmodell.** Eigenschaften von NoC-Architekturen definieren sich zum einen aus der Topologie des On-Chip-Netzwerks und zum anderen aus dem verwendeten Protokoll oder, im Falle von heterogenen oder hierarchischen NoCs, den verwendeten Protokollen. NoC-Protokolle sind, in Anlehnung an das *ISO/OSI(International Organization for Standardization / Open Systems Interconnection)*-Referenzmodell [34] typischerweise schichtenartig [35] organisiert (vgl. auch DIN ISO 7498). Dieses Referenzmodell vereinheitlicht und regelt den Transport von Daten in Kommunikationsmedien (vgl. Abbildung 2-10). So werden die verschiedenen Anwendungsbereiche der Kommunikation in Schichten unterteilt, die in sich geschlossen und unabhängig voneinander abgearbeitet werden können. Jede Schicht erfüllt eine definierte Funktionalität. Auf Sender- und Empfängerseite des Kommunikationskanals existieren identische Schichten, zwischen denen logische Verbindungen aufgebaut werden. Die Art und Weise, wie die Daten transportiert werden, ist in einem Protokoll festgelegt, das beide Teilnehmer beherrschen. Innerhalb eines Gerätes erfolgt die physikalische Datenweitergabe in vertikaler Weise, wobei beim Sender Paketköpfe (*Header*) hinzugefügt und beim Empfänger wieder entfernt, also Protokolle „terminiert“ werden. Die sender-/empfängerseitigen Schichten tauschen sich horizontal über diese Protokolle aus. Der Grad der Abstraktion wird in Richtung der höheren Schichten immer größer. Der untersten Schicht ist die physikalische Übertragung der einzelnen Bits zugewiesen, während die oberste Schicht mit der initiierten Anwendung kommuniziert und ggf. Anwenderinteraktion einbezieht.

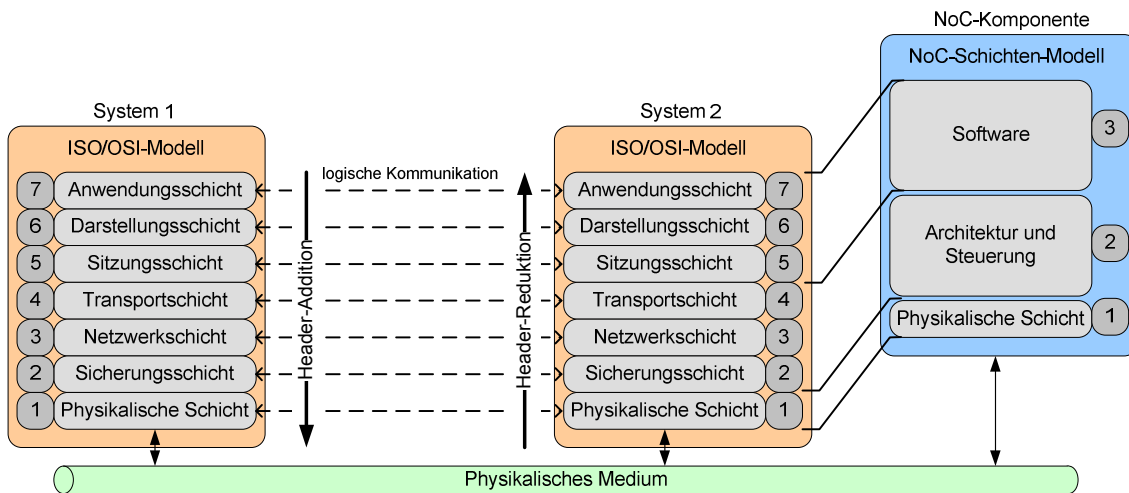


Abbildung 2-10: Schichtenmodell für Netzwerk-Kommunikationskomponenten

Beim ISO/OSI-Modell sind sieben Schichten vorgesehen, die sich üblicherweise bei On-Chip-Netzwerken auf drei zusammenfassen lassen [35]. Zur detaillierten Definition der einzelnen Funktionen der sieben Schichten des ISO/OSI-Referenzmodells sei auf die einschlägige Literatur [34] verwiesen. Im Folgenden werden die drei Schichten des NoC-Referenzmodells kurz vorgestellt: physikalische Schicht, Architektur- und Steuerungsschicht, Software-Schicht.

Die *physikalische Schicht* umfasst die technische Realisierung der Kommunikationskanäle, also Verdrahtung sowie Treiber- und Empfängerstufen, aber auch Datenpuffer zur Zwischenspeicherung der Signale.

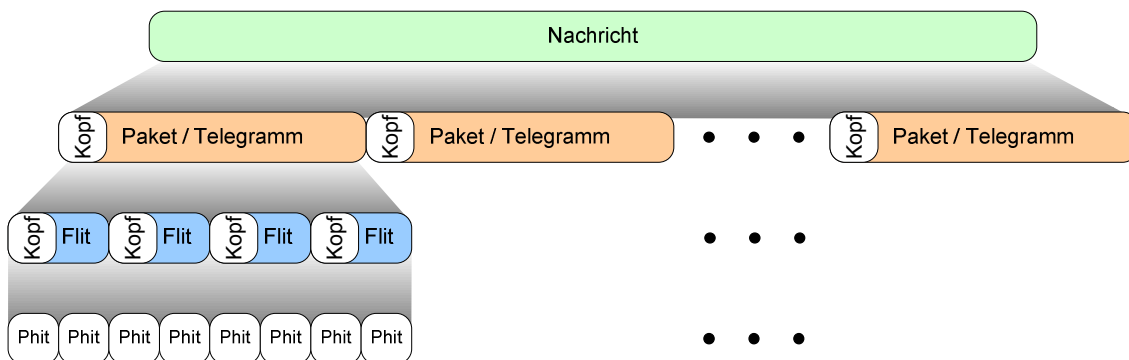


Abbildung 2-11: Die zu versendende Nachricht dargestellt in Stufen der Paketverarbeitung

Die *Architektur- und Steuerungsschicht* vereint in sich die ursprünglichen Schichten zwei bis vier des ISO/OSI-Referenzmodells. Die enthaltene Sicherungsschicht (*Data-Link-Layer*) ist für die zuverlässige Übertragung der Daten, z. B. durch Hinzufügen von Redundanz für fehlererkennende bzw. sogar fehlerkorrigierende Codes zuständig. Sie regelt außerdem den Zugriff auf das Kommunikationsmedium und teilt die Daten, falls vorgesehen, in Blöcke auf. Die Aufteilung von Daten-Telegrammen in einzelne Pakete ist besonders für die Vermeidung und Behebung von Übertragungsfehlern von Vorteil, da die Absicherung der Daten hierbei durch etablierte Prüfsummenverfahren, z. B. zyklische Codes (CRC-Prüfsummen) blockweise geschehen kann. Abbildung 2-11 zeigt die logischen bzw. physikalischen Formen einer zu übertragenden Nachricht auf. Die *Nachricht* kann in ein oder mehrere *Pakete* bzw. *Telegramme* aufgeteilt werden, denen Informationen in Form von Paket-Köpfen, auch Paket-Headern genannt, hinzugefügt werden. Die einzelnen Pakete wiederum werden bei vielen NoCs nochmals aufgeteilt in Flits (*Flow Control Digits*), die atomaren Übertragungseinheiten mit Paketeigenschaften. Auch die Flits bekommen einen zusätzlichen Kopf,

der in den meisten Fällen deutlich kleiner als bei den Paketen ausfällt und häufig nur wenige Bytes beansprucht. Schließlich kann es noch, je nach Implementierung, vorkommen, dass diese Flits, oder aber auch die Pakete direkt in *Phits* (*Physical Units*) übertragen werden. Unter einem *Phit* versteht man die Menge an Bits, die innerhalb eines Taktes über die Verbindung, auch Link genannt, zur Gegenstelle der Übertragungsleitung transferiert werden können. *Phits* haben üblicherweise keine eigenen Köpfe, sondern werden ggf. mit *Handshake*-Mechanismen kontrolliert.

Weiterhin sind in der Architektur- und Steuerungsschicht Mechanismen enthalten, die der Netzwerkschicht (*Network Layer*) zuzuordnen sind. Dies betrifft das Schalten der Verbindungen (*Switching*) für den Datentransport und letztendlich das Weiterleiten und die Wegsuche (*Routing*) der Pakete durch das Netzwerk.

**Switching-Verfahren.** Switching bezeichnet im Hinblick auf die technische Realisierung die Art und Weise, wie Daten zum nächsten Knoten weitergeleitet werden. Zu den bekanntesten **Switching-Verfahren** zählen u. a. das *Circuit Switching* (CS), das *Packet- bzw. Store-And-Forward* (SAF)-Switching, das *Virtual-Cut-Through* (VCT)-, das *Wormhole* (WH)- und das *Mad-Postman*-Switching.

Im Folgenden werden die oben genannten Switching-Verfahren kurz bewertet:

- CS reserviert vor der eigentlichen Übertragung den gesamten Pfad vom Quellknoten bis zum Zielknoten der Übertragung. Hierzu wird häufig ein „Routing Header Flit“ zum Zielknoten geschickt, das den Pfad durchläuft. Nach erfolgreicher Übertragung wird eine Bestätigung zum Quellknoten versendet und die Verbindung zwischen den beiden Teilnehmern ist exklusiv eingerichtet. Somit setzt sich die Zeit  $t_{CS}$  aus der Zeit des Verbindungsaufbaus und der eigentlichen Übertragung zusammen:  $t_{CS} = t_{\text{Verbindungsaufbau}} + t_{\text{Daten}}$ . CS ist empfehlenswert, wenn Pakete relativ selten zwischen den Teilnehmern versendet werden und die Datenmengen relativ umfangreich sind. Dies relativiert den zusätzlichen Aufwand des Verbindungsaufbaus. Ein großer Nachteil ist die Blockade von anderen Paketen, die für die Zeit der Reservierung keine Verbindung aufbauen können. Speziell bei stark frequentierten Übertragungswegen und relativ geringen Datenmengen, also z. B. für Anwendungsszenarien von Single-Chip-Multiprozessoren, empfiehlt es sich meistens, das *Packet-Switching*-Verfahren vorzuziehen (vgl. [26]).
- SAF bzw. *Packet-Switching* benötigt eine große Menge an Pufferspeicher in den einzelnen Kommunikationsknoten, da mindestens ein Paket zwischengespeichert werden muss. Zusätzliche Latenz wird hinzugefügt, allerdings erlaubt dieser Switching-Algorithmus komplexe Routingverfahren, da für jeden Knoten Einblick in den gesamten Inhalt des Pakets besteht. Ein Datentelegramm wird ggf. in mehrere Pakete aufgeteilt und ohne eine feste Reservierung des Kanals zu benötigen, transportiert. Jedes einzelne Paket kann gegebenenfalls über einen anderen Weg zum Ziel geleitet werden. *Packet-Switching* ist besonders bei häufigem Datenverkehr mit relativ geringem Datenvolumen von Vorteil. Nachteilig kann sich die etwaige Aufteilung des Telegramms in Pakete aufgrund des damit verbundenen Overheads (z. B. redundante Headerinformationen in jedem Flit etc.) auswirken.
- VCT-Switching reduziert die absolut benötigte Speichermenge in den Routingknoten und verringert ggf. die Latenzzeiten, falls das Netzwerk nicht blockiert sein sollte. Ist die Gegenstelle allerdings nicht verfügbar, so muss auch hier im schlimmsten Fall das gesamte Paket

innerhalb eines Knotens zwischengespeichert werden können. Sind die Verbindungen nicht belegt, so wird bereits nach dem Empfang des Headers die Nachricht zum nächsten Knoten weitergeleitet. Dieser Mechanismus wird als „*Virtual Cut-Through*“-Switching bezeichnet. Man bezeichnet diese Art der Übertragung dann auch als Pipelining der Nachricht, da sich die einzelnen Segmente in aufeinanderfolgenden Routern befinden. Aufgrund der möglichen Blockierung der Paketentitäten verhält sich das *VCT*-Switching bei Überlastung des Netzes wie das oben erwähnte Packet-Switching.

- Beim *WH*-Switching wird das Datentelegramm nicht nur in Pakete, sondern noch weiter in die oben genannten *Flits* unterteilt. Das Header-Flit enthält die Informationen, die für das Routing benötigt werden, alle weiteren Flits des Pakets folgen ihm über die gleichen Wege durchs Netz. Auch hier findet ein Pipelining der Nachrichtenübertragung statt. Vorteile bei diesem Verfahren sind die geringere Speichermenge, die benötigt wird, da nur wenige Flits pro Router gespeichert werden müssen. Weiterhin zeichnet sich *WH*-Switching durch die deutlich kleineren *SAF*-Latenzzeiten bei der Übertragung aus. Nachteilig kann sich die Blockierung (*Blocking*) eines Übertragungsweges durch eine Flit-Kette erweisen, wenn hier nicht durch Soft- oder Hardware Abhilfe geschaffen wird. Die Eigenschaften, die das *WH*-Verfahren bietet, ermöglichen die Realisierung von flächenextensiven, kompakten und schnellen Routern (vgl. [26]). Die Latenz ist die gleiche wie beim *VCT*-Switching.
- Eine Weiterentwicklung des *VCT*-Switchings in Kombination mit dem *WH*-Switching ist das „*Mad Postman*“-Switching. Bei diesem wird versucht, die Latenz der Paketsegmente nochmals zu minimieren. Ein eintreffendes Flit wird bereits während des Empfangs spekulativ zum gegenüberliegenden Ausgang weitergeleitet. Speziell in 2D-Gittern kann eine solche Strategie die Performanz steigern, da hier häufig die Richtung eines Flits gleich bleibt. Das *MP*-Switching eignet sich vor allem für bit-serielle Übertragungen, und wenn die physikalische Übertragung innerhalb des Netzwerks so geartet ist, dass sich ein Flit mehrere Zyklen auf der Übertragungsleitung befindet. Handelt es sich bei dem zugrunde liegenden Netzwerk hingegen um eine Infrastruktur, die in der Lage ist, relativ breite Flits innerhalb eines Zyklus zum nächsten Knoten zu transportieren, so bietet dieses Verfahren nahezu keinen Vorteil gegenüber dem *VCT*- und dem *WH*-Switching (vgl. [26]).

Die Methode der **virtuellen Kanäle** (*Virtual Channels / VC*) ermöglicht es einem NoC, einen einzigen physikalischen Kanal, z. B. einen Inter-Switch-Box-Link (vgl. 4.2.1), für mehrere Datenströme zu multiplexen und somit  $n$  virtuelle Kanäle zur Verfügung zu stellen. Dazu werden mehrere Puffer für einen physikalischen Kanal benötigt. Diese Methode erhöht den Ressourcenbedarf, kann jedoch zur Latenzminimierung einzelner Paketklassen herangezogen werden und steigert ggf. den gesamten Netzwerkdurchsatz. Virtuelle Kanäle ermöglichen u. a. auch den Einsatz von Verfahren zur Bandbreitenkontrolle und damit *Quality-of-Service(QoS)*-Mechanismen. Der Einsatz einer Vielzahl von virtuellen Kanälen kann jedoch die Latenz, die ein Paket innerhalb eines Knotens erfährt, aufgrund der Entscheidungsfindung über die Prioritätseinstufung erhöhen.

Durch die Einführung von virtuellen Kanälen konnten so genannte *hybride Switchingverfahren* entwickelt werden. Zu diesen Techniken zählen u. a. das *Buffered Wormhole Switching*, *Pipelined Circuit Switching* sowie das *Scouting Switching*. Diese Verfahren sind im Gegensatz zu den oben genannten *optimistischen* Switching-Algorithmen als *konservativ* einzustufen und verbessern besonders die Eignung des Netzwerks im Hinblick auf Fehlertoleranz.

Zusammenfassend ist zu sagen, dass Switching-Verfahren, neben Topologie, Routingverfahren und Flußkontrollmechanismen, einen großen Einfluss auf die Leistungsfähigkeit des Netzwerks haben. Das Switching-Verfahren bestimmt nicht zuletzt den Aufbau und damit auch die benötigten Ressourcen des einzelnen Routingknotens. Wormhole-Switching ist das derzeit am häufigsten verwendete Verfahren bei parallelen Rechnerarchitekturen [26]. Es wurde bereits 1986 vorgestellt [36][37] und ermöglicht kleine und zugleich schnelle Routingknoten, die in der Lage sind, Nachrichten jeder Länge effizient zu übertragen.

**Wegwahl-/Routingstrategien.** *Routing* bezeichnet die Art und Weise, wie die Wegwahl der Nachrichtenströme in Netzwerken entschieden wird. Bei paketvermittelten Datennetzen ist prinzipiell zwischen *Routing* und *Forwarding* zu unterscheiden: Während das *Routing* die komplette Wahl des Weges durch das Netzwerk bestimmt, entscheidet das *Forwarding* nur über den zu wählenden Nachbarknoten, über den die Nachricht weitergeleitet werden soll.

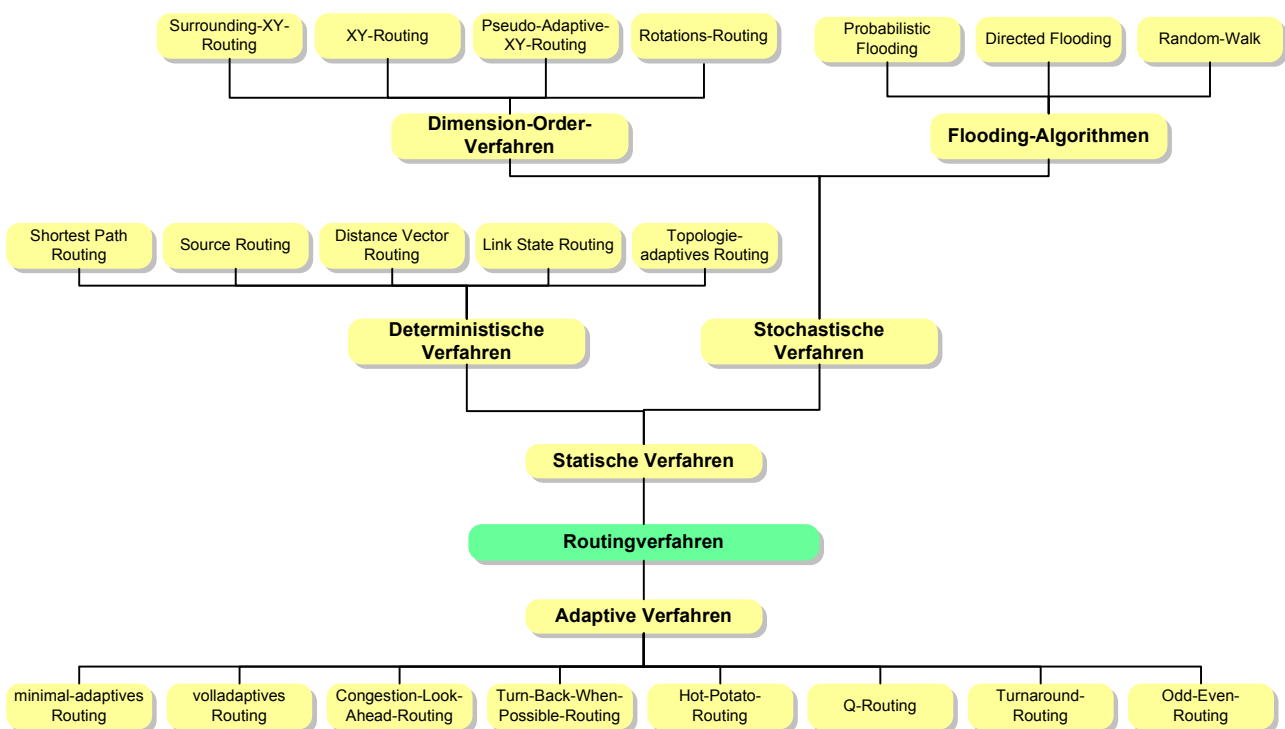


Abbildung 2-12: Übersicht über etablierte Routingmechanismen

Beim Routing kann zwischen *adaptiven* und *gedächtnislosen* bzw. *statischen* Routingverfahren unterschieden werden. Während adaptive Verfahren auf Auslastung (*Contention*) und Blockade bzw. Überlastung (*Congestion*) der einzelnen Kommunikationswege reagieren können und die Wegwahl von solchen und anderen Parametern abhängig machen, nehmen statische bzw. gedächtnislose Verfahren (auch „Oblivious“ Routing genannt) keine Rücksicht auf die derzeitigen Verkehrsverhältnisse des Netzwerks. Beim statischen / gedächtnislosen Routing wird zwischen *deterministischen* und *stochastischen* Verfahren unterschieden. Während deterministische Verfahren immer zur gleichen Entscheidung bezüglich des Pfades führen, wird die Wegwahl beim stochastischen Routing zufallsbasiert entschieden. Abbildung 2-12 gibt einen Überblick über etablierte Routingmechanismen, die im Folgenden näher diskutiert werden.

Gerade für nicht vorhersehbares Kommunikationsaufkommen sind zwar adaptive Verfahren zu empfehlen, sie benötigen jedoch für die Realisierung in NoC-Komponenten meist deutlich mehr Ressourcen bzw. Fläche. Die nächsthöhere Aufgabe, die von Architektur- und Steuerungsschicht

übernommen wird und im Allgemeinen der Transportschicht zuzuschreiben ist, ist die Segmentierung von Nachrichten in Pakete und ggf. in Flits. Die Schichten, die oberhalb dieser Schicht angesiedelt sind, brauchen die Eigenschaften des Kommunikationsnetzes nicht in Betracht zu ziehen, da auf der Transportschicht eine vollständige „Ende-zu-Ende“-Kommunikation stattfindet. Aufgrund der häufig recht hohen Empfindlichkeit eines Netzes bezüglich der verwendeten Paketgrößen stellt die richtige Segmentierung oft eine Herausforderung dar. Dies beinhaltet sowohl Aspekte der Performanz als auch der Leistungsaufnahme und der benötigten Fläche.

Probleme, die sowohl beim gedächtnislosen als auch beim adaptiven Routing auftreten, sind: *Deadlock*, *Livelock* und *Starvation*. Beim *Deadlock* blockieren sich zwei Pakete gegenseitig im Routingknoten aufgrund der Belegung von ausschließlichen Ressourcen. Dies macht ein Weiterleiten der jeweiligen Daten für den Router unmöglich. Beim *Livelock* zirkuliert das Paket unendlich im Netz, ohne seinen Zielknoten zu erreichen. Dieses Phänomen tritt bei nicht-minimalen Routing-Algorithmen auf. *Livelock* beeinträchtigt die Leistungsfähigkeit des Netzwerks und sollte durch geeignete Maßnahmen, wie z. B. Reduzierung der Lebensdauer („*Time To Live*“-Parameter) verhindert werden. *Starvation* bezeichnet die Situation, wenn das Netzwerk kontinuierlich Daten hoher Priorität transportieren muss und Pakete geringer Priorität aufgrund der vorherrschenden Sättigung des Netzes nicht weitergeleitet werden können.

Im Folgenden werden einige relevante Routingstrategien näher vorgestellt.

**Gedächtnisloses / statisches Routing.** *Dimension-Order-Routing*-Verfahren leiten ein Paket dimensionsweise ans Ziel. D. h. es wird der kürzeste Weg zum Zielknoten bestimmt, dann werden die Kanten einer Dimension nach der jeweiligen Vorschrift des Routingalgorithmus reduziert, bis letztendlich das Paket beim Zielknoten angekommen ist. Im Allgemeinen haben *Dimension-Order*-Verfahren keine vorteilhaften „Load-Balancing“- also Lastverteilungseigenschaften, sind allerdings sehr leicht und mit relativ überschaubarem Hardwareaufwand zu realisieren.

Das *XY-Routingverfahren* zählt zu einem der bekanntesten *Dimension-Order*-Verfahren. Es eignet sich für zweidimensionale Topologien wie Gitter oder Tori. Hierbei werden die Pakete zunächst horizontal (also in X-Richtung) und anschließend vertikal (in Y-Richtung) bis zum Zielknoten weitergeleitet. Bezüglich der Lastverteilung entsteht in der Mitte des Netzwerks die größte Belastung, welche durch geeignete, überlagerte Algorithmen nivelliert werden kann. Bedeutender Vorteil beim XY-Routing ist die Tatsache, dass keine *Livelocks* bzw. *Deadlocks* in mehr als einer Dimension auftreten [38].

Das *Pseudo-Adaptive-XY-Routing* arbeitet in zwei unterschiedlichen Modi, je nach Auslastung des Netzwerks. Bei geringer Auslastung kommt die deterministische Variante zum Tragen. Treten gehäuft Blockierungen des Netzes auf, so wechselt das Verfahren auf den adaptiven Modus, in dem weniger ausgelastete Verbindungen für die Wegewahl ausgesucht werden. Dieses Verfahren wird vornehmlich bei zweidimensionalen Gittern eingesetzt, bei denen die Router über 5 Ports verfügen, wobei vier nach Norden, Osten, Süden und Westen gerichtet sind, und der fünfte den lokalen Anschluss anbindet. Das *Pseudo-Adaptive-XY-Routingverfahren* versucht im Gegensatz zum herkömmlichen XY-Routing die Mitte des Netzwerks zu entlasten und eine Gleichverteilung des Datenverkehrs herzustellen.

Beim *Surrounding-XY-Routing* besteht die Möglichkeit, Knoten als blockiert zu kennzeichnen und somit den Routingalgorithmus zur Wahl einer „Umleitung“ zu veranlassen. In diesem Fall unter-

scheidet man die Ausweichmodi *SH-XY*- und *SV-XY-Routing*. Bei der ersten Variante wird die alternative Wegstrecke in der Horizontalen gesucht, und bei Blockierung in vertikaler Richtung findet das Paket auf einer vertikalen Ausweichroute zum Ziel.

Bei den *Rotations-Routingverfahren* wird vorgeschrieben, welche Richtungswechsel ein Paket bei der Wegwahl erfahren darf. Hier gibt es zahlreiche Implementierungen wie z. B. das *North-last-Routing* oder das *West-first-Routing*, wobei der Name schon die Vorschrift verrät.

**Deterministische Routing-Algorithmen** verwenden immer den gleichen Pfad, ohne die Auslastung der betreffenden Kanäle zu berücksichtigen. In blockadefreien Netzen zeichnen sie sich durch kurze Latenzzeiten und ihre Zuverlässigkeit aus. Deshalb sind sie besonders für Echtzeit-Anwendungen geeignet. Pakete treffen in der Reihenfolge ein, in der sie versendet worden sind, so dass das Problem des „*Packet Reordering*“ nicht auftreten kann. Im einfachsten Fall hat ein Router eine fixe Routingtabelle, die alle Routen zu seinen Nachbarknoten beinhaltet.

Mit Hilfe dieses routingtabellenbasierten Ansatzes lassen sich verschiedene Formen des deterministischen Routings realisieren. Eines der bekanntesten Verfahren ist das „*Shortest Path Routing*“. Hierbei werden die Pakete entlang des kürzesten Pfades geleitet. Zu den Varianten des *Shortest-Path-Routings* zählen das *Distance Vector Routing* und das *Link State Routing*. Beim *Distance Vector Routing* beinhaltet die Routingtabelle des jeweiligen Knotens Informationen über die Konnektivität der Nachbarschaft. Pakete werden über die jeweils kürzeste Verbindung zum Ziel geleitet. Beim *Link State Routing* wird die Routingtabelle zwischen allen Routern ausgetauscht bzw. geteilt. In SoCs werden die Routingtabellen schon während der Produktion voreingestellt und nur in Sonderfällen, z. B. bei Ausfällen während des Betriebs, umprogrammiert. Beim *Source Routing* werden bereits beim Quellknoten alle Entscheidungen bezüglich der Wegwahl getroffen. Das *Topology-adaptive Routing* erweitert das deterministische Routingverfahren um die Eigenschaft, auf Veränderungen der Topologie einzugehen, indem die Inhalte der Routingtabellen angepasst werden können. Man spricht hierbei auch von *Online Oblivious Routing*.

**Stochastische Routingverfahren** sind zum einen einfach zu realisieren und fehlertolerant, zum anderen brauchen sie mehr Netzwerkressourcen, als notwendig wäre. Für diese Algorithmen ist eine Begrenzung der Lebenszeit von Paketen unabdingbar, da es sonst zu dem Problem der *Starvation* kommen könnte. Zu den bekanntesten stochastischen Routingverfahren gehören die so genannten *Flooding-Algorithmen*. Bei der einfachsten Implementierung, dem *Probabilistic Flooding*, wird ein Paket zu allen Verbindungen weitergeleitet. Es findet somit eine „Überflutung“ des Netzwerks statt. Sobald eine Kopie den Zielknoten erreicht, werden alle weiteren Kopien beim Eintreffen am Zielknoten gelöscht. Eine Weiterentwicklung spiegelt der *Directed-Flood-Algorithmus* wieder. Hierbei werden die Pakete in die ungefähre Richtung des Zielknotens ins Netz injiziert, somit weniger Netzwerkressourcen benötigt. Als weitere Einschränkung dieses Algorithmus ist das *Random-Walk-Verfahren* zu nennen. Hier wird jeweils nur eine definierte Anzahl von gerichteten Paketen von jedem Routingknoten weitergeleitet. Dies verringert deutlich die beanspruchten Netzwerkressourcen. Um die hohen Belastungen an Datenaufkommen für das Netzwerk zu reduzieren, wurde der *Valiant-Algorithmus* entwickelt. Bei dieser Methode wird die Belastung des Netzes reduziert. Er eignet sich besonders für hoch dimensionierte Netzwerke mit einem großen Grad. Es wird zufällig ein Knoten im Netz ausgewählt, zu dem das Paket zunächst geroutet wird. Anschließend werden von diesem Knoten herkömmliche, gedächtnislose Verfahren zur Wegwahl angewendet.

**Adaptive Routing-Algorithmen** zeichnen sich dadurch aus, dass sie sich dem Zustand des Netzwerks entsprechend ihrer Wegewahl anpassen. Zu diesen Routingverfahren zählen u. a.: Das *minimal adaptive Routing*, das *Hot-Potato-Routing*, das *Q-Routing*, das *volladaptive Routing*, das *Congestion-Look-Ahead-Routing*, die *Turnaround-Routingverfahren*, das *Turn-Back-When-Possible-Routing* und das *Odd-Even-Routing*.

Beim *minimal adaptiven Routing* wird immer versucht, den kürzesten Pfad zu wählen, und bei alternativen Routen entscheidet sich der Algorithmus für die geringer ausgelastete Verbindung. Beim *Hot-Potato-Routing* werden Pakete ohne Aufenthalt weitergeleitet. Sollte eine Verbindung blockiert sein, wird das Paket wahllos in eine andere, freie Richtung geschickt, gleichsam wie eine „heiße Kartoffel“ weitergereicht. Bei einer Aneinanderreihung von blockierten Pfaden kann das Paket komplett in die entgegengesetzte Richtung transportiert werden, man spricht dann auch von „Misrouting“. Die Hardwareressourcen für dieses Verfahren sind bzgl. der Speichermenge relativ gering, da so gut wie kein Pufferspeicher in den Routern eingesetzt wird [39]. Ein Beispiel für ein Routingverfahren, das das Verkehrsaufkommen innerhalb des Netzwerks statistisch auswertet und die Wegewahl basierend auf den resultierenden Ergebnissen trifft, ist das *Q-Routing* [40]. Hierbei werden Merkmale wie Latenz und Auslastung der Pfade berücksichtigt. Beim *volladaptiven Routing* wird immer der am wenigsten belastete Kanal ausgewählt, auch wenn damit ein Umweg verbunden ist. Allerdings wird bei unterschiedlichen Möglichkeiten die kürzeste Strecke bevorzugt gewählt. Das *Congestion-Look-Ahead-Routing* verwendet Informationen der potentiell involvierten Router über die Auslastung relevanter Kanäle, um so „vorausschauend“ Staus zu umgehen. *Turnaround-Routingverfahren* werden bei Baum- und Butterfly-Topologien eingesetzt und zeichnen sich dadurch aus, dass Pakete zunächst in die entgegengesetzte Richtung und dann von der entfernten Netzwerkseite zurück zum eigentlichen Empfänger versendet werden. Diese Methode wird durch das *Turn-Back-When-Possible-Routing* optimiert, indem die Auslastung des potentiellen Rückweges mit in die Wegewahl einbezogen wird und ggf. eine Alternative ausgesucht wird. Beim *Odd-Even-Routing* wird zwischen erlaubten und nicht erlaubten Richtungswechseln von Knoten zu Knoten und Spalte zu Spalte bzw. Reihe zu Reihe im Gitter unterschieden. Diese variieren und ermöglichen eine *Deadlock*-freie Wegewahl.

Außer den hier genannten Verfahren gibt es weitere, weniger relevante Routingverfahren und Abwandlungen der oben genannten Methoden. Zur Vertiefung sei auf die weiterführende Literatur [22][26][35][41] verwiesen. Basierend auf den bereits vorgestellten Switching- und Routingverfahren können nun weiterreichende Netzwerkmechanismen eingesetzt werden, die die Leistungsfähigkeit des NoCs für die jeweiligen Einsatzgebiete deutlich steigern können. Hierzu zählt auch die garantierte Bandbreitenzuweisung / *Quality-of-Service (QoS)*.

**Congestion- / Flow-Control-Techniken.** Um die entsprechenden Prioritäten der Datenübertragung und die damit verbundenen maximalen Latenzen einhalten zu können, müssen zwei grundlegende Phänomene erläutert werden. Die Problematik, dass Pakete um Netzwerkressourcen konkurrieren und damit zunächst eine hohe Auslastung (*Contention*) herbeiführen, die in einer Überlastung (*Congestion*) enden kann, erfordert *Congestion-Control*-Techniken. Ein korreliertes Problem tritt auf, wenn einzelne Endknoten unterschiedliche Bandbreiten injizieren bzw. absorbieren können. Um diese Problematik zu lösen, müssen *Flow-Control*(Flußkontroll)-Techniken eingesetzt werden. *Congestion-Control*-Techniken arbeiten gegen eine Überlastung des Netzwerks und seiner Routingknoten im Allgemeinen, während *Flow-Control*-Mechanismen den reibungslosen Datenfluss



zwischen Sender- und Empfängerknoten regeln. Durch Einsatz dieser beiden Techniken wird die Einhaltung einer minimal garantierten Bandbreite, einer garantierten maximalen Latenz und eines maximal zulässigen Jitters und damit QoS erst ermöglicht.

Durch räumliche Ressourcenverwaltung, also Zuweisung von Routingkanälen und Speicher, sowie zeitliche Zuteilung kann eine Stauung und letztendlich Überlastung der Ressourcen vermieden werden. *Congestion-Control*-Mechanismen können in rückgekoppelte (*feedback*) und offene bzw. vorbeugende (*preventive*) Verfahren eingeteilt werden [42]. Außerdem kann eine Einteilung in ressourcenreservierende und in nicht-ressourcenreservierende Maßnahmen vorgenommen werden.

Zu den Letzteren zählt das Verfahren, bei dem Pakete, die aufgrund von Stauungen nicht weitergeleitet werden können, einfach verworfen werden (*dropping*). Zunächst können mit nicht-ressourcenreservierenden Verfahren Verstopfungen aufgelöst werden, insgesamt jedoch müssen die Pakete erneut versendet werden, was zu einem höheren Datenaufkommen führt. Dies reduziert folglich die effektiv übertragene Datenmenge. Derzeit sind noch keine NoCs bekannt, die dieses in Datennetzen etablierte Verfahren ebenfalls anwenden [35]. Eine weitere Methode stellen dynamische Routingverfahren dar, die so genannte „Hot Spots“, also Punkte besonders hohen Datenaufkommens, vermeiden [43]. Dieses Verfahren hat jedoch den Nachteil, dass die Reihenfolge der Pakete verändert werden kann (*Packet Reordering*). Ein anderer Ansatz steuert die Paketinjektionsrate basierend auf statistischen Daten, die während des Betriebs ausgewertet werden [44]. BENINI charakterisiert die bisher erläuterten Verfahren als reaktiv, d. h. er argumentiert, dass sie erst greifen können, wenn das Netzwerk bereits überlastet ist [35]. Dies trifft meiner Ansicht jedoch nicht bei dynamischen Routingverfahren zu, die zufällig über die Wegewahl entscheiden. Ebenso lässt sich durch eine geeignete Schwelle bei den Monitoren schon vor der Überlastung des Netzwerks eine Reduktion der Paketinjektion einleiten und damit der Verstopfung vorbeugen.

Ressourcenreservierende Maßnahmen teilen vor der Übertragung die Ressourcen zeitlich und örtlich zu. Zunächst werden alle geplanten Übertragungen festgestellt und anhand ihrer Erfordernisse eine Ressourcenzuteilung (*Admission Control*) vorgenommen. Zusätzlich zu dieser Initialisierungsphase (*setup*) wird am Ende einer Übertragung noch eine Informationsphase (*tear-down*) benötigt, in der den anderen Netzwerknoten mitgeteilt wird, dass die Daten komplett übermittelt worden sind und die Ressourcen für neue Transfers zur Verfügung stehen. Sollten alle Übertragungen und deren Anforderungen bereits im Vorfeld bekannt sein, so ließe sich mit einem Zeit-Multiplex-Verfahren (*Time Division Multiplex / TDM*) die Überlastungsfreiheit des Netzwerks garantieren. Globale Einplanungsverfahren (*Global Scheduling*), die u. a. auf dem *TDM*-Prinzip basieren, werden in NuMesh [45], Nostrum [46] und Æthereal [47] eingesetzt. Diese globalen Einplanungsverfahren sind jedoch nur bedingt für einen universellen Einsatz geeignet und wenig flexibel bei nicht-vorhersehbarem Lastaufkommen, wie dies u. a. häufig bei Netzwerkanwendungen der Fall ist. Nachteilig an diesem Ansatz ist die relativ hohe durchschnittliche Latenz, die ein Paket aufgrund der Initialisierungs- und Informationsphase erfährt. Dafür erhält man ein verstopfungsfreies (*congestion-free*) und konkurrenzfreies (*contention-free*) Netzwerk.

Ein anderer Ansatz bei ressourcenreservierenden Maßnahmen sind bandbreitenkontrollierende Verfahren (*rate-control schemes*). Hierbei werden berechnete Injektionsraten für die Quellknoten vorgegeben, so dass eine obere Schranke für die Auslastung des Netzwerks und Latenz definiert ist. Zur Vermeidung von, bei diesem Verfahren häufiger entstehenden, Übertragungsspitzen (*Bursts*) werden zusätzliche Pufferspeicher benötigt, die sich nachteilig auf den Flächenbedarf auswirken.

Vorteilhaft ist die Eigenschaft, dass die durchschnittliche Latenz geringer als beim TDM-Verfahren ist. Einsatz finden Varianten dieser Methode z. B. beim MANGO(*Message-passing Asynchronous Network-on-Chip providing Guaranteed services through OCP interfaces*)-On-Chip-Netzwerk [48][49].

Die *Flusskontrolle* übernimmt die Aufgabe, dass Pakete, trotz vorhandener Auslastungskontrolle, nicht in Routern stecken bleiben, deren Puffer nicht entleert werden können, weil ein Endknoten nicht genug Bandbreite zur Verfügung stellt. Flusskontrolle dient zur Vermeidung von „Deadlocks“, auch wenn keine Auslastungskontrolle verwendet wird. Es gibt eine Vielzahl von Algorithmen zur Flusskontrolle. Die einfachste Art ist, Pakete, die nicht weitergeleitet werden können, zu verwerfen. Eine weitere Methode besteht darin, inakzeptable Pakete zum Sender zurückzuleiten, der diese garantiert annehmen muss. Diese beiden Methoden werden derzeit in bekannten NoCs im Gegensatz zu Computernetzen nicht eingesetzt. Beim SPIN (*Scalable Programmable Interconnection Network*)-On-Chip-Netzwerk [50] wird ein Verfahren eingesetzt, bei dem bei Blockierung kurzfristig Pakete zu freien Knoten weitergeleitet werden, um nach kurzer Zeit erneut zum Endknoten geschickt zu werden.

Die folgenden Mechanismen zur Flusskontrolle basieren auf der Reservierung von Ressourcen. Um zu garantieren, dass genügend Pufferplatz in den einzelnen Übertragungsknoten vorhanden ist, können Ende-zu-Ende-Quittierungsmechanismen (*Handshake*-Verfahren) eingesetzt werden, bei denen ggf. bei nicht ausreichender Speichermenge die Erlaubnis zur Übertragung zum Zielknoten verzögert wird, auf die der Quellknoten zu warten hat. Ein anderer Ansatz wird durch so genannte Kreditpunkte-basierte (*Credit-based*) Ende-zu-Ende-Flusskontrollen aufgezeigt. Die Routingknoten verfügen über ein gewisses Kontingent an *Credits*, das ihrer Speichermenge entspricht. Soll ein Paket in das Netzwerk injiziert werden, so benötigt der Sender zunächst entsprechende Credits seitens des Empfängers. Aus diesem Grund werden regelmäßig Credits von freien Routingknoten an Nachbar-knoten versendet. Bei erfolgreicher Versendung wird im Sender der Credit-Zähler um die speichermengen-entsprechende Anzahl dekrementiert und die „verbrauchten“ Credits an den Empfangsknoten zurückgeschickt. Dieser zusätzliche Verkehr an „Credit-Paketen“ kann bis zu 31% der verfügbaren Bandbreite beanspruchen [43]. Speziell bei kleinen Datenmengen ist der Overhead beträchtlich und dieses Verfahren relativ ineffizient. Es gibt zahlreiche Abwandlungen dieses Verfahrens, um die Nachteile zu nivellieren. *Æthereal*, *Nostrum*, *QNoC* und *SPIN* verwenden u. a. Varianten dieser Methode zur Flusskontrolle. Nach [35] verwenden derzeit nur wenige On-Chip-Netzwerke sowohl Congestion- als auch Flow-Control-Mechanismen, um harte Bandbreitengarantien zu gewährleisten. Hierzu zählen ebenfalls *Æthereal*, *MANGO* und das *SonicsMX*-On-Chip-Netzwerk [51]. Zusammenfassend ist zu sagen, dass es neben den Vorteilen der Ressourcenreservierung auch zahlreiche Nachteile dieses Verfahrens gibt. Es gilt also von Fall zu Fall abzuwägen, welche Maßnahmen für die Implementierung eines ressourceneffizienten On-Chip-Netzwerks zu treffen sind. Je flexibler ein On-Chip-Netzwerk auf Veränderungen hinsichtlich des Verkehrsaufkommens und der Topologie bzw. der Anzahl der Teilnehmer reagieren kann, sowohl im Betrieb, als auch bei der Konzeption neuer Hardware, desto effizienter wird es sich auch in zukünftige SoCs integrieren lassen. Zukünftig werden immer stärker Entwicklungswerkzeuge, die speziell für den Entwurf der Kommunikationsinfrastruktur von Systems-On-Chip konzipiert sind, in die Entwurfs-kette komplexer digitaler Schaltkreise Einzug halten, wie sie z. B. von *ARTERIS* bereits vorgestellt wurden [52].

**Dienstqualität.** *Quality-of-Service(QoS)* ist für viele Anwendungsbereiche ein weiteres wichtiges Merkmal für Netzwerke und stellt ebenso eine Herausforderung für On-Chip-Netzwerke dar. Unter QoS werden Mechanismen verstanden, die es ermöglichen die Übertragungsbandbreite für unterschiedliche Datentypen oder zwischen einzelnen Endknoten unter Berücksichtigung spezifizierter Transportklassen dediziert zuzuweisen. Gerade bei limitierter Bandbreite des Netzwerks und speziell auch für „Echtzeit“-Anwendungen ist diese Funktionalität von besonderer Bedeutung. QoS erlaubt ein priorisiertes Übertragen unterschiedlicher Datentypen, was nicht zuletzt bei heutigen Breitbandnetzwerkanwendungen wie „Tripple Play“, also Datenverkehr, Telefonie und multimediale Video-Inhalte, von großer Bedeutung ist. Für QoS sind Mechanismen zur Überwachung des Transportaufkommens notwendig, die u. a. Entscheidungen über die Wegewahl und die zeitliche Ablaufsteuerung der Kommunikation treffen. Wenn ein Netzwerk über keine QoS-Mechanismen verfügt, spricht man von einem „Best-Effort-Ansatz“. Hierunter versteht man das Prinzip, dass alle Daten und Verbindungen gleich behandelt werden und der Transport nach den Möglichkeiten des Systems geschieht. Es können keine festen Zusagen bezüglich der zur Verfügung gestellten Bandbreite gemacht werden. Ein häufig eingesetztes Mittel, ein Netzwerk QoS-tauglich zu machen, besteht darin, das Netzwerk großzügiger bezüglich seiner Bandbreite zu dimensionieren, damit die anfallende Datenlast mühelos bewältigt werden kann. Hierbei sind die Wege, die das Rückgrat des Netzes ausmachen, so leistungsfähig zu gestalten, dass die Endknoten nicht annähernd die Datenmenge ins Netz injizieren können, die zu einer Überlastung (*Congestion*) des Netzwerks führen könnte. Diese relativ einfache Methode wird auch *Over-Provisioning* genannt. Diese Form der QoS-Implementierung ist jedoch immer kritisch in Relation zu der Anwendung und der Art des Datenaufkommens sowie der Anzahl der Teilnehmer zu sehen. Für höhere Anforderungen setzt man häufig auf ein Markieren der einzelnen Datenpakete im Hinblick auf ihre Anforderungen bezüglich Latenz, Jitter und Zuverlässigkeit ihrer Zustellung. In diesem Zusammenhang spricht man auch von „differenzierten Diensten“ oder auch *DiffServ (Differentiated Services)*. Jedes Datenpaket trägt eine Kennung in sich, in der die Anforderungen kodiert sind. Die Knoten im Netz können dann den anfallenden Datenverkehr anhand ihrer Auslastung und der Kennungen innerhalb der Pakete steuern. Allerdings kann auch ein solches Netzwerk nur in dem Maße befriedigende Leistung bieten, als es physikalisch angemessen für das entsprechende Anwendungsszenario ausgelegt ist. In diesem Falle steigert die QoS-Fähigkeit die Performanz.

Die *Software-Schicht* umfasst die abstrakteren Schichten fünf bis sieben des ISO/OSI-Referenzmodells. Diese Schicht bezieht sich sowohl auf System- als auch auf Anwendungssoftware. Die Systemsoftware abstrahiert von der eigentlichen Hardware und fungiert als *Hardware Abstraction Layer (HAL)*. Dies entkoppelt die Anwendungssoftware von der Hardware und gewährleistet eine höchstmögliche Flexibilität. Die Systemsoftware ist eng an systemspezifische Schnittstellen für Hardwarebeschleuniger (vgl. Abbildung 4-19) und IPs gebunden (vgl. Abbildung 4-4). Sie gestaltet sich als kontrolllastig mit geringerem Kommunikationsaufkommen. Inwieweit die Systemsoftware ausgeprägt ist, also ob es sich um einzelne Aufgaben / Tasks handelt, oder ob ein komplettes eingebettetes Betriebssystem zum Einsatz kommt, hängt maßgeblich von der jeweiligen Anwendung und dem Einsatzgebiet ab. Die Anwendungssoftware sollte für ein massiv paralleles System einem geeigneten Programmiermodell unterliegen (vgl. Abschnitt 4.5) und ist häufig stark kommunikationslastig.

Abbildung 2-13 zeigt zusammenfassend die in den vorangegangenen Abschnitten diskutierten wesentlichen Mechanismen zur Gestaltung und Leistungssteigerung von On-Chip-Netzwerken. In Ab-

hängigkeit vom jeweiligen Anwendungsszenario und den damit verbundenen Anforderungen werden Hardware und Protokolle unter Verwendung der vorgestellten Maßnahmen zur Optimierung konzipiert.

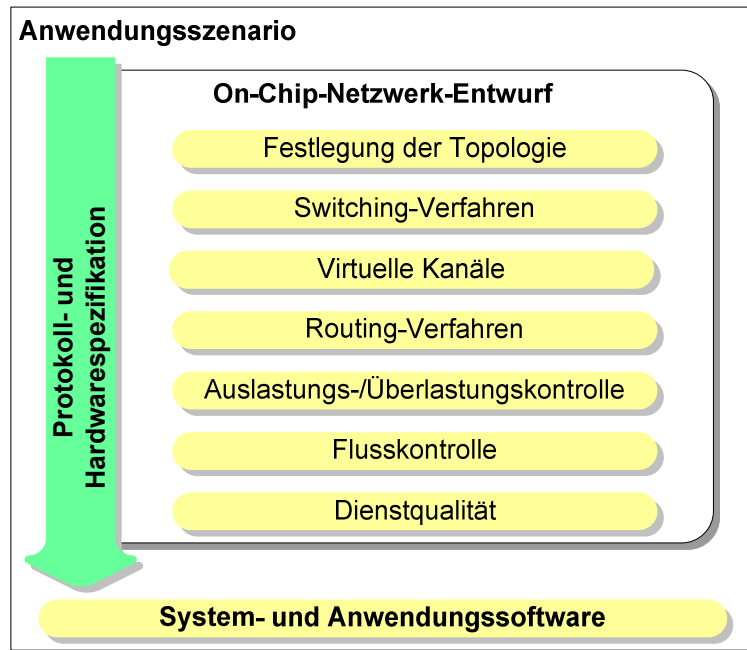


Abbildung 2-13: Mechanismen zur Gestaltung und Leistungssteigerung von On-Chip-Netzwerken

### 2.3.3 Beispiele von On-Chip-Netzwerken

In Tabelle 2-2 werden wesentliche Merkmale etablierter NoC-Varianten vorgestellt. Die bevorzugte Topologie der aufgeführten Netzwerke ist das 2D-Gitter, das aufgrund seiner regelmäßigen Struktur relativ gut zu implementieren ist. Die Größe der Kommunikationskanäle, also die Breite der Übertragung, liegt zwischen 19 Bits beim Marescaux-Ansatz und 294 Bits beim Dally-NoC. Einige NoCs sind in dieser Hinsicht auch parametrisierbar, wie z. B. QNoC, Hermes-NoC oder auch das GigaNoC. Bei diesen Netzwerktopologien besteht noch Spielraum, und es kann auf Anforderungen seitens der Anwendung eingegangen werden, um so eine möglichst effiziente Realisierung zu treffen. Die vorherrschende Routingstrategie ist das *Wormhole*-Switching mit XY-Routing-Ansatz. Beim *Æthereal*-NoC wird *Wormhole*-Switching nur für *Best-Effort*(BE)-Pakete verwendet und für *Guaranteed-Throughput-Traffic*(GT)-*Circuit-Switching* genutzt. Weitere Details zu einigen der aufgeführten NoCs sind in [53] aufgeführt.

Die Art der Pufferung entscheidet, wie anfällig die NoCs gegenüber dem Phänomen des *Head-of-Line-Blocking*(HOL)-Problems sind. Einige Ansätze verwenden Eingangspuffer bzw. EingangsfIFOs, um die eintreffenden Pakete/Flits zwischenspeichern, andere benutzen diese Speicherelemente im Ausgang. Besonders flächenintensiv sind zumeist die Varianten, die sowohl Ein- als auch Ausgänge mit Speicher ausrüsten. Dies schlägt sich dann allerdings positiv auf die Performanz nieder. Die Realisierung des Kreuzschienenverteilers innerhalb des Routers kann zum einen vollständig (voll) sein. In diesem Fall können alle virtuellen Kanäle (VC), die der Router zur Verfügung stellt, direkt zugewiesen werden, d. h. die Anzahl der Ein-/Ausgangsports des Kreuzschienenverteilers ist gleich der Zahl der virtuellen Kanäle  $m$  multipliziert mit der Anzahl der Routerports  $n$ . Zum anderen kann ein Multiplexer eingesetzt werden, der die jeweiligen Verbindungen arbitriert herstellt. Hierdurch entsteht ggf. ein Performanznachteil, allerdings ist die benötigte Fläche für Verbin-

dungsleitungen geringer. Beim multiplexerbasierten Kreuzschienenverteiler entspricht die Anzahl der Ein- und Ausgangsverbindungen jeweils  $n$ . Diese Art der Realisierung eignet sich speziell für Implementierungen, die eine hohe Anzahl von virtuellen Kanälen erlauben [54].

**Tabelle 2-2: Charakteristika ausgewählter On-Chip-Netzwerke**

Netzwerk/ Router	Topologie	Kommuni- kationskanäle	Switching/ Routing- Strategie	Pufferung	Kreuzschienen- verteiler	Fluss- kontrolle	VC	VC-Auswahl	QoS- Unterstützung
<b>Kavaldijev</b> [55]	2D-Gitter	NA	Wormhole Source	Input Queue	Voll	NA	4	TDM	Ja
<b>QNoC</b> [56]	2D-Gitter regulär oder irregulär	16 Datenbit (parametrierbar) +10 Kontrollbit	Wormhole XY	Input Queue	Voll	Credits	4	Priorität und Puffer- Verfügbarkeit	Ja
<b>Dally</b> [57]	2D-gefalteter Torus	256 Datenbit +38 Kontrollbit	Wormhole XY Source	Input Queue +1 Output Position	Multiplexer	Credits	8	NA	Ja
<b>Marescaux</b> [58]	2D-Torus	16 Datenbit +3 Kontrollbit	Wormhole XY	2 Output Positions	Multiplexer	Handshake	2	TDM	Ja
<b>Xpipes</b> [59]	Variabel (zur Entwurfszeit)	32, 64 oder 128 Bits	Wormhole Street Sign	Output Queue	Multiplexer	Handshake	parametrierbar	Priorität	Nein
<b>Æthereal</b> [60]	2D-Gitter	32 Bits	Circuit Switching (GT) Worm- hole Source (BE)	Output Queue	NA	NA	3	Priorität	Ja
<b>MediaWorm</b> [54]	Nicht zuord- nungs-f.	NA	Wormhole	Input und Output Queue	Multiplexer	NA	2	Virtuelle Uhr	Ja
<b>Hermes NoC</b> [53]	2D-Gitter	16 Datenbit (parametrierbar) +6 Kontrollbit	Wormhole XY / partiell adaptiv	Input Queue	Voll	Credits	2-4	TDM-adaptiv	Nein
<b>MANGO</b> [48]	2D-Gitter Clockless	32 Datenbit +5 Kontrollbit	GS und BE XY	Output Queue	Multiplexer	Handshake / Credits	8	Priorität	Ja
<b>SPIN</b> [50]	Fat Tree	32 Datenbit +4 Kontrollbit	Wormhole, adaptiv	Input und Output Queue	Voll	Handshake / Credits	parametrierbar	NA	Ja
<b>Nostrum</b> [46]	2D-Torus	NA	Deflective (Hot-Potato) Routing Wormhole	NA	NA	Credits	parametrierbar	TDM	Ja
<b>GigaNoC</b>	2D-Gitter regulär oder irregulär 2D-Torus Variabel (zur Entwurfszeit)	64 Datenbit (parametrierbar) +29 Kontrollbit	Wormhole XY / adaptiv d. Routing-tabelle	Input Queues, Output Queue	Multiplexer	Handshake	2 / parametrierbar	Variabel / tabellen- basiert	Vorgesehen, z. Zt. nicht implementiert

Die Flusskontrolle ist besonders bei VC-basierten NoCs ein weiteres Merkmal zur Differenzierung. Bei *creditbasierter* Flusskontrolle halten die Router Zähler für die zur Verfügung stehende Menge an Pufferspeicher vor. Sind keine Pufferressourcen mehr verfügbar, so werden eingehende Pakete/Flits abgelehnt und dem Sender der *Credit*-Stand des Empfängers über Benachrichtigungskanäle bei freiem Pufferspeicher mitgeteilt [22]. Bei der *Handshake*-Flusskontrolle wird dem Sender seitens des Empfängers per dedizierter Leitung oder Protokollpaket signalisiert, ob das empfangene Paket verarbeitet werden kann oder verworfen wird. Hardwareseitig bedeutet dies deutlich weniger Aufwand, allerdings ist dieses Verfahren nicht so effizient wie das creditbasierte, da ggf. erfolglose Übertragungen stattfinden oder Pufferspeicher unnötig lang belegt bleiben. Zudem muss senderseitig auf eine Bestätigung (*Acknowledge*) gewartet werden.

Die nächsten drei Spalten von Tabelle 2-2 enthalten VC-spezifische Merkmale. Zunächst wird die Anzahl der möglichen virtuellen Kanäle pro Router angegeben, in der Folgespalte der Zuweisungsmodus. Beim GigaNoC sind virtuelle Kanäle vorgesehen, deren Anzahl parametrisierbar ge-

halten ist. Sie können zum einen über die Auswahl der Routingstrategie eingerichtet werden, zum anderen erlauben die Kommandoflits des GigaNoCs eine höherprioräre Verarbeitung im Vergleich zu den Datenflits (vgl. Abschnitt 4.2.2.1), so dass man derzeit über eine relativ einfache Variante mit zwei virtuellen Kanälen verfügt. Durch eine zusätzliche Auswertung von Tabelleneinträgen kann dieser Wert bei Bedarf angepasst werden. Dies gilt ebenso für die Unterstützung bezüglich verschiedener Qualitätsklassen der Kommunikation (*Quality-of-Service / QoS*). Die Mehrzahl der anderen NoCs sieht eine Unterstützung von Qualitätsklassen bei der Kommunikation vor. Beim GigaNoC kann dies in Abhängigkeit von der Anwendung abgewogen werden. Die Entscheidung über die Auswahl der VCs wird teilweise durch Zeitmultiplex (*Time Division Multiplex / TDM*) oder aber durch Prioritäten-Vergabe, wie es z. B. beim Internet-Protokoll [61] vorgesehen ist, getroffen. Beim MediaWorm-NoC wird die Allokation der virtuellen Kanäle mit Hilfe einer virtuellen Uhr geregelt. Eine globale Zeit im Zusammenhang mit einem individuellen, paketzugeordneten Zeitinkrement (*Vtick*) und einer Ankunftszeit (*AuxVC*) dient zur Kanalvergabe. Je kleiner der Wert von *Vtick* ist, desto schneller muss das Paket weitergeleitet werden.

### 2.3.4 Anforderungen an On-Chip-Netzwerke

Die sich ergebenden wesentlichen Anforderungen an On-Chip-Netzwerke, gilt es je nach Einsatzzweck und unter Berücksichtigung aller Randbedingungen, gewichtet miteinander in Beziehung zu setzen und eine möglichst optimale Konstellation zu wählen. Neben den für Systementwürfe bekannten Kriterien wie Performanz (aufgeteilt in Durchsatz, Latenz und Jitter), Leistungsaufnahme, Flächenbedarf und Kosten, sind weitere Faktoren von großer Relevanz. So ist speziell für große Systeme die Skalierbarkeit oder auch die Unterstützung unterschiedlicher Topologien sehr wichtig. Je nach Leistungsfähigkeit der Verarbeitungseinheiten (vgl. Abschnitt 2.4) kann auch der Funktionsumfang des NoCs von Bedeutung sein. Bei nicht so leistungsfähigen Verarbeitungseinheiten kann ein NoC diese durch eigene Intelligenz entlasten (*Communication Off-load Engines*) und so die Gesamtleistung des Systems steigern. Ebenso können Merkmale wie *Quality of Service* oder Fehlertoleranz in dem jeweiligen Anwendungsszenario von besonderer Wichtigkeit sein. Sehr positive Eigenschaften für eine vielseitige Verwendung eines On-Chip-Netzwerks sind Flexibilität bzw. Robustheit gegenüber sich ändernden Lastaufkommen bzw. Bandbreitenansprüchen einzelner Netzwerkpfade. In Abbildung 2-16, Abschnitt 2.7 werden die allgemeinen und NoC-spezifischen Anforderungen und ihre gegenseitigen Wechselwirkungen in einer Merkmalsmatrix für Chip-Multiprozessoren gegenüber gestellt. Als weiterführende Literatur zu On-Chip-Netzwerken sei auf die Beiträge [22][26][41][53][62][63][64] verwiesen, die tiefer gehende Details über On-Chip-Netzwerke liefern.

## 2.4 Eingebettete Verarbeitungseinheiten

Neben der Kommunikationsinfrastruktur stellen die eingebetteten Verarbeitungseinheiten die maßgebliche funktionale Einheit eines massiv-parallelen eingebetteten Systems dar. Ihre Aufgabe ist es, die zur Verfügung gestellten Daten möglichst effizient zu verarbeiten und in spezifizierter Form den Ausgangsschnittstellen zur Verfügung zu stellen.

### 2.4.1 Anforderungen an eingebettete Verarbeitungseinheiten

Im Folgenden soll kurz auf die Anforderungen bzw. besonderen Merkmale von eingebetteten Verarbeitungseinheiten (PEs) eingegangen werden. Wobei auch hier die Performanz, die Leistungsaufnahme und die benötigte Chipfläche bzw. die Kosten zu den allgemeinen Merkmalen bzw. Anforderungen zählen, die je nach Anwendungsgebiet unterschiedliche Stellenwerte haben. Ebenso ist die Flexibilität bzw. die Art und Weise der Programmierbarkeit einer Verarbeitungseinheit je nach Einsatzzweck von entscheidender oder weniger bedeutender Rolle. Verarbeitungseinheiten z. B. in Funkweckern sind aufgrund des sich nicht ändernden Einsatzzweckes, des hohen Preisdrucks und der limitierten Ressourcen sehr unflexibel, aber zugleich klein und energieeffizient realisiert. Bei zentralen Verarbeitungseinheiten in PDAs hingegen spielt der Preis im Gegensatz zu Flexibilität und Leistungsfähigkeit eine untergeordnete Rolle, da die Anwendungsgebiete dieser Geräte sehr variabel ausfallen und zugleich hohe Leistungsansprüche haben können. In diesem Szenario kommt zusätzlich die Wiederverwendbarkeit zum Tragen, da Softwareentwicklungen auch für zukünftige Architekturen verwendet werden sollen. Besonders für komplexe Systeme ist die Verifizierbarkeit der Funktionalität von großer Bedeutung. Dies setzt eine ausgereifte, komfortable und umfangreiche Entwicklungsumgebung mit guten „*Debug*-Möglichkeiten“ voraus. Im Hinblick auf die Kosten spielen ebenfalls die von der Verarbeitungseinheit benötigten Ressourcen, wie z. B. Programmspeicher und Kommunikationsschnittstellen eine nicht zu vernachlässigende Rolle. Abbildung 2-16, Abschnitt 2.7 zeigt u. a. wesentliche Anforderungen an eingebettete Verarbeitungseinheiten, die es bei der Konzeption eines Systems zu berücksichtigen und gegeneinander abzuwägen gilt. Sich ergebende Wechselwirkungen mit anderen Anforderungen bei Veränderungen einzelner Merkmale können anhand von Trendsymbolen abgelesen werden.

### 2.4.2 Klassen eingebetteter Verarbeitungseinheiten

Abbildung 2-14 zeigt Varianten eingebetteter Verarbeitungseinheiten und gibt eine qualitative Einstufung ihrer Eigenschaften bezüglich ihrer Performanz, ihrer Flexibilität und der Zeit bis zur Marktreife bzw. Verfügbarkeit (vgl. Kapitel 3). Mit der Zeit bis zur Verfügbarkeit ist bei anwendungsspezifischen Hardwareeinheiten wie den rekonfigurierbaren FPGA-Zellen und den spezialisierten, meist Standardzellen-basierten Hardwarebeschleunigern der Entwurf und die Programmierung bzw. Fertigung mit berücksichtigt. Bei den programmierbaren Prozessoren (*Central Processing Unit / CPU*), Co-Prozessoren (*Co Processing Unit / Co-PU*) und anwendungsspezifischen Prozessoren (*Application Specific Instruction Set Processor / ASIP*) wird eine Abstufung bzgl. der Einfachheit ihrer Programmierung vorgenommen. Die Größe der Kugeln spiegelt den ungefähren Flächenbedarf in Bezug auf eine vergleichbare Performanz der einzelnen Verarbeitungseinheiten wieder. Die dargestellten Werte dienen zur groben Einstufung der durchschnittlichen Eigenschaften und können mitunter für Spezialfälle abweichen.

Zu den hier vorgestellten Verarbeitungseinheiten zählen die *Universal-Prozessoren* (CPU), die mittels einer Hochsprache sehr flexibel oder mit Hilfe einer maschinennahen Sprache bzgl. der Performanz zumeist effizienter, aber einhergehend mit deutlich größerem Zeitaufwand programmiert werden können. Implementierungen einer geforderten Funktionalität gestalten sich besonders bei Verwendung einer Hochsprache und der entsprechenden Werkzeuge wie Compiler und Linker etc. sehr schnell. Auch die Verifikation der programmierten Funktionalität wird häufig durch eine Vielzahl von Hilfswerkzeugen oder durch integrierte Entwicklungsumgebungen komfortabel unterstützt.

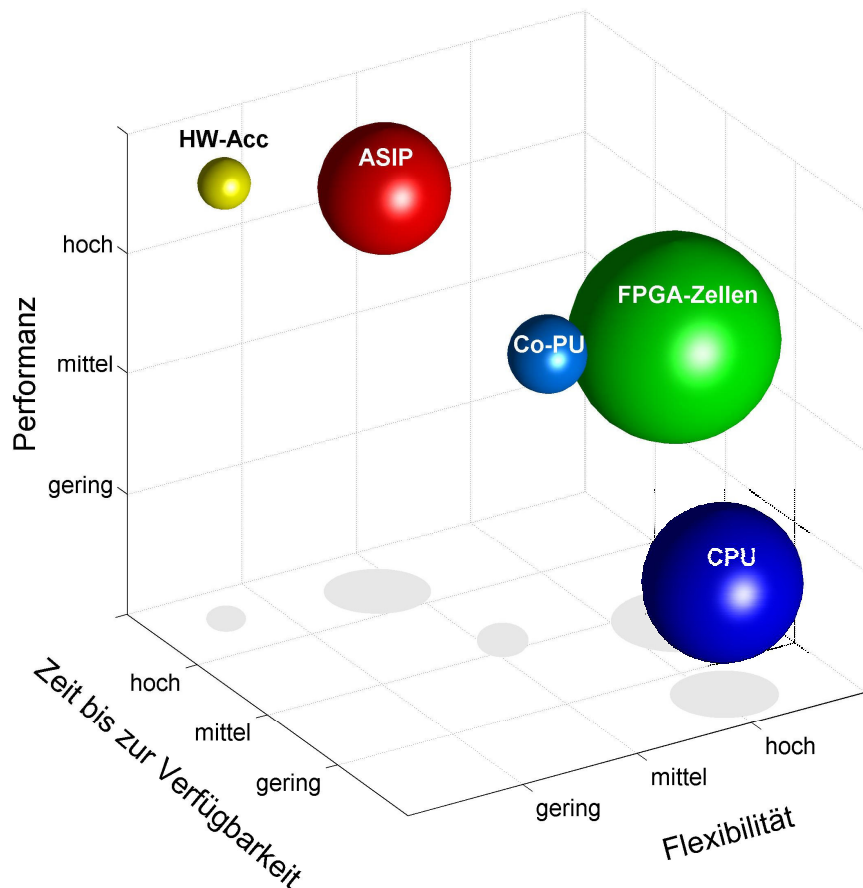


Abbildung 2-14: Entwurfsraum eingebetteter Verarbeitungseinheiten

Zur Performanzsteigerung von Universal-Prozessoren werden häufig spezialisiertere *Coprozessoren* eingesetzt, die über einen eingeschränkten, zumeist anwendungsspezifischen Befehlssatz verfügen. Je nach Unterstützung durch die Entwicklungsumgebung muss bestehender Code umgeschrieben bzw. erweitert werden, um die volle Funktionalität solcher Zusatzprozessoren ausnutzen zu können. Dadurch erhöht sich bei ihrer Verwendung die Zeitspanne, bis eine zu realisierende Anwendung verfügbar ist. Sie benötigen zwangsläufig eine CPU für übergeordnete Kontrollaufgaben. Aufgrund ihrer zumeist eingeschränkten Funktionalität sind sie von der Fläche meist kleiner als der eigentliche Prozessor, erzielen dennoch eine höhere Performanz in ihrem Einsatzbereich.

**Anwendungsspezifische Prozessoren** (*Application-Specific Instruction Set Processors / ASIPs*) grenzen sich von den Universal-Prozessoren in der Art ab, dass sie speziell für ausgewählte Anwendungsgebiete konzipiert sind. Deshalb ist ihre Performanz höher, ihre Flexibilität aufgrund eines eingeschränkten / spezialisierten Befehlssatzes und ggf. weniger stark ausgeprägter Unterstützung von Hochsprachen geringer als die der Universal-Prozessoren. Ihre Programmierung gestaltet sich zumeist komplexer, so dass die Zeitspanne bis zur Verfügbarkeit einer Lösung in der Regel größer ausfällt als bei Universalprozessoren, aber mit einer höheren Performanz einhergeht. Zu der Klasse der ASIPs lassen sich auch Netzwerkprozessoren [65] zählen, die für spezielle Aufgaben bei der Datenverarbeitung in Computernetzen eingesetzt werden, vgl. Kapitel 7 und 8.

**Eingebettete rekonfigurierbare Hardware**, auch *FPGA(Field Programmable Gate Array)*-Blöcke genannt, lassen sich durch Formulierungen in Hardwarebeschreibungssprachen und anschließender Synthese auf die Zieltechnologie in ihrer Funktionalität beliebig oft konfigurieren. Sie bieten den Vorteil der Möglichkeit von massiv paralleler Verarbeitung bei Betriebsfrequenzen, die derzeit be-



reits über 500 MHz liegen können [66]. Die Realisierung und Verifikation wird ebenfalls durch ausgereifte Werkzeugketten unterstützt, gestaltet sich allerdings vom Zeitaufwand in der Regel umfangreicher als beim reinen Softwareentwurf. Dafür kann bei vielen Anwendungen aufgrund der realisierbaren Parallelität eine respektable Performanz erzielt werden. Die Fläche ist hingegen aufgrund der flexibleren Strukturen höher anzusetzen.

Die zumeist schnellste und zugleich energieeffizienteste Lösung bei geringster Fläche sind die *spezialisierten Hardwarebeschleuniger (HW-Acc)*. Sie bieten jedoch am wenigsten Flexibilität und benötigen die längste Implementierungszeit. Sie eignen sich für Aufgaben, deren Spezifikation sich nicht mehr ändert und die besonders viel Rechenleistung ohne große Flexibilitätsanforderungen verlangen. Die Implementierung erfolgt ebenfalls durch Modellierung in einer Hardwarebeschreibungssprache und anschließende Abbildung auf die entsprechende Zieltechnologie, z. B. FPGA- oder Standardzellentechnologie. Bei der letzteren Zieltechnologie ist der Aufwand deutlich höher und damit die Zeit bis zur Verfügbarkeit verglichen mit den anderen Varianten am längsten.

Letztendlich sollte von Fall zu Fall abgewogen werden, welche Art von Verarbeitungseinheit für welchen Zweck zum Einsatz kommt. Wünschenswert ist eine globale Kommunikationsinfrastruktur, die eine problemlose Anbindung aller Varianten von eingebetteten Verarbeitungseinheiten zulässt und für den Systemprogrammierer leicht ansprechbar integriert.

Hardwarebeschleuniger, FPGA-Blöcke und anwendungsspezifische Prozessoren sind Einheiten, die zumeist kunden- bzw. anwendungsspezifisch realisiert werden. Sie finden Einsatz in sehr eingeschränkten Anwendungsklassen mit besonderen Anforderungen an die Leistungsfähigkeit oder an die Kosten. Eingebettete Universal-Prozessoren zusammen mit ihren Coprozessoren bzw. Coprozessorschnittstellen sind nicht ausschließlich für Kontrollaufgaben geeignet, sondern werden zusätzlich für ein großes Spektrum von Anwendungen eingesetzt. Neben den bereits verifizierten Hardwareentwürfen versprechen ausgereifte Entwicklungsumgebungen zudem eine kurze Entwurfszeit des Gesamtsystems. Detaillierte Analysen zu den unterschiedlichen eingebetteten Verarbeitungseinheiten in Bezug auf die GigaNetIC-Architektur werden in den Kapiteln 6, 7 und 8 für dedizierte Anwendungen aus dem Netzbereich vorgestellt.

Im Folgenden wird ein Überblick über eine Auswahl der verbreitetsten Prozessorkerne und ihre wesentlichen Merkmale gegeben. Selbst wenn für sehr rechenlastige Anwendungen spezielle Hardware Einsatz findet, so werden Universalprozessoren sehr häufig für die bereits erwähnten übergeordneten Kontrollaufgaben und Schnittstellenfunktionen eingesetzt, so dass aktuelle SoCs zumeist mehrere Prozessorkerne integrieren.

### 2.4.3 Charakteristika von eingebetteten Prozessoren

Betrachtet man die Kopplung Speicher und Prozessorkern, unterscheidet man bei den eingebetteten Prozessoren im Wesentlichen zwischen der Von-Neumann-Architektur und der Harvard-Architektur. Bei der *Von-Neumann-Architektur* werden Daten und Instruktionen über einen gemeinsamen Bus transferiert. Zunächst werden die Instruktion und im Anschluss die benötigten Daten geholt. Diese sequentielle Vorgehensweise kann eine gewisse Geschwindigkeitsreduktion bedeuten, da der Prozessor ggf. erst auf die Bereitstellung der benötigten Daten warten muss. Die Entwicklung der letzten Jahre hat gezeigt, dass die Geschwindigkeit von Prozessoren schneller zunimmt als die der Speicher und Bussysteme [67]. Man spricht in diesem Zusammenhang auch von

dem „Von-Neumann-Flaschenhals“, dessen Effekt durch Einsatz von Caches, also schnellen Zwischenspeichern in enger räumlicher Nähe zum Prozessorkern, verringert werden kann.

Bei der **Harvard-Architektur** sind getrennte Busse für Daten und Instruktionen vorgesehen. Hierdurch wird der Ablauf nicht so stark wie bei der Von-Neumann-Architektur durch die limitierenden Ressourcen Bus und Speicher beeinträchtigt. Daten und Instruktionen können gleichzeitig geladen werden. Aus Sicherheitsgründen ist eine strikte Trennung zwischen Daten und Instruktionen ebenfalls empfehlenswert, da so die Gefahr von Schadcode, der zu Pufferüberläufen führt, reduziert wird. Allerdings benötigt die Implementierung der Harvard-Architektur zusätzliche Chipfläche für die Realisierung der beiden Bussysteme und zusätzlich benötigter Kontrolllogik.

Bezüglich des Befehlssatzes von Prozessoren wird allgemein zwischen *RISC(Reduced Instruction Set Computer)*- und *CISC(Complex Instruction Set Computer)*-Architekturen unterschieden. Die *RISC-Prozessor-Architektur* ist gekennzeichnet durch einen relativ geringen Befehlsvorrat, daher auch der Name *Reduced Instruction Set Computer*. Wesentliche Eigenschaften dieser Architektur sind vor allem: Die enthaltenen Befehle sind weniger spezialisiert, besitzen nahezu alle die gleiche Länge und können meist in einem Taktzyklus abgearbeitet werden, was u. a. durch Pipelining erreicht wird. RISC-Prozessoren sind zumeist als *Load-Store-Architektur* realisiert, d. h. nur Befehle aus der *Load/Store*-Gruppe können auf den Speicher zugreifen. Alle weiteren Befehle arbeiten auf Registerinhalten. RISC-Kerne verfügen deshalb zumeist über eine größere Anzahl von Registern, da dies die beschränkten Möglichkeiten der Speicherzugriffe seitens der Befehle kompensieren hilft.

*CISC-Prozessor-Architekturen* verfügen im Gegensatz zu RISC-CPU's über einen weitaus größeren Befehlsvorrat. CISC-Befehle sind in der Regel in ihrer Funktion deutlich komplexer als RISC-Befehle. Viele der Befehle sind hochgradig spezialisiert, benötigen jedoch meist mehrere Taktzyklen. Es stehen mehr Adressierungsmöglichkeiten als bei RISC-Architekturen zur Verfügung. Im Gegensatz zu den meist „festverdrahteten“ Funktionen der RISC-Kerne liegt die Funktionalität eines CISC-Kerns häufig als Microcode in einem internen Speicher des Prozessors vor. Dieser wird dann in einzelne einfachere Befehle übersetzt bzw. enthält diese direkt.

Wesentliche Vorteile der RISC-Architektur gegenüber der CISC-Architektur beim Einsatz in eingebetteten Systemen sind die geringere Komplexität der Hardware und damit die geringere Fläche als auch die geringere Leistungsaufnahme. Zudem lassen sich höhere Taktraten erzielen. Die eingesparte Fläche kann ggf. für anwendungsspezifische Hardwarebeschleuniger verwendet werden. Die CISC-Architektur zeichnet sich durch die makroartigen Befehle aus, die dem Softwareentwickler bei der Programmierung auf Assemblerebene viel Arbeit abnehmen können. Geschieht die Programmierung hingegen in einer Hochsprache, die mittels eines *Übersetzers*, im Weiteren auch *Compiler* genannt, auf den Prozessor übertragen wird, so entfällt dieser Vorteil.

#### 2.4.4 Methoden zur Erhöhung der Leistungsfähigkeit von Prozessoren

Methoden, die auf Prozessebene die Performanz und damit verbunden auch meist die Ressourceneffizienz des Systems steigern können, gründen sich häufig auf das Ausnutzen von Parallelität. Hierbei kann zwischen der feingranularen Parallelität auf Instruktionsebene, man spricht hier auch von *Instruction-Level Parallelism (ILP)*, und der grobgranularen Parallelität auf Funktionsebene, auch *Task-Level Parallelism* bzw. *Thread-Level Parallelism (TLP)* genannt, unterschieden werden.

ILP lässt sich unterschiedlich erfolgreich ausnutzen. Wie erfolgreich, also wie stark die Beschleunigung ausfällt, hängt stark von der Anwendung ab. Allerdings trägt auch die Leistungsfähigkeit des Compilers zum Erfolg bei. Je nach Fähigkeit des Erkennens von Parallelität auf Instruktionsebene seitens des Compilers kann ein Programmablauf mehr oder weniger stark beschleunigt ausgeführt werden. Der Compiler versucht hierbei durch ein Überlappen bzw. auch ein Vertauschen der Reihenfolge der Befehle die Ressourcen der Hardware zeitlich besser einzusetzen.

Zu den Techniken, die auf Basis der Mikroarchitektur eingesetzt werden, um ILP auszunutzen, gehören: **Instruktionspipelining**, **superskalare Ausführung**, **Out-of-Order-Verarbeitung**, **Register-Renaming**, **spekulative Ausführung**, **Branch Prediction** und **Multithreading**. Durch **Instruktionspipelining** wird eine teilweise Überlappung der Ausführung von Instruktionen ermöglicht. Hierdurch kann u. a. die Taktfrequenz erhöht werden. Als Steigerung kann die **superskalare Ausführung** eingesetzt werden, bei der parallele Einheiten benachbarte Instruktionen ausführen, deren Ausführung auch wieder im *Pipelining*-Verfahren stattfinden kann. Eine weitere Technik verwendet die **Out-of-Order-Verarbeitung**. Bei dieser Methode nutzt man die Existenz von Programmcode aus, der in keiner Datenabhängigkeit zu anderen Programmsequenzen steht, so dass dieser parallel ausgeführt werden kann. Die *Out-of-Order-Verarbeitung* ist orthogonal zum *Instruktionspipelining* und zur superskalaren Ausführung zu sehen und kann deshalb auch mit beiden Techniken kombiniert eingesetzt werden. Das Arbeiten mit Schattenregistern, auch **Register-Renaming** genannt, ermöglicht die Vermeidung von Konflikten bei der *Out-of-Order-Verarbeitung*, was durch diese versteckten internen Register ermöglicht wird. Die **spekulative Ausführung** von zukünftigen, wahrscheinlichen Codesequenzen mit möglichen Eingangsdaten zählt ebenso zu den Techniken zur Performanzsteigerung wie die Sprungvorhersage (**Branch Prediction**). Bei dieser werden im Vorfeld mögliche Ergebnisse von zukünftigen Sprungadressen berechnet, um den potentiellen Programmablauf parallel zur derzeitigen Operation spekulativ fortzusetzen. Die hier vorgestellten Techniken sind sowohl in Hardware realisierbar als auch mögliche Einsatzgebiete von Compilern.

Bei der Ausnutzung von *Thread-Level*-Parallelität muss die Anwendung in mehrere Funktionen bzw. *Threads* einteilbar sein, die nebenläufig abgearbeitet werden können. In diesem Zusammenhang spricht man auch von **Multithreading**. Dieses Verfahren nutzt eine gröbere Granularität als die des *ILP* aus. Die immer größer werdenden Wartezeiten auf Speicher oder andere Systemressourcen können z. B. durch das Ausführen eines anderen *Threads* sinnvoll vom Prozessor genutzt werden. Dieses „neue“ Programmierparadigma führt zu neuen Ansätzen bei der Anwendungsprogrammierung, bei der nicht mehr nur durch schnellere Prozessoren ein Leistungszuwachs erzielt wird, sondern zusätzlich durch Ausnutzen inhärenter Anwendungsparallelität und das damit verbundene Einführen von *Threads*.

### 2.4.5 Beispiele eingebetteter Prozessorkerne

Im Folgenden werden einige relevante, aktuelle eingebettete Prozessorkerne vorgestellt. Tabelle 2-3 stellt wesentliche Merkmale dieser eingebetteten Prozessorkerne gegenüber. Mit ihrer Hilfe lässt sich ein erster Eindruck über die Einordnung der GigaNetIC-Prozessorarchitektur gewinnen, deren Aufbau in Abschnitt 4.3 näher beschrieben wird. Die aufgezeigten Synthesewerte können in Kapitel 8 nachvollzogen werden. Zu den hier aufgetragenen Charakteristika zählen u. a. die maximale Arbeitsfrequenz, die stark abhängig von Technologie und Architektur respektive Anzahl der Pipeline-stufen sein kann: von 33 MHz beim MCore in 360-nm-Technologie von Freescale mit nur drei Pipeline-stufen bis zu 1000 MHz beim IBM Power464 in 90-nm-Technologie mit siebenstufiger Pipe-

line. Die typische Leistungsaufnahme variiert von 0,05 mW/MHz beim GigaNetIC-N-Core bis hin zu 0,58 mW/MHz beim MIPS32 M24K und liegt damit um zwei bis drei Größenordnungen unter der heutiger Desktop-CPU's (vgl. Abschnitt 8.3.1). Der Flächenbedarf (die *Die*-Größe) des Prozessorkerns bzw. des Kerns mit zusätzlichem Speicher in Zusammenhang mit der Technologie und der zu fertigenden Stückzahl entscheidet über die Herstellungskosten und damit auch über den Preis des Endprodukts und ist daher ebenfalls von großer Bedeutung für das Gesamtsystem. Die Größen sind abhängig von der Technologie, der Menge des integrierten Speichers und der zusätzlich integrierten Peripherie. Die angegebenen Daten liegen hier zwischen 0,1 mm<sup>2</sup> beim ARM7TDMI-S, gefertigt in 90-nm-Technologie, und 5,8 mm<sup>2</sup> beim IBM Power 464, dem zugleich leistungsfähigsten Prozessor der dargestellten Auswahl. Die Performanz der Prozessoren wird in dem Bewertungsmaß (vgl. Definition 4) DMIPS<sup>4</sup> [68] angegeben. Dieser bereits 1984 von WEICKER vorgestellte Benchmark gilt zwar seit einiger Zeit für die Bewertung von Desktop-CPU's als veraltet, wird aber im Bereich der eingebetteten Prozessorkerne immer noch zur Charakterisierung der Leistungsfähigkeit verwendet. Der Dhrystone-Benchmark ist ein synthetischer Benchmark, der die Leistungsfähigkeit bzgl. Integer- und Stringoperationen der Rechnerarchitektur bewertet. Er ist u. a. stark abhängig von der Architektur, vom Compiler, dessen Codeoptimierung, dem Linker, und der eingesetzten Cachearchitektur. Dies spiegelt sich auch in den aufgetragenen Werten wider. Der N-Core mit einer sehr einfach gehaltenen Struktur erreicht nur 0,51 DMIPS/MHz, wohingegen der flächengrößte Prozessorkern Power 464 von IBM auf einen nahezu vierfach so hohen Effizienzwert (vgl. Definition 38) von 2 DMIPS/MHz kommt.

Der ARM11MP-Prozessorkern ist, ebenso wie der GigaNetIC-N-Core-Prozessorkern multiprozessorfähig, allerdings nur konfigurierbar mit bis zu vier Prozessoren, im Gegensatz zum GigaNetIC-Processorcluster, der derzeit bis zu acht Prozessoren als eng-gekoppeltes MP-System unterstützt (vgl. Kapitel 4). Für die Realisierung eines massiv-parallelen Chip-Multiprozessorsystems ist eine kleine *Die*-Größe der Prozessorkerne von besonderer Bedeutung, vgl. Kapitel 8.

Zu den bedeutendsten Prozessorkernen in eingebetteten Systemen zählen derzeit die Produkte der Firmen ARC, ARM, Freescale, Hitachi, MIPS und Tensilica. Im Bereich der FPGA-Softcores, also der für FPGAs optimierten Prozessorkerne, sind der MicroBlaze von Xilinx, wie auch der NiosII von Altera zu nennen, deren *Die*-Größen aufgrund der abweichenden Zieltechnologie in

Tabelle 2-3 nicht aufgelistet sind. Wichtig zu erwähnen ist außerdem, dass eine Vielzahl weiterer Firmen als Lizenznehmer der vorgestellten Architekturen darauf aufbauende, eigene Produkte anbieten. Derzeit wird der Markt der eingebetteten Mikroprozessoren mit über 80 % Marktanteil von ARM dominiert. Mehr als 2,3 Milliarden ARM-basierte Mikroprozessordesigns werden derzeit jährlich gefertigt [69]. Weitere 200 Millionen integrierte Schaltkreise werden mit ARC-basierten Mikroprozessoren pro Jahr gefertigt [70]. Bei eingebetteten Mikrocontrollern, also Systemen, die zusätzlich Speicher, Peripherie und erweiterte Schnittstellen aufweisen, wird der Markt gegenwärtig unter 40 Herstellern, die insgesamt mit mehr als 50 Architekturvarianten vertreten sind, aufgeteilt.

---

<sup>4</sup> **DMIPS** steht für **Dhrystone MIPS** und beziffert die Anzahl erreichter Dhrystone-Benchmark-Durchläufe die eine Verarbeitungseinheit pro Sekunde bewältigt, geteilt durch 1757. 1757 bezeichnet die Anzahl an Durchläufen, die eine VAX11/780 Maschine erzielte. Diese galt als eine Ein-MIPS(Millionen Instruktionen pro Sekunde)-Maschine.

Tabelle 2-3: Kenndaten ausgewählter eingebetteter Prozessorkerne

Hersteller	Typ	CPU-Frequenz [MHz]	Pipeline Stufen	DMIPS	Typische Leistungsaufnahme [mW/MHz]	Die-Größe [mm <sup>2</sup> ]	Technologie [nm]	Speicher Flash / SRAM
Altera	NiosII	200	6	250	k. A.	k. A.	65	- / -
ARC	ARC 605	400	5	520	0,06	0,31	130	- / -
ARC	ARC 625D	350	5	455	0,08	0,71	130	- / -
ARC	ARC 710D	533	7	800	0,16	0,93	130	- / -
ARM	ARM7TDMI-S	245	3	220	0,09	0,1	90	- / -
ARM	966E-S	470	5	517	0,11	0,38	90	- / -
ARM	ARM11MP (mit Cache)	620	8	650	0,43	2,54	130	- / 16K+16K
ARM	ARM11MP (ohne Cache)	620	8	k. A.	0,37	1,8	130	- / -
Freescall	M-Core	33	3	31	0,41	2,2	360	- / -
IBM	Power 405	400	5	608	0,19	2,0	90	0 / 32K
IBM	Power 464-H90	1000	7	2000	0,53	5,8	90	0 / 64K
MIPS Technologies	MIPS32 M4K Core	240	5	367	0,05	0,4	130	- / -
MIPS Technologies	MIPS32 M24K Core	625	8	900	0,58	2,8	130	- / -
Renesas	SH4-202	266	5	400	0,06	0,93	130	- / -
Tensilica	Diamond 108Mini	250	5	300	0,11	0,46	130	- / -
Tensilica	Diamond 570T	233	5	380	0,28	1,46	130	- / -
Xilinx	MicroBlaze 5.00	210	5	240	k. A.	k. A.	65	- / -
GigaNetIC	N-Core	285	3	144	0,05	0,12	90	- / -
GigaNetIC	N-Core-Subsystem	285	3	144	0,2	0,96	90	- / 32K

## 2.5 Speicher für eingebettete Systeme

Der Speicher stellt neben der Kommunikationsinfrastruktur und den Verarbeitungseinheiten die dritte wichtige Hardwarekomponente für eingebettete parallele Systeme dar. Komplexe Systeme verfügen häufig nicht nur über eine Art Speicher, sondern verwenden eine besonders aufeinander abgestimmte Speicherstruktur. Dies liegt darin begründet, dass die einzelnen Speichervarianten unterschiedliche Eigenschaften und damit auch Vor- und Nachteile mit sich bringen, die es gilt möglichst gut und an die potentielle Anwendung angepasst zu kombinieren.

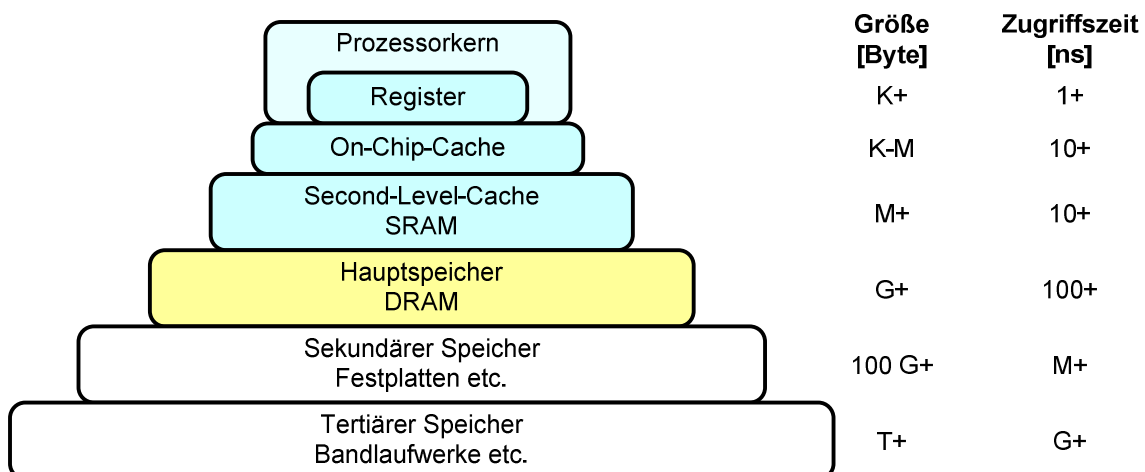


Abbildung 2-15: Speicher-Hierarchie bei Prozessorsystemen

Als Grundregel lässt sich folgende Aussage treffen: Schneller Speicher benötigt viel Fläche und Energie, großer (im Sinne der Speicherkapazität) Speicher ist langsam und benötigt weniger Energie pro Bit. Eine effiziente Speicherhierarchie versucht die Vorteile beider Speicherarten nutzbar zu machen und maskiert bestenfalls deren Nachteile. Begünstigt werden diese Ansätze häufig durch

die Anwendungen selbst, die oft eine deutliche „Lokalität“ in ihren Daten und Instruktionen aufweisen. Diese Lokalität wird u. a. auch besonders von schnellen Zwischenspeichern, den so genannten Caches (vgl. Abschnitt 4.4.2) ausgenutzt. Man unterscheidet zwischen *räumlicher* und *temporaler Lokalität*. Unter räumlicher Lokalität werden Zugriffe auf Daten bzw. Instruktionen verstanden, die in gleichen bzw. benachbarten Speicherbereichen liegen. Temporale Lokalität hingegen bezieht sich auf zeitlich nah aufeinanderfolgende Zugriffe auf gleiche Daten bzw. Instruktionen. Abbildung 2-15 gibt Aufschluss über Kennwerte unterschiedlicher Speichervarianten und die sich hieraus ergebende Speicher-Hierarchie für Prozessorsysteme.

Algorithmen sollten, wenn möglich, die Lokalität der Daten und Instruktionen ausnutzen bzw. so implementiert werden, dass möglichst effizient mit der vorgegebenen Speicher-Hierarchie gearbeitet werden kann. Z. B. kann die Verwendung mehrerer Threads helfen, Speicherlatenzen zu verdecken. Bei Multiprozessorsystemen ist normalerweise Kommunikation zwischen den Übergängen der einzelnen Hierarchiestufen bzw. zwischen verschiedenen Verarbeitungseinheiten notwendig. Für den Algorithmus ist in dieser Situation wichtig, abwägen zu können, wie teuer, im Sinne von Takten, die unterschiedlichen Speicherzugriffe bzw. die Kommunikation zu anderen Verarbeitungseinheiten und der Zugriff auf deren Speicher kommt. Das bedeutet, dass eine genauere Charakterisierung dieser Zugriffszeiten für die spätere Anwendung und deren Realisierung von entscheidender Bedeutung sein kann. Allerdings sind hier Mittelwerte anzunehmen, da z. B. bei wahlfreiem Zugriff konkurrierende Anfragen mehrerer Verarbeitungseinheiten auftreten können. Vorteilhaft sind hier Simulationen im Vorfeld mit der Möglichkeit der Parametrisierung der einzelnen Komponenten unter Anwendung der vorgesehenen Zielapplikation, vgl. Kapitel 5. Ein auf diese Belange eingestelltes Programmiermodell (vgl. Abschnitt 4.5) kann ebenfalls die Performanz des Gesamtsystems deutlich optimieren.

### 2.5.1 Wesentliche Charakteristika von Speicherstrukturen

Neben der in der Speicherhierarchie vorgestellten Einteilung in primäre, sekundäre und tertiäre Speicher, die durch die Distanz zur Verarbeitungseinheit definiert sind, gibt es weitere wichtige Merkmale für Speicher.

Man unterscheidet zwischen *flüchtigem* und *nicht-flüchtigem Speicher*. Eine Eigenschaft, die besagt, ob der Inhalt des Speichers erhalten bleibt, wenn er nicht mehr mit Spannung versorgt wird. Mit den immer geringer werdenden Zugriffszeiten auch für nicht-flüchtige Bausteine finden diese immer mehr Verwendung bei eingebetteten Systemen. Flüchtiger Speicher wird normalerweise nur für Primärspeicher eingesetzt, da er eine hohe Performanz bietet, allerdings auch permanent mit Spannung versorgt werden muss. Flüchtiger Speicher wiederum lässt sich unterteilen in die zwei Hauptgruppen, den statischen und den dynamischen Speicher. Statischer Speicher wird normalerweise mit sechs Transistoren pro Bit realisiert und muss durchgängig mit Spannung versorgt werden. Dynamischer Speicher hingegen kann mit einem Transistor und einer Kapazität als speicherndem Element pro Bit realisiert werden. Bei diesem Speichertyp müssen allerdings die Speicherstellen periodisch aufgefrischt werden, was einen zusätzlichen Aufwand an Kontrolllogik bedeutet.

Eine weitere Eigenschaft ist die Art und Weise, wie auf die Daten zugegriffen werden kann: Es wird grundsätzlich zwischen wahlfreiem und sequentielltem Zugriff unterschieden. Bei wahlfreiem Zugriff ist es möglich, beliebig oder in einem sehr großzügigen Rahmen auf Speicherzellen zuzugreifen. Bei sequentielltem Zugriff hingegen kann der Zugriff nur kontinuierlich in einer geordneten

Reihenfolge erfolgen. Ein weiteres Merkmal ist die Möglichkeit der Informationsänderung. Handelt es sich um einen reinen Lesespeicher, *ROM (Read Only Memory)*, einen einmalig beschreibbaren Speicher, *Write Once Read Many (WORM)*, oder aber einen einmalig programmierbaren Speicher (*One-Time-Programmable / OTP*), der z. B. schon bei der Fertigung programmiert wird. Speicher, der wahlfreien Zugriff gestattet, *RAM (Random Access Memory)*, wird zusätzlich nach seiner Technologie bzw. Realisierungsform näher unterschieden: *SRAM, DRAM, DDRAM, VRAM* und weitere. Die Möglichkeiten der Adressierung sind ebenfalls von Bedeutung. So kann zwischen numerisch adressiertem Speicher, bei dem jede Information mit einer numerischen Adresse erreichbar ist, und inhaltsbasierter Adressierung unterschieden werden. Bei der inhaltsbasierten Adressierung, auch *Content Addressable Memories (CAM)* genannt, werden für die Speicherinhalte mittels einer Streuwertfunktion, auch als *Hash-Funktion* bezeichnet, fingerabdruckähnliche Identifizierungsmerkmale mit geringem Speicherbedarf erzeugt, mit deren Hilfe später die abgespeicherte Information wieder abgerufen werden kann. Diese Art der Adressierung ist besonders für spezielle Speicher in Netzwerkkomponenten von großer Bedeutung (vgl. Abschnitt 6.5). Die Realisierung kann in Software unter Verwendung von herkömmlichem Speicher erfolgen. Diese Variante ist kostengünstig, aber langsam. Oder aber die Funktionalität wird durch Implementierung spezieller Hardware, die vor die Speicherzellen vorgeschaltet ist, realisiert, welches deutlich aufwändiger, aber auch wesentlich schneller ist. Eine weitere Möglichkeit ist die Überlagerung der numerischen, maschinenlesbaren Adressierung durch menschenlesbare Zuordnungen, wie es z. B. bei Dateisystemen üblich ist, die ggf. durch ein Betriebssystem gepflegt werden.

Neben den bisher genannten Methoden und Technologien lassen sich Speicher auch durch messbare, qualitativ fassbare Parameter definieren:

Die *Speicherkapazität*  $C_M$  gibt die Gesamtheit der zu speichernden bzw. abrufbaren Information des Speichers in Bit bzw. Byte und deren Vielfachen an. Ein *Bit* ist die atomare Informationseinheit in der Digitaltechnik und repräsentiert die zweiwertige Logik durch eine logische „0“ bzw. „1“. Ein *Nibble* besteht aus vier Bits, und das *Byte* setzt sich aus zwei *Nibbles* zusammen. Die Definition, dass ein Byte acht Bits umfasst, gilt für alle IBM-PCs und deren Nachfolger. Grundsätzlich bezeichnet ein Byte die Anzahl an Bits, die notwendig sind, um ein Symbol des Basis-Symbolvorrats des Systems darzustellen. Die Notation größerer Speichermengen erfolgt häufig nicht SI-konform durch dezimale Vielfache (k, M, G, etc.), sondern durch Vielfache von Zweierpotenzen ( $2^{10} = 1024 \hat{=} K$ ,  $2^{20} \hat{=} M$  etc.).

Die *Speicherdichte*  $\rho_M$  wird angegeben in *Speicherkapazität/Fläche*, also Bit/mm<sup>2</sup>. Sie ist ein gutes Maß, um die Flächenintensität verschiedener Speichervarianten abzuschätzen.

Die *Latenz*  $L_M$  ist das Maß für die Zeitspanne, die benötigt wird, um die Informationen aus einer bestimmten Speicherzelle auszulesen bzw. Daten in diese Speicherzelle zu schreiben. Aufgrund des technologiebedingten, teilweise unterschiedlichen Zeitverhaltens von Speichern beim Lesen bzw. Schreiben ist sinnvollerweise zwischen Lese-Latenz  $L_{M(R)}$  und Schreib-Latenz  $L_{M(W)}$  zu unterscheiden. Ebenso sind *minimale, maximale* und *durchschnittliche Latenz* (*min / max / avg*  $L_{M(R/W)}$ ) besonders bei sequentiellen Speichermedien als kennzeichnendes Leistungsmerkmal zu nennen.

Der *Durchsatz*  $D_M$  gibt die zur Verfügung gestellte Datenmenge in Byte pro Sekunde [B/s] an. Auch hier wird in Abhängigkeit von der Zugriffsart ebenso wie bei der Latenz differenziert.

### 2.5.2 Anforderungen an eingebettete Speicher

Offensichtlich gibt es zwischen vielen der Anforderungen proportionale aber auch antiproportionale Wechselwirkungen. So erhöht eine deutlich größere Kapazität die Kosten, und ein höherer Durchsatz, z. B. hervorgerufen durch eine höhere Taktfrequenz, vergrößert die Leistungsaufnahme. Diese konkurrierenden Anforderungen können durch Wahl einer anderen Technologie bzw. Speicherform ggf. umgangen bzw. verringert werden. Allerdings sind dies zumeist Kompromisslösungen, bei denen nie alle Ziele optimal erreicht werden können. Die Performanz eines Speichers wird im Allgemeinen durch seine Latenz und den erzielbaren Durchsatz definiert. Die Kosten bzw. der Preis gehen einher mit der Speicherkapazität, der dafür benötigten Fläche und der verwendeten Technologie. Die Leistungsaufnahme hängt, wie bereits erwähnt, von zahlreichen Faktoren ab, wobei für viele Einsatzgebiete zusätzliche Funktionen wie z. B. flexibler Zugriff auf Speicherinhalte bestimmter Größe, Fehlererkennungs- und Fehlerkorrekturmechanismen sowie Stromspar-Modi von besonderer Bedeutung sein können. Letztendlich spielen auch Eigenschaften wie Integrationsaufwand, also die Existenz genormter Schnittstellen, bzw. eine genaue Spezifikation des benötigten Zeitverhaltens eine Rolle. Eine Zusammenfassung der wesentlichen Anforderungen an eingebettete Speicher im Kontext von Chip-Multiprozessoren wird in Abbildung 2-16, Abschnitt 2.7 gegeben.

## 2.6 Anwendungsgebiete von On-Chip-Parallelrechnern

Setzt man aus den in den vorherigen Abschnitten diskutierten Kernkomponenten *On-Chip-Netzwerk*, *eingebettete Verarbeitungseinheiten* und *Speicher*, baukastenartig Systeme mit mehreren Prozessorkernen zusammen und integriert diese auf einem Siliziumträger, so erhält man **On-Chip-Parallelrechner** oder auch **Chip-Multiprozessoren (CMP)**.

Heutige Anwendungen erfordern aufgrund wachsender Anforderungen und extrem rechenintensiver Algorithmen in vielen Bereichen bereits parallele Verarbeitung. Dabei gestalten sich die Anwendungsgebiete entgegen der weitläufig verbreiteten Meinung sehr vielfältig und beschränken sich nicht nur auf Wissenschaft und Forschung, wie es vor einer Dekade noch vorwiegend der Fall war. Zu den herausfordernden Einsatzgebieten für parallele Verarbeitung zählen u. a.: Berechnung globaler Klimamodelle, Crashtest-Simulationen, dreidimensionale Modellierung auf Basis finiter Elemente, Erdbebenvorhersage, Genforschung, militärische Forschung, quantentechnische Simulationen, Weltraumforschung, Wirtschaftsanalysen und medizinische Forschung allgemein. All diese Anwendungsszenarien kommen wie erwartet aus den Bereichen Wissenschaft und Forschung. Immer stärker jedoch treten Gebiete aus alltäglichen Lebensbereichen in Erscheinung und fordern immensen Zuwachs an Rechenleistung. Zu diesen „neuen“ potentiellen Einsatzgebieten paralleler Rechensysteme, die häufig kompakte, SoC-basierte Realisierungen erfordern, zählen: Computerspiele und Computergrafik allgemein, Multimedia-Anwendungen allgemein (z. B. Videobearbeitung in Echtzeit, MPEG4 etc.), Physikbeschleuniger zur Simulation komplexer physikalischer Effekte (z. B. in 3D-Spielen), Spracherkennung zur Computer-Maschinensteuerung etc. [71], Virtualisierung (Emulation vieler / verschiedener Betriebssysteme auf einer Hardware z. B. bei *Web-Hostern*), *World-Wide-Web*-basierte Suchmaschinen, massiv parallele Datenverarbeitung im Sinne von Netzwerkanwendungen (z. B. *Voice-over-IP*, *Tripple Play*, *Home-Video-Entertainment* etc.). Diese Bereiche haben mittlerweile eine weitaus größere Marktmacht als die oben genannten Forschungsgebiete und sind somit bereits als treibende Kräfte für die Entwicklung leistungsfähiger Parallelrechnerarchitekturen zu sehen. Dies wird u. a. durch die Entwicklung des bereits anfangs dieses Kapitels



erwähnten Cell-Prozessors deutlich, der hauptsächlich für den Home-Entertainmentbereich z. B. in Spielekonsolen hergestellt wird. Die Tatsache, dass mit ihm auch Supercomputer realisiert werden können ist aus kommerzieller Sicht zweitrangig.

Die Fertigung von SoCs in Nanometertechnologie wird immer teurer und die *NRE(Non-Recurring-Engineering)*-Kosten für solche Entwürfe übersteigen die Millionen-Euro-Grenze bei weitem. Die Einsetzbarkeit einer CMP-Architektur für den Massenmarkt und für viele Bereiche dieses Marktes sind somit für einen wirtschaftlichen Erfolg zwingend notwendig. Skalierbarkeit, Wiederverwendbarkeit und Flexibilität gepaart mit angemessener Leistung sind hier ausschlaggebende Kriterien. Die weltweite Vernetzung hält mehr und mehr Einzug in unser tägliches Leben, und der damit verbundene Informationsaustausch sowie die damit verbundene Informationsverarbeitung sind wesentliche Treiber für SoC-Designs. Folglich erscheint dieses Anwendungsgebiet als prädestiniert für massiv-parallele Systeme. In Kapitel 7 wird deshalb die in dieser Arbeit entworfene GigaNetIC-Architektur speziell für Netzwerkanwendungen analysiert und optimiert.

## 2.7 Anforderungen an Chip-Multiprozessoren

Im Folgenden werden zusammenfassend die wesentlichen Anforderungsmerkmale an Chip-Multiprozessoren aufgezeigt (vgl. Abbildung 2-16). Die dargestellte Merkmalsmatrix setzt allgemeine Anforderungen an Schaltungsentwürfe sowie die speziellen Charakteristika der zuvor diskutierten Kernkomponenten (On-Chip-Netzwerk / NoC, Verarbeitungseinheiten / PEs und Speicher / Mems) und des resultierenden Chip-Multiprozessors (CMP) mit einander in Beziehung.

Die bilateralen Abhängigkeiten der Merkmale untereinander werden dabei in fünf Kategorien unterteilt. Bei Änderung eines Merkmals kann sich dies proportional, antiproportional oder auch unbestimmt auf ein anderes Merkmal auswirken. Bei der Proportionalität wird zwischen zumeist proportional/antiproportional und proportional/antiproportional unterschieden. Diese Abstufung differenziert so zwischen zwei Qualitäten. Bei einer zumeist proportionalen/antiproportionalen Beziehung wird eine Tendenz angegeben, wohingegen bei proportionalen/antiproportionalen Zusammenhang in der Regel die Aussage stets zutrifft<sup>5</sup>. Beispielsweise bedeutet ein höherer Durchsatz mehr Performanz und steht somit in proportionaler Beziehung mit diesem Merkmal. Besteht der gleiche Zusammenhang bei umgekehrter Reihenfolge der Merkmale (Vertauschung von Ursache und Wirkung) ebenfalls, so wird aus Gründen der Übersichtlichkeit das entsprechende Symbol weggelassen. Die Felder in der rechten oberen Hälfte der Matrix sind dann grau unterlegt und „identisch“ zu ihren Pendant in der linken unteren Hälfte der Matrix. Bei einer nicht kommutativen Beziehung der Merkmale<sup>6</sup> werden beide Hälften der Matrix zur Kennzeichnung genutzt. So bedingt z. B. ein höherer Preis nicht zwangsläufig eine höhere Performanz aber aufgrund höherer Performanz lässt sich am Markt ein höherer Preis vertreten.

---

<sup>5</sup> Abbildung 2-16 zeigt die typischen Beziehungen die im allgemeinen gelten, allerdings lassen sich immer Ausnahmeszenarien finden, für die einige Gewichtungen anders formuliert werden können.

<sup>6</sup> Siehe auch grafische Erläuterung in der Grafik selbst im linken oberen Bereich.



Ein interessanter Ausspruch von HALFHILL, Microprocessor Report, lautet: „The Key to Massive Parallelism: Think Small“ [72]. HALFHILL sieht dies im Zusammenhang mit dem Anwendungsgebiet des Systems. Die Geschichte habe gezeigt, dass massiv parallele Systeme stets erfolgreich waren, wenn sie für eng eingegrenzte Problemstellungen verwendet wurden, anstatt den Anspruch der Universalität verfolgt zu haben. Bei der GigaNetIC-Architektur wird dieses Prinzip in gewisser Weise zweifach verfolgt, ohne jedoch den Anspruch als universelle Chip-Multiprozessor-Architektur aufzugeben. Zum einen wird zunächst ein universell einsetzbares System entworfen (vgl. Kapitel 4), das dann durch die speziell entwickelte Werkzeugkette auf eine dedizierte Anwendung optimiert werden kann (vgl. Kapitel 5 und 6). Zum anderen wird bei der Architekturkonzeption auf überschaubare Blöcke mittlerer Komplexität geachtet. Der vorgesehene Prozessorkern ist absichtlich „klein“ im Sinne von Befehlssatz, Pipelintiefe, Sprungvorhersage und spekulativer Ausführung etc. gehalten, um die Grundstruktur des Parallelsystems nicht zu überladen. Zusätzlich benötigte Funktionen oder Hardwarebeschleuniger lassen sich vor Fertigstellung des Chips leicht integrieren, vgl. Kapitel 6 und 7.

## 2.8 Varianten eingebetteter paralleler Rechnerarchitekturen

Im Folgenden werden ausgewählte Ansätze für CMPs vorgestellt. Kann dieser Überblick zwar nicht den Anspruch an Vollständigkeit erheben, da aufgrund der neuesten Paradigmenwechsel im Bereich von Prozessorstrukturen eine Vielzahl von Forschungsaktivitäten veröffentlicht wird, so gibt er doch einen repräsentativen Einblick in aktuelle Architekturen.

Chip-Multiprozessoren werden häufig auch als Single-Chip-Multiprozessoren bezeichnet, da sie mehrere eigenständige Prozessoren auf einen Chip vereinen. Hierbei lassen sich verschiedene Organisationsformen bezüglich der Speicherorganisation und der Kommunikationsmöglichkeiten (*Interconnection*) der einzelnen Prozessoren untereinander unterscheiden. Die geläufigsten Organisationsformen von Chip-Multiprozessoren sind der *Symmetric Multiprocessor (SMP)*, der *Distributed Shared Memory Multiprocessor (DSM)* und der *Message-Passing Shared-Nothing Multiprocessor* (vgl. [73]). Beim *SMP* und *DSM* teilen sich die Prozessorelemente einen gemeinsamen Adressraum.

Der *SMP* verfügt über einen globalen Hauptspeicher, der von allen Prozessoren gemeinsam genutzt wird. Die Speicherzugriffszeit ist für jede Adresse und für jeden Prozessor gleich, weshalb der Zugriff auch als *Uniform Memory Access (UMA)* bezeichnet wird.

Beim *DSM* ist dieser Hauptspeicher auf die einzelnen Prozessoren verteilt, so dass jeder einen lokalen Speicher besitzt, der aber in einen globalen Adressraum eingegliedert ist. Hierbei sind allerdings die unterschiedlichen Zugriffszeiten zu berücksichtigen, denn lokaler Speicher kann schneller erreicht werden als ein Speichersegment, das zu einem fremden Prozessorelement gehört. Man spricht hier auch von *Nonuniform Memory Access (NUMA)*.

*Shared-Nothing-Prozessoren* haben keinen gemeinsamen Adressraum und der Arbeitsspeicher ist physikalisch auf die einzelnen Prozessorelemente verteilt. Deshalb können die einzelnen Prozessoren nur über das *Message-Passing-Verfahren* miteinander kommunizieren. Diese Art von Multiprozessor lässt sich aufgrund ihrer regelmäßigen Struktur sehr hoch integrieren, allerdings schwerer programmieren als die Multiprozessoren, die sich den Adressraum teilen. Abbildung 2-17 führt die

besprochenen Architekturen kurz ein, berücksichtigt jedoch nicht die Cache-Organisation der einzelnen Prozessor-Elemente.

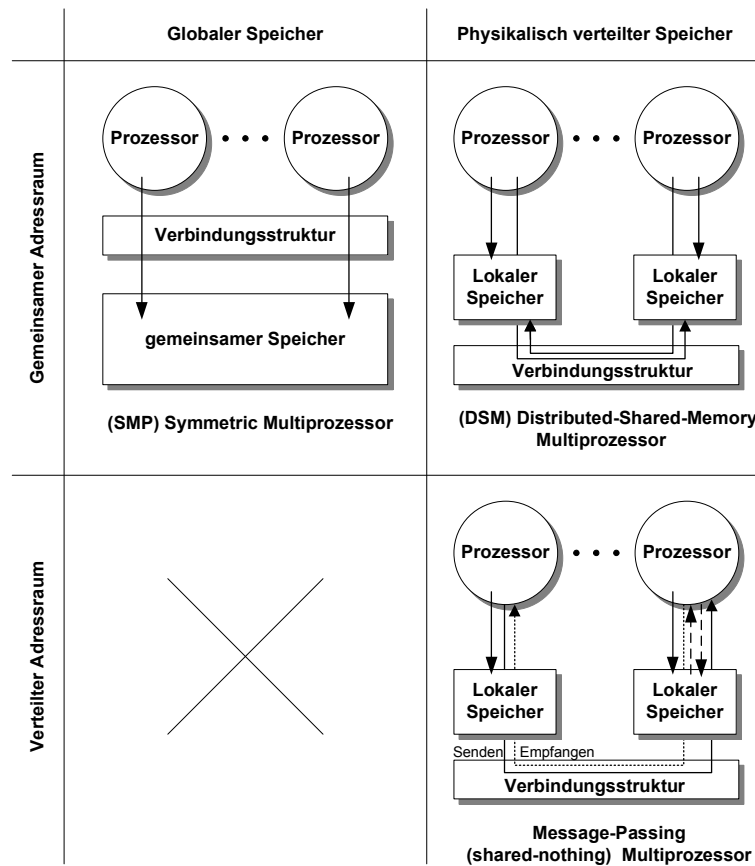


Abbildung 2-17: Organisationsformen von Multiprozessoren (vgl. [73])

Bei CMPs werden bzgl. der Parallelität folgende *Grain-Level* (Granularitätsstufen) unterschieden:

- mehrere Prozesse bzw. Anwendungen, die parallel abgearbeitet werden,
- mehrere Threads, die zu einer Anwendung gehören und die parallel abgearbeitet werden,
- Threads, die aus einem sequentiellen Programm extrahiert werden.

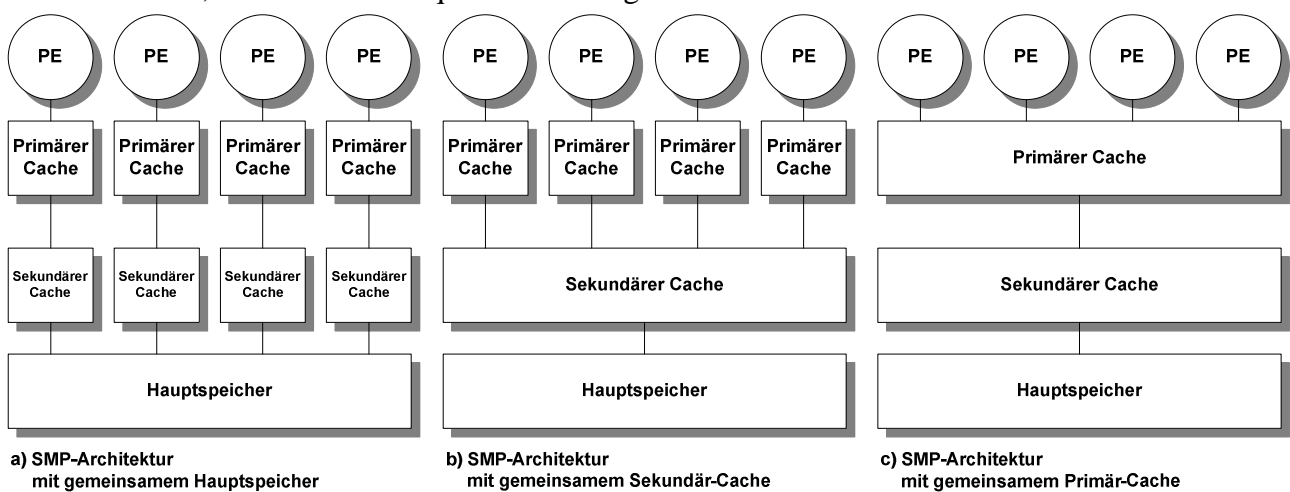


Abbildung 2-18: Typische Varianten von SMP-Architekturen im Überblick (vgl. [73])

In Abbildung 2-18 werden einige typische Implementierungsformen von *Shared Memory-CMPs* gezeigt. Je nach Implementierung kann sich die gemeinsame Nutzung auf den Hauptspeicher beschränken oder aber bis hin zur gemeinsamen Benutzung des Primär-Caches gehen.

### 2.8.1 Beispiele zu Chip-Multiprozessoren

Selbstverständlich sprengte eine „enzyklopädische“ Gesamtschau aller existierenden Chip-Multiprozessoren den Rahmen dieser Arbeit. Vielmehr sollen, mit Hilfe der hier getroffenen Auswahl, repräsentative Beispiele geliefert werden, die ganz besondere Charakteristika von CMPs aufweisen. Diese lassen sich teilweise so oder ähnlich in der entworfenen GigaNetIC-Architektur wiederfinden. Wie bereits in Abbildung 2-6 aufgezeigt, kann die Konzeption eines Chip-Multiprozessors nach einem Baukastenprinzip erfolgen, bei dem jeweils aufeinander angepasste Komponenten das Gesamtsystem formen. Wesentlich ist, dass hier nicht die Performanz einer einzelnen Kernkomponente über den Erfolg entscheidet, sondern das Zusammenspiel aller Elemente die Leistungsfähigkeit der Architektur definiert.

Zahlreiche Ansätze aus der Literatur verwenden *busbasierte Architekturvarianten*, vgl. Abbildung 2-7 a). Zu diesen Architekturen gehört z. B. die ATLAS Chip-Multiprozessorarchitektur [74]. OLUKOTUN et. al verfolgen mit ihrem vierfach-parallelen Zwei-Wege-CMP ein ähnliches Konzept [75]. Hier fungiert allerdings ein Kreuzschienenverteiler (ähnlich Abbildung 2-7 b)) als Schnittstelle zum gemeinsamen Cache und würde für eine größere Anzahl von Verarbeitungseinheiten einen Flaschenhals bedeuten. CMPs, bei denen die Verarbeitungseinheiten über nicht-hierarchische Topologien, sondern basierend auf Bussen, Multiplexern oder Kreuzschienenverteilern miteinander verbunden sind, sind nur als vorübergehende Lösungen zu sehen. Diese Topologievarianten ohne Hierarchie skalieren im Sinne von Definition 35 nicht und eignen sich nicht für massiv parallele eingebettete Systeme, sondern erlauben nur die effiziente Integration einiger weniger Verarbeitungseinheiten, siehe auch [76]. In [77] wird ein Ansatz gezeigt, wie busbasierte CMP-Ansätze mit Hilfe eines überlagerten „Butterfly-Fat-Tree“-Netzwerks, also einer zusätzlichen Hierarchie, zu *skalierbaren System-on-Chip-Architekturen* erweitert werden können. Inwieweit dies allerdings für reale Systeme umsetzbar ist, wurde noch nicht gezeigt.

Viele der in der Literatur untersuchten CMP-Architekturen sind noch nicht in Hardware realisiert, sondern wurden auf Basis von Simulationen und Abschätzungen untersucht, so dass eine abschließende Verifikation dieser Systeme noch aussteht.

### 2.8.2 Ansätze für Chip-Multiprozessoren mit akademischem Ursprung

**Raw, MIT.** Der Raw-Chip-Multiprozessor unterteilt sich in 16 gleichförmige Kacheln, die aus einem MIPS-basierten Prozessor mit acht Pipelinestufen, einem programmierbaren Routerblock, einer vierstufigen Fließpunkt-Einheit, 32 KByte Datencache und 96 KByte Instruktionscache bestehen. Der Gesamtentwurf umfasst 122 Millionen Transistoren und ermöglicht eine Taktfrequenz von 225 MHz in einer 150-nm-Technologie. Die 16 Kacheln beanspruchen insgesamt 331 mm<sup>2</sup>. Die Leistungsaufnahme wird mit 25 W abgeschätzt [78][79]. Die in vier Richtungen gehenden Busse zur Verbindung der Nachbarkacheln ermöglichen die Übertragung der Daten innerhalb eines Taktes. Die maximale Länge der Verbindungsleitungen zwischen den Nachbarkacheln entspricht genau der Kantenlänge einer Kachel. *Global Wires* werden somit kurz gehalten. Die Verbindungsstruktur bietet zwei statische und zwei dynamische (*Wormhole-Switching*-basierte) Routen zu den jeweiligen Nachbarkacheln. Die Steuerung des On-Chip-Netzwerks wird komplett in Software gelöst und erlaubt dem Programmierer oder dem Compiler so größtmögliche Realisierungsfreiheit. Das Routing des Ergebnisses einer Arithmetik-Berechnung zur nächsten Kachel benötigt drei Takte. Teile der Netzwerkverbindungen sind direkt in die Pipeline des jeweiligen Prozessors eingebunden und

ermöglichen so direkten Zugriff seitens des Prozessors auf Netzwerkressourcen. Der Raw-Prozessor ist in Hochsprachen wie C oder Java programmierbar und stellt einen eigenen Compiler zur Verfügung. Extern lassen sich bis zu 64 Raw-Chips in beliebiger rechteckiger Anordnung kombinieren und ermöglichen so ein Konglomerat von insgesamt 1024 Kacheln bzw. Einzelprozessoren.

**Hydra, Stanford.** Der Hydra-Chip-Multiprozessor ist im Rahmen eines Forschungsprojekts der Universität Stanford entstanden [80]. Er umfasst vier MIPS-basierte Prozessorkerne mit eigenem Level-1-Daten- und Instruktions-Cache und gemeinsamem Level-2-Cache. Die Verbindung ist busbasiert. Die Autoren skizzieren eine maximale Ausbaumöglichkeit des Systems von bis zu acht Prozessoren, wobei jeder dieser Prozessorkerne einen direkt gekoppelten Coprozessor ansteuern kann. Sie geben allerdings zu bedenken, dass für eine größere Anzahl von Prozessoren andere, z. B. hierarchische Verbindungsstrukturen zu implementieren wären. Die benötigte Fläche in einer 250-nm-Technologie mit vier Prozessorkernen wird mit 90 mm<sup>2</sup> angegeben, mit je 8 KByte Daten- und Instruktioncaches und 128 KByte gemeinsamem Cache. Die angestrebte Taktfrequenz wird mit 250 MHz angegeben.

**Daytona Multiprozessor DSP.** Der Daytona ist ein Vierfach-MIMD(*Multiple Instruction Multiple Data*)-DSP mit vier 64-Bit-Verarbeitungseinheiten [81]. Die vier 32-Bit-RISC-Prozessorkerne vom Typ Sparc V8 mit einer fünfstufigen Pipeline verfügen über DSP-Erweiterungen sowie je eine enggekoppelte Coprozessoreinheit. Die Verarbeitungseinheiten sind über rekonfigurierbare Level-1-Caches an einen *Split-Transaction-Bus (STBus)* angeschlossen. Als Cache-Kohärenzprotokoll wird ein modifiziertes MESI-Protokoll verwendet (vgl. Abschnitt 4.4.2). Ein eingebettetes *RTOS (Real Time Operating System / Echtzeitbetriebssystem)* übernimmt die Einteilung der Verarbeitungseinheiten auf anfallende Aufgaben. Eine Synchronisierung der Prozesse wird über Semaphore erreicht. Der 200 mm<sup>2</sup> große, in 250-nm-Technologie gefertigte Chip arbeitet mit 100 MHz und nimmt ca. 4 W auf.

**PipeRench, Carnegie Mellon University.** Die PipeRench-Architektur ist eine rekonfigurierbare Architektur, die ohne spezielle Hardwareerweiterungen an anwendungsspezifische Probleme angepasst werden kann [82]. Allerdings ist laut [82] der Ansatz hier anders als bei herkömmlichen FPGAs (*Field Programmable Gate Arrays*). PipeRench ist speziell für Berechnungen ausgelegt, die durch die „*Pipeline Reconfiguration*“-Technik besonders unterstützt werden. Hierunter wird eine Rekonfiguration der einzelnen Pipelinestufen verstanden, bei  $p$  physikalisch vorhandenen Stufen werden  $v$  Stufen emuliert, mit  $p < v$ . Die Konfigurationszeit beeinflusst die Verarbeitung nicht nachteilig und geschieht innerhalb eines Taktes. Die globale Verbindungsstruktur ist busbasiert. Die einzelnen Verarbeitungseinheiten sind acht Bit breit, können aber zu größeren Bitbreiten kombiniert werden. Wesentliche Bestandteile der Architektur sind die globalen Busse, die in Streifen angeordneten Verarbeitungseinheiten und Übergaberegister. Das System umfasst 256 Verarbeitungseinheiten, die in 16 Streifen kaskadierbar angeordnet sind, und lässt sich in C, mit Hilfe des mitgelieferten Compilers, programmieren.

Bevor sich die Diskussion der Merkmale der vorgestellten Chip-Multiprozessoren anschließt, sollen im Anschluss an die geschilderten Ansätze akademischer Herkunft nun weitere Ansätze für Chip-Multiprozessoren aus der Industrie charakterisiert werden, die ebenfalls zum Vergleich mit der GigaNetIC-Architektur herangezogen werden.

### 2.8.3 Ansätze für Chip-Multiprozessoren aus der Industrie

Um einen Überblick über die Anstrengungen, die im Bereich von Chip-Multiprozessoren unternommen werden, zu geben, reicht es nicht aus, nur Ansätze aus dem Bereich der akademischen Forschung zu beleuchten. Einfluss auf zukünftige Architekturen (vgl. Definition 3) werden in nächster Zeit besonders die ausgewählten Beispiele der großen, etablierten Universal-Prozessorhersteller wie Intel, AMD und IBM nehmen. Diese Unternehmen verfügen über das Kapital und die Humanressourcen, um solche komplexen und extrem kostenaufwändigen Entwicklungen zu realisieren. Die Produktstrategien bzw. „Roadmaps“ und Architekturvarianten dieser und anderer relevanter Hersteller im Zusammenhang mit CMPs werden im Folgenden kurz vorgestellt.

**DRP, NEC.** NEC hat mit dem *Dynamically Reconfigurable Processor* ein synchrones Prozessorfeld bestehend aus bis zu 512 Verarbeitungseinheiten entwickelt [83]. Der DRP verfügt über eine 8-Bit-Genauigkeit und ist in 150-nm-Technologie gefertigt und mit 133 MHz zu betreiben. Seine Datenpfade sind innerhalb eines Taktes umkonfigurierbar. Sein Haupteinsatzgebiet sind Bereiche, in denen bisher DSPs (Digitale Signalverarbeitungs-Prozessoren) verwendet wurden, er lässt sich aber auch als Netzwerkprozessor nutzen.

**XPU128 / XPP III, PACT.** Die XPU- und XPP-Serie von PACT sind rekonfigurierbare Felder mit einfachen Arithmetikeinheiten (ALUs), die sich besonders für *Stream*-Anwendungen, also z. B. Videoverarbeitung eignen. Die PACT-Architekturen verfügen über ein effizientes Rekonfigurationsmanagement und lassen sich so auf verschiedene Anwendungsgebiete optimieren. Horizontal geschieht die Punkt-zu-Punkt-Verbindung der ALUs über Routingbusse, vertikal sind die ALUs direkt miteinander verbunden. Zusätzlich sind dedizierte Verbindungen der einzelnen Einheiten vorgesehen. Die Datenbreite kann zwischen 16, 24 und 32 Bit gewählt werden. Die Datenübertragung innerhalb des Chips geschieht über das paketorientierte On-Chip-Netzwerk. Die XPU128 war im Jahr 2000 der erste einsatzbereite Chip der Firma PACT und umfasste bereits 128 ALU-Verarbeitungseinheiten, die mit angestrebten 100 MHz in 150-nm-Technologie betrieben werden konnten. Die XPP-III-Architektur ist der kommerziell eingesetzte, in C programmierbare Nachfolger [84][85].

**Piranha, Compaq.** Die in der Forschung von Compaq entwickelte Piranha-Architektur [86] beschreibt eine achtfach-parallele Struktur mit einfachen Alpha-Prozessorkernen, die über jeweils einen eigenen Daten- und Instruktions-Cache mit je 64 KByte verfügen und sich einen Level-2-Cache teilen, mit dem sie über einen so genannten *Intra-Chip-Switch* verbunden sind. Die Prozessoren sollen in einer 180 nm-Technologie mit 500 MHz betrieben werden können. Zur weiteren Erhöhung der CPU-Anzahl verfügt das System über einen integrierten Router mit zwei Protokoll-Einheiten, der den Anschluss weiterer Piranha-Chips übernimmt. Die von Compaq verfolgte Entwurfsmethodik begann mit einer Spezifizierung des Systems in C++, dem eine Verfeinerung der einzelnen Blöcke in der Hardwarebeschreibungssprache Verilog folgte.

**KiloCore, Rapport.** Die derzeit aktuelle Chip-Variante KC256 von Rapport basiert auf Rapports KiloCore-Architektur. Er umfasst 256 Prozessorkerne, die mit 8-Bit-Daten- und Befehlsbreite arbeiten und ist in 180-nm-Technologie realisiert [87]. Die maximal erreichbare Betriebsfrequenz wird mit bis zu 125 MHz angegeben. Die Leistungsaufnahme des KC256 soll Rapport zufolge unter 500 mW liegen. Der 1025 Kerne umfassende KiloCore soll 2007 in 90-nm-Technologie realisiert werden. Er wird über 1024 8-Bit-Verarbeitungseinheiten verfügen, die von einem PowerPC-Kern kontrolliert werden. Die Architektur soll außerdem rekonfigurierbar sein, wobei die Rekonfigurati-

on nur einen Takt benötigt. Über die Kommunikationsstruktur sind noch keine Details bekannt, allerdings lässt sich eine überlagerte Busstruktur anhand der Blockdiagramme vermuten. Einsatzgebiete sollen u. a. Videoverarbeitungsalgorithmen sein, bei denen er um den Faktor zehn schneller sein soll als aktuelle Prozessoren.

**GPUs, nVIDIA und ATI.** nVIDIA und ATI sind namhafte Hersteller für Grafikchips für PCs und stellen bereits seit mehr als einer Dekade massiv parallele Coprozessoren zur Grafikbeschleunigung her. Die neueste Generation der nVIDIA Grafikbeschleuniger, die GeForce 8800, integriert 128 so genannte „StreamProzessoren“, die bei Verwendung einer 90-nm-Technologie mit bis zu 1,35 GHz getaktet werden [88]. nVIDIA kooperiert mit Forschungsinstituten, die die nVIDIA-Grafikprozessoren für komplexe Berechnungen, z. B. für medizinische Zwecke einsetzen und dadurch eine Beschleunigung von mehr als 25, verglichen zu herkömmlichen CPUs, erreichen. ATIs neueste Consumer-Grafikkarten werden in 80- bzw. 65-nm-Technologie gefertigt und vereinen bis zu 720 Millionen Transistoren auf einem Chip, die unter anderem für 64 parallele komplexe 4-Wege-SIMD(*Single Instruction Multiple Data*)-Einheiten benötigt werden [89].

**PhysX, Ageia.** Ageia hat mit dem PhysX einen speziell für die Beschleunigung physikalischer Phänomene konzipierten Chip-Multiprozessor (Physikbeschleuniger) entwickelt, der über Dutzende von eingebetteten Prozessoren verfügt und vor allem für 3D-Computer-Spiele auf Coprozessor-Karten eingesetzt wird [90]. Der Chip soll über 125 Millionen Transistoren integrieren und eine *Die*-Größe von 182 mm<sup>2</sup> in 130-nm-Technologie haben. Die Leistungsaufnahme beschränkt sich auf 25 W.

**Intels Weg vom „Single Core“ über „Multi Cores“ zu „Many Cores“.** Intel hat seit der Einführung des ersten Ein-Chip-Mikroprozessors 1971, des Intel 4004, eine Flut von Verbesserungen und neuartigen Prozessor-Architekturen präsentiert. Mit steigender Taktfrequenz, beim Pentium 4 der Netburst-Architektur bis zu 3,8 GHz, stieg jedoch auch die Leistungsaufnahme trotz neuester Technologie bis auf 115 W (Intel Pentium 4, 672) weiter an. Dies bedeutete eine Stromaufnahme von 119 A, die von einem *Die* der Größe von nur 135 mm<sup>2</sup> (in 90 nm-Technologie, 165 Millionen Transistoren) aufgenommen werden musste. Die zuvor von Intel angestrebte maximale Frequenz von 10 GHz für diese Architektur konnte aus zahlreichen technischen Gründen nicht erreicht werden, so dass die Entwicklung bereits bei unter 4 GHz eingestellt wurde [3]. Das von nun an verfolgte Architektur-Paradigma lautete: Mehrere Prozessoren, „Multi Cores“, bei geringerer Taktfrequenz. Es wurde mit der Intel-Core-Architektur erfolgreich umgesetzt. Die Core-Architektur wird auch als achte x86-Architektur geführt. Die Mitte 2006 eingeführte Core-2-Prozessorlinie, in 65-nm-Technologie gefertigt und bis zu 2,93 GHz operabel, hat eine deutlich reduzierte Leistungsaufnahme von 65 W bei einer Chipfläche von 144 mm<sup>2</sup> und 291 Millionen Transistoren. Sie verfügt über zwei Prozessorkerne, 64 KByte Level-1-Cache und 4 MByte Level-2-Cache. Seit Anfang 2007 halten die ersten Vierkern-Prozessoren Einzug in Standard Desktop-PCs. Hierbei handelt es sich um die „Core 2 Quad“-Familie, bei der zwei einzelne Dies in einem Prozessorgehäuse untergebracht sind. Die Variante Q6600 arbeitet mit 2,4 GHz Kerntakt und beansprucht pro Die 143 mm<sup>2</sup> bei einer Strukturgröße von 65 nm. Die Leistungsaufnahme des gesamten Chips beziffert Intel zu 130 W. Der neuartige Cache wird jeweils von den beiden zugehörigen Verarbeitungseinheiten genutzt, wobei die Speicherzuteilung dynamisch geschieht. Außerdem können die beiden Kerne gegenseitig Daten über ihn austauschen, was verglichen mit Hauptspeicherzugriffen äußerst effizient ist. Auch wurde hinsichtlich der Reduktion der Leistungsaufnahme eine Vielzahl von Mechanismen einge-



baut, wie z. B. bereichsbezogene Spannungsvariation und *Clock Gating* sowie Abschaltung des Caches. Die Core-Architektur ist die erste vierfach-superskalare Intel-Architektur, die eine simultane Abarbeitung von vier Mikrooperationen ermöglicht.

**Intels** zukunftsweisendes **Tera-Scale-Projekt** zielt auf massiv parallele Systeme mit einigen hundert integrierten Prozessorkernen ab. Zu den bedeutendsten Anforderungen, die durch die Tera-Scale-Architektur erfüllt werden sollen, zählt Intel die einfache, effektive Programmierbarkeit („Programmability“) mit einem ausgereiften Programmiermodell und komfortabler Entwicklungsumgebung. Die nächste Anforderung ist „Adaptability“, d. h. die Plattform soll anpassbar an verschiedene Aufgaben und Lastverteilungen sein. Außerdem soll die Hardware an verschiedene Umgebungen in Bezug auf z. B. Leistungsaufnahme und Formfaktor anpassbar sein. „Reliability“ nennt Intel als weiteres Schlagwort. D. h. die Plattform soll trotz der großen Komplexität noch zuverlässiger als die bisherigen Entwürfe sein. „Trust“, also Vertrauen soll die Plattform erwecken, um so Kunden von dem neuen Konzept zu überzeugen. Abschließend wird „Scalability“, die Skalierbarkeit der Architektur in Bezug auf Hardware und Software als wesentliche Anforderung formuliert. Die sehr modular gehaltene Architektur integriert Hardwarebeschleuniger, optimierte Prozessorkerne und skalierbare On-Chip-Verbindungsstrukturen. Dieser Ansatz wird von Intel als vielversprechende Lösung für die Abdeckung vieler Märkte propagiert<sup>7</sup>. Als Treiber für den Kurswechsel in der Architektur gibt Intel Performanz, Leistungsaufnahme und kurze Entwicklungszyklen an. Intel stellte diesen neuen Ansatz erstmals 2006 der Öffentlichkeit vor [91]. Drei bzw. vier Jahre vorher postulierten wir in [6] und [14] bereits sehr ähnliche Ansätze. Hieran zeigt sich, dass der von uns damals beschrittene Weg zukunftsweisend ist.

**Intels** aktuelle 80-Prozessor-Architektur „Polaris“, die zunächst unabhängig vom Tera-Scale-Projekt entwickelt wurde, basiert auf einem Gitter mit  $8 \times 10$  identischen Verarbeitungseinheiten, wobei der einzelne Kern bis zu acht Instruktionen gleichzeitig verarbeiten kann [92]. Der lokale Speicher ist sehr klein bemessen. Jeder Kern verfügt über 2 KByte Daten- und 3 KByte Instruktionsspeicher. Die Anbindung an das On-Chip-Netzwerk übernimmt ein leistungsstarker Routingknoten. Er ermöglicht einen akkumulierten Durchsatz von 80 GByte/s<sup>8</sup> bei 4 GHz. Jeder Port ist dabei über je  $2 \times 39$  Signalleitungen mit dem Nachbarknoten verbunden. Die Struktur eines solchen Routers sieht vier Verbindungsschnittstellen (*Ports*) zu anderen Knoten vor und einen Port für die Verbindung eines zusätzlichen SRAMs. Ein Polaris-Kern benötigt ca. 1,2 Millionen Transistoren, die auf einer Fläche von 3 mm<sup>2</sup> in 65 nm-Technologie Platz finden. Die Recheneinheiten beinhalten eine neun-stufige Pipeline. Bei der Polaris-Architektur wurden zahlreiche Optimierungen wie z. B. *Clock Gating*, bezüglich der Leistungsaufnahme vorgenommen, so dass der gesamte Chip bei 3,16 GHz nur ca. 62 W aufnimmt. Erhöht man die Taktfrequenz auf 5,7 GHz, so nimmt die Leistungsaufnahme um mehr als das Vierfache auf 265 W zu. Auch hier zeigt sich der nichtlineare Zusammenhang zwischen Steigerung der Betriebsfrequenz und der resultierenden Leistungsaufnahme eines Chips. Der Polaris-Chip bietet auf der Fläche eines Daumennagels genauso viel Rechenleis-

---

<sup>7</sup> Intel arbeitet u. a. an Grafikchips mit dem Codenamen „Larrybee“, die über 16 eingebettete Prozessorkerne verfügen, die auf x86/SSE-Codebasis beruhen und den gleichen Befehlssatz wie die Tera-Scale-Architektur verwenden.

<sup>8</sup> Lt. Intel, nach bestehender Nomenklatur 74,51 GByte/s.

tung wie vor 11 Jahren in Dienst gestellte Supercomputer, die auf über 200 m<sup>2</sup> Fläche untergebracht werden mussten [92].

**Opteron, AMD.** AMD ist seit dem Zeitalter der 386er-Prozessorgeneration Intels größter Konkurrent, der mal schnellere und mal weniger effiziente Architekturansätze, verglichen mit Intel, hervorgebracht hat. Interessant ist, dass AMD eher als Intel in Bezug auf das Streben nach immer höheren Taktfrequenzen einen anderen Weg eingeschlagen hat und recht bald nicht mehr die tatsächliche Frequenz seiner CPUs als Verkaufsargument verwendete, sondern seine Prozessoren mit einer fiktiven Taktfrequenz bewarb, die mit der Leistung einer entsprechenden Intel-Architektur vergleichbar waren. AMD stellte 2007 die erste, auf einem *Die* integrierte Vierfach-Desktop-/Server-CPU, den Opteron Quad-Core-Prozessor vor [93]. Dieser arbeitet *on-chip* mit dem MOESI-Cachekohärenzprotokoll.

Mit der *Torrenza*-Initiative setzt AMD bei zukünftigen Prozessorgenerationen verstärkt auf eine Kopplung von wenigen Universalprozessorkernen und anwendungsspezifischen Hardwarebeschleunigern, um sich so gegenüber den von Intel propagierten *ManyCore*-Architekturen, zu behaupten, ein „Kern-Wettrüsten“ solle es vorerst nicht geben [93]. Allerdings setzt auch Intel auf so genannte *Fixed Function Units* [92], also ebenfalls auf Hardwarebeschleuniger, die spezielle Aufgaben effizienter lösen können als Universalprozessoren.

**Cell-Architektur, IBM, Toshiba und Sony.** Die Entwicklung der Cell-Architektur [12][94] wurde bereits zu Anfang dieses Kapitels grob skizziert. Sie integriert 241 Millionen Transistoren unter anfänglicher Verwendung einer 90-nm-Technologie, die recht bald durch eine 65-nm-Variante ersetzt wurde. Es werden Frequenzen von über 4 GHz angestrebt. Motivation für die Realisierung dieser Chip-Multiprozessor-Architektur war die „Vision, Supercomputerleistung ins tägliche Leben zu bringen“ [94]. Cell umfasst einen PowerPC-Kern als Kontrollprozessor für derzeit acht zusätzlich integrierte „Synergistic Processor Elements (SPE)“ als spezielle Coprozessoren, die kohärente DMA(*Direct Memory Access*)-Operationen zur Verfügung haben. Die *SPEs* haben lokalen Speicher und zusätzlich einen gemeinsamen Level-2-Cache. Als On-Chip-Verbindungsstruktur dient der „Element Interconnect Bus“ (*EIB*). Außerdem besitzt der Cell-Prozessor einen eigenen Speichercontroller, „Memory Interface Controller (*MIC*)“, und zwei konfigurierbare, Ein-/ Ausgangsschnittstellen. Letztendlich wurde noch eine umfangreiche Monitor- und Debugereinheit implementiert. Den Autoren von [94] zufolge sind 40 % des Schaltungsentwurfs Syntheseergebnisse. Die dominierenden 60 % des Chips sind als „Full Custom Design“ entworfen worden. Dies war aufgrund der hohen Rechenleistungs- und Verlustleistungsanforderungen notwendig. Um eine hohe Ausbeute und Zuverlässigkeit garantieren zu können, wurden zahlreiche Funktionen für Build-in-Self-Test (*BIST*) und Fehlerumgehung (*Array Repair Fuses*) eingebaut. Insgesamt wurde auch beim Cell-Chip versucht, das Design möglichst modular zu halten. Es wurde ein immenser Simulationsaufwand mit mehr als zwei Millionen Stunden Simulationszeit (dies entspricht 1,5 Trillionen simulierter Taktzyklen) im Vorfeld der Chipfertigung getrieben [94]. Dies lässt auf eine mittlere Simulatorgeschwindigkeit von ca. 140 Takten/Sekunde schließen. Der Cell-Architektur steht eine umfangreiche Werkzeugkette zur Verfügung, die verschiedene Betriebssysteme auf der Cell-Architektur unterstützt. Tabelle 2-4 gibt einen Überblick über ausgewählte Chip-Multiprozessoren und einiger ihrer wesentlichen Eigenschaften.

Tabelle 2-4: Charakteristika ausgewählter Chip-Multiprozessoren

For- schungs- einrich- tung, Her- steller Typ	CPU- Fre- quenz [MHz]	Anzahl Pro- zessorkerne	Leis- tungs- aufnah- me [W]	Chipflä- che [mm <sup>2</sup> ]	Technolo- gie [nm]	Speicher	PE	Kommuni- kationsstruktur	Besonderheiten
MIT Raw	225	16	25	331	150	>16*(32K+96K)	MIPS, 8 Pipeline- stufen	Busse zwi- schen Nach- barknoten, WH-Switching	Skalierbarkeit
Stanford Hydra	250	4	k. A.	90	250	>4*(8K+8K)+128 K	MIPS	Bus	
Daytona DSP	100	4	4	200	250	>4*8K	Sparc V8, 5 Pipeline- stufen	STBus	Gekoppelte Hardwa- rebeschleuniger MESI-Cache- Kohärenzprotokoll RTOS zur PE- Kontrolle
Ageia PhysX	533	>20 PEs	25	182	130	k. A.	Physikengi- ne	k. A.	Physikbeschleuniger
Intel Polaris	3.160 (bis 5.700)	80	62 (265)	275 (3 pro Kachel)	65	>80*(2K+3K)	X86	1 Router pro Kachel	1,01 Terra Flops (1,81 Terra Flops)
IBM, To- shiba und Sony Cell	3.200 (>4.000)	1+8	200	221	90	1*(32K L1 +512K L2) +8*256K	PowerPC + 8 SPEs	Element Inter- connect Bus	Max. 204 GByte/s Durchsatz des EIB
Rapport KiloCore	125	256	0,5	k. A.	180	k. A.	8-Bit-PEs	k. A.	
Nvidia 8800 Ultra / G80	1.500	128	175	k. A.	90	k. A.	Stream Prozessoren	k. A.	681 Mio. Transistoren
Compaq Piranha	500	8	k. A.	k. A.	180	>8*(64K+64K) +L2	Alpha CPU	Intra Chip Switch	
GigaNetIC- Projekt GigaNetIC	285	32	1,8	43,7	90	1.280K	N-Core	GigaNoC	Skalierbarkeit, Res- sourceneffizienz

### 2.8.4 Resümierender Vergleich mit dem GigaNetIC-Ansatz

Bei dem *MIT-Ansatz* handelt es sich, wie bei der GigaNetIC-Architektur, ebenfalls um einen kachelartigen, WH-Switching-basierten CMP, der mit seinen 16 Prozessorkernen ca. halb so viele Transistoren benötigt wie das in Tabelle 8-5 vorgestellte GigaNetIC-System mit 80 N-Cores. Aufgrund der flächenmäßig überlegenen GigaNetIC-Prozessorarchitektur lässt sich mehr Parallelität auf einen Chip bringen, die allerdings in der Standardversion zunächst über weniger leistungsfähige Verarbeitungseinheiten verfügt. Diese können allerdings speziell auf ein Anwendungsgebiet hin optimiert werden, ohne unnötigen Flächenbedarf durch unbenötigte Funktionen zu verursachen. Ein kachelartiger Floorplan bietet zahlreiche Vorteile im Bezug auf heutige Fertigungstechniken und wird deshalb auch beim GigaNetIC-Projekt berücksichtigt (vgl. Abschnitt 8.2.3).

Der *Hydra CMP* entspricht in etwa einem Cluster des GigaNetIC-Systems, wies aber seinerzeit kein übergeordnetes Konzept zur weiteren Skalierung auf, wie es u. a. durch das GigaNoC und durch die GigaNetIC-Switch-Boxen (vgl. Abschnitt 4.2) gegeben ist.

Der *Daytona CMP* integriert einen konfigurierbaren Multiprozessorcache für seine vier Verarbeitungseinheiten, der mit einem ähnlichen Cache-Kohärenzprotokoll arbeitet, wie es beim GigaNetIC-Multiprozessorcache zum Einsatz kommt, vgl. Abschnitt 4.4.2. Zusätzlich wurde ein Echtzeitbetriebssystem (*RTOS*) zur Steuerung der Prozessabläufe entwickelt, das die Leistungsfähigkeit eines CMP erheblich steigern kann. Für den GigaNetIC-CMP stehen mehrere Programmiermodelle zur

Auswahl, vgl. Abschnitt 4.5, allerdings wurde noch kein echtzeitfähiges Betriebssystem mit dem Gesamtsystem getestet.

Der *PipeRench*-Ansatz ist als Konzept für eine rekonfigurierbare, compilerunterstützte Architektur, die ohne zusätzliche Hardwarebeschleuniger an Anwendungsszenarien angepasst werden kann, ähnlich dem *DRP*-Ansatz von NEC zu sehen. Diese Art der Systemkonfiguration kann beim GigaNetIC-CMP ebenfalls für einzelne Cluster umgesetzt werden (vgl. Abschnitt 4.3.3), allerdings wäre an dieser Stelle die GigaNetIC-Compiler-Werkzeugkette (vgl. Abschnitt 5.6) auf diese neue Konstellation anzupassen, was derzeit<sup>9</sup> noch mit nicht zu vernachlässigendem Aufwand verbunden ist.

Der *XPP-III* verfolgt einen ähnlichen, wenn auch nicht vollständig auf FPGA-Technologie basierendes Konzept. Bei ihm wird zusätzlich ein paketorientiertes On-Chip-Netzwerk eingesetzt, wie es auch bei der GigaNetIC-Architektur durch das GigaNoC gegeben ist.

Beim Entwurf des *Piranha*-CMP wurde bei Compaq bereits vor der Einführung von SystemC eine C++-basierte Spezifikation des Gesamtsystems vorgenommen. Die GigaNetIC-Entwicklungsumgebung umfasst ebenfalls für die Spezifikation und zugleich zur schnellen, SoC-umspannenden, zyklenakkuraten Simulation ein SystemC-Modell (vgl. Abschnitt 5.2), eine erweiterte C++-Klassenbibliothek, die seit 2005 IEEE-Standard ist.

Die Architektur des *KiloCore* verfügt mit 256 bzw. in einer weiteren Ausbaustufe mit 1024 Verarbeitungseinheiten über die größte Anzahl integrierter Prozessorkerne in diesem Vergleich, allerdings handelt es sich auch um sehr einfach geartete Verarbeitungseinheiten, da sie u. a. nur über einen 8-Bit-breitem Datenpfad verfügen. In wieweit sich diese sehr geringe Berechnungsbandbreite, die nur durch einen nennenswerten Mehraufwand zu breiteren Datenpfaden konfiguriert werden kann, bewährt, bleibt abzuwarten. Bei dem Prozessorkern der GigaNetIC-Architektur wurde bewusst auf eine, zwar einfache, aber bereits 32-Bit-breite CPU gesetzt, um einen möglichst effizienten Kompromiss zwischen Leistungsfähigkeit und Flächeneffizienz zu erhalten. Außerdem wird durch die Möglichkeit der Integration beliebiger Hardwarebeschleuniger bzw. Hardwareblöcke (vgl. Abschnitt 4.3.3) genügend Flexibilität gewährleistet, um auf spezielle Anforderungen seitens des Anwendungsgebiets reagieren zu können.

Die stark auf Performanz optimierten Verarbeitungseinheiten der *Grafikbeschleuniger / GPUs* hingegen sind auf eingeschränkte Anwendungsgebiete spezialisiert und bieten wenig Spielraum. Eine Auslegung auf möglichst geringe Fläche oder minimale Verlustleistungsaufnahme stand bei der Entwicklung dieser Architekturen deutlich im Hintergrund. Dies verhält sich bei der grundsätzlich als Universalrechner ausgelegten GigaNetIC-Architektur anders. Diese ermöglicht den Entwicklern, die Positionierung des späteren Systems während der Spezifikation bzgl. der vier Dimensionen des Entwurfsraums (vgl. Definition 11) in alle Richtungen zu variieren. Ähnlich verhält sich der Vergleich mit den spezialisierten Physikbeschleuniger *PhysX*.

Die *Polaris*-Architektur von Intel kommt der Art des GigaNetIC-Ansatzes in vielen Aspekten sehr nahe, nicht zuletzt werden durch Intels Tera-Scale-Projekt ähnliche Anforderungen an die Rechnerarchitekturen der Zukunft definiert, wie es sie auch bei der Konzeptionierung der GigaNetIC-

---

<sup>9</sup> Die Fachgebiete Schaltungstechnik, Prof. Dr.-Ing. Ulrich Rückert und Programmiersprachen und Übersetzer, Prof. Dr. Uwe Kastens, forschen mittlerweile aktiv an dieser Thematik.

Architektur zu berücksichtigen gab. Auch hier wird eine gitterförmige, kachelbasierte Anordnung angewendet. Sind die Kacheln beim Polaris-Chip je 3 mm<sup>2</sup> groß und beinhalten nur einen, für Intels Verhältnisse wenig komplexen, Prozessor mit relativ wenig Speicher, so gestalten sich die Kacheln der GigaNetIC-Cluster mit 4,84 mm<sup>2</sup> in der 90-nm-Technologie in einer vergleichbaren Größenordnung. Skaliert man diese auf die von Intel verwendete 65-nm-Technologie (vgl. Definition 30) so entspräche sie einer Fläche von nur noch 2,5 mm<sup>2</sup>. Allerdings beinhalten die GigaNetIC-Kacheln vier Prozessoren mit insgesamt 32-mal soviel Speicher (160 KB) und bereits reservierter Fläche für spezielle Hardwarebeschleuniger. Bzgl. der Rechenleistung liegt die Intel-Architektur um Größenordnungen vorne, zumal sie je zwei Fließpunkteinheiten pro Kachel integriert, die beim GigaNetIC, aufgrund der betrachteten Anwendungsgebiete (vgl. Kapitel 7), derzeit nicht vorgesehen sind. Der Polaris-Router beherrscht einen Durchsatz von 72,51GByte/s bei 4 GHz gegenüber 26,6 GByte/s (Netto-Datendurchsatz) der GigaNetIC-Switch-Box mit nur 714 MHz. Nach den Skalierungsgesetzen ergäbe sich für die Switch-Box in der 65-nm-Technologie ein theoretischer Netto-Datendurchsatz von 36,84 GByte/s bei 989 MHz. Diese Zahlen zeigen, dass die GigaNetIC-Architektur in vielen Bereichen durchaus konkurrenzfähig ist. Abschließend bleibt zu erwähnen, dass Polaris über kein hierarchisches On-Chip-Netzwerk verfügt, sondern nur eine Hierarchieebene vorsieht. Durch die ausgeprägt generische Struktur der GigaNetIC-Architektur (vgl. Abschnitt 4.2.1.2) lassen sich nahezu beliebige Topologievarianten konstruieren, welches mit der Polaris-Router-Struktur aufgrund der festen Anzahl von vier Ports nicht in dem Maße zu verwirklichen ist. Außerdem ist nicht bekannt, in wieweit die Prozessoren durch eine, beim GigaNetIC-System vorhandene, „Offload-Engine“ zur Koordination des On-Chip-Datenverkehrs (vgl. 4.2.1.1) entlastet werden. Ansonsten würde Rechenleistung der Prozessoren für die Datenübertragung benötigt.

Bei den *Opteron*-Vierfach-Kernen setzt AMD bereits erfolgreich das *MOESI*-Cachekohärenzprotokoll ein, das gleiche, wie es der GigaNetIC-Multiprozessorcache auf Prozessor-Cluster-Ebene verwendet (vgl. Abschnitt 4.4.2). Allerdings wird beim GigaNetIC-Projekt eine deutlich höhere Parallelität auf SoC-Ebene, als derzeit bei AMD, angestrebt. Mit der zukünftigen *Torrenza*-Initiative verfolgt AMD einen Ansatz, bei dem heterogene Systeme von Prozessoren und Hardwarebeschleunigern auf einem Chip größtmögliche Effizienz bieten sollen. Dies wird ebenfalls bei der hybriden Struktur der GigaNetIC-Architektur wirkungsvoll eingesetzt, vgl. Kapitel 7 und 8.

Der *Cell*-Chip-Multiprozessor integriert derzeit die meisten Prozessorkerne der wirtschaftlich erfolgreichen Systeme dieses Vergleichs. Die GigaNetIC-Architektur zielt auf noch höhere Parallelität ab, die nicht zuletzt durch ihre regelmäßige Struktur und der guten Abbildbarkeit auf verschiedene Zieltechnologien aufgrund ihrer synthetisierbaren Beschreibung ermöglicht wird. Beim Cell-Chip sticht hingegen ein hoher Anteil an „Full Custom Design“; der bei den anderen Hochleistungsprozessoren in ähnlichen Dimensionen liegen wird, ins Auge. Dies bedeutet einen Nachteil bzgl. einfacher Skalierbarkeit und zukünftiger, anwendungsspezifischer Erweiterungen. Dadurch kann kein automatisierter Synthese-Prozess greifen, geometrisches Herunterskalieren bei derartigen Strukturgrößen ist ebenfalls nicht ohne Weiteres möglich, deshalb bedeutet eine Portierung auf neuere Technologien deutlich mehr Aufwand als es bei der GigaNetIC-Architektur der Fall ist. Der Cell-Chip verfügt über eine integrierte, leistungsstarke „Debug“-Einheit, was für eine produktive Softwareentwicklung von großer Bedeutung ist. Diese Option ist bei aktuellen GigaNetIC-Systemen nur ansatzweise vorhanden und ausbaufähig. Aufwändige Mechanismen zur Umgehung bzw. Feststellung von Fertigungsfehlern wurden beim Cell-Chip integriert – ebenfalls ein Punkt der bei der GigaNetIC-Architektur noch weiterer Anstrengungen bedarf.

**Schlussbemerkung.** Jeder der gezeigten Ansätze weist Besonderheiten und spezielle Mechanismen zur Ausnutzung der Parallelität auf. Die GigaNetIC-Architektur wurde ohne eine zuvor angestellte, tiefer gehende Analyse dieser Architekturen entworfen, um so weitestgehend unvoreingenommen einen neuartigen, innovativen massiv parallelen Chip-Multiprozessor zu erhalten. Diese, durch Analyse der grundlegenden Methoden und eigene Überlegungen getriebene Herangehensweise mag zunächst gewagt erscheinen. Der resümierende Vergleich zeigt allerdings, dass mit der GigaNetIC-Architektur ein CMP entwickelt wurde, der zahlreiche der Besonderheiten in einem Ansatz vereint und teilweise neue Methoden einsetzt, um so ein möglichst ressourceneffizientes Konzept zu verwirklichen.

## 2.9 Zusammenfassung

In diesem Kapitel wurden grundlegende Abschätzungen zur Leistungssteigerung durch paralleles Rechnen und die damit verbundenen Anforderungen an die Systeme aufgezeigt. Es wurden elementare Grundlagen zu den Kernkomponenten eingebetteter Parallelrechner vorgestellt. Hierzu zählen die *On-Chip-Netzwerke* und die in diesem Zusammenhang relevanten Methoden, die eine möglichst effiziente Verbindung *eingebetteter Verarbeitungseinheiten* gewährleisten. Mit Hilfe dieser *Kommunikationsinfrastruktur*, durch die Integration einer angepassten *Speicherhierarchie* und unter Einbeziehung angepasster *Algorithmen* und für die Parallelverarbeitung geeigneter *Anwendungen* wird so eine funktionale Parallelverarbeitung mittels der eingebetteten Verarbeitungseinheiten möglich (vgl. Abbildung 2-6). Anhand von Prognosen und den vorgestellten Anwendungsszenarien zeigt sich ein immer größer werdender Bedarf an solch leistungsfähigen Architekturen. So prognostiziert die ITRS bereits für das Jahr 2020 eine Beherrschbarkeit von Chip-Multiprozessoren mit mehr als 800 Prozessorkernen [2].

Beispielhaft wurden innovative Ansätze aus Wissenschaft und Industrie zu On-Chip-Netzwerken, Verarbeitungseinheiten und Chip-Multiprozessoren aufgezeigt und diskutiert. Hieraus wird das derzeit technisch Mögliche ersichtlich. Zukünftige Trends werden abgeleitet. Diese Entwicklungen bedeuten Herausforderungen, die es auch im GigaNetIC-Projekt ganzheitlich zu lösen galt. Unterschiede und Gemeinsamkeiten der vorgestellten Ansätze im Hinblick auf die GigaNetIC-Architektur wurden in einem Resümee herausgearbeitet und geben so bereits einen ersten Eindruck über die von mir entworfene Systemarchitektur. Die Charakterisierung und analytische Modellierung eines skalierbaren, ressourceneffizienten massiv parallelen eingebetteten Prozessorsystems sowie dessen praktische Umsetzung sind Bestandteil der folgenden Kapitel.

### 3 Charakterisierung und analytische Modellierung

Im Verlauf dieses Kapitels wird eine analytische Modellierung des Gesamtsystems entwickelt. Hierbei wird auf Größen wie Flächenbedarf, Leistungsaufnahme, Performanz und Zukunftssicherheit der jeweiligen Systemkomponenten und letztendlich des gesamten SoCs eingegangen. Anhand dieser Charakterisierungen wird mit Hilfe von definierten Kostenfunktionen ein Modell entwickelt, das zur Bewertung von Chip-Multiprozessorsystemen speziell auch im Hinblick auf ihre Ressourceneffizienz herangezogen werden kann. Die in diesem Kapitel erarbeiteten Bewertungsmaßstäbe finden in den Folgekapiteln Einsatz zur Bewertung und Optimierung der GigaNetIC-Architektur in Bezug auf dedizierte Anwendungsgebiete.

#### 3.1 Ressourceneffizienz eingebetteter Systeme

Thema dieser Arbeit ist die „Ressourceneffiziente Schaltungstechnik eingebetteter Parallelrechner“. Was aber ist unter diesem Leitgedanken genau zu verstehen? **Schaltungstechnik**, also der Entwurf von analogen und digitalen Halbleiterschaltungen, hier im Speziellen basierend auf CMOS-Standardzellentechnologie, ist in der heutigen Elektrotechnik eine etablierte und wohl erklärte Technik. **Eingebettete Parallelrechner**, die im vorigen Kapitel bereits diskutiert wurden, sind begrifflich ebenfalls etabliert. Wie aber lässt sich das Kompositum **Ressourceneffizienz** für System-on-Chip-Entwürfe definieren, und welche wesentlichen Kriterien gilt es zu beachten?

Das Wort **Ressource** kommt aus dem Französischen und meint laut Duden Hilfsmittel, Rohstoffe bzw. Grundlagen oder Geldmittel. Im öffentlichen Leben wird der Begriff meist mit Vernunft, Ethik und einem langfristigen Ökonomieverständnis in Verbindung gebracht. Der Brockhaus unterscheidet zwischen zwei fachsprachlichen Verwendungsweisen. In den Wirtschaftswissenschaften diene *Ressource* der Bezeichnung „im weiteren Sinne von Produktionsfaktoren (Arbeit, Boden und Kapital)“ und bezeichne „im engeren Sinne Rohstoffe (natürliche Ressourcen)“. Der zweite Bereich, für den der Brockhaus den Begriff definiert, die Datenverarbeitung, ist treffender für die Thematik dieser Arbeit. Der Brockhaus formuliert: „Ressourcen, Mittel die genutzt werden können. Die Ressourcen eines PCs sind: 1. die inneren (Prozessorleistung, Arbeitsspeicher, Massenspeicher) und 2. die äußeren (Peripheriegeräte). In einem Netzwerk werden alle gemeinsam nutzbaren Mittel als Ressourcen bezeichnet, auch Software (Anwendungsprogramme, Datenbestände)“.

Der zweite Bestandteil ist **Effizienz** (lateinisch *efficientia*: Wirksamkeit), unter dem der Brockhaus „etwas besonders Wirksames und Wirtschaftliches bzw. etwas besonders Leistungsfähiges“ versteht. Der Duden beschränkt sich auf die Bedeutung „wirksam und wirtschaftlich“.

Schlösse man nun aus diesen Definitionen auf eine möglichst treffende Interpretation des Begriffes Ressourceneffizienz für die Schaltungstechnik eingebetteter Parallelrechner, so ergäbe sich folgende Begrifflichkeit:

**Definition 1** Unter *ressourceneffizienter Schaltungstechnik* wird – im Allgemeinen und unter Zuhilfenahme der allgemeinen Definitionen der beiden Wortbestandteile **Ressource** und **Effizienz** – eine im Hinblick auf die nutzbaren Mittel und Grundlagen besonders leistungsfähige, wirtschaftliche Realisierung eines Halbleiterbausteins verstanden.

Im Folgenden werden weitere Begriffsdefinitionen gegeben, die im Rahmen meiner Charakterisierung eingebetteter Parallelrechner hinsichtlich ihrer Ressourceneffizienz essentiell sind. Es wird ferner eine formale Beschreibung vorgestellt, die die Bewertung der Ressourceneffizienz solcher Systeme ermöglicht.

**Definition 2** Als *Systementität*  $\mathbb{S}_e$  wird eine zum *System*  $\mathbb{S}$  gehörige Einheit bezeichnet, die funktional in sich geschlossen ist und einen wesentlichen Bestandteil des Systems in materieller oder auch immaterieller Hinsicht darstellt.

$$\mathbb{S} \supseteq \mathbb{S}_e \quad (3.1)$$

**Definition 3** Die *Architektur*  $\mathbb{A}$  definiert sich aus der Gesamtheit aller Bestandteile des Chip-Multiprozessors. Hierunter werden zum einen die Hardware-Systementitäten und deren Verschaltung  $\mathbb{S}_{e(HW)_i}$  und zum anderen alle nichtmateriellen Systementitäten  $\mathbb{S}_{e(SW)_i}$  verstanden.

$$\mathbb{A} \supset \mathbb{S}_{e(HW)_i} \cup \mathbb{S}_{e(SW)_i} \quad (3.2)$$

Zu den Hardware-Systementitäten (vgl. Abschnitte 4.2 bis 4.4)  $\mathbb{S}_{e(HW)_i}$  zählen z. B. Prozessorelemente, Hardwarebeschleuniger, Speicher, Caches, On-Chip-Netzwerke sowie Peripherieblöcke.

Die nichtmateriellen Komponenten  $\mathbb{S}_{e(SW)_i}$  sind z. B. der Compiler auf Clusterebene (vgl. Abschnitt 4.5), der zugehörige Assembler, das On-Chip-Kommunikationsprotokoll (vgl. Abschnitt 4.2.2), das übergeordnete Programmiermodell sowie die Entwurfswerkzeuge und die Zielapplikationen (vgl. Kapitel 5, 6 und 7).

**Definition 4** Ein *Bewertungsmaß*  $BM$  ist ein charakteristisches Maß, das zur Bewertung von Systementitäten  $\mathbb{S}_e$  bzw. Systemen  $\mathbb{S}_i$  oder auch Architekturen  $\mathbb{A}_i$  herangezogen werden kann.

**Definition 5** *Kostenmaße*  $\mathbb{K}$  definieren Obermengen ausgewählter und thematisch verwandter Bewertungsmaße  $BM$ .

$$\mathbb{K}_i \supset BM_{n(i)} \quad (3.3)$$

Im weiteren Verlauf dieser Arbeit werden die Kostenmaße Leistungsaufnahme  $\mathbf{P}$ , Flächenbedarf  $\mathbf{A}$ , (Rechen-)Leistung/Performanz  $\mathbf{T}$  und Zukunftssicherheit bzw. Flexibilität  $\mathbf{F}$  verwendet,  $\mathbb{K} = \{\mathbf{P}, \mathbf{A}, \mathbf{T}, \mathbf{F}\}$ . Kostenmaße sind nicht zwangsläufig invariant gegenüber einander, vielmehr stehen sie häufig in diametraler Wechselwirkung miteinander, vgl. Abbildung 3-2.

**Definition 6** Die *Zielfunktion*  $ZF$  beinhaltet ausgewählte, mit einem *Gewichtungsfaktor*  $c_i$  gewichtete *Bewertungsmaße*  $BM_i$  eines Kostenmaßes  $\mathbb{K}_i$ .

$$ZF = c_1 BM_1 + \dots + c_n BM_n \quad (3.4)$$

Es sind auch andere Zusammenhänge der einzelnen Bewertungsmaße denkbar, wie z. B. die Verwendung eines multiplikativen Zusammenhangs:

$$ZF = c_1 BM_1 \cdot \dots \cdot c_n BM_n \quad (3.5)$$

Auch exponentielle Gewichtungsfaktoren zur Differenzierung der Bewertungsmaße sind denkbar:



$$ZF = BM_1^{c_1} + \dots + BM_n^{c_n} \quad (3.6)$$

bzw.

$$ZF = BM_1^{c_1} \cdot \dots \cdot BM_n^{c_n} \quad (3.7)$$

Im weiteren Verlauf dieser Arbeit beschränke ich mich, um den Rahmen dieser Arbeit nicht zu sprengen, bei Kostenanalysen auf die Anwendung von (3.4).

Die Gewichtungen  $c_i$  sind entsprechend den Randbedingungen und Anforderungen an die jeweiligen Bewertungsmaße zu wählen. Im weiteren Verlauf dieser Arbeit werden die Gewichtungen gemäß des Zusammenhangs  $\sum_i c_i = 1$  gewählt, wobei  $0 \leq c_i \ll 1$  für eine geringere Gewichtung von  $BM_i$  und  $0 \ll c_i \leq 1$  für eine entsprechend größere Bedeutung des jeweiligen Bewertungsmaßes angesetzt wird.

**Definition 7** Eine *Randbedingung*  $R$  charakterisiert spezifische Besonderheiten, die bei der Erstellung bzw. beim Einsatz einer Systementität  $S_e$ , eines Systems  $S$  oder einer *Architektur*  $A$  auftreten.

Die Randbedingungen  $R_i$  sind bei der Spezifikation bekannt und müssen bei der Realisierung eingehalten werden.

**Definition 8** Als *untere Schranke*  $S_u$  wird eine quantitative untere Grenze und als *obere Schranke*  $S_o$  eine obere Grenze einer Zielfunktion  $ZF$  bezeichnet, die aufgrund der speziellen Randbedingungen  $R_i$  zulässig sind:

$$ZF \stackrel{!}{\geq} S_u, \quad ZF \stackrel{!}{\leq} S_o, \quad \forall R_i \quad (3.8)$$

**Definition 9** Die *Kostenfunktion*  $CF$  ist eine gewichtete Verknüpfung der einzelnen Zielfunktionen  $ZF_i$ . Die Gewichtungen  $\alpha_i$  sind hierbei ggf. subjektiv und durch die Spezifikation bzw. Randbedingungen  $R_i$  und Schranken  $S$  zu wählen (vgl. Definition 6). Für einen additiv gewählten Zusammenhang der Zielfunktionen  $ZF_i$  ergibt sich:

$$CF = \sum_{i \in \mathbb{K}} \alpha_i ZF_i$$

bzw. (3.9)

$$CF = \alpha_P \mathbf{P} + \alpha_A \mathbf{A} + \alpha_T \mathbf{T} + \alpha_F \mathbf{F} \quad \text{mit } \alpha_i \in \mathbb{R}$$

Hier wird ebenfalls so vorgegangen, dass  $\sum_i \alpha_i = 1$  gesetzt wird, wobei für besonders relevante Zielfunktionen  $ZF_i$  bzgl. der Anforderungen des Anwendungsszenarios  $0 \ll \alpha_i \leq 1$  angesetzt wird, und Zielfunktionen geringerer Bedeutung mit  $0 \leq \alpha_i \ll 1$  gewichtet werden. Varianten, ähnlich zu (3.5), (3.6) und (3.7) bezogen auf den Aufbau einer Kostenfunktion sind denkbar, werden aber im Rahmen dieser Arbeit nicht angewendet.

Sollten die Werte der einzelnen Zielfunktionen bzgl. der betrachteten Kostenmaße quantitativ sehr unterschiedlich sein, so ist eine Normierung der Werte der jeweiligen Realisierungsvarianten  $RV_i$  hilfreich:

$$ZF_{RV_i}(\mathbb{K}_i)_{\text{normiert}} : \frac{ZF_{RV_i}(\mathbb{K}_i)}{\max(ZF_{RV}(\mathbb{K}_i))}, \text{ mit } n = \text{Anzahl}\{RV\}, i = \{1, \dots, n\} \quad (3.10)$$

Die Normierung ist für alle Realisierungsvarianten  $RV_i$  des betrachteten Systems  $\mathbb{S}$  bzw. der Systementität  $\mathbb{S}_e$  für das betreffende Kostenmaß  $\mathbb{K}_i$ , oder auch alle Kostenmaße durchzuführen.

**Definition 10 Pareto-Optimierung** (nach VILFREDO PARETO) bezeichnet die Lösung eines multikriteriellen Problems. Bei einer Mehrzieloptimierung, bei der die Zielkriterien konkurrieren, kann eine gemeinsame Kostenfunktion  $CF$  mit Gewichtung der einzelnen Zielfunktionen  $ZF_i$  aufgestellt werden. Eine Lösung  $CF_{\text{pareto}}$  des Problems, für gegebene Gewichtungen der Zielfunktionen, wird als **pareto-optimal** bezeichnet, wenn eine weitere Verbesserung (im Sinne einer Minimierung von  $CF$ ) eines beliebigen Zielfunktionswertes  $\downarrow ZF_m$  stets in einer Verschlechterung (im Sinne einer Vergrößerung) eines anderen Zielfunktionswertes  $\uparrow ZF_n : n \neq m$  und somit auch  $\uparrow CF$  resultiert.

$$CF_{\text{pareto}} : \sum_{i \in \mathbb{K}} \alpha_i ZF_i = \mathbf{min!} \quad (3.11)$$

**Definition 11** Der **Entwurfsraum**  $\mathbb{E}$  umfasst die **Realisierungsvarianten**  $RV_i$  einzelner Systementitäten  $\mathbb{S}_e$ , Systeme  $\mathbb{S}$  bzw. Architekturen  $\mathbb{A}$ . Dimensionen des Entwurfsraums sind die Kostenmaße  $\mathbb{K}_i$ , nach Definition 5, vgl. auch Abbildung 2-14.

**Definition 12** Als **pareto-optimaler Punkt**  $P_{\text{pareto}}$  im Entwurfsraum  $\mathbb{E}$  wird eine Lösung im Sinne von (3.11) bezeichnet. Die Bezeichnung **pareto-optimaler Punkt** kann auf Realisierungsvarianten einzelner Systementitäten  $\mathbb{S}_{e(i)}$  mit  $i \in HW, SW$  auf Systeme  $\mathbb{S}_i$  oder Architekturen  $\mathbb{A}_i$  angewendet werden.

Da im weiteren Verlauf dieser Arbeit Untersuchungen unter Berücksichtigung implementierter Systementitäten  $\mathbb{S}_e$ , Systeme  $\mathbb{S}_i$  und Architekturen  $\mathbb{A}_i$  angestrebt werden, die keine vollständige Abdeckung des Entwurfsraums zulassen<sup>10</sup>, muss an dieser Stelle eine starke Einschränkung von Definition 12 getroffen werden.

**Definition 13** Als **diskreter pareto-optimaler Punkt**  $P_{\text{pareto, diskret}}$  im Entwurfsraum  $\mathbb{E}$  wird eine Lösung nach (3.11) bezeichnet, wobei die Menge der Realisierungsvarianten diskret und zumeist stark eingeschränkt ist. Es kann beliebig viele pareto-optimale Punkte nach Definition 12 geben, die nicht in der betrachteten Menge liegen. Ein nur auf diese Auswahl bezogener, pareto-optimaler Punkt wird dann **diskret pareto-optimal** genannt.

Auf Basis der bisher definierten Begriffe lässt sich nun eine formale Definition der Ressourceneffizienz treffen:

**Definition 14 Ressourceneffizienz**  $R_E$  im Sinne des schaltungstechnischen Entwurfs eines Chip-Multiprozessorsystems bzw. seiner Bestandteile und zur Verfeinerung der allgemeinen

<sup>10</sup> Es handelt sich zum einen um diskrete Punkte im Entwurfsraum, die nicht die Gesamtheit aller möglichen Realisierungsvarianten abdecken und zum anderen um Syntheseergebnisse, die für spezielle Randbedingungen erzielt wurden.

Definition 1 wird nun im Weiteren spezifiziert als: Realisierung von Systementitäten  $\mathbb{S}_{e_i}$ , Systemen  $\mathbb{S}_i$  bzw. Architekturen  $\mathbb{A}_i$ , die unter Berücksichtigung der vorgegebenen Schranken  $S$  bzw. Randbedingungen  $R$ , den damit verbundenen Zielfunktionen  $ZF$  und den daraus resultierenden Ergebnissen der Kostenfunktion  $CF$ , pareto-optimale Punkte  $P_{pareto}$  bzw., stark abgeschwächt, diskrete pareto-optimale Punkte  $P_{pareto, diskret}$  im Entwurfsraum  $\mathbb{E}$  darstellen. Auch der Versuch der Annäherung einer Realisierung an solche Punkte kann als Ansatz einer ressourceneffizienten Implementierung gewertet werden.

Mit den vorangegangenen Definitionen sind nun die relevanten Begriffe, die als Basis für die formale Bestimmung der Ressourceneffizienz  $R_E$  dienen, geprägt. Im Folgenden müssen nun Kriterien gefunden werden, die sich als Bewertungsmaße für eingebettete Systeme einsetzen lassen und die Zielfunktionen bilden.

## 3.2 Bewertungsmaße für Ressourceneffizienz

Zur Bewertung des Grades der Ressourceneffizienz eines Chip-Multiprozessorsystems müssen Bewertungsmaße  $BM$  (auch als *Benchmarks* bezeichnet) gefunden werden, die als Grundkomponenten für die jeweiligen Zielfunktionen  $ZF$  dienen. In der Literatur ist eine Vielzahl von Bewertungsmaßen für elektronische Schaltungen zu finden, von denen im Folgenden einige relevante kurz eingeführt werden.

### 3.2.1 Bewertungsmaße zur Performanz

Die nachstehenden Bewertungsmaße charakterisieren die Performanz  $T$  (eingeschränkter die Rechenleistung) von einzelnen Systementitäten, Systemen bzw. Architekturen.

**Definition 15** *Taktfrequenz*  $f$  [MHz] =  $1/\text{Taktperiode } T$ , mit der die Schaltung betrieben werden kann.

**Definition 16** Die *Latenz*  $L$  [1] einer Systementität ist die Anzahl der Taktperioden (häufig: Takte), nach der ein Ergebnis am Ausgang der Schaltung anliegt.

Im nicht-digitalen Fall wird sie einheitenbehaftet als Zeitspanne  $L$  [s] angegeben, die vergeht, bis eine Eingangssignaländerung als Ausgangssignaländerung erkennbar wird.

**Definition 17** Der *Jitter*  $J$  [s] kennzeichnet die maximale Varianz der Latenz  $L$  und wird im Weiteren im digitalen Sinne verwendet, also in ganzzahligen Taktperioden  $T$  angegeben.

Jitter ist ein Phänomen, das z. B. aufgrund konkurrierender Prozesse, die auf dieselbe Ressource zugreifen wollen und nicht immer in gleicher Weise die Zuteilung bekommen, entsteht.

**Definition 18** Die *Ausführungszeit*  $T_{ex} = T \cdot L$  [s] ist die benötigte Zeit für die Lösung bzw. die Bereitstellung des Ergebnisses für eine gegebene Aufgabe.

**Definition 19** Der *Durchsatz*  $D = \frac{1}{T_{ex}}$  [1/s] kennzeichnet die Anzahl der gelieferten Ergebnisse bzw. Daten pro Sekunde.

Im Folgenden werden allgemeine Bewertungsmaße zur Charakterisierung der Leistungsfähigkeit bzw. Performanz von Verarbeitungseinheiten und im Speziellen von Prozessoren vorgestellt:

Ein verbreitetes, zumeist weniger aussagekräftiges Maß<sup>11</sup> zur Angabe der Leistungsfähigkeit ist die Klassifikation in *MIPS* (*Million Instructions Per Second*).

**Definition 20** *MIPS* beziffern Millionen Instruktionen bzw. Maschinenbefehle pro Sekunde  $MIPS = \text{Instruktionen} \cdot 10^{-6} \text{ s}^{-1}$ , die durch die Verarbeitungseinheit abgearbeitet werden können:

Ähnliche Bewertungsmaße zur Angabe der Instruktionen pro Sekunde sind die Angabe der Instruktionen pro Takt bzw. der Kehrwert in Takten pro Instruktion.

**Definition 21** Instruktionen pro Takt<sup>12</sup> (*Instructions Per Cycle / IPC*) bzw. dessen Kehrwert Takte pro Instruktion (*Cycles Per Instruction / CPI*)  $IPC = \frac{\text{Instruktionen}}{\text{Taktzyklus}} = \frac{1}{CPI}$ . Es gilt ferner:  $IPC \cdot f \cdot 10^{-6} = MIPS$

Aus den bisherigen Bewertungsmaßen lässt sich nun ein objektiveres Maß zur Bewertung von Instruktionssatz-basierten Prozessoren definieren, vgl. [95]:

**Definition 22** Die *gemittelte Ausführungszeit für Instruktionssatz-basierte Prozessoren*  $T_{ex}(PE)$  [s] ist die benötigte Zeit für die Lösung und die Bereitstellung des Ergebnisses für eine gegebene Aufgabe.  $IC$  stellt die dynamische Instruktionsanzahl (*dynamic Instruction Count*)<sup>13</sup>, die für die Abarbeitung der Aufgabe benötigt wird, dar.  $\overline{CPI}$  ist die durchschnittliche Anzahl der benötigten Takte bzw. Zyklen zur Verarbeitung einer Instruktion der betreffenden Prozessorarchitektur.

$$T_{ex}(PE) = IC \cdot \overline{CPI} \cdot f^{-1} \quad (3.12)$$

Dieses Bewertungsmaß ist sinnvoll, wenn die zur Verfügung stehende Entwicklungsumgebung keine zyklenakkurate Laufzeitauswertung bereitstellt und stattdessen nur die Anzahl abgearbeiteter Instruktionen zählt. Die GigaNetIC-Entwicklungsumgebung hingegen erlaubt in allen Simulatoren (vgl. Kapitel 5) eine taktgenaue Simulation und Laufzeitauswertung der eingebetteten Prozessoren. Deshalb kann mit der exakten Ausführungszeit  $T_{ex}$  gearbeitet werden, im Gegensatz zum Ansatz von LI und MARTÍNEZ [96]. Die in [96] aus (3.12) gefolgerte nominelle Effizienz paralleler Verarbeitung (*nominal parallel efficiency*) wird in Definition 23 dargelegt. Es handelt sich hierbei um eine vereinfachte Formulierung des Gesetzes von GUSTAFSON (2.2). Beim *DSLAM-System-Explorer* (vgl. Abschnitt 7.5) dient ein ähnliches Modell als Grundlage für die Hochrechnung der Leistungsfähigkeit GigaNetIC-basierter Parallelprozessorsysteme. Es verwendet allerdings die exakten Laufzeiten zur Bestimmung der Leistungsfähigkeit. Zusätzlich findet ein einstellbarer *Overhead* der parallelen Architektur Berücksichtigung.

<sup>11</sup> Aufgrund der unterschiedlichen Mächtigkeit der Instruktionssätze verschiedener Prozessorarchitekturen (z. B. RISC vs. CISC, Abschnitt 2.4.3) lässt sich zunächst kein objektiver Vergleich auf Basis dieses Bewertungsmaßes treffen.

<sup>12</sup> Dieser Wert hängt stark von der zugrundeliegenden Architektur (z. B. von der *Pipeline*, der *Superskalarität*, der Mächtigkeit der eigentlichen Instruktion etc.) ab und ist deshalb kein eindeutiges Maß zur Bewertung der Leistungsfähigkeit von verschiedenen Architekturen.

<sup>13</sup> Diese kann u. a. von den jeweiligen Daten und Speicherhierarchien abhängen.

**Definition 23** Die *nominelle Effizienz paralleler Verarbeitung* (*nominal parallel efficiency*)  $\varepsilon_n(N)$  nach [96] mit  $N$  Prozessoren ergibt sich aus dem Verhältnis der benötigten Takte eines Prozessors  $IC_1$  zu denen von  $N$  Prozessoren  $IC_N$  (3.13).  $\varepsilon_n(N)$  gibt Aufschluss über die Eigenschaften der Anwendung bzgl. paralleler Verarbeitung mittels CMPs. Bei  $\varepsilon_n(N) < 1$  zeigen sich Performanzeinbußen, z. B. verursacht durch Kommunikation etc. Bei  $\varepsilon_n(N) > 1$  hingegen zeigen sich superlineare Effekte der parallelen Verarbeitung, z. B. hervorgerufen durch Multiprozessorcaches (vgl. Abschnitte 4.4.2 und 6.7).

$$\varepsilon_n(N) = \frac{IC_1 \cdot \overline{CPI}_1}{IC_N \cdot \overline{CPI}_N} \quad (3.13)$$

Ein sich an diese Definitionen anschließendes Bewertungsmaß, das vor allem für digitale Signalprozessoren (*DSPs*) und im Bereich des wissenschaftlichen Rechnens von Relevanz ist, ist die Angabe der Gleitkommaoperationen eines Systems pro Sekunde. Der Standard-Prozessor der GigaNetIC-Architektur, der N-Core, verfügt nicht über eine derartige Gleitkommaeinheit, weshalb derartige Operationen sehr teuer emuliert werden, bzw. eine Integration eines anwendungsspezifischen Hardwarebeschleunigers notwendig wäre. Da die im Zusammenhang mit dem GigaNetIC-CMP betrachteten Anwendungen jedoch keine relevante Verwendung für diese Operationen zeigen, wird diesbezüglich keine Einstufung vorgenommen.

**Definition 24** Gleitkommaoperationen pro Sekunde (*Floating Point Operations Per Second / FLOPS*), gibt die Anzahl möglicher Berechnungen im Gleitkommabereich pro Sekunde an.

Da die Bewertung der Leistungsfähigkeit von eingebetteten Verarbeitungseinheiten anhand der Definitionen 20 bis 22 sehr fraglich ist und das Bewertungsmaß aus Definition 24 nicht für die Hauptanwendungsgebiete der GigaNetIC-Architektur herangezogen werden kann, werden im weiteren Verlauf dieser Arbeit komplexere Bewertungsmaße bzw. *Benchmarks* eingeführt, die eine kontextbezogene Einstufung der betrachteten Verarbeitungseinheiten erlauben (vgl. Kapitel 7 und 8).

### 3.2.2 Bewertungsmaße zur Leistungsaufnahme

Die folgenden Bewertungsmaße werden häufig zur Charakterisierung von digitalen Schaltungen im Hinblick auf ihre Leistungsaufnahme verwendet.

**Definition 25** Die *Leistungsaufnahme* bzw. *Verlustleistung*  $P$  einer CMOS-Schaltung setzt sich zusammen aus statischer Verlustleistung  $P_{stat}$  und dynamischer Verlustleistung  $P_{dyn}$  und stellt die Umsetzung von elektrischer Energie in Wärme dar:

$$P = P_{stat} + P_{dyn} \quad (3.14)$$

**Definition 26** Die *statische Verlustleistung* entsteht aufgrund von Ruhestromen. Sie setzt sich aus den Anteilen  $P_{quer}$ , hervorgerufen durch die Querströme  $I_{quer}$ , und aus  $P_{leck}$ , der aus den Leckströmen  $I_{leck}$  im Halbleitermaterial resultiert, zusammen:

$$P_{stat} = P_{quer} + P_{leck} \quad (3.15)$$

**Definition 27** Die *dynamische Verlustleistung* entsteht aufgrund von Schaltvorgängen und hat derzeit bei CMOS-basierten Schaltkreisen den überwiegenden Anteil an der gesamten Verlustleistung (vgl. Abbildung 3-1). Sie setzt sich zusammen aus der Lastumladeverlustleistung

tung  $P_{last}$ , die beim Laden bzw. Entladen der Lastkapazitäten entsteht und durch die Lade- bzw. Entladeströme  $I_{last}$  hervorgerufen wird. Die Schaltverlustleistung  $P_{schalt}$  entsteht bei beiden Umladevorgängen durch die Kurzschlussströme  $I_{schalt}$ , die fließen können, wenn beide Transistoren durchgeschaltet sind:

$$P_{dyn} = P_{last} + P_{schalt} \quad (3.16)$$

Berücksichtigt man die Schalthäufigkeiten  $\alpha$ , mit denen die Lastkapazitäten  $C_{last}$  umgeladen werden, so ergibt sich für die Lastumladeverlustleistung  $P_{last}$  (auch häufig  $P_{load}$  genannt), die bei derzeitigen CMOS-Schaltungen einen Großteil der dynamischen Verlustleistung (ca. 80 bis 90%) ausmacht [97][98], folgende Beziehung (3.17):

$$P_{last} = \frac{1}{2} \alpha \cdot C_{last} \cdot f \cdot U_B \cdot \Delta U \quad (3.17)$$

Da der Signalhub  $\Delta U$  häufig den vollen Bereich der Versorgungsspannung  $U_B$  ausmacht, lässt sich die folgende Vereinfachung treffen:

$$P_{dyn} = \frac{1}{2} \alpha \cdot C_{last} \cdot f \cdot U_B^2, \text{ mit } \Delta U = U_B \quad (3.18)$$

Aufgrund des wesentlichen Anteils der dynamischen Verlustleistung an der Gesamtverlustleistung digitaler Schaltkreise wird im Folgenden hauptsächlich an der Minimierung dieser Leistungsaufnahmeart im Sinne der Ressourceneffizienz (nach Definition 14) gearbeitet. Allerdings nehmen mit zunehmender Miniaturisierung die anderen Anteile der Verlustleistung zu [2], so dass auch hier bereits Lösungen zur Minimierung bzw. Optimierung dieser Effekte aufgezeigt werden [99][100].

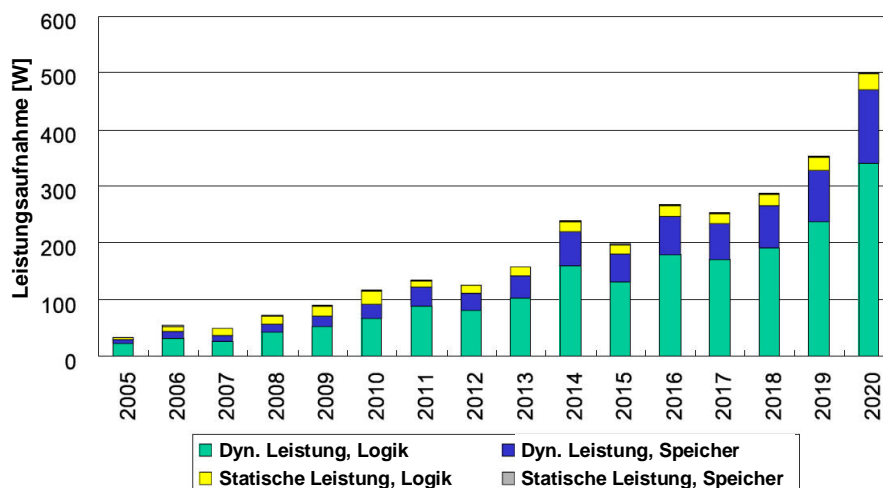


Abbildung 3-1: Trends der Leistungsaufnahme bei CMOS-basierten SoCs [67]

Einen Überblick bzw. eine Prognose über die Entwicklung für die Anteile von Logik und Speicher an der Leistungsaufnahme für zukünftige „Consumer Stationary“ SoCs gibt die ITRS in [67] (vgl. Abbildung 3-1). Es zeigt sich, dass die erwartete statische Verlustleistung, wie bereits erwähnt, klein gegenüber der dynamischen Verlustleistung ist. Die Leistungsaufnahme der Logik ist gegenüber der des Speichers deutlich größer.

Mit Hilfe der in (3.18) gezeigten Beziehung zur dynamischen Verlustleistung ist es möglich, den Komponenten eines Systems bzw. den Systementitäten (vgl. Definition 2) ihre charakteristische, effektive Kapazität anstelle einer allgemeinen Lastkapazität  $C_{last}$  zuzuordnen (vgl. [101]).

**Definition 28** Die charakteristische, *effektive Kapazität* einer Systementität  $\mathbb{S}_{e_i}$  bestimmt sich zu:

$$C_{\text{eff}}(\mathbb{S}_{e_i}) = C_{\text{last}}(\mathbb{S}_{e_i}) = \frac{2 \cdot P_{\text{dyn}}(\mathbb{S}_{e_i})}{\alpha \cdot f \cdot U_B^2} \quad (3.19)$$

Die in den Kapiteln 5 und 6 vorgestellte GigaNetIC-Werkzeugkette ermöglicht eine komfortable Ermittlung von  $C_{\text{eff}}$  bzw. der benötigten Verlustleistung  $P_{\text{dyn}}(\mathbb{S}_{e_i})$  der betreffenden Hard- und Software-Systementitäten. Die ermittelten Werte können dann in die Zielfunktion bzw. letztendlich in die Kostenfunktion und Optimierungswerkzeuge einfließen und zu neuen, ressourceneffizienteren Systemkonstellationen führen [102][103][104].

### 3.2.3 Bewertungsmaße zur Fläche

Ein Maß anderer physikalischer Natur als Performanz und Leistungsaufnahme ist die Fläche, die eine Schaltungsimplementierung benötigt. Die Fläche wird zum einen beeinflusst durch die realisierte Funktion und die Art der Implementierung und zum anderen durch die verwendete Technologie.

**Definition 29** Die *Fläche*  $A_{\mathbb{S}}$  des Systems  $\mathbb{S}$  setzt sich zusammen aus der Summe der benötigten Einzelflächen sowohl aller Hardwaresystementitäten als auch der Verbindungsstruktur:

$$A_{\mathbb{S}} = \sum \left( A_{\mathbb{S}_{e_i}(\text{HW})} + A_{\text{Verbindungsstruktur}} \right) \quad (3.20)$$

Die durch Synthese und ggf. anschließende Platzierungs- und Verdrahtungsschritte erhaltenen Flächenwerte sind technologieabhängig und lassen sich über Skalierungsregeln (vgl. Definition 30) miteinander in Beziehung setzen.

Im Rahmen dieser Arbeit werden hauptsächlich 130-nm- und 90-nm-CMOS-Standardzellentechnologien zur Schaltungsrealisierung verwendet. Um besonders im Hinblick auf die benötigte Fläche Vergleiche mit Standardzellentechnologien anderer Strukturgrößen treffen zu können (vgl. [14]), lassen sich idealisierte Skalierungsregeln („Scaling-“Gesetze) anwenden [105][106].

**Definition 30** Zwischen zwei CMOS-Standardzellentechnologien kann ein konstanter *Skalierungsfaktor*  $S$  ermittelt werden, vgl. (3.21).

$$\frac{\text{Strukturgröße}_{\text{verwendete Technologie}}}{\text{Strukturgröße}_{\text{neue Technologie}}} = S > 1 \quad (3.21)$$

Der Skalierungsfaktor  $S$ , mit einem entsprechenden Exponenten  $c$  gewichtet, ermöglicht die Skalierung eines Merkmals, insbesondere der Fläche, auf eine Zieltechnologie anderer minimaler Strukturgröße. Mit Hilfe von (3.22) lässt sich der entsprechende Wert des jeweiligen Merkmals abschätzen.

$$\text{Merkmal}_{\text{verwendete Technologie}} \cdot S^c = \text{Merkmal}_{\text{neue Technologie}} \quad (3.22)$$

Tabelle 3-1 gibt Aufschluss über die, den Merkmalen entsprechenden, gewichteten Skalierungsfaktoren. Hierbei wird zwischen einer linearen Skalierung bei konstantem E-Feld und einer linearen Skalierung bei konstanter Spannung unterschieden. Die angegebenen gewichteten Skalierungsfaktoren sind idealisiert zu sehen, allerdings kommen sie derzeit der Realität aktueller CMOS-Technologien sehr nahe [2].

Tabelle 3-1: Idealisierte Skalierungsregeln bei Strukturverkleinerungen von Halbleiterschaltungen

Merkmal	gewichteter Skalierungsfaktor konstantes E-Feld ( $E = \text{const.}$ )	gewichteter Skalierungsfaktor konstante Spannung ( $U = \text{const.}$ )
Fläche $A$	$S^{-2}$	$S^{-2}$
$W, L, t_{ox}$	$S^{-1}$	$S^{-1}$
$U_{DD}, U_T, U$	$S^{-1}$	1
Feldstärke $E$	1	$S$
$c_{ox}$	$S$	$S$
$C_{ox}, C$	$S^{-1}$	$S^{-1}$
$I_D, I$	$S^{-1}$	$S$
Schaltzeit $\tau$	$S^{-1}$	$S^{-2}$

Eine eng mit der Fläche gekoppelte Größe ist der Preis für die fertige Halbleiterrealisierung. Der Preis ist zumeist direkt abhängig von der benötigten Fläche und zusätzlich gekoppelt an die Technologie. Die zu produzierende Stückzahl, technologieabhängige Fertigungskosten und die Ausbeute (*Yield*) bestimmen u. a. die Kosten CMOS-basierter Schaltungen.

**Definition 31** Der *Preis*  $P$  für eine Hardwaressystementität  $\mathbb{S}_{e(HW)}$ , ein System  $\mathbb{S}$  oder eine *Architektur*  $\mathbb{A}$  wird in € angegeben und beziffert die finanziellen Mittel, die am Markt für die besagte Komponente aufgewendet werden müssen, um sie zu erwerben.

Der Preis kann zu Vergleichszwecken in Relation zur Fläche in €/mm<sup>2</sup> angegeben werden. Hierbei ist zu beachten, dass der Preis sich nicht immer zwangsläufig proportional zu den Kosten der betrachteten Komponente verhält, sondern z. B. aufgrund von Alleinstellungsmerkmalen oder Gründen der Marktpenetration komplett von den Kosten entkoppelt gestaltet wird.

**Definition 32** Die *Kosten*  $K$  für eine Hardwaressystementität  $\mathbb{S}_{e(HW)}$ , ein System  $\mathbb{S}$  oder eine *Architektur*  $\mathbb{A}$  werden in € angegeben und beziffern die für die Realisierung der betreffenden Komponente aufzuwendenden finanziellen Mittel.

### 3.2.4 Bewertungsmaße zur Zukunftssicherheit und Flexibilität

Bewertungsmaße zur Zukunftssicherheit bzw. Flexibilität einzelner Systementitäten oder ganzer Systeme sind physikalisch weniger fassbar als die drei bisher vorgestellten Bewertungsmaße Performanz, Leistungsaufnahme und Fläche. Vielmehr sind sie im Kontext der Randbedingungen und Schranken (vgl. Definitionen 7 und 8) für das System und die jeweilige Anwendung zu sehen, die häufig subjektiver Formulierungen der Systemarchitekten unterliegen. Dennoch sind diese Bewertungsmaße von ebenso großer Bedeutung, wenn nicht sogar zukünftig noch bedeutsamer als die bisher genannten Maße. Nachstehend werden deshalb die wichtigsten Bewertungsmaße dieser Kategorie eingeführt.

Im Folgenden werden die Begriffe Fehlertoleranz und Fehlerimmunität, letztere eine noch stärkere Form der Toleranz bezüglich des Verbergens von Software- oder Hardwaredefekten, festgelegt. Auch sie können als Bewertungsmaße für das Kostenmaß Flexibilität herangezogen werden. Da jedoch mit der Realisierung von fehlertoleranten bzw. fehlerimmunen Systemen zusätzlich Redundanz vorgesehen werden muss, erhöht sich zwangsläufig die benötigte Fläche (vgl. Definition 5).



**Definition 33** Die *Fehlertoleranz*  $FT$  eines eingebetteten Systems definiere ich als immanente Fähigkeit des Systems, auf Ausfälle oder Fehlfunktion von Systementitäten (nach Definition 2) selbstständig zu reagieren und diese Fehler in gewissen Grenzen zu kompensieren.

Das bedeutet, dass nach außen die grundsätzlich spezifizierte Funktionalität des Systems erhalten bleibt. Die Maßnahmen, die angewendet werden müssen, um ein System fehlertolerant aufzubauen, sind vielschichtig und umfassen alle wesentlichen Systementitäten. Die Toleranz gegenüber auftretenden Fehlern ist allerdings nicht unbeschränkt und ist u. a. mit dem Grad der Redundanz der Hardware verknüpft. Die Software muss u. a. angemessene Detektions- und Verriegelungsmechanismen aufweisen, um Fehlertoleranz in gewissen Grenzen zu ermöglichen. Toleriert ein System Fehler, so muss das korrekte Verhalten nach außen – also zu den externen Schnittstellen – grundsätzlich gewährleistet sein. Dies beinhaltet z. B. Protokolle und Datenkonsistenz, also die Korrektheit der Ergebnisse der Verarbeitung. Bezüglich der Latenz, des Durchsatzes und des Jitters (vgl. Definitionen 16, 17 und 19) hingegen können Leistungseinbußen auftreten.

**Definition 34** Als *Fehlerimmunität*  $FI$  eines eingebetteten Systems formuliere ich die Eigenschaft eines Systems, alle Anforderungen, die sich für ein fehlertolerantes System aus Definition 33 ergeben, zu erfüllen. Darüber hinaus muss ein fehlerimmunes System alle Spezifikationen bezüglich der Performanz einhalten.

Das bedeutet keine Reduktion des Durchsatzes (nach Definition 19), keine Erhöhung der Latenz (nach Definition 16) und keine Vergrößerung des Jitters (nach Definition 17). Die Erfüllung dieser Bedingungen ist nur mit massivem Einsatz von Redundanz und großem Aufwand auf Seiten der Software möglich.

**Definition 35** *Skalierbarkeit* bezeichnet die Möglichkeit, auf Basis der bestehenden Minimal- bzw. Grundstruktur eines Systems ein neues System gleichgearteter Struktur zu realisieren, welches über weitaus mehr Systementitäten verfügt als das Ursprungssystem. Die wesentlichen Systemeigenschaften in Form der beschriebenen Kostenmaße sollten jedoch von dieser Erweiterung weitgehend unbeeinflusst bleiben oder maximal linear proportional ansteigen.

Ein System *skaliert* also, wenn bei  $n$ -facher Anzahl von Verarbeitungseinheiten die resultierende Fläche maximal  $n$ -mal so groß ist, die maximale Taktfrequenz des resultierenden Systems gleich oder zumindest  $\frac{1}{n} \cdot f$  beträgt und die Leistungsaufnahme maximal um das  $n$ -fache zunimmt. Für

ein Softwareprogramm bedeutet dies, dass ein Algorithmus *skaliert*, wenn bei einer Erhöhung des Eingangsproblems um den Faktor  $n$  die Berechnungsdauer auf der gleichen Maschine maximal linear ansteigt und somit  $n$ -mal so viele Berechnungen gemacht werden müssen.

Erfüllt ein erzeugtes  $n$ -faches System zusätzlich die Anforderung, dass es durch Reduktion oder Abschalten von  $m$  Einheiten der Systemgrundstruktur auch nur eine  $m$ -fache Herabsetzung seiner Eigenschaften erfährt, so ist es *herunterskalierbar*. Diese Eigenschaft ist z. B. für den Einsatz in mobilen Geräten, im Falle geringerer Last, zur Reduktion der Verlustleistung und damit einhergehend zur Verlängerung der Betriebszeit von besonderem Vorteil.

Im Folgenden werden einige Aussagen zur Wiederverwertbarkeit (*Reuse*) wiedergegeben bzw. getroffen, die letztendlich zu einer Definition dieses Begriffs im Hinblick auf die in dieser Arbeit relevante Thematik führten.

Ein Ausschnitt aus der ITRS-Roadmap [2] zeigt, dass auf System-Ebene im Bereich der Entwurfsanforderungen der nahen Zukunft (*System Level Design Requirements — Near-term Years*) der Anteil an Design Reuse, also an Wiederverwertbarkeit von Hardwareblöcken, mit fortschreitender Strukturverkleinerung in den kommenden Jahren bis 2013 stetig und insgesamt um mehr als 37 % ansteigen muss (vgl. Tabelle 3-2). Dies zeigt deutlich die Vorteile, die sich durch einen flexiblen, NoC-basierten SoC-Entwurf ergeben werden.

**Tabelle 3-2: Entwurfsanforderungen / Design Reuse der nahen und fernen Zukunft auf System-Ebene [2]**

Jahr der Produktion	Nahe Zukunft ( <i>Near Term</i> )									Ferne Zukunft ( <i>Long Term</i> )						
	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020
<i>DRAM 1/2 Pitch(nm) (contacted)</i>	80	70	65	57	50	45	40	36	32	28	25	22	20	18	16	14
<i>Design Block Reuse [% zur Gesamten Logikgröße]</i>	32%	33%	35%	36%	38%	40%	41%	42%	44%	46%	48%	49%	51%	52%	54%	55%

Nach den Annahmen der ITRS steigert sich der Wert für wiederverwendete Blöcke ausgehend von 2005 um bis zu 71 % im Jahr 2020, so dass 2020 mehr als die Hälfte (55 %) der gesamten Logik eines Chipentwurfs auf vorhandenen Schaltungsblöcken aufbaut. Diese Prognose zeigt das immer wichtiger werdende Entwurfsparadigma der *IP-Block*-basierten System-on-Chips auf, die sich durch ihre immanenten Eigenschaften besonders für ressourceneffiziente Implementierungen eignen. Deshalb ist auch die GigaNetIC-Architektur besonders prädestiniert für zukünftige Systeme. Die ITRS gibt folgende Berechnungsformel zur Vorhersage des „Reuse“-Wertes an:

$$reuse(n) = 1 - (1 - reuse) \cdot \left( \frac{(1 + p_{Entwicklung})^n}{(1 + c_{Wachstum})^n} \right) \quad (3.23)$$

Mit  $reuse(n)$  ist die Anzahl wiederverwendeter Blöcke im Jahre  $n$  vom Referenzjahr gemeint. Reuse ist einer der Hauptfaktoren, die die Entwurfsproduktivität steigern und eines der Schlüsselkonzepte, die hinter dem System-Ebenen-Entwurf (*System Level Design*) stehen.  $p_{Entwicklung}$  steht für die erwartete jährliche, durchschnittliche Produktivitätswachstumsrate, ohne den Effekt von Reuse.  $c_{Wachstum}$  bezeichnet die erwartete jährliche, durchschnittliche Wachstumsrate der Entwurfskomplexität. Diese Formel setzt voraus, dass die Größe des Entwicklungsteams genauso wie die Entwurfszeit während der betrachteten Zeitspanne konstant bleibt. Die Begründung dieser Formel liegt darin, dass die Lücke zwischen Produktivitätswachstum und Wachstumsrate der Entwurfskomplexität (*Design Productivity Gap*) durch Wiederverwendung (*Reuse*) ausgefüllt werden muss, soll der Fortschritt im SoC-Entwurf vollständig genutzt werden [2]. Neben der unter (3.23) genannten Gleichung zur Wiederverwendbarkeit soll nun noch eine textuelle Definition für diesen Begriff gegeben werden.

**Definition 36 Wiederverwendbarkeit WV:** Ein *wiederverwendbarer* Hardware- oder Softwareentwurf zeichnet sich dadurch aus, dass durch verifizierte Funktion, verifizierte Implementierung, bestehende Testumgebungen und gute Dokumentation der Einsatz einer bereits realisierten Systementität (nach Definition 2), verglichen mit einer Neuimplementierung, einen deutlich geringeren Entwurfsaufwand (maximal 50 % eines Neuentwurfs) bedeutet.

Wiederverwendbarkeit ist eine essentielle Anforderung heutiger und zukünftiger SoC-Entwürfe, um die Entwurfsproduktivitätslücke (vgl. Kapitel 1) schließen zu können. Wiederverwendbare Systementitäten helfen somit den Entwurfsprozess drastisch zu verkürzen.

Im Zusammenhang mit der Zukunftssicherheit und Flexibilität ist auch die Programmierbarkeit einer Systementität bzw. des Systems zu sehen. Diese steht, im Rahmen der implementierten Möglichkeiten, für Flexibilität gegenüber einer starren, durch größtenteils fest verdrahtete Logikfunktionen definierten Funktion eines Hardwareblocks. Je vielfältiger die Möglichkeiten der Programmierbarkeit ausfallen, desto flexibler kann auf veränderte Anforderungen seitens der Anwendung reagiert werden. Allerdings geht dies zumeist mit einer Vergrößerung der Fläche einher. Je universeller die Art der Programmierung ist, desto leichter lassen sich entwickelte Programme auf neue Hardwarevarianten portieren. Skalierbare Programmiermodelle (vgl. Abschnitt 4.5) ermöglichen effizient nutzbare Hardwarestrukturen. Die Programmierbarkeit in einer Hochsprache wie z. B. C und die effiziente Abbildbarkeit auf das System durch geeignete Werkzeugketten (vgl. Abschnitt 5.6) erhöhen ebenfalls die Wiederverwendbarkeit und Flexibilität des Systems.

**Definition 37** Die *Programmierbarkeit*  $PG$  eines Systems beschreibt den Grad der Veränderbarkeit der Funktion der Schaltung durch Veränderung von immateriellen Bestandteilen (Software-Systementitäten  $\mathbb{S}_{e(SW)_i}$  (nach Definition 2)) des Systems. Die Hardware-Systementitäten  $\mathbb{S}_{e(HW)_i}$  bleiben dabei unverändert.

### 3.2.5 Effizienzmaße zur Bewertung

Eine Kopplung von Bewertungsmaßen führt zu komplexeren Maßen, die, zumeist bezogen auf eine spezielle Anwendung, eine höhere Aussagefähigkeit besitzen. Es handelt sich hierbei um so genannte Effizienzmaße.

**Definition 38** *Effizienzmaße*  $EM$  setzen unterschiedliche Bewertungsmaße  $BM$  in Relation zueinander und ermöglichen so tiefer gehende Vergleiche verschiedener Systemrealisierungen in Bezug auf charakteristische Eigenschaften des betrachteten Systems.

$$\frac{BM_i}{BM_k} = EM_{i,k}, \text{ mit } i \neq k \quad (3.24)$$

Weitere Schachtelungen von (3.24) sind möglich. Handelt es sich bei Dividend und Divisor um Bewertungsmaße der Dimension Energie, so definiert das sich ergebende Effizienzmaß den Wirkungsgrad.

Ein Effizienzmaß zur Verknüpfung von Performanz und Leistungsaufnahme ist die Anzahl der möglichen „*Millionen Operationen pro Sekunde pro Watt*“ ( $MOPS/Watt$ ), wobei hier nicht direkt auf den tatsächlichen Durchsatz geschlossen werden kann, da die Anzahl der Operationen und deren Leistungsfähigkeit nicht zwangsläufig bei jeder Architektur gleich zu bewerten sind, vgl. Abschnitt 6.2.3. Dies wird prinzipiell schon bei den Ansätzen von RISC- und CISC-Architekturen deutlich. Sicherlich aussagekräftiger ist der *Durchsatz pro Watt* (z. B.  $MBit/s/Watt$ ), bei einer gegebenen Anwendung, allerdings nicht so gut geeignet für generelle Vergleiche. Für Kommunikationssysteme von besonderer Aussagekraft ist das Effizienzmaß *benötigte Takte pro Datenbit*, was zum einen eine Bewertung unterschiedlicher Architekturen erlaubt und zum anderen eine Charakterisierung unterschiedlicher Algorithmen ermöglicht, vgl. Kapitel 7.

Ein Effizienzmaß zur Abschätzung der Zukunftssicherheit ist der Anteil wiederverwendeter bzw. wiederverwendbarer Hardwareblöcke an der Gesamtfläche einer neuen Architekturvariante. Entsprechendes lässt sich sinngemäß auch für Softwarebestandteile definieren.

Zur Abschätzung der Wirtschaftlichkeit lassen sich Effizienzmaße wie *Preis pro Chipfläche* oder *Performanz pro Chipfläche* oder auch *Leistungsaufnahme bezogen auf den Durchsatz* für eine gegebene Anwendung einsetzen.

### 3.3 Die vier bestimmenden Kostenmaße der Ressourceneffizienz

Im Folgenden wird die prinzipielle Vorgehensweise zur Bewertung und Realisierung ressourceneffizienter Systeme vorgestellt. Die vier Kostenmaße, die den Entwurfsraum definieren, und auf diese wesentlich Einfluss nehmenden Faktoren werden näher charakterisiert. Die zur Bewertung der Ressourceneffizienz nach Definition 9 aufzustellende Kostenfunktion  $CF$  beinhaltet folgende, von mir für wesentlich befundene Kostenmaße  $\mathbb{K}$ : **Leistungsaufnahme**, **Performanz** bzw. **Rechenleistung**, **Chipfläche** bzw. **Preis** und **Zukunftssicherheit** bzw. **Flexibilität** (vgl. Definition 5). Zumeist besteht eine diametrale Wechselwirkung zwischen den einzelnen Kostenmaßen, vgl. Abbildung 3-2.



Abbildung 3-2: Die vier Kostenmaße zur Bestimmung der Ressourceneffizienz

Zur Abschätzung der Ressourceneffizienz eines Systems müssen zunächst die Systemanforderungen und die damit verbundenen Randbedingungen festgelegt werden. Diese spiegeln sich zum einen in den Gewichtungsfaktoren  $c$  für die relevanten Bewertungsmaße  $BM$  der jeweiligen Zielfunktionen  $ZF$  der vier Kostenmaße  $\mathbb{K}$  wider. Zum anderen wird die Relevanz jedes einzelnen Kostenmaßes mit Hilfe von, den Randbedingungen angepassten, Gewichtungen  $\alpha$  in der Kostenfunktion  $CF$  nach Definition 9 berücksichtigt. Ziel ist es, die Kostenfunktion, also die eigentlichen Kosten des zu realisierenden Systems, zu minimieren, um so eine den Anforderungen des Einsatzgebietes entsprechende, möglichst optimale Implementierung zu erzielen (vgl. Definition 10). Im Rahmen des GigaNetIC-Projekts geschieht dies durch die in den Kapiteln 5 und 6 vorgestellte Werkzeugkette und den verfolgten hierarchischen Optimierungsansatz der Systemarchitektur. Abbildung 3-3 verdeutlicht die wesentlichen Schritte und deren Reihenfolge zur Feststellung und Bewertung der Ressourceneffizienz von verschiedenen Systemkonzepten.

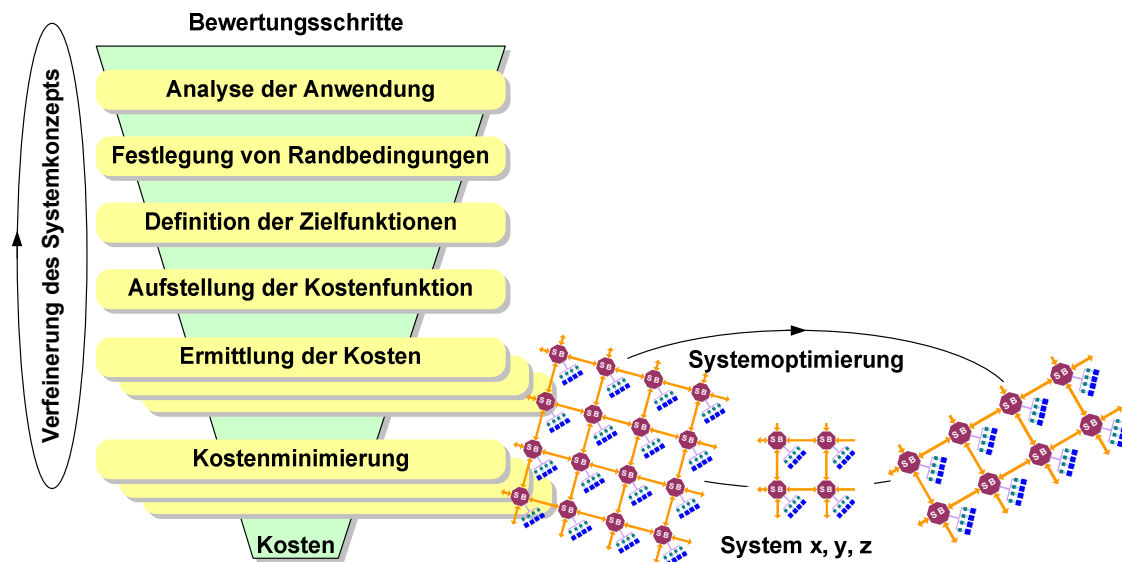


Abbildung 3-3: Vorgehensweise zur Bewertung der Ressourceneffizienz

Abbildung 3-4 veranschaulicht den durch diese vier Kostenmaße nach Definition 11 aufgespannten *Entwurfsraum* und zeigt das wesentliche Für und Wider der einzelnen Kostenmaße auf. So lässt sich z. B. die *Rechenleistung* durch Steigern der Parallelität oder der Taktrate erhöhen. Auch eine Optimierung der Hardware und der Software bedeutet einen Leistungszuwachs. Allerdings erhöht sich z. B. durch stärkere Parallelität die Fläche, was die diametrale Verknüpfung der einzelnen Dimensionen verdeutlicht. Durch Verwendung einer älteren Technologie, nicht optimierter Hardware oder Software hingegen wird die Rechenleistung reduziert. Die *Leistungsaufnahme* einer Multiprozessorarchitektur wird u. a. durch Faktoren wie Technologie, Taktfrequenz und Fläche beeinflusst.

Die *Chipfläche* und damit einhergehend auch der *Preis* des Bausteins können ebenfalls durch den Grad der Parallelisierung und die Art der Kommunikationsinfrastruktur des Chips (seriell / parallel / heterogen) variiert werden. Die Artung der Kommunikationsinfrastruktur beeinflusst nach außen hin zudem die Anzahl der Ein- und Ausgänge (*I/Os*) und damit die Gehäusekosten.

Als ein eher abstraktes Kostenmaß, das nicht so leicht zu messen bzw. auch zu bewerten ist, gestaltet sich die *Zukunftssicherheit* bzw. *Flexibilität* der Architektur. Dieses neue Kostenmaß ergänzt die bereits in der Literatur etablierten drei Kostenmaße (ebendort oft auch Kostenfaktoren genannt). In Zukunft wird es vermutlich eine immer wichtigere Rolle einnehmen. In Zeiten, in denen die Wertschöpfungskette nicht mehr nur in dem Verkauf des Produkts allein besteht, sondern mehr und mehr auch durch kostenpflichtige Dienste Einnahmen erzielt werden, kommt es besonders auf die Flexibilität und die Erweiterbarkeit der Merkmale eines Produktes an. Nicht zuletzt auch deshalb, um das Produkt möglichst lange am Markt zu halten und auf kommende Anforderungen ohne Hardwaremodifikationen reagieren zu können.

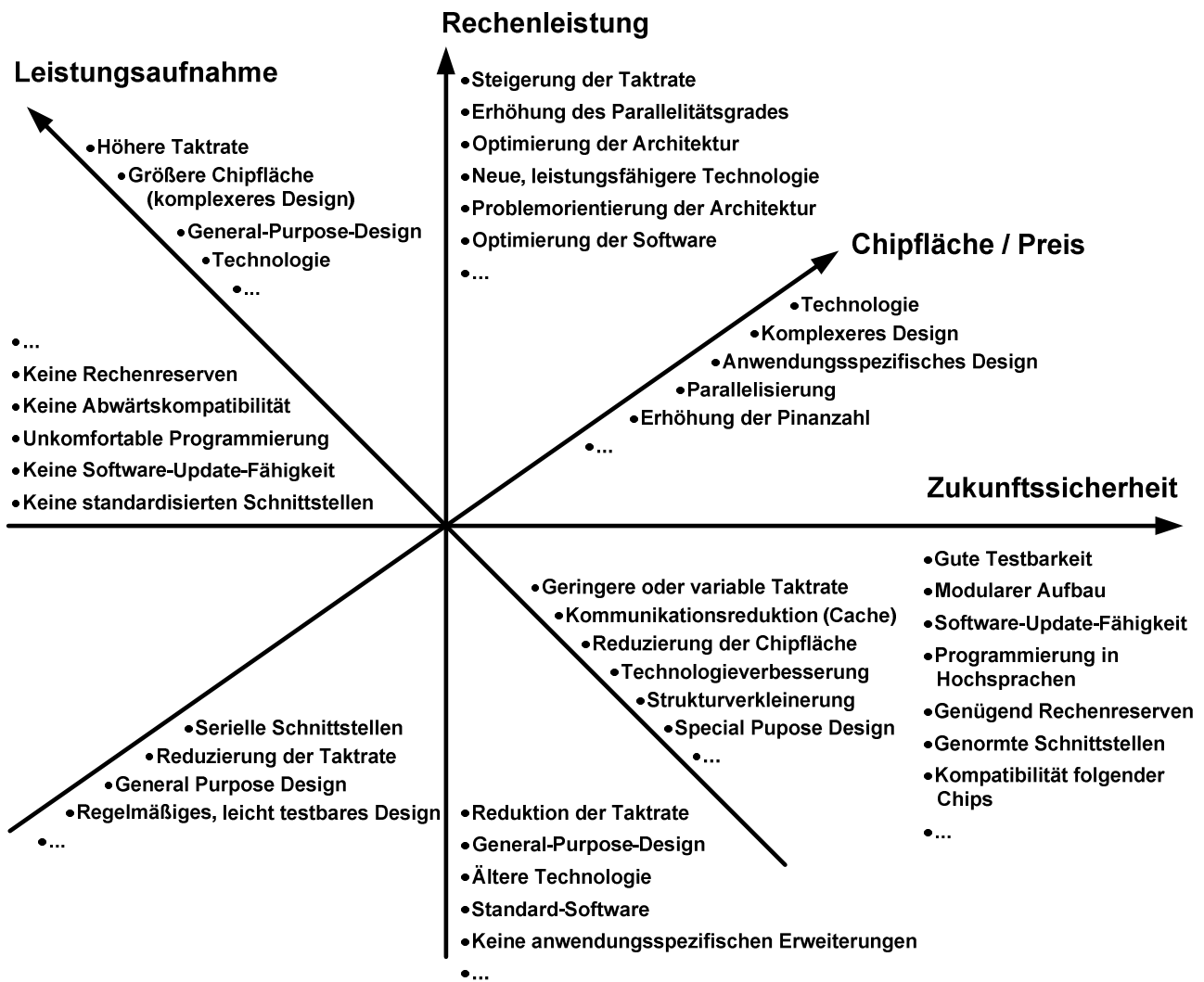


Abbildung 3-4: Vier Dimensionen des Entwurfsraums, aufgespannt durch die vier Kostenmaße

### 3.4 Zusammenfassung

In diesem Kapitel wurden wesentliche Begriffe eingeführt, die zur grundlegenden Definition der Ressourceneffizienz und zur Bewertung eingebetteter Parallelrechner und ihrer Komponenten führen. Mit Hilfe der aufgeführten Formalismen können im Folgenden die Systemimplementierungen aus Kapitel 4 sowie die Erweiterungen und Optimierungen aus den Kapiteln 6 und 7 mit Hilfe der ganzheitlichen GigaNetIC-Werkzeugkette (vgl. Kapitel 5) charakterisiert und im Sinne der unter Definition 14 gegebenen Begrifflichkeit im Hinblick auf Ressourceneffizienz bewertet werden. Eine exemplarische Untersuchung von unterschiedlichen Realisierungsvarianten zur effizienten Paketverarbeitung mit Hilfe der hier geschilderten Methode wird in Abschnitt 6.4 vorgestellt.

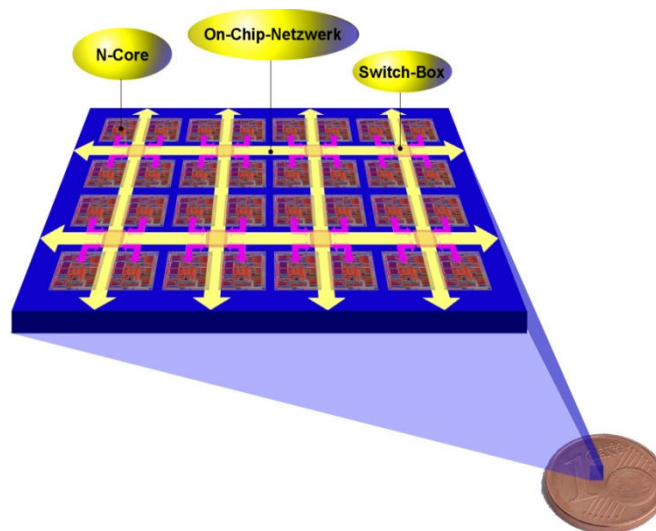
## 4 Die GigaNetIC-Systemarchitektur

In diesem Kapitel wird ein neuartiges Systemkonzept für Chip-Multiprozessoren vorgestellt. Die Besonderheiten dieser Architektur sind die Skalierbarkeit, die hohe Flexibilität und der konsequent verfolgte ganzheitliche Ansatz bei der Umsetzung des Architekturkonzepts. So wurde nicht nur am Entwurf und der Optimierung der Hardware gearbeitet, sondern in Kooperation mit den Fachgebieten „Programmiersprachen und Übersetzer“, Prof. Uwe Kastens, und „Algorithmen und Komplexität“, Prof. Friedhelm Meyer auf der Heide, der Universität Paderborn eine geschlossene Werkzeugkette mit Compiler, Programmiermodell und Simulatoren geschaffen. Diese Ganzheitlichkeit zeichnet die GigaNetIC-Architektur in erster Linie aus, da man ein solch komplexes Multiprozessorsystem nicht nur von einer Seite betrachten darf.

### 4.1 Neuartiges, ressourceneffizientes und skalierbares CMP-Systemkonzept

BENINI et al. zeigen in [107] die Notwendigkeit der Skalierbarkeit für zukünftige Systeme auf. *Ambient Intelligence (AmI)*, also allumgebende „maschinelle“ Intelligenz in Form von komplexen Systemen, wird in Zukunft eine immer größere Rolle in unserem täglichen Leben spielen. AmI erfordert energieeffiziente, hoch-performante Rechensysteme mit intelligenten Sensoren und Aktoren, die in jeder Hinsicht hoch skalierbar, für den jeweiligen Einsatzzweck optimal konfiguriert sein müssen. Sowohl Hardware, Kommunikationsstrukturen als auch Software müssen skalierbar gehalten sein, um die kommenden Anforderungen des Marktes erfüllen zu können.

Die im Folgenden vorgestellte GigaNetIC-Architektur ist konzipiert worden, um all diese Ansprüche zu erfüllen. Auch Aspekte der Hochverfügbarkeit, also der Fehlertoleranz und Ausfallsicherheit, können durch ein redundant ausgelegtes GigaNetIC-System in gewissen Grenzen realisiert werden.



**Abbildung 4-1: GigaNetIC-Architektur – 32 Prozessoren untergebracht auf einem 20tel der Fläche eines Cents**

Abbildung 4-1 verdeutlicht die hohe Integrationsdichte und die Flächenverteilung einer Architekturvariante des GigaNetIC-Systems, die 32 N-Core-Prozessoren (vgl. Abschnitt 4.3.1) und acht Switch-Boxen (vgl. Abschnitt 4.2) umfasst. Diese Realisierung lässt sich in einer aktuellen 90-nm-Technologie auf ca. einem 20tel der Fläche eines Centstücks integrieren. Zusammen mit 1,3 MByte

On-Chip-Speicher findet das gesamte System auf weniger als der Hälfte der Fläche des besagten Geldstücks Platz. Abweichend von den Ansätzen immer komplexer werdender, hoch spezialisierter Recheneinheiten der letzten Jahre (vgl. Kapitel 2), bei denen eine möglichst hohe Taktfrequenz an oberster Stelle steht, wird bei dem GigaNetIC-System das Prinzip einer massiv parallelen Architektur verfolgt. Hierbei werden weniger "hochgezüchtete" CPUs mit geringer Pipelintiefe vielfach instanziiert. Dies geschieht frei nach dem Prinzip: „Teile und herrsche“, d. h. die Arbeit wird auf eine Vielzahl relativ einfacher Verarbeitungseinheiten verteilt. Allerdings ist der Erfolg dieser Methode abhängig von der Anwendung und ihrer Parallelisierbarkeit<sup>14</sup>.

Die GigaNetIC-Architektur ist im Grundsatz eine „General Purpose“-Systemarchitektur, also eine zunächst universell einsetzbare Multiprozessorrealisierung. Bei vielen der im Folgenden vorgestellten Aspekte wurde sie insbesondere für den Einsatz als Netzwerkprozessor getestet, durch anwendungsspezifische Hardwarebeschleuniger erweitert und optimiert (vgl. Kapitel 6 und 7). Bei der prototypischen Realisierung (vgl. Kapitel 8) wurde ebenfalls besonderes Augenmerk auf die Integration der Architektur in ein Netzwerkanwendungsszenario gelegt.

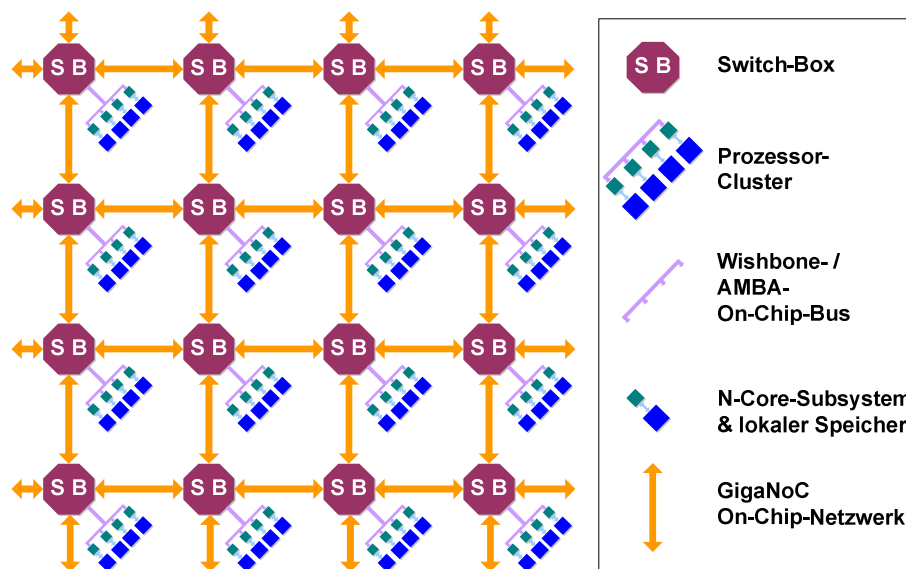


Abbildung 4-2: Schematischer Aufbau der massiv parallelen GigaNetIC-Architektur

Die entworfene GigaNetIC-Architektur (vgl. Abbildung 4-2) beruht auf massiv paralleler Verarbeitung, die durch eine Vielzahl homogener Verarbeitungseinheiten ermöglicht wird. Diese Rechenknoten basieren auf einem im Fachgebiet Schaltungstechnik der Universität Paderborn, Prof. Rückert, entworfenen 32-Bit-RISC-Prozessorkern [108]. Diese werden in einer hierarchischen System-Topologie über eine leistungsfähige, mehrstufige Kommunikationsinfrastruktur, den GigaNoC [109][110], miteinander verbunden. Die GigaNetIC-Architektur lässt sich strukturell in drei Ebenen [6] unterteilen:

- **Prozessor-Ebene**

Diese Ebene kennzeichnet die unterste Ebene des GigaNetIC-Systems. Auf dieser Ebene finden feingranulare Modifizierungen und Optimierungen wie z. B. Befehlssatzerweiterungen des Prozessors statt (vgl. Abschnitt 6.2 und 7.3). Dies betrifft jeweils einzelne Module, also alle *Intellectual Property(IP)*-Blöcke, die hier zum Einsatz kommen, wie z. B. Prozes-

<sup>14</sup> Vgl. hierzu die Gesetze von AMDAHL (2.1) und GUSTAFSON (2.2), Abschnitte 2.1.1 und 2.1.2.



sorelemente (*PE*), Hardwarebeschleuniger (*HW-Acc*) (vgl. Abschnitt 6.3, 7.4.2.3 und 7.7), Speicher oder Peripherie-Blöcke. Im Falle der zentralen Verarbeitungseinheit (*CPU*) konzentrieren sich hier die Aktivitäten auf den N-Core [108][111] (vgl. auch Abschnitt 6.2).

- **Cluster-Ebene**

Auf dieser Ebene sind die einzelnen Module über einen lokalen Bus oder eine Switchmatrix verbunden. Der verwendete Compiler [111][112] kann sowohl Optimierungen auf Instruktionsebene (*Instruction Level Parallelism / ILP*) vornehmen, als auch die Existenz von anwendungsspezifischen Hardwarebeschleunigern ausnutzen. Auf dieser Ebene kann bei Bedarf ein eigens entworfener Multiprozessor-Cache eingesetzt werden [113] (vgl. auch Abschnitt 4.4.2). Switch-Boxen [114] (vgl. Abschnitt 4.2.1) agieren als Hochgeschwindigkeits-Routingknoten und entlasten die Prozessoren bzw. die lokalen Hardwareblöcke als so genannte „NoC Offload Engines“ beim Empfangen und Senden von Daten von bzw. zu anderen Clustern bzw. externen Schnittstellen des SoCs.

- **SoC(System-on-Chip)-Ebene**

Rückgrat der *SoC-Ebene* ist die Kommunikationsinfrastruktur, die sowohl die Kommunikation auf dem Chip als auch die Anbindung des Off-Chip-Speichers und der IOs gewährleistet. Für den Programmierer geschieht dies transparent, da die Wegewahl und das Speichermanagement von den Switch-Boxen (vgl. Abschnitt 4.2.1) gesteuert werden. Zur komfortablen Nutzung der Kommunikationsinfrastruktur werden spezielle intrinsische Funktionen [115][109] als Software-Bibliothek zur Verfügung gestellt (vgl. Abschnitt 4.2.2). Darüber hinaus wurde ein globales Programmiermodell zur Ausnutzung der SoC-weiten Parallelität eingeführt (vgl. Abschnitt 4.5).

Ein Hauptziel meines Ansatzes ist, dass der resultierende Multiprozessor in Bezug auf die Anzahl der Cluster, der pro Cluster instanziierten Prozessoren sowie die zur Verfügung gestellte Bandbreite durch die Kommunikationskanäle leicht parametrisierbar sein soll. Außerdem sollen definierte Schnittstellen zum einfachen Integrieren von kundenspezifischen Hardwarebeschleunigern und Peripherieblöcken die Performanz und die Zukunftssicherheit des Konzepts maximieren. Auf diese Weise kann eine große Wiederverwendbarkeit (vgl. Abschnitt 3.2.4, insbesondere Definition 36) dieser Architektur durch Skalierung auf vielfältige Einsatzgebiete gewährleistet werden (vgl. Abschnitt 3.3). Dies erhöht auch die Ressourceneffizienz des Systems (vgl. Abschnitt 3.1, insbesondere Definition 14).

Die drei Hierarchieebenen bieten dem Entwickler definierte Ansatzpunkte zur Optimierung bzw. Parametrisierung des Systems in Bezug auf den späteren Einsatzzweck und die damit verbundenen Anforderungen (vgl. Kapitel 6). Dies wird zudem von der entwickelten Werkzeugkette komfortabel unterstützt (vgl. Kapitel 5) und ermöglicht somit schnelle Entwicklungszyklen. Weitere Vorteile einer solchen homogenen Systemarchitektur liegen in dem einheitlichen Programmiermodell und der vereinfachten Testbarkeit und Verifikation. Dies wirkt sich ebenfalls positiv auf die Entwicklungszeit des Gesamtsystems aus.

Die spezielle parallele und redundant auslegbare Architektur des GigaNetIC-Systems bietet zudem großes Potential für die Realisierung nach Definition 33 fehlertoleranter Chip-Multiprozessoren und ermöglicht somit eine Erhöhung der Chipausbeute (*Production Yield*), was zur Steigerung der Ressourceneffizienz beiträgt. Vorausgesetzt die Hardware und die Betriebssoftware ist dafür ausgelegt, erreicht man sogar – in gewissen Grenzen – eine Immunität des Systems gegenüber Fehlern (nach

Definition 34) und damit gegebenenfalls eine weitere Steigerung der Ressourceneffizienz. Mit der zunehmenden Miniaturisierung und den hervorragenden Eigenschaften der GigaNetIC-Architektur bezüglich Skalierbarkeit und Programmiermodell bieten sich optimale Voraussetzungen, um in Zukunft fehlertolerante bzw. sogar fehlerimmune Systeme zu konstruieren.

Abbildung 4-3 zeigt die Zuordnung der einzelnen GigaNetIC-Systementitäten zu dem bereits in Abschnitt 2.3.2 vorgestellten NoC-Schichtenmodell. Zur physikalischen Schicht sind die Verbindungsleitungen zwischen den Switch-Boxen (*Inter-Switch-Box-Links*) und die spezifischen Pufferspeicher für die Flits zur Vermeidung von Blockaden (*Advanced Buffer*) zu zählen. Die Architektur und Steuerungsschicht umfassen die funktionalen Einheiten der Switch-Boxen und die vordefinierten Software-Bibliotheksfunktionen, die den Prozessorelementen eine komfortable Schnittstelle zu den Funktionen des On-Chip-Netzwerks zur Verfügung stellen. Zur Schicht 3 und damit zur Softwareschicht gehören die Anwendungssoftware, wie z. B. Paketverarbeitungsalgorithmen beim Einsatzgebiet als Netzwerkprozessor (vgl. Kapitel 7), und das Programmiermodell des Gesamtsystems (vgl. Abschnitt 4.5).

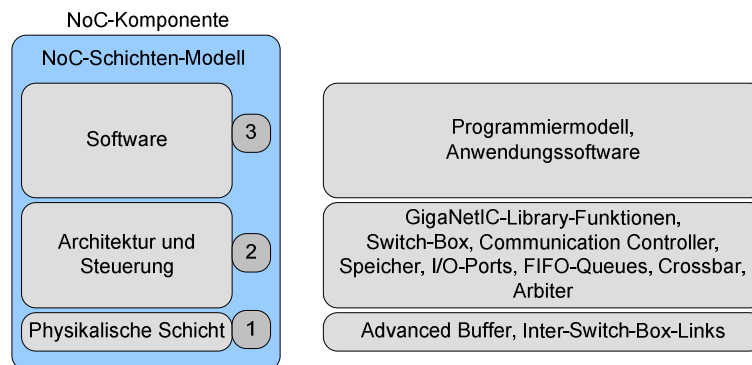


Abbildung 4-3: Zuordnung wesentlicher Systementitäten zum NoC-Schichtenmodell

Im Folgenden wird der Aufbau der GigaNetIC-Architektur im Detail beschrieben, angefangen bei der *GigaNoC*-On-Chip-Kommunikationsstruktur und dem hierfür eigens entworfenen On-Chip-Kommunikationsprotokoll [110], über die Anbindung von Verarbeitungseinheiten [108][6][111] und IP-Blöcken [116][117], die Implementierung eines GigaNetIC-konformen Multiprozessor-Caches [113], bis hin zum Programmiermodell des Gesamtsystems [6] und zu einer Werkzeugkette zur modularen, effizienten Modellierung von Netzwerkanwendungen [118][119].

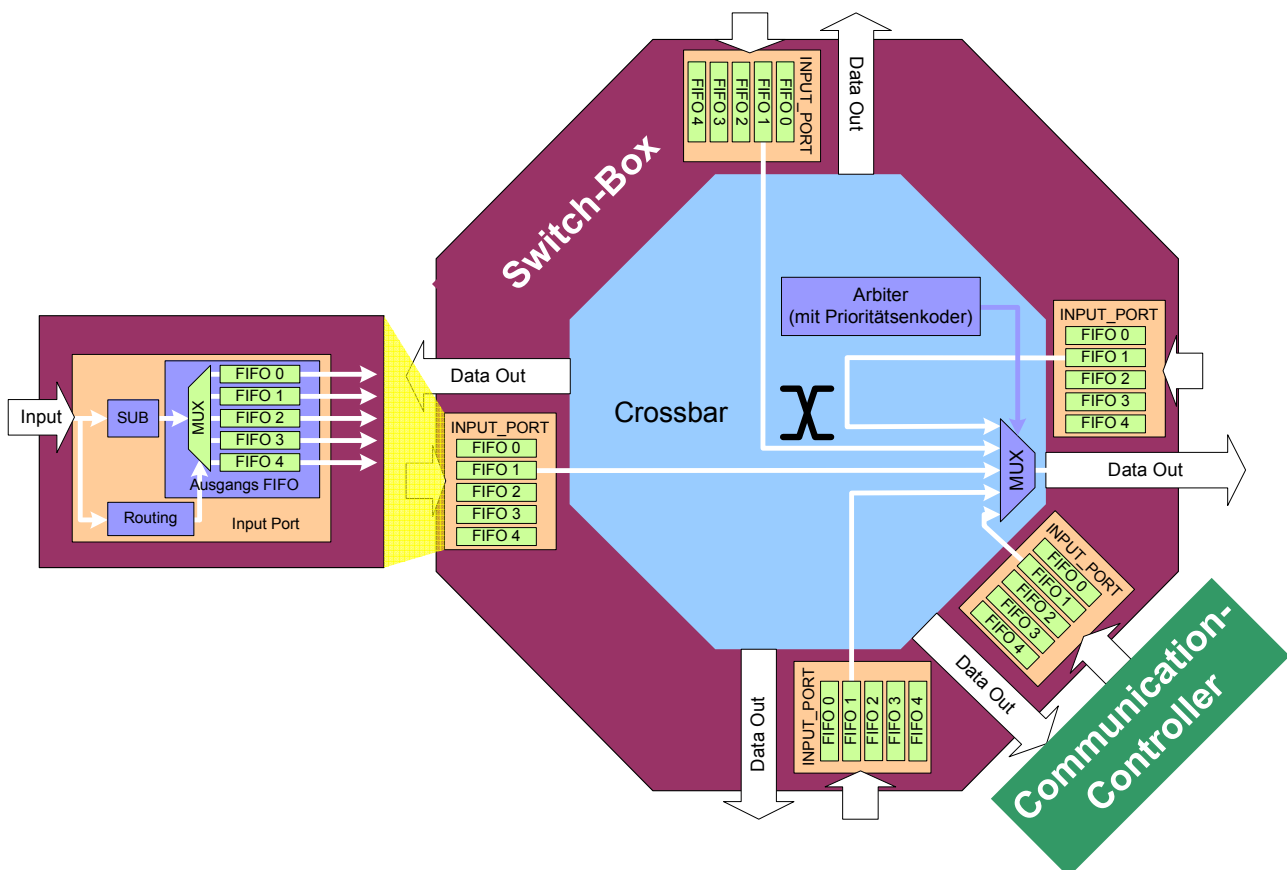
## 4.2 GigaNoC-On-Chip-Kommunikationsstruktur

Bei dem GigaNetIC-Prototypen (vgl. Kapitel 8) wurde ein zweidimensionales Gitter (vgl. Abbildung 2-8) für die obere Topologiehierarchiestufe realisiert. Diese Topologie ist mittlerweile technologisch aufgrund symmetrischer Leitungslängen und Laufzeiten beherrschbar und bietet weitere Vorteile für On-Chip-Netzwerke (vgl. Abschnitt 2.3). Die Module innerhalb der Cluster der unteren Hierarchiestufe werden über lokale Bussysteme, wie z. B. Wishbone oder AMBA, oder über Switchmatrixen miteinander verbunden. Abbildung 2-9 b) stellt eine freie, hierarchische Topologie dar, die keinerlei Restriktionen hinsichtlich der Verbindungen zwischen den einzelnen Clustern unterliegt. Die GigaNetIC-Architektur ist zwar für eine solche Variante vorbereitet, technologisch werfen sich allerdings für derzeitige Standardzellprozesse mitunter große Probleme auf, so dass eine Realisierung einer derart unregelmäßigen Struktur momentan nicht angedacht ist.

Für NoCs gibt es, wie in Abbildung 2-16 gezeigt, konkurrierende Faktoren, die gegeneinander abgewogen werden müssen, allerdings besteht hier der Vorteil, dass nicht zwangsläufig existierende strenge Kommunikationsstandards eingehalten werden müssen, da diese Netzwerke normalerweise in sich abgeschlossen sind. Zu diesen Kosten zählen die bereits erwähnten Faktoren wie *Leistungsaufnahme* und damit einhergehend eine möglichst direkte Wegwahl. Als nächstes ist der *Flächenbedarf* zu nennen, der direkt mit der Router-Architektur verknüpft ist, die zudem direkt maßgeblich die *Leistungsfähigkeit* der Datenübertragung bestimmt. Ein weiterer sehr bedeutender Punkt ist die Robustheit des Netzwerks gegenüber sich verändernden Datenverkehrsverteilungen. Hier wird die Qualität des Netzwerks nicht zuletzt durch die verwendeten Routingalgorithmen definiert. Statische Wegwahlverfahren eignen sich vor allem für deterministisches Verkehrsaufkommen, wohingegen sich adaptive, also dynamische Routingverfahren durch Flexibilität bei veränderlichem Lastverhalten auszeichnen. Durch die Wahl des Routingverfahrens wird deshalb die *Flexibilität* und somit die *Zukunftssicherheit* des NoCs definiert.

#### 4.2.1 Switch-Boxen als zentrale Kommunikationsknoten auf SoC-Ebene

**System-on-Chip-Ebene.** Auf *SoC-Ebene* fungieren Switch-Boxen (vgl. Abbildung 4-4) als Hochgeschwindigkeits-Routingknoten, die die einzelnen Cluster des SoCs miteinander verbinden [114]. Die On-Chip-Kommunikation ist paketbasiert (vgl. Abschnitt 4.2.2).



**Abbildung 4-4: Switch-Box-IP-Block mit Struktur eines Ports und des Kreuzschienenverteilers.  
Explizite Darstellung für Eingangsport 3 und den Ausgang von Port 1.**

Aufgrund der generischen Beschreibung in VHDL, die u. a. Variationen der Anzahl der Ports zulässt, können nahezu beliebige Netzwerktopologien aufgebaut werden. Für die prototypische Realisierung wird zunächst ein Gitter implementiert. Trotz dieses regelmäßigen und einfachen Aufbaus

erlaubt diese Architektur Pipelining und Parallelverarbeitung der Prozessorfelder. Außerdem garantiert diese Topologie gleichlange Verbindungsleitungen und damit gleichlange Signal-Laufzeiten zwischen den Switch-Boxen. Die parallele, Switch-Box-basierte Architektur erlaubt des Weiteren eine hohe Fehler- bzw. Ausfalltoleranz. Sollte ein Prozessorfeld ausfallen, so kann, sofern die Software dies unterstützt, ein anderer Cluster dessen Funktionalität übernehmen.

Die Switch-Box besteht als aktiver Netzwerkknoten aus zwei Hauptteilen. Der eine Teil bildet mit den Eingangsports und dem Kreuzschienenverteiler (*Crossbar*) mit integriertem Prioritäts-Enkoder-basierten Round-Robin-Arbiter die Kommunikationsstruktur. Diese sorgt dafür, dass die Datenpakete problemlos durch den Knoten geleitet werden und mit der Routing-Strategie den korrekten Ausgang der Switch-Box erreichen. Der zweite Teil der Switch-Box umfasst Kontrollstrukturen, die als Schnittstelle zwischen dem Prozessorfeld und dem Chipnetzwerk dienen. Diese Kopplung übernimmt der *Communication-Controller (CC)*, welcher zwischen dem Port 0 und dem Bussystem des Prozessorfeldes angeordnet ist (vgl. Abbildung 4-4).

#### 4.2.1.1 Communication-Controller

Der *Communication-Controller* wird in Abbildung 4-5 detaillierter dargestellt, zu seinen wesentlichen Aufgaben zählen:

Die Bereitstellung der für den lokalen Cluster bestimmten Pakete geschieht auf Anfrage der Prozessoren. Die Eingangsports der Switch-Box erkennen, dass ein Flit für den lokalen Port gedacht ist, daran, dass die Koordinaten im Kopf eines Flits null sind. Die Flits müssen nicht zwangsläufig nach Paketen sortiert ankommen, noch müssen die Flits eines Pakets in der richtigen Reihenfolge sortiert sein. Die Pakete werden also gegebenenfalls vom *CC* geordnet. Er übernimmt die Speicherverwaltung.

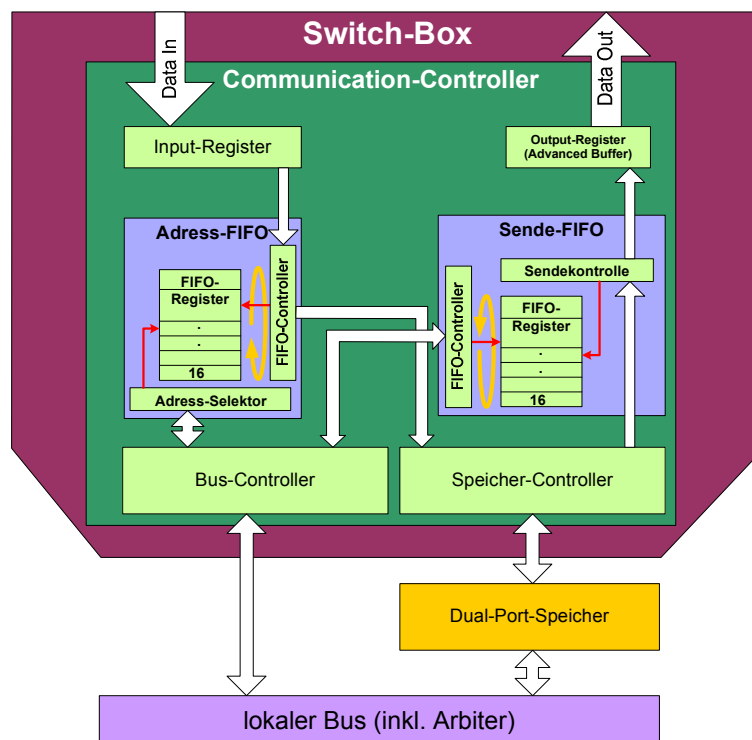


Abbildung 4-5: Der Communication-Controller, die NoC-Kommunikations-"Offload-Engine"

Eine weitere Hauptaufgabe des *CCs* ist das Versenden von Paketen, wenn ein Prozessor die Bearbeitung seiner Daten beendet hat, und diese weitergeleitet werden müssen. Dabei kann das Ziel ein

anderes Prozessorfeld sein, welches zusätzliche Funktionen hat, oder eine Ausgangsschnittstelle des Chips. Zusätzlich ist vorgesehen, dass sich die Programme, die auf den Prozessoren laufen, gegenseitig Nachrichten schicken können (*Message Passing*). Auch zu diesem Zweck müssen Pakete verschickt werden. Der Ablauf dabei ist wie folgt: Ein Prozessor teilt dem *CC* mit, dass es ein zu versendendes Paket gibt, und informiert ihn über die Adresse, an der die Daten im Speicher beginnen. Anschließend sorgt der *CC* dafür, dass die Daten zu einem Paket zusammengestellt werden und organisiert deren Transport zum Eingangsport Null. Von dort nehmen die Flits vollkommen automatisiert ihren Weg durch die Kommunikationsstruktur zum gewünschten Zielort.

In der Initialisierungsphase fungiert der *CC* als „Bootloader“. Er schreibt die Programmflits vollautomatisch in die entsprechenden Speicherbereiche für den Programmcode der Prozessorelemente. Dieses Verfahren wird selbständig nach dem *Hard Reset* angestoßen. Die entsprechenden Daten werden aus einem nichtflüchtigen, an einer Switch-Box angeschlossenen Speicher geladen. Hierzu wird ebenfalls die inhärente Funktionalität des GigaNoCs genutzt. Eine Übersicht der Befehle des Communication-Controllers wird in Abbildung Anhang A-2 gegeben.

#### 4.2.1.2 Topologie des Netzwerks

Wie in Abschnitt 2.3.1 ausführlich erläutert eignen sich zweidimensionale Strukturen für aktuelle Standardzell-Technologien derzeit am besten. Deshalb wird momentan auch beim GigaNoC eine gitterförmige Struktur präferiert, vgl. Abbildung 4-2. Durch Parametrisierung (vgl. Abschnitt 4.7) bzw. Änderung der Anordnung kann die Architektur aber auch in komplexere Topologien überführt werden. Hierzu können auch die zusätzlichen diagonalen Ports der Switch-Boxen eingesetzt werden, die den Grad  $\delta$  des einzelnen Netzwerkknotens bis hin zu derzeit 16 steigern können (vgl. Abbildung 4-6).

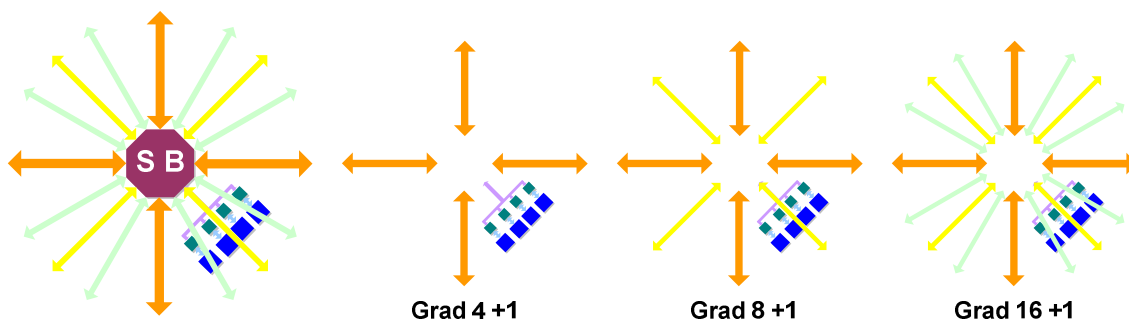


Abbildung 4-6: Parametrisierbarer Grad  $\delta$  der Switch-Box

Ebenso sind Topologien mit mehr als zwei Hierarchiestufen realisierbar. D. h. dem regelmäßigen Gitter aus Abbildung 4-2 kann z. B. durch Anschluss von Switch-Boxen anstelle der lokalen Cluster an den Port 0 des jeweils übergeordneten Knotens eine weitere Hierarchiestufe implementiert werden (vgl. Abbildung 4-7). Die Abbildung zeigt ein  $n$ -Hierarchieebenen umfassendes System, wobei die  $n$ -te Dimension durch den lokalen Cluster repräsentiert wird.

Die einfache Parametrisierbarkeit des Systems erlaubt die entsprechende Realisierung der benötigten Bandbreiten. Natürlich sind auch andere Konstellationen, z. B. wie in Abbildung 2-9 b), denkbar, allerdings würden diese Topologien ggf. Erweiterungen bzgl. des Routings erfordern.

### 4.2.1.3 Switching-Methode

Eingangsdaten werden, solange noch Kapazitäten frei sind, in den Ports einer Switch-Box direkt in eine FIFO-Warteschlange (*Queue*) eingereiht. Ansonsten wird dem Quellknoten signalisiert, dass die beabsichtigte Richtung bzw. Warteschlange blockiert bzw. gefüllt ist. Die Übertragung wird dann, um eine definierte Anzahl an Takten verzögert, erneut seitens des Quellknotens initiiert. Die FIFO-Puffer arbeiten mit zwei unabhängigen Zeigern für den Lese- und Schreibzugriff, die von einer speziellen Kontrolleinheit gesteuert werden. Diese sorgt dafür, dass zum einen das FIFO-Prinzip eingehalten wird, und zum anderen, sollte die Warteschlange leer sein, die Daten nicht erst taktweise durch eine Registerkette hindurchgeschleust werden müssen, sondern sofort zum Ausgang gelangen. Obwohl die FIFO-Struktur physikalisch am Eingangsport platziert ist, handelt es sich dennoch um eine Form des *Output Queueings*, das *Virtual Output Queueing (VOQ)* [120]. KAROL et al. [121] konnten zeigen, dass die Methode des Output Queueings deutliche Vorteile gegenüber dem *Input Queueing* aufweist. Nicht zuletzt deshalb wird *VOQ* auch beim GigaNoC eingesetzt. Da getrennte Warteschlangen für jeden Ausgang vorhanden sind, wird das Problem des so genannten *Head-of-Line-Blockings* (vgl. Abschnitt 2.3) in Abhängigkeit von der Größe des Warteraums verringert.

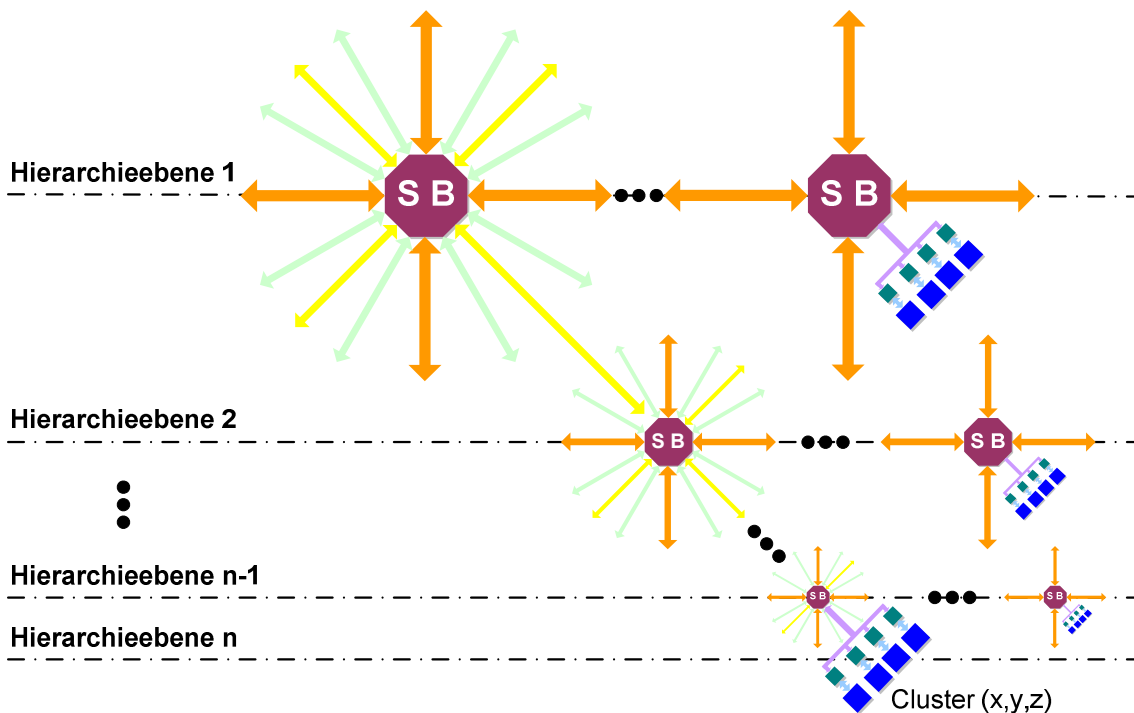


Abbildung 4-7: On-Chip-Netzwerk höherer Hierarchie, basierend auf dem GigaNoC

Bei der Verbindungsart der Ports sind verschiedene Lösungen wie Bussysteme, Kreuzschienenverteiler oder Punkt-zu-Punkt-Verbindungen denkbar. Letztere bieten zwar eine absolute Blockierungsfreiheit, aber dafür müssen an jedem Ausgang mehrere Anfragen parallel verarbeitet werden. Zusammen mit der hohen Anzahl der benötigten Verbindungen steigert dies die Kosten wesentlich. Ein Bus hat vergleichsweise geringe Kosten, doch werden alle Anfragen bis auf eine blockiert. Der Kreuzschienenverteiler stellt ein nicht-blockierendes Netz mit mittleren Kosten und hohem Durchsatzpotential dar. Er kommt deshalb als Verbindungsstruktur innerhalb der Switch-Box in einer auf Geschwindigkeit optimierten Variante, der in [122] dargestellten Form, zum Einsatz.

Grundlegend war zu entscheiden, ob die Daten paketbasiert durch die Knoten geleitet werden (*Packet Switching / PS*), oder ob vor der Übertragung physikalische Kanäle reserviert werden (*Circuit Switching / CS*). Wie in Abschnitt 2.3.2 erläutert, bietet sich für universelle Multiprozessorsysteme, bei denen die Aufgaben und Lastverteilungen nicht im Vorfeld genauestens bekannt sind, eine Variante der Packet-Switching-Verfahren an. Deshalb wurde beim GigaNoC eine Form des *Wormhole-Switching-Verfahrens*, das sich durch die in Abschnitt 2.3.2 erwähnten Vorteile (geringer Pufferbedarf, kleinere *SAF*-Latenzzeiten, gute Eignung für kompakte und leistungsfähige Hardwarerealisierung) deutlich von anderen Switching-Verfahren absetzt, implementiert. Der Nachteil, dass es zu Blockierungen kommen kann, wird durch die spezielle Art des bei der Switch-Box eingesetzten *Virtual Output Queueings* deutlich reduziert. Für eine einfache Gitterstruktur des GigaNetIC-CMPs wäre das in Abschnitt 2.3.2 vorgestellte „*Mad Postman*“-Switching ebenfalls eine vielversprechende Methode, die sich aufgrund der Switch-Box-Struktur leicht in den bestehenden Entwurf integrieren ließe. Für zukünftige Realisierungen kann mit Hilfe der SiMPLE-Entwicklungs-umgebung (vgl. Abschnitt 5.2) im Vorfeld der Chipfertigung ein Vergleich der vielversprechenden Switching-Varianten für das jeweilige Anwendungsszenario erfolgen, um so die Leistungsfähigkeit der Kommunikationsinfrastruktur optimal an die Anwendungsanforderungen anzupassen.

#### 4.2.1.4 Routing

Das GigaNoC unterstützt mehrere Routingmechanismen, um möglichst flexibel auf spezielle Anwendungsszenarien oder Lastverteilungen reagieren zu können. Standardmäßig ist das *XY-Routing* (vgl. Abschnitt 2.3.2) aktiviert, da es sich, wie bereits diskutiert, als gedächtnisloses, deterministisches Routing besonders für Gitter- und Tori-Topologien eignet. Es bietet den Vorteil, dass keine *Live-* bzw. *Deadlocks* in mehr als einer Dimension auftreten können. Als Alternative kann komfortabel, mittels so genannter Instruktionsflits (vgl. Abschnitt 4.2.1.1) auf ein adaptives Routingverfahren, das auf Kostentabellen in den einzelnen Switch-Boxen basiert, umgeschaltet werden. Zusätzlich lassen sich bei Bedarf weitere anwendungsspezifische Routingmechanismen integrieren. Zwischen den einzelnen Verfahren kann während des Betriebs dynamisch umgeschaltet werden.

Durch Erweiterung der Funktionalität der Routingflits könnten Mechanismen wie *Broadcast* (das Weiterleiten eines Pakets an alle Teilnehmer des NoCs) oder *Multicast* (das Weiterleiten eines Pakets an eine Gruppe ausgewählter Teilnehmer des NoCs) realisiert werden. Ein anderer Ansatz wäre die Erweiterung und spezielle Auswertung der Adressierungsfelder der Flits ähnlich dem Internet-Protokoll. Beide Varianten bedeuteten Erweiterungen der Switch-Box-Struktur, wobei die sich ergebenden neuen Möglichkeiten sicherlich den Aufwand rechtfertigten.

#### 4.2.1.5 Flächenverteilung der Switch-Box-Komponenten basierend auf Synthesergebnissen

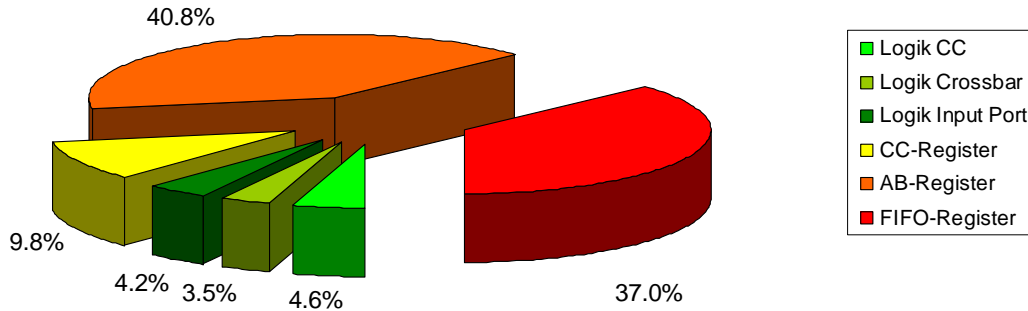
Abbildung 4-8 zeigt die qualitative Flächenaufteilung der wesentlichen Switch-Box-Komponenten nach der Synthese auf Standardzellen auf. Auffällig ist, dass bereits bei der FIFO-Tiefe = 3 mehr als 77 % der Fläche für FIFO- und Ausgangsregister benötigt werden. Durch zukünftigen Einsatz von modernster SRAM-Technologie kann dieser Flächenanteil, verglichen mit den derzeit verwendeten Registerzellen, deutlich reduziert werden [110].

**Abschätzung der Switch-Box-Fläche.** Die Gesamtfläche der Switch-Box  $A_{SB}$  setzt sich zusammen aus der Addition der Flächen der Hauptkomponenten *Communication-Controller*  $A_{CC}$ , *Crossbar*  $A_{Crossbar}$  und aller *Inputports*  $A_{Inputports}$ .



$$A_{SB} = A_{CC} + A_{Crossbar} + A_{Inputports} \quad (4.1)$$

In der VHDL-Implementierung ist eine Vielzahl von variierbaren Parametern enthalten, durch die das Gesamtsystem vor der Synthese flexibel gestaltbar und damit speziell auf Anforderungen neuer Anwendungsszenarien bzw. auf neue Technologien anpassbar ist. Die wesentlichen Parameter der Switch-Box, die teilweise direkt durch das On-Chip-Kommunikationsprotokoll (vgl. Abschnitt



4.2.2) beeinflusst werden, sind in Anhang B aufgeführt.

**Abbildung 4-8: Flächenaufteilung der einzelnen Switch-Box-Komponenten bei einer FIFO-Tiefe von drei**

Die Fläche des *Communication-Controllers* setzt sich aus zwei wesentlichen Bestandteilen zusammen. Die Fläche teilt sich in Bereiche auf, die entweder abhängig von oder aber invariant gegenüber der Flitbreite sind. Da in den FIFO-Strukturen des *CCs* keine vollständigen Flits, sondern nur Speicheradressen und Datenlängen gespeichert werden, hängen diese nicht vom Parameter *DATA\_WIDTH* (Gesamtbreite eines Flits, vgl. Abbildung 4-11) ab. Auch die Speicherschnittstelle *MEMORY\_CONTROLLER* und die Busanbindung *COM\_BUS\_CONTROLLER* sind von der Flitbreite unabhängig, da derzeit, unabhängig von der Anzahl der Datenworte eines Flits, 32 Bit breite Schnittstellen verwendet werden. Das Ausgangsregister *OUTPUT\_BUFFER* und das Eingangsregister *INPUT\_REG* hängen hingegen als Blöcke des *CCs* von der Flitbreite ab.

Alle folgenden Werte wurden für die Flitbreite *DATA\_WIDTH* = 93 Bit anhand umfangreicher Synthesen ermittelt. Die Syntheseresultate der Switch-Box ergeben, dass der gesamte *CC* zu 13,42 % von der Flitbreite abhängig ist, da dieser Teil hauptsächlich aus den Registern für die Flits besteht. Da sich diese Registerfläche linear zu der Flitbreite verhält, kann für die Gesamtfläche  $A_{CC}$  des *CCs* die Formel (4.2) aufgestellt werden.  $A_{93}^{CC}$  ist die Fläche des *CCs* bei einer Flitbreite von 93 Bit.

$$A_{CC} = (A_{93}^{CC} \cdot 0,8658) + \left( (A_{93}^{CC} / 93) \cdot DATA\_WIDTH \cdot 0,1342 \right) \quad (4.2)$$

Der Kreuzschienenverteiler enthält keine Registerstrukturen, in denen Flits zwischengespeichert werden. Allerdings besteht seine Fläche zu ca. 82 % aus Multiplexern, die sich ähnlich wie Register nahezu linear mit der Breite der Flits vergrößern. Zur Berechnung der Fläche  $A_{Crossbar}$  des Kreuzschienenverters wird in Analogie zu (4.2) die Fläche des *Kreuzschienenverters*  $A_{93}^{Crossbar}$  verwendet.

$$A_{Crossbar} = (A_{93}^{Crossbar} \cdot 0,18) + \left( (A_{93}^{Crossbar} / 93) \cdot DATA\_WIDTH \cdot 0,82 \right) \quad (4.3)$$

In den Eingangsports befinden sich die *FIFO-Register* der Breite *DATA\_WIDTH*, deren Tiefe mit *FIFO\_DEPTH* eingestellt wird. Die Registerfläche der FIFOs  $A_{Fiforeg}$  kann daher, analog zu (4.2)



und (4.3), wie folgt angegeben werden, und zwar mit  $A_{93}^{Fiforeg}$  für die erhaltene Fläche für ein *FIFO-Register* mit 93 Bit Breite:

$$A_{Fiforeg} = \left( A_{93}^{Fiforeg} / 93 \right) \cdot DATA\_WIDTH \cdot FIFO\_DEPTH \quad (4.4)$$

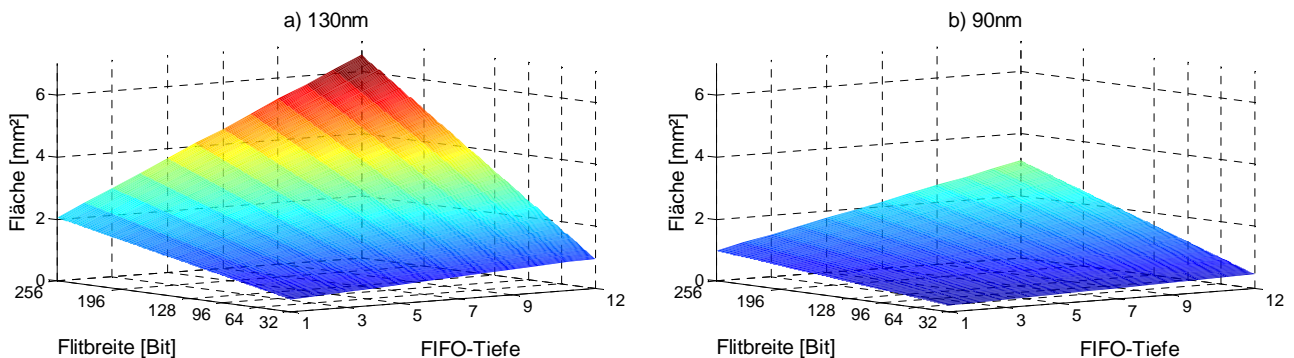
Jeder FIFO-Kette ist ein *ADVANCED\_BUFFER* nachgeschaltet, der u. a. das Head-of-Line-Blocking-Problem zusätzlich minimiert. Er ist ebenfalls von der Breite *DATA\_WIDTH* abhängig, so dass sich seine Fläche  $A_{AB}$  wie durch (4.5) beschrieben ergibt, unter Einbeziehung der Fläche eines *ADVANCED\_BUFFERS*  $A_{93}^{AB}$ .

$$A_{AB} = \left( A_{93}^{AB} / 93 \right) \cdot DATA\_WIDTH \quad (4.5)$$

Sowohl FIFO-Warteschlangen als auch Puffer werden in einer Anzahl generiert, die dem Quadrat der Anzahl der Switch-Box-Ports  $P_{SB}$  entspricht. Die Synthesergebnisse zeigen, dass diese Ports ca. 3,725% Logik enthalten, die weder von *DATA\_WIDTH* noch von *FIFO\_DEPTH* beeinflusst wird. Hieraus folgt (4.6) für die Fläche der *Inputports*  $A_{inputports}$ .

$$A_{inputports} = \frac{P_{SB}^2}{(1 - 0,03725)} \cdot \left( A_{Fiforeg} + A_{AB} \right) \quad (4.6)$$

Die Annäherungsformel (4.1) zur Bestimmung der Gesamtfläche der Switch-Box ist damit vollständig definiert. Abbildung 4-9 zeigt die sich daraus rechnerisch ergebenden Flächen für ausgewählte Switch-Box-Implementierungen unter Variation der beiden Parameter *FIFO\_DEPTH* und *DATA\_WIDTH* sowie einer Portanzahl von 5. (4.1) ist natürlich nur als Abschätzung zu sehen, da durch explizite Synthesen und durch die heuristischen Verfahren der Synthesewerkzeuge sowie deren Optimierungsmechanismen (Fläche bzw. Geschwindigkeit etc.) sich teilweise abweichende Werte ergeben können. Es hat sich allerdings gezeigt, dass mit (4.1) im Allgemeinen eine durchaus akzeptable Prognose erreicht wird, die einen maximalen Fehler von bisher kleiner 3 % zwischen Abschätzung und Synthese geliefert hat (vgl. [123]). Vorteil dieser Abschätzung ist die immense Zeitersparnis bei der Entwurfsraumexploration, die durchaus im Bereich von derzeit zehn Stunden und mehr pro Synthese liegt. Im Vorfeld einer Implementierung, bei der Eruiierung globaler Systemparameter können solche relativ geringen Abweichungen im einstelligen Prozentbereich problemlos toleriert werden.



**Abbildung 4-9: Abschätzung der Switch-Box-Gesamtfläche in Abhängigkeit von Flitbreite und FIFO-Tiefe bei einer Switch-Box-Portanzahl von fünf in a) 130-nm- und b) 90-nm-Standardzellentechnologie**

Abbildung 4-10 zeigt die Flächenabschätzung in Abhängigkeit der Portanzahl und der Flitbreite, wobei die FIFO-Tiefe konstant zu drei gesetzt wird. Bei höheren Portanzahlen wächst der resultie-

rende Flächenbedarf der Switch-Box auf über 20 mm<sup>2</sup> bei der 130-nm-Technologie bzw. 10 mm<sup>2</sup> bei der 90-nm-Technologie an, so dass für ASIC-Realisierungen genauestens abgewogen werden sollte, ob ein derart hoher Grad  $\delta$  benötigt wird. Dies lässt sich unter anderem mit einigen der in Kapitel 5 vorgestellten GigaNetIC-Simulatoren analysieren.

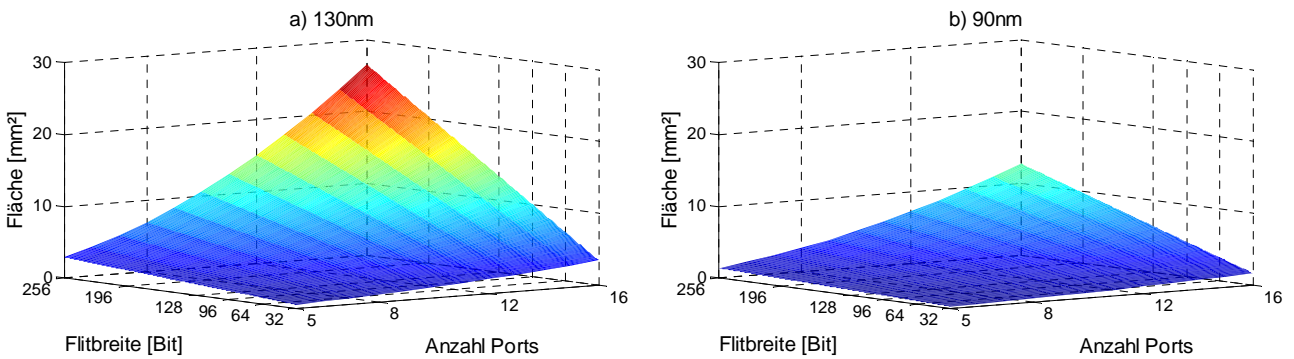


Abbildung 4-10: Abschätzung der Switch-Box-Gesamtfläche in Abhängigkeit von Flitbreite und Anzahl der Switch-Box-Ports bei einer FIFO-Tiefe von drei in a) 130-nm- und b) 90-nm-Standardzellentechnologie

### 4.2.2 On-Chip-Kommunikationsprotokoll

Die Kommunikation auf SoC-Ebene ist paketbasiert. Pakete können maximal 16 KB groß sein und werden in Flits (vgl. Abbildung 4-11) segmentiert, um sie über das On-Chip-Netzwerk versenden zu können. Flits stellen die atomare Informationseinheit des GigaNoCs dar.

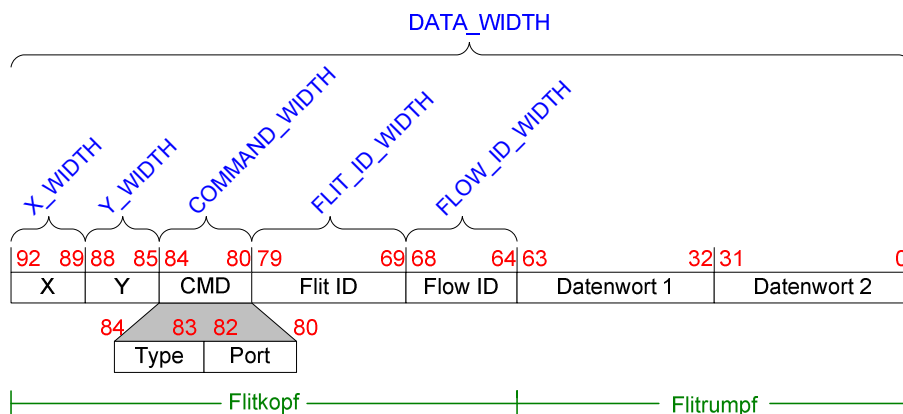


Abbildung 4-11: Flit Aufbau – On-Chip-Kommunikationsrahmen

#### 4.2.2.1 Aufbau der Pakete/Flits

In einem Flit sind alle Daten, die die Switch-Boxen und die *Communication-Controller* zum Routing benötigen, sowie eine parametrisierbare Anzahl von 32-Bit-Datenwörtern enthalten. Flits, die zu dem gleichen Paket gehören, nehmen alle denselben Weg durch das On-Chip-Netzwerk, vergleichbar dem Wormhole-Switching. Die Flitfelder *X*, *Y* und *Port* bilden die Zielkoordinaten. Ausgehend von einer Gitterstruktur geben die Werte *X* und *Y* den Zielknoten relativ zur sendenden Switch-Box an. Die 4-Bit-breiten Koordinaten sind binär kodiert und vorzeichenbehaftet, d. h. das höchstwertige Bit (*MSB*) legt eine positive (*MSB*=0) bzw. negative (*MSB*=1) *X*- oder *Y*-Richtung fest. Mit den restlichen 3 Bits der Koordinatenfelder können die Flits daher bis zu  $2^3 = 8$  Kommunikationsknoten in jede Richtung (*X*, *Y*) geschickt werden. Dies ist mehr als ausreichend für die  $2 \times 4$ -Gitterstruktur des GigaNetIC-Systems. Werden größere Systeme als  $8 \times 8$ -Gitter benötigt, lässt sich dies aufgrund der generischen Struktur leicht ändern. Die dritte Zielkoordinate (*Port*) ist 3 Bit breit

und gibt den Ausgangsport am Zielknoten an. Die Felder *Flit ID* und *Flow ID* dienen zur Identifikation eines Flits. Jeder Kommunikationsknoten im GigaNetIC-Netzwerk besitzt eine eindeutige Kennnummer. Diese wird bei einer Datenübertragung im Feld *Flow ID* eingetragen, um die Datenflits im Empfangsknoten dem entsprechenden Datenpaket eines Sendeknotens zuordnen zu können. Damit die richtige Reihenfolge der Datenflits erhalten bleibt und der Empfänger das Ende einer Datenübertragung erkennt, werden die Flits durchnummeriert. Dies geschieht im 11-Bit-breiten *Flit-ID*-Feld, das die maximale Paketgröße in der realisierten Variante auf  $2^{11} \times 64 \text{ Bit} = 16 \text{ KB}$  begrenzt.

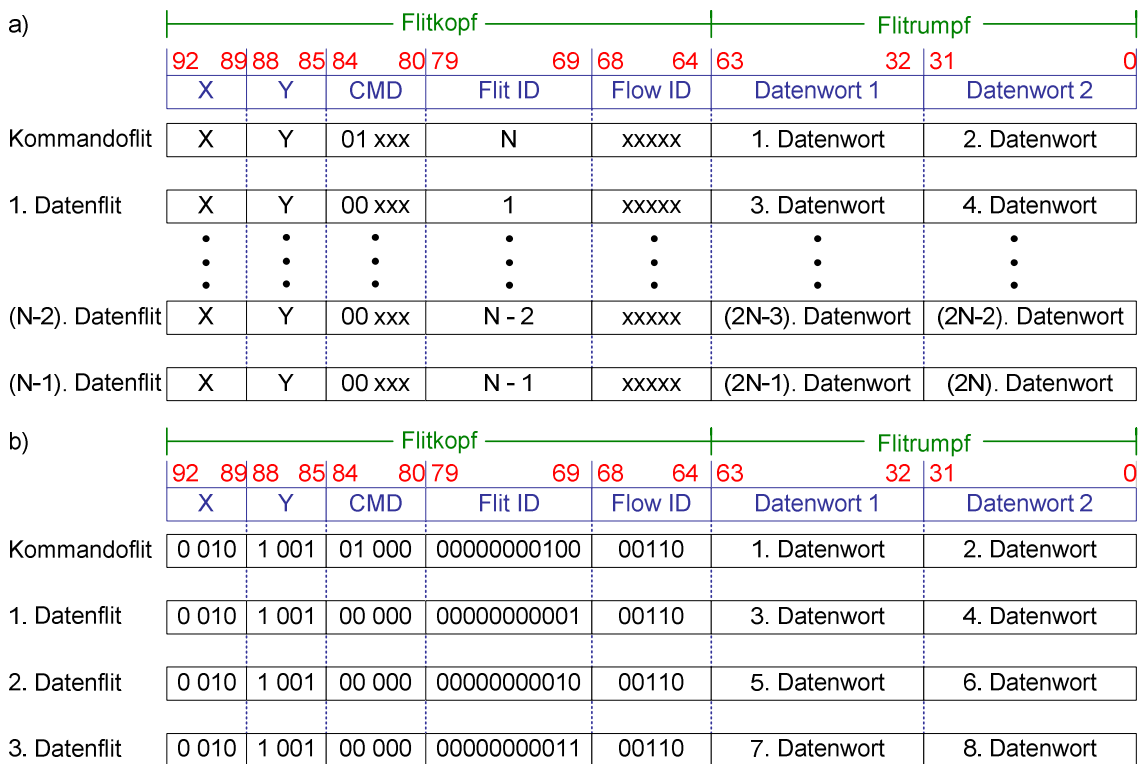


Abbildung 4-12: Struktur eines Pakets:

a) Aufbau eines Pakets mit der allgemeinen Flitanzahl  $N$ ; b) Beispiel eines Pakets mit  $N=4$

Zu Beginn einer Datenübertragung wird jeweils ein so genanntes *Kommandoflit* gesendet, das die Anzahl  $N$  der insgesamt zu übertragenden Flits im Feld *Flit ID* angibt. Bei jedem Transfer der darauf folgenden *Datenflits* wird das *Flit ID*-Feld von  $N = 1$  beginnend inkrementiert, so dass der Empfänger das Übertragungsende ( $Flit ID = N - 1$ ) erkennen kann. Ein Flit kann außer der reinen Datenübertragung noch zusätzliche Aufgaben erfüllen. Das *Type*-Feld gibt die jeweilige Funktion eines Flits an. Neben den bereits erwähnten *Kommandoflits* und *Datenflits* existieren noch *Instruktionsflits* und *Programmflits*.

Mit einem *Instruktionsflit* können Befehle direkt an eine Switch-Box übergeben werden. Z. B. kann durch ein *Instruktionsflit* das Verfahren der Wegewahl (Routing-Strategie) geändert werden. Im GigaNetIC-System ist standardmäßig das X-Y-Routing (*Fast-Routing*) in der Switch-Box aktiviert. Hierbei werden die Flits zunächst in X-Richtung zur jeweiligen Zielspalte des Gitternetzwerks geleitet und anschließend in Y-Richtung dem Zielknoten zugeführt.

Mit Hilfe der *Programmflits* kann auf die Speicherbereiche der Prozessorelemente zugegriffen werden. Nach einem *Reset* des GigaNetIC-Systems wird so jedem Prozessorelement sein Instruktions-

code übermittelt. Das erste 32-Bit-Datenwort eines Programmflits wird hierbei als Adresse genutzt, an die das zweite 32-Bit-Datenwort geschrieben wird.

**Flit-Typen im Detail.** Im Folgenden werden beispielhaft die einzelnen Pakettypen und deren Segmentierung in die entsprechenden Flits aufgezeigt.

Ein Datenpaket wird segmentiert zu einem Kommandoflit und einer variablen Anzahl von Datenflits  $n_{Flits}-1$ , wobei sich die Gesamtzahl der benötigten Flits wie folgt errechnen lässt:

$$n_{Flits} = \left\lceil \frac{Paketdaten[Bit]}{8Bit \cdot 4 \cdot W_{Flit}} \right\rceil \quad (4.7)$$

$W_{Flit}$  gibt hierbei die ganzzahlige Anzahl der 32-Bit-Datenworte an, die in einem Flit enthalten sind. Sollten die Paketdaten nicht ein ganzzahliges Vielfaches von 32 sein, so wird seitens der Hardware der Rest der Flitdatenbits mit Nullen gefüllt. Bei den weiteren Betrachtungen wird, soweit nicht anders erwähnt,  $W_{Flit}=2$  gesetzt. In der Hardwarebeschreibung des GigaNetIC-Systems und in den entsprechenden Simulatoren ist dies ein Parameter, der variiert werden kann, um so die Ressourceneffizienz für den jeweiligen Einsatzzweck zu erhöhen.

In Abbildung 4-12 a) ist ein Datenpaket mit der Länge  $N$  dargestellt. Die Koordinaten sind mit den großen Buchstaben „X“ und „Y“ symbolisiert. Das kleingeschriebene „x“ markiert die Positionen, die für den allgemeinen Aufbau des Flits nicht relevant sind. Im unteren Teil b) der Abbildung 4-12 ist ein Datenpaket mit 256 Bit zu sehen, das folglich in vier Flits segmentiert wurde. Zuerst wird das Kommandoflit übertragen, zu erkennen an der Bitfolge „01“ im Bereich *CMD*, während bei den folgenden Datenflits an dieser Stelle „00“ steht. Die Zielkoordinaten des Pakets sind vom Senderknoten aus gesehen zwei Gittereinheiten in positiver X-Richtung und eine Einheit in negativer Y-Richtung. Der Senderknoten hat die Identifikationsnummer sechs, was im Bereich *Flow ID* eingetragen ist.

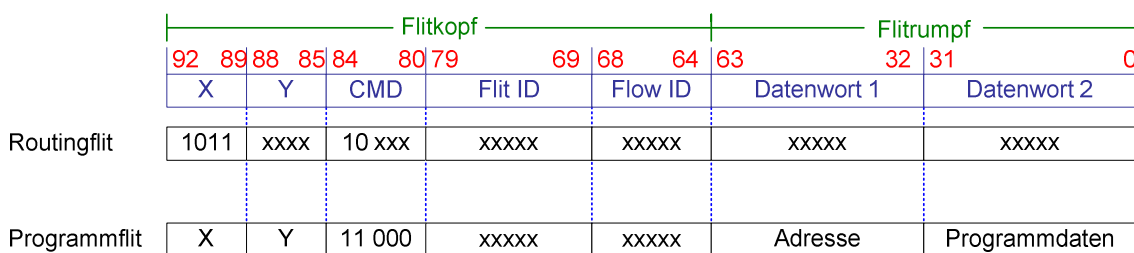


Abbildung 4-13: Aufbau eines Routing- und eines Programmflits

Der Aufbau eines Routingflits und eines Programmflits ist in Abbildung 4-13 dargestellt. Ein *Routingflit* hat die Aufgabe, die Routing-Strategie, die im Netzwerk verwendet wird, zu wechseln. Es ist durch den Eintrag „10“ auf den ersten zwei Bits im *CMD*-Feld gekennzeichnet. Im X-Bereich ist kodiert, wie das Routing-Verfahren verändert werden soll. Die zurzeit geplanten Routing-Verfahren sind in Tabelle 4-1 aufgelistet. Das erweiterte X-Y-Routing ist für Switch-Box-Implementierungen vorgesehen, die über mehr als vier Ports zu Nachbarknoten verfügen. Als effizientes, gut für zweidimensionale Gitterstrukturen geeignetes Verfahren ist das X-Y-Routing (vgl. Abschnitt 2.3.2) als Standardverfahren vorgesehen. Für adaptives Routing lässt sich das *Kosten*-Routing einstellen, das dynamisch auf Auslastungen einzelner Pfade reagieren kann. Für alle Routing-Varianten müssen die Routingflits komplett durch das Netzwerk propagiert, also als spezieller *Broadcast* (vgl. Abschnitt 4.6) versendet werden. Ein Routingflit lässt die Implementierung zusätzlicher Netzwerkkommandos bzw. Steuerbefehle aufgrund der ungenutzten Bereiche (mit „x“ markiert) zu. Um die

Leistungsaufnahme des Chips zu optimieren, ließe sich z. B. ein netzwerkweiter Befehl einführen, der je nach Auslastung einzelne CPUs oder ganze Prozessorfelder in einen Stromsparmodus versetzt, um die Leistungsaufnahme des Chips bei geringer Auslastung zu reduzieren. Auch Statusinformationen über Verkehrsaufkommen oder Auslastung von GigaNetIC-Systementitäten könnten so im Netz propagiert werden.

**Tabelle 4-1: Umstellung implementierter Routing-Verfahren durch Routingflits**

<i>CMD-Feld</i>	<i>DATA_IN[X_WIDTH]</i>	<b>Routing-Verfahren</b>
10xxx	1010	erweitertes X-Y-Routing
10xxx	1011	X-Y-Routing
10xxx	1100	Kosten-Routing
00xxx	xxxx	vorherige Einstellung

Nach dem Einschalten oder einem *Reset* des Systems kann der Programmcode der einzelnen Prozessoren durch das On-Chip-Netzwerk geleitet werden. Dies geschieht durch die so genannten *Programmflits* (siehe Abbildung 4-13) in der Initialisierungsphase des Chips. Die Adresse für die CPU-eigenen Speicherbereiche wird im Flit mitgeschickt, so dass unterschiedliche Programme für die einzelnen Prozessoren in die Instruktionsspeicher geschrieben werden können. Die zwei Datenworte enthalten die Speicheradresse für den Programmspeicher des betreffenden Prozessors gefolgt vom Programmcode. Mit diesem Format ist keine Berücksichtigung der Reihenfolge der Programmflits notwendig. Auch eine Kennzeichnung der CPU im Prozessorfeld, für die das Flit bestimmt ist, ist nicht erforderlich, da die Speicherbereiche durch die Adresse eindeutig festgelegt sind.

#### 4.2.2.2 Funktionsumfang der GigaNoC-Software-Bibliothek

Um das Empfangen und Senden von Paketen sowie diverse Kontrollfunktionen möglichst einfach für den Programmierer zu gestalten, werden spezielle Bibliotheksfunktionen in der GigaNetIC-Software-Bibliothek in der Hochsprache C und in optimiertem Assembler für den eingesetzten Prozessorkern als *Intrinsics* zur Verfügung gestellt. Sie wurden auf Geschwindigkeit und Platzbedarf für die N-Core-Architektur optimiert und sind größtenteils in C geschrieben; wenige Ausnahmen nutzen optimierten Assemblercode, der jedoch ebenfalls in gleicher Funktionalität in C vorliegt und somit leicht auf andere eingebettete Prozessorkerne portierbar ist.

Im Anhang A befindet sich ein C-Codebeispiel, anhand dessen die Nutzung der GigaNetIC-Software-Bibliothek detailliert beschrieben wird. Der Einsatz dieser Funktionen gestaltet sich sehr einfach und komfortabel für den Softwareentwickler, so dass keine nennenswerten bzw. teuren Einarbeitungszeiten, und, damit verbunden, Verzögerungen bei der Softwareentwicklung für das GigaNetIC-System entstehen.

Der prinzipielle zeitliche Ablauf einer Paketinjektion und Terminierung sowie die damit verbundenen Mechanismen werden in Anhang C erläutert.

#### 4.2.3 Performanzanalyse der Kommunikationsinfrastruktur

Der theoretisch erreichbare Durchsatz  $D_{SB}$  einer Switch-Box mit  $P_{SB}$  Ports und einer möglichen Betriebsfrequenz  $f$  lässt sich zu (4.8) bestimmen. Die Anzahl der *Headerbyte* wird in  $m_h$  und die der *Datenbyte* in  $m_f$  angegeben,  $m_h + m_f$  umfasst somit ein komplettes Flit.

$$D_{SB,brutto} [Bit] = P_{SB} \cdot f \cdot 8 \cdot (m_h + m_f) \quad (4.8)$$

$$D_{SB,netto} [Bit] = P_{SB} \cdot f \cdot 8 \cdot m_f$$

Switch-Boxen mit fünf Ports in der 90-nm-Variante ermöglichen theoretische Durchsätze von 332 GBit/s (brutto) bzw. 228 GBit/s (netto), bei  $f=714$  MHz. Die gleiche Variante in 130-nm-Technologie erreicht immerhin noch 273 GBit/s (brutto) bzw. 188 GBit/s (netto) bei  $f=588$  MHz<sup>15</sup>.

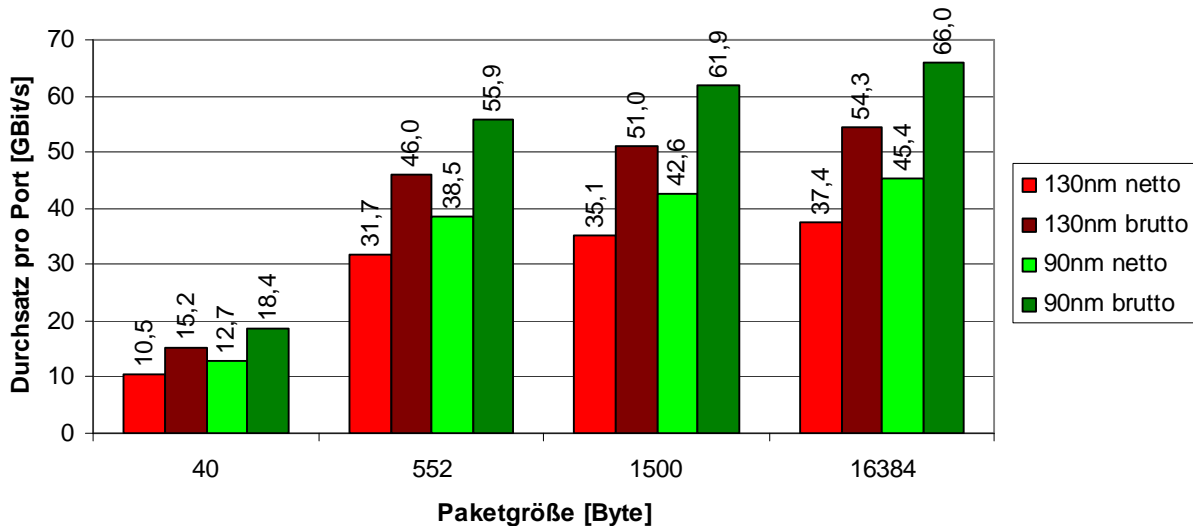


Abbildung 4-14: Leistungsfähigkeit der Switch-Box-basierten On-Chip-Kommunikation

Berücksichtigt man nun die Mechanismen, die zur Paketinjektion und Weiterleitung von Flits seitens der Switch-Box notwendig sind, so lässt sich die Anzahl der Takte bzw. die Latenz, die die Pakete erfahren, die über  $h$  Hops zu Zielknoten geleitet werden, wie in (4.9) angeben. Hier wird zugrunde gelegt, dass eine beliebige Switch-Box Pakete ins Netz injiziert und zugleich von anderen Knoten injizierte Pakete weiterleitet.  $S$  gibt hierbei die Anzahl konkurrierender FIFO-Ketten am Eingang einer Switch-Box an, die über den Kreuzschienenverteiler Pakete auf den gleichen Ausgangsport leiten wollen. Im Falle einer konkurrenzfreien Nutzung der Ports ergibt sich für  $S_i$  ein Wert von 1 für die Verzögerung beim  $i$ -ten Knoten. Sollten  $P_i$  Eingangspports auf den gleichen Ausgangsport zugreifen wollen, ist  $S_i$  entsprechend gleich  $P_i$  zu setzen.

$$Latenz_{Paket} [s] = 11 + \sum_{i=1}^h S_i \cdot 3 \cdot \left( \underbrace{(1)}_{\text{Kommandoflit}} + \underbrace{\left( \left\lfloor \frac{m}{m_f} \right\rfloor - 1 \right)}_{\text{Datenflits, wenn } \frac{m}{m_f} > 1} \right) \cdot f \quad (4.9)$$

Abbildung 4-14 zeigt den Durchsatz pro Port einer Switch-Box in GBit/s bezogen auf die Größe der versendeten Pakete. Die Resultate beziehen sich auf die möglichen Betriebsfrequenzen der Switch-Box für die beiden Standardzellentechnologien in 130 nm (588 MHz) und 90 nm (714 MHz). Die Leistungsfähigkeit wird hier für unterschiedliche, speziell für Netzwerkanwendungsszenarien relevante Paketgrößen untersucht. Zum einen wird der Brutto-Durchsatz angegeben, der die *Flit-Header* mit einbezieht, und zum anderen der reine Netto-Nutzdatenanteil. Es wird deutlich, dass der Durchsatz bei größeren Paketen deutlich über dem der kleineren Paketgrößen liegt. Dies ist begrün-

<sup>15</sup> Diesem Beispiel liegen die Werte  $m_h=3,625$  und  $m_f=8$  zugrunde.



det durch den geringeren Verwaltungsaufwand aufgrund weniger Kommando-Flits, die den Datenpaketen jeweils vorangehen. Daraus resultiert eine Steigerung der effektiven Übertragungsleistung. Der Nettodurchsatz liegt zwischen 12,7 bis 45,4 GBit/s pro Port (90 nm) bzw. 10,5 bis 37,4 GBit/s pro Port (130 nm).

Abbildung 4-15 zeigt drei wesentliche Verkehrsbelastungsfälle einer Switch-Box, die Daten über einen beliebigen Port  $P$  zu einem benachbarten Knoten versendet<sup>16</sup>. Bei der dargestellten Analyse wird explizit die Paketgröße und damit einhergehend die Anzahl der zusammengehörigen Flits erhöht, bis schließlich maximal große Pakete von 16 KByte versendet werden.

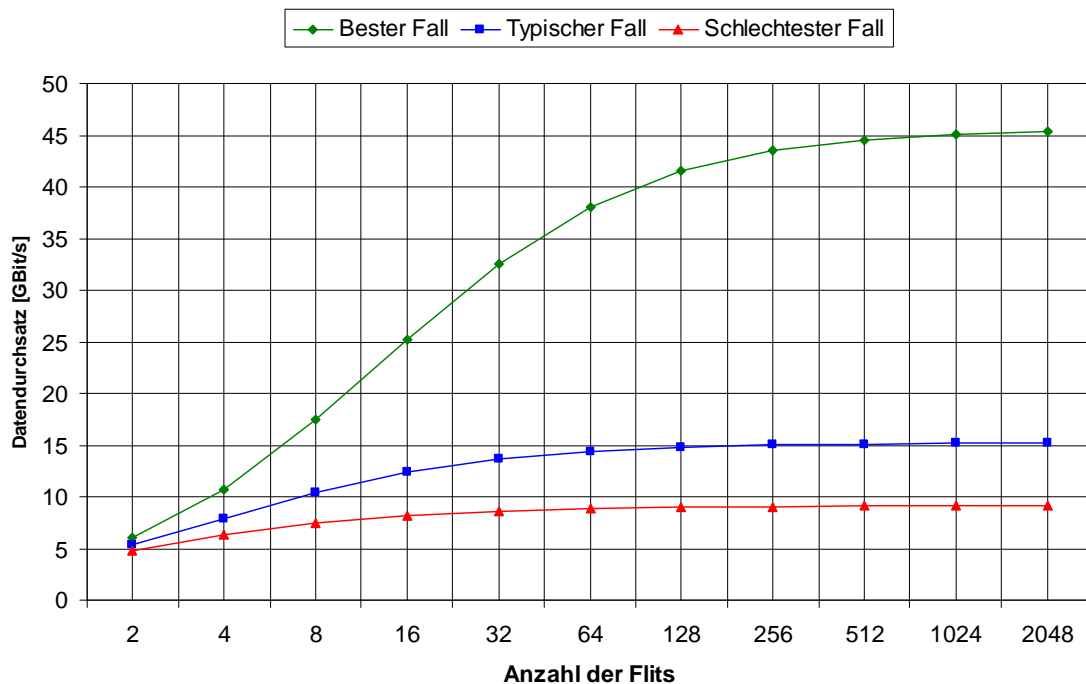


Abbildung 4-15: Datendurchsatz einer Switch-Box für unterschiedliche Verkehrsmodelle

Es zeigt sich in allen drei Fällen, dass mit zunehmender Paketgröße der Datendurchsatz merklich gesteigert werden kann. Der erste Fall „Bester Fall“ beschreibt ein Szenario, in dem, wie bereits in Abbildung 4-14 geschildert, keine konkurrierenden Zugriffe seitens der Eingangsports auftreten. Die Daten können ungehindert das Ziel erreichen. Hier ergeben sich Übertragungsraten von über 45 GBit/s. Der zweite Fall „Typischer Fall“ beschreibt insofern ein eher typisches Szenario für das Verkehrsaufkommen einer Switch-Box, als noch zwei weitere Datenströme auf die Ausgangsports geleitet werden, die sich somit die Bandbreite teilen müssen. Aufgrund dieses Umstands verläuft die Durchsatzkurve deutlich flacher und nähert sich asymptotisch an 15,2 GBit/s an. Der dritte Fall spiegelt den „Schlechtesten Fall“ wider, bei dem alle fünf Ports der Switch-Box um den Ausgangs-port konkurrieren und so das maximale vorfindbare Datenaufkommen darstellen. Hier verläuft die

<sup>16</sup> Auch hier wird die Konfiguration wie bei Abbildung 4-14 verwendet, jedoch wird nur die 90-nm-Realisierung betrachtet, die 130-nm-Variante verhält sich analog. Die Kennwerte lauten somit:  $f = 714$  MHz,  $P_{SB} = 5$ ,  $SB_{FIFO-Tiefe} = 5$ ,  $m_h = 3,625$  und  $m_f = 8$ ,  $S_{Bester\ Fall} = 1$ ,  $S_{Typischer\ Fall} = 3$ ,  $S_{Schlechtester\ Fall} = 5$ . Betrachtet werden ferner nur die Nutzdaten, so dass der Nettodurchsatz dargestellt ist.

Kurve noch flacher und nähert sich schnell dem maximal erreichbaren Nutzdurchsatz von 9,1 GBit/s an.

**Fazit.** Für kleine Pakete kann in allen Verkehrsbelastungsfällen, solange keine Blockade durch Pufferüberlauf eintritt, ein Durchsatz von ca. 5 GBit/s erreicht werden (vgl. Abbildung 4-13). Große Pakete profitieren mehr von gering ausgelasteten Switch-Boxen, so dass eine Verdreifachung des Datendurchsatzes gegenüber einem „vollkonkurrierenden“ Szenario zu beobachten ist. Wesentlich ist, dass kleine Pakete mit sehr geringer Latenz weitergeleitet werden können, allerdings aufgrund der Protokollstruktur einen gewissen Verwaltungsmehraufwand bedeuten. Große Pakete, z. B. Hauptspeicherzugriffe im Sinne von *Bursts*, profitieren von konkurrenzfreien Übertragungen, die durch Ausnutzen der Routingtabellen der Switch-Boxen ggf. ermöglicht werden können. Die GigaNetIC-Architektur verfügt folglich über ein außerordentlich leistungsfähiges On-Chip-Netzwerk, welches für eine Vielzahl von Anwendungsgebieten eingesetzt werden und ggf. zielgerichtet parametrisiert (vgl. Abschnitt 4.7) werden kann.

**Übertragungseffizienz.** Nach Definition 38 wird die Übertragungseffizienz gebildet aus:

$$\varepsilon_{Tr} = \frac{\text{Nettodatenmenge}}{\text{Bruttodatenmenge}}.$$

Trotz der weitreichenden Möglichkeiten des GigaNetIC-On-Chip-

Kommunikationsprotokolls ist der Verwaltungsaufwand, verglichen z. B. mit dem Internet-Protokoll (*IP*) [61], das für Computernetze allgemein entworfen wurde, signifikant geringer. Zwar ist das Internetprotokoll deutlich umfangreicher bzgl. seiner Möglichkeiten, wäre aber sicherlich für ein On-Chip-Netzwerk derzeit funktional überdimensioniert. Es ließe sich zudem keine derart kompakte und zugleich leistungsfähige Hardware realisieren, die mit dem gleichen Flächenaufwand wie die Switch-Box das Internet-Protokoll unterstützen könnte.

Allein die Größe für den Paketkopf von 20 Byte gekoppelt mit den zusätzlichen Paketinformationen der Netzzugangsschicht (z. B. Ethernet von mindestens 18 Byte) bedeutete einen zusätzlichen *Overhead* bei der Flitübertragung von 38 Byte pro Paket. Wollte man das Internet-Protokoll-basierte Datenpaket zudem innerhalb eines Taktes übertragen, so käme man unter Berücksichtigung einer minimalen Framelänge des Internet-Protokoll-Pakets von 46 Byte auf 512 parallele Datenleitungen. Die Übertragungseffizienz liegt bei 40,6 % verglichen mit 68,8 % beim GigaNetIC-Protokoll. Wäre die Parallelität von deutlich mehr Leitungen technisch unproblematisch, so ließe sich für ein Paket maximaler Länge (1500 Byte) ein merklich besseres Verhältnis von Kopfdaten zu Nutzdaten (Übertragungseffizienz = 97,5 %) erzielen. Jedoch sind speziell bei vielen Anwendungen häufig Pakete minimaler Länge zu verzeichnen. Dies wird auch bei den heutigen Lastmodellen (vgl. *iMix*, Abschnitt 7.2.3) berücksichtigt.

Zukünftig sind durchaus Szenarien denkbar, in denen sich die Übergänge von diskreten Netzwerken (wie z. B. des *World Wide Web*) zum Chip fließend gestalten und aufgrund der technischen Möglichkeiten keine Unterschiede zwischen On-Chip- und Off-Chip-Protokollen gemacht werden müssen. In nicht allzu ferner Zukunft wird es möglich sein, On-Chip-Netzwerkknoten mit mehr als 12.000 Intra-Chip-Verbindungen miteinander zu verbinden, diese Anzahl würde eine Weiterleitung ganzer IP-Pakete maximaler Länge innerhalb eines Takts erlauben. Alternativ dazu sind auch besonders schnelle serielle Verbindungen denkbar, die bei sehr hohen Taktfrequenzen eine derartige Übertragung ebenfalls möglich machen könnten. Derzeit hingegen stellen die weniger als 200 Intra-Chip-Verbindungen (Full-Duplexverbindung) zwischen benachbarten Switch-Boxen einen guten Kompromiss zwischen Parallelität und Machbarkeit sowie Taktfrequenz und Durchsatz dar.



#### 4.2.4 Bussysteme auf Cluster-Ebene

Innerhalb der Cluster werden derzeit Busstrukturen zur Anbindung der Verarbeitungseinheiten (*PEs*) eingesetzt (vgl. Abbildung 4-2). Prinzipiell unterstützt die GigaNetIC-Architektur nahezu alle verfügbaren Bussysteme aufgrund der parametrisierbaren Schnittstellen des Communication-Controllers. Bei den Daten- und Adressleitungen müssen lediglich die Bitbreiten angepasst und das erforderliche Zeitverhalten nachgebildet werden. Da der Communication-Controller als passiver Teilnehmer in das Bussystem integriert wird, sind damit die Voraussetzungen zur Integration des Busses in die GigaNetIC-Struktur erfüllt. Derzeit kann beim GigaNetIC-System zwischen dem **Wishbone-Bus** und einer von uns implementierten **AMBA-AHB-Bus-Variante** [113] sowie einer **AMBA-Switch-Matrix** gewählt werden. Die beiden Bus-Realisierungen differieren in ihren Eigenschaften und empfehlen sich daher für unterschiedliche Einsatzgebiete, die AMBA-Switch-Matrix findet u. a. Verwendung beim GigaNetIC-Multiprozessorcache, vgl. Abschnitt 4.4.2.

Im Falle der Verwendung des Wishbone-Busses kann auf eine stetig steigende Anzahl von frei verfügbaren IP-Cores, die dieser weit verbreiteten Schnittstellenspezifikation folgen, zurückgegriffen werden<sup>17</sup>. Hardwareblöcke, die der AMBA-Spezifikation genügen sind größtenteils kostenpflichtige Module. Dies erfordert während der Spezifikation des Systems eine genaue Analyse der benötigten Komponenten. Je nach Verfügbarkeit und Kostenbudget kann die passende Auswahl getroffen werden. Natürlich ist es für beide Bussysteme möglich, eigene IP-Blöcke zu erstellen, sollten dies die zur Verfügung stehenden Ressourcen (Entwicklungskosten, Humankapital, Time-To-Market-Spanne etc.) erlauben. Für beide Busvarianten stellt die GigaNetIC-Architektur die benötigten *Master-* bzw. *Slave-*Schnittstellen bereits zur Verfügung.

Der Wishbone-Bus [124] zeichnet sich durch eine kompakte, relativ einfache Implementierung aus. Er benötigt im Durchschnitt 10 % weniger Fläche als die AMBA-Realisierung [125]. Die AMBA-AHB-Realisierung hingegen zeichnet sich durch eine leistungsfähigere Architektur mit einer weit verbreiteten Standardschnittstelle aus, deren realisierbare Taktfrequenz ca. 5 % höher als die des Wishbone-Systems liegt. Die AMBA-Implementierung nimmt etwas mehr Fläche in Anspruch und erlaubt geringfügig höhere Systemtaktfrequenzen. Die Leistungsaufnahme liegt bei beiden Implementierungen bei einer Konfiguration für vier Verarbeitungseinheiten bei ca. 9,5  $\mu\text{W}/\text{MHz}$  in 130-nm-Technologie und bei 8,6  $\mu\text{W}/\text{MHz}$  in der 90-nm-Technologie<sup>18</sup>.

Im Falle der Realisierung als AMBA-AHB-Interconnection-Matrix sind sogar zeitgleiche, disjunkte Zugriffe mehrerer Master möglich, was bei einer einfachen Busrealisierung wie z. B. bei unserer Wishbone-Bus-Realisierung [117] nicht möglich ist. Für die Integration eines angepassten Multiprozessorcaches (vgl. Abschnitt 4.4.2) wird die AMBA-Implementierung durch einen Snooping-Bus erweitert [113]. Wesentliche Merkmale der beiden Verbindungsstrukturen, die beim GigaNetIC-System derzeit auf Clusterebene zum Einsatz kommen, fasst Tabelle 4-2 zusammen.

---

<sup>17</sup> So gibt es z. B. auf <http://www.opencores.org> (Stand: Juni 2007) eine große Anzahl unterschiedlichster Wishbone-IP-Blöcke zur freien Verfügung.

<sup>18</sup> Alle technologiespezifischen Angaben für Standardzellen-Implementierungen beziehen sich, wenn nicht explizit gekennzeichnet, auf Realisierungen und Synthesergebnisse für normale Betriebsbedingungen der jeweils betrachteten CMOS-Technologie (*Typical Case*).

Tabelle 4-2: Eigenschaften der realisierten Bussysteme, der derzeitigen GigaNetIC-Architektur

Kriterium	Wishbone	AMBA-AHB
<b>Bandbreite:</b> Adress-Pipelining, Busbreite, Verzicht auf Tristate-Busse, Parallel-Übertragungen	Kein Adress-Pipelining (→geringe Taktfrequenzen), geringe Busbreite, Parallel-Übertragung bei der Verwendung von Kreuzschienen-Verteilern oder Datenfluss-Verbindungen, Multicasting und Broadcasting von Schreib-Zugriffen möglich	Parallel-Übertragungen bei der Verwendung einer Matrix möglich
<b>Latenz:</b> Mehrfachzugriffe (Bursts)	Bursts werden unterstützt	Bursts werden unterstützt
<b>Fläche &amp; Verlustleistung</b>	relativ gering	geringfügig mehr als bei Wishbone-Realisierung mit gleicher Anzahl Teilnehmer
<b>Multimasterfähig</b>	Ja	Ja
<b>Kosten:</b> Lizenz-, Versicherungs-, Entwicklungs- & Anpassungskosten	gering	mittel
<b>Wiederverwendbarkeit:</b> Technologie-Unabhängigkeit, (Soft-Core)	Ja	Ja
<b>Snooping</b>	Ja (Schreibzugriffe)	Nein
<b>Besonderheiten</b>	Benutzerdefinierbare Tags	Bursts fester Länge, sehr hohe Busbreiten
<b>Sonstige Einschränkungen</b>	keine	keine atomaren Operationen; kein vorzeitiger Abbruch von Burst fester Länge durch den Master
<b>Fläche (System mit 4 PEs) [mm<sup>2</sup>]</b> 130 / 90 nm	0,05 / 0,02	0,044 / 0,018
<b>Taktfrequenz (System mit 4 PEs) [MHz]</b> 130 / 90 nm	211 / 290	222 / 303
<b>Leistungsaufnahme (System mit 4 PEs) [mW/MHz]</b> 130 / 90 nm	0,0095 / 0,0086	0,0095 / 0,0086

In der zusammen mit Infineon Technologies entwickelten Nova-Architektur [126], die ebenfalls auf dem modularen GigaNetIC-Konzept basiert, kommt der *OCP(Open Core Protocol)*-Bus auf Cluster-Ebene nebst N-Core- bzw. MIPS-4k-Prozessoren zum Einsatz. Aufgrund der Modularität kann auf alle weiteren Funktionen der GigaNetIC-Architektur auf SoC-Ebene zugegriffen werden.

Mit den beiden implementierten Bussystemen bzw. der Interconnection-Matrix kann die GigaNetIC-Architektur auf spezifische Anwendungsgebiete angepasst werden, je nach deren Anforderungen kann eine flächensparende oder aber performantere<sup>19</sup> Variante für die lokale Verbindungsstruktur auf Cluster-Ebene integriert werden.

### 4.3 Verarbeitungseinheiten auf PE-, Cluster- und SoC-Ebene

Die GigaNetIC-Architektur ermöglicht ein flexibles Anschließen von Verarbeitungseinheiten auf allen Hierarchie-Ebenen. Die generischen Schnittstellen seitens der Bussysteme und des Communication-Controllers erlauben die Integration einer breiten Menge von fertigen *IP(Intellectual Proper-*

<sup>19</sup> Der nominelle Durchsatz der implementierten Bussysteme wird an dieser Stelle nicht quantitativ angegeben, da dieser zu sehr vom jeweiligen Anwendungsszenario, von der Art der Verarbeitungseinheiten und der Parametrisierung der Systeme abhängt, als dass eine fundierte Aussage zu treffen wäre. In Kapitel 7 werden für spezielle Anwendungen und Konfigurationen detaillierte Ergebnisse vorgestellt.

ty)-Blöcken, bieten damit ein hohes Potential an Zukunftssicherheit und helfen, die Entwurfs-Produktivitätslücke (*Design Productivity Gap*) zu schließen.

Derzeit sind bereits zwei Prozessortypen erfolgreich in das GigaNetIC-System integriert worden. Hauptbestandteil der an der Universität Paderborn eingesetzten Architekturvariante ist ein am Fachgebiet Schaltungstechnik entwickelter Prozessorkern, der N-Core [108][127][111], der im Folgenden (vgl. Abschnitt 4.3.1) näher beschrieben wird. Außerdem wurden seitens Infineon Technologies MIPS-Prozessorkerne auf Cluster-Ebene implementiert [126].

Neben den CPUs und den lokalen Speichern können weitere IP-Blöcke in das System integriert werden. Häufig unterstützen bereits die Prozessoren auf PE-Ebene über eine Coprozessor-Schnittstelle die Anbindung von Hardwarebeschleunigern, wie z. B. auch der von uns entworfene N-Core. Soll ein Beschleuniger mehreren Prozessoren zur Verfügung stehen, so kann dieser auf Cluster-Ebene über den lokalen Bus angekoppelt werden. Diese eng an das jeweilige Prozessorfeld gekoppelten IP-Blöcke werden über zusätzliche Master/Slave-Schnittstellen des lokalen Bussystems eingegliedert und z. B. über *Memory-Mapped-I/O* angesprochen. Neben Hardwarebeschleunigern können aber auch zusätzliche Module wie z. B. *UARTs (Universal Asynchronous Receiver Transmitter)* für Debuggingzwecke in die lokalen Cluster integriert werden. Diese ermöglichen dann ein Interagieren mit den Prozessoren (z. B. *Touchscreens* wie beim RAPTOR2000-basierten GigaNetIC-Demonstrator, vgl. Abbildung 8-2).

Auf SoC-Ebene können autonomere bzw. global verfügbare Hardwarebeschleuniger und IP-Blöcke über das GigaNoC-On-Chip-Netzwerk angeschlossen werden. Dies können lose gekoppelte Hardwarebeschleuniger sein, die im Datenpfad eingereiht werden, wie z. B. Verschlüsselungs- oder Checksummen-Prüfmodule, vgl. Abschnitt 6.3.1.1. Es können aber auch Einheiten sein, die die Verbindungen nach außen realisieren, wie z. B. Speichercontroller für externen Speicher oder Ethernetcontroller (vgl. Abschnitte 8.1 und 8.2), die die Anbindung an das externe Netzwerk übernehmen. Diese Einheiten können theoretisch an beliebiger Stelle des On-Chip-Netzwerks angeschlossen werden, üblicherweise jedoch an den Rändern des Gitters, um die Hopanzahl zu den Pads des Chips so gering wie möglich zu halten. Zur Integration auf SoC-Ebene wird lediglich eine Instanz des bereits vorgestellten *Communication-Controllers* an die jeweilige Komponente angeschlossen. Er übernimmt die Konvertierung der Daten in das GigaNoC-Flit-Protokoll bzw. die Terminierung des Protokolls und stellt die Daten für den Hardwarebeschleuniger zur Verfügung. Aufgrund dieser Anschlussart sind die Einheiten universell einsetzbar und erlauben eine leichte Adaption des Systems für neue Einsatzgebiete.

Im Folgenden werden der von uns entworfene und erweiterte Prozessorkern, Systemerweiterungen des Prozessorsubsystems sowie Hardwarebeschleuniger und weitere IP-Blöcke des GigaNetIC-Systems vorgestellt.

### 4.3.1 Prozessorkern

Weitere Kernkomponenten des Systems neben der Switch-Box sind die Prozessorkerne. Am Fachgebiet Schaltungstechnik wurde ein 32-Bit-RISC-Prozessorkern in der Hardwarebeschreibungssprache VHDL (*Very High Scale IC (Hardware) Description Language*) entwickelt und in einer aktuellen Standardzellentechnologie implementiert. Der Vorgänger des aktuellen Prozessorkerns, der S-Core, konnte bereits bei einer Taktrate von 160 MHz erfolgreich getestet (Infineon, 130 nm, 0,18 mm<sup>2</sup>) werden [108][127]. Er ist binärkompatibel zum Motorola M-Core [128] gehalten, um die

Softwarewerkzeuge wie z. B. Compiler, Linker, Assembler und Simulator der GNU-Compiler-Entwicklungswerkzeuge, GCC [129], nutzen zu können. Zum Zeitpunkt der Entstehung des S-Cores gab es noch keine derart enge Kooperation mit dem Fachgebiet von Prof. Dr. Kastens, Programmiersprachen und Übersetzer, so dass eine eigene Werkzeugkette fehlte und somit auf öffentlich verfügbare Quellen zurückgegriffen werden musste. Im Verlauf des GigaNetIC-Projekts wurde eine ausgereifte Entwicklungsumgebung realisiert, die speziell auf die Eigenschaften des S-Core-Nachfolgers abgestimmt ist und schnell auf Erweiterungen und Optimierungen der Hardware angepasst werden kann (vgl. Abschnitt 5.6). Neben dieser können aber auch weiterhin die aktuellen GCC M-Core-Werkzeuge zur Erzeugung des Programmcodes verwendet werden, allerdings bleiben dann die Befehlssatzerweiterungen der neuen Architektur ungenutzt.

Der N-Core ist die Weiterentwicklung des S-Cores und der derzeitige Standardprozessorkern der GigaNetIC-Architektur [117][111][130][131][113][110]. Er wurde ebenfalls als Soft-Core in VHDL realisiert und kann frei nach den Bedürfnissen des jeweiligen Einsatzgebietes, unterstützt durch die Paderborner Werkzeugkette, angepasst werden. Ein weiterer Vorteil einer Eigenentwicklung ist die freie Verwendung und Vervielfältigung eines solchen Kerns anstelle von etwaigen Lizenzgebühren, die bei kommerziellen Produkten auftreten können. Je nach Geschäftsmodell kann hier u. a. die Anzahl der verwendeten Kerne als Berechnungsgrundlage dienen, was für ein massiv paralleles System ggf. sehr kostspielig werden könnte.

Der N-Core ist eine Zwei-Address-Maschine mit einer typischen *Load-Store*-Architektur. Er hat eine dreistufige Pipeline (*Fetch*, *Decode* und *Execute*) und besitzt zwei Registerbänke mit je 16 32-Bit-Registern und 13 zusätzliche Spezialregister. Zur Unterstützung von normalen und bevorzugten Interruptquellen wurde ein spezieller Interruptcontroller realisiert. In Verbindung mit dem zweiten Registersatz können auf diese Weise sehr schnelle Interruptbehandlungen unterstützt werden. Außerdem verfügt der N-Core über ein so genanntes (*Global Control Register / GCR*) und ein globales Status-Register (*Global Status Register / GSR*), mit deren Hilfe direkt Daten ausgegeben bzw. im letzteren Fall eingelesen werden können. Hierdurch können sehr schnell Informationen speicherunabhängig kommuniziert werden. Der schematische Aufbau des N-Cores ist in Abbildung 4-16 dargestellt.

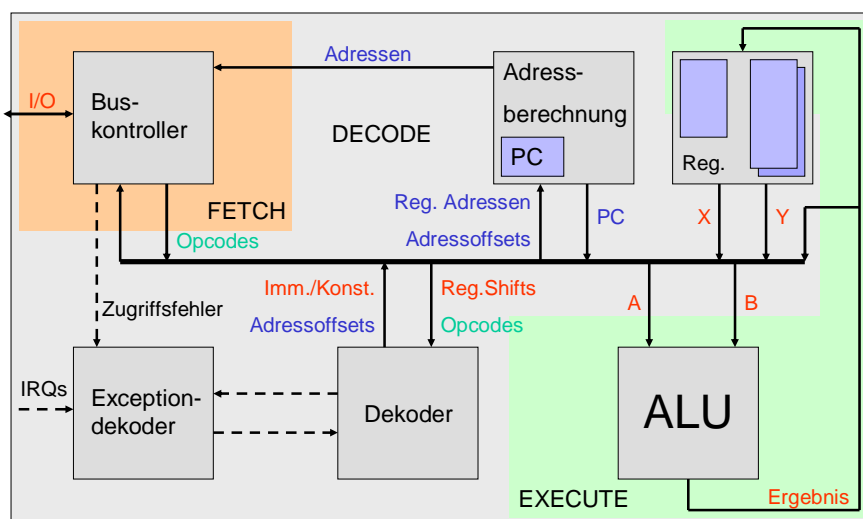


Abbildung 4-16: Schematischer Aufbau des N-Core-Prozessorkerns

Die Instruktionen haben eine feste Breite von 16 Bit, wodurch eine hohe Codedichte erreicht wird. Dies ist bei eingebetteten Systemen mit limitierten Speicherressourcen von besonderer Bedeutung. Der Befehlssatz lässt sich durch zusätzliche Instruktionen erweitern, da noch 11% an freiem Opcode zur Verfügung stehen (vgl. Kapitel 6 und 7). Weiterhin stellt der N-Core eine Coprozessor-schnittstelle für Hardwarebeschleuniger zur Verfügung. Der N-Core unterstützt eine byteweise Adressierung des Speicherinhalts und arbeitet im *Big-Endian-Format*, was besonders für Netzwerkapplikationen von Bedeutung und vorteilhaft ist. Die Mehrzahl aller verfügbaren Instruktionen benötigt zur Ausführung einen Takt, Speicherbefehle zwei bzw. mehr bei größerer Speicherlatenz. Ohne spezialisierte Multiplikations- und Divisionsbeschleuniger können Multiplikationen bis zu 18 und Divisionen bis zu 37 Takte beanspruchen.

**Tabelle 4-3: Kenndaten der ursprünglichen S-Core-Realisierung [108]**

Technologie	Fläche [mm <sup>2</sup> ] bzw. [Slices]	Taktfrequenz [MHz]	Verlustleistung [mW/MHz]	Versorgungsspannung [V]
<b>Standardzellen</b>				
Infineon 130 nm	0,25	160	0,165	1,2
AMS 600 nm	30	61	20	5,0
<b>FPGA</b>				
Xilinx Virtex 1000-4 (220 nm)	3727 (von 12288)	12	25	2,5

Die Komplexität des S-Core-Prozessorkerns entspricht ca. 23.000 Gatteräquivalenten<sup>20</sup> und umfasst ca. 7300 kommentierte VHDL-Codezeilen in 24 Dateien. Der originale Motorola M-Core-Prozessor benötigt in der 0,36 µm- bzw. in der 0,25 µm-Technologie 2,2 mm<sup>2</sup> bzw. 1,6 mm<sup>2</sup> Chipfläche. Die wesentlichen Kenndaten der Ursprungsversion des S-Core sind in Tabelle 4-3 dargestellt [108].

Die wesentlichen Daten der aktuellen N-Core-Implementierung [118] ohne Spezialinstruktionen, unter Berücksichtigung einer aktuellen 90-nm-Standardzellentechnologie zeigt Tabelle 4-4 auf. Detaillierte Informationen zu den einzelnen Optimierungen und Instruktionssatzerweiterungen des N-Cores werden in Kapitel 6 gegeben. Hier wird der von mir entworfene, hierarchisch gerichtete Optimierungsansatz zur GigaNetIC-Architektur für gegebene Anwendungsszenarien vorgestellt.

**Tabelle 4-4: Kenndaten des aktuellen N-Core-Prozessorkerns der GigaNetIC-Architektur [118]**

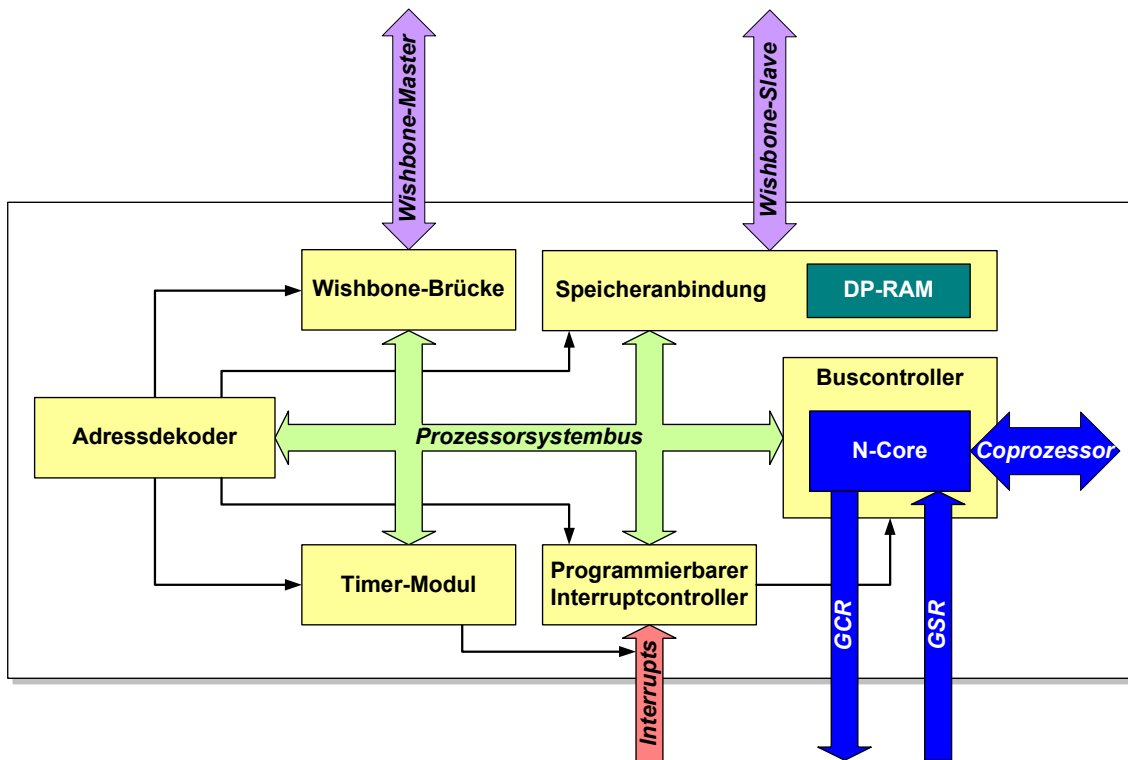
Technologie	Fläche [mm <sup>2</sup> ]/[slices]	Taktfrequenz [MHz]	Leistungsaufnahme [mW/MHz]	Versorgungsspannung [V]
<b>Standardzellen</b>				
UMC 130 nm	0,158	205	0,049	1,2
90 nm	0,127	285	0,032	1,2
<b>FPGA</b>				
Xilinx Virtex II 8000-4 (150 nm)	3206 (von 46592)	17,5	k. A.	1,5 Kern / 3,3 IO

### 4.3.2 Systemerweiterungen und Peripherie – das Prozessorsubsystem

Der S-Core-/N-Core-Prozessorkern allein wäre nicht effizient einsetzbar im GigaNetIC-Chip-Multiprozessorsystem, deshalb wurden Systemerweiterungen und Schnittstellen für Peripherieblöcke integriert und so ein leistungsfähiges Prozessorsubsystem geschaffen, vgl. Abbildung 4-17.

<sup>20</sup> Hierunter wird die Anzahl des Flächenäquivalents in Standard-NAND2-Gattern der entsprechenden CMOS-Standardzellen-Technologie verstanden, also die Fläche der Realisierung, ausgedrückt in der Anzahl der NAND2-Gatter, die auf dieser untergebracht werden können.

Bei der Realisierung des Prozessorsubsystems wurde der S-Core zunächst zum Net-S-Core erweitert. Der Net-S-Core verfügt über Erweiterungen wie einen programmierbaren *Timerblock*, einen programmierbaren *Interruptcontroller*, integrierte *Performanzbewerter* und einen erweiterten *Adressdekoder*, der die komfortable Ansteuerung von Hardwarebeschleunigern über Memory-Mapped-IO-Zugriffe gestattet. Die Integration all dieser Komponenten zusammen mit den *Wishbone-* bzw. *AMBA-Busschnittstellen* und die Anbindung von lokalem Dual-Port-Speicher zusammen mit den anwendungsspezifischen Optimierungen unter Verwendung der geschlossenen Software-Werkzeugkette (vgl. Kapitel 5 und 6) formen letztendlich den N-Core [111][117].



**Abbildung 4-17: Das Prozessorsubsystem des N-Core, am Beispiel der Wishbone-Bus-Implementierung**

Mit Hilfe der *Wishbone-/AMBA-Bridge* lässt sich das *N-Core-Prozessorsubsystem* an den jeweiligen lokalen Bus des GigaNetIC-Systems anknüpfen. Der *DP-RAM-Block* ermöglicht gleichzeitiges Lesen bzw. Schreiben vom Prozessorkern bzw. vom Bus aus. Die Kontrolle des Prozessorsystembusses übernimmt der *N-Core-Buscontroller*, der ebenfalls auf etwaige Adressverletzungen seitens der Software reagiert und entsprechende Ausnahmebehandlungsroutinen (*Exceptions*) auslöst.

Der N-Core kann 32-Bit-breit adressieren, dies entspricht einem Speicherbereich von theoretisch 4 GByte. Dies erscheint für eingebettete Prozessoren derzeit mehr als ausreichend, so dass ein Teil des Adressraums für weitere Zwecke genutzt werden kann. Der *Adressdekoder* fungiert als zentrale Steuereinheit der Buszugriffe seitens des Prozessors. Er generiert Selektionssignale für angesprochene Hardwareblöcke und übernimmt die Adressübersetzung. Die Anzahl der zu verwaltenden Komponenten und die Größen der einzelnen Speicherbereiche sind generisch anpassbar, so dass leicht zusätzliche Einheiten an den Prozessorsystembus angeschlossen werden können und flexibel auf anwendungsspezifische Anforderungen reagiert werden kann. Die Speicherbereiche auf Clusterebene sowie die Adressierung der beiden zusätzlichen Hardwareeinheiten Timer und programmierbarer Interruptcontroller des Prozessorsubsystems sind in Abbildung 4-18 dargestellt.

Der programmierbare Interruptcontroller (*PIC*) ermöglicht eine Priorisierung und Auswahl der Interruptsignale. Der N-Core differenziert zwischen normalen (*normal interrupt, nint*) und hochprioren Interrupts (*fast interrupt, fint*). Die hochprioren Interrupts verwenden den zweiten Registersatz des Prozessorkerns und ersparen so das Sichern der Registerinhalte auf den Stapel (*Stack*). Das *Normal Interrupt Enable Register (NIER)* bestimmt, welche 32-Bit-Signale einen normalen Interrupt auslösen. Analog entscheidet der Inhalt der *Fast Interrupt Enable Registers (FIER)*, welche hochprioren Interrupts zum Prozessor geleitet werden. Hierzu werden die externen Signale mit den Registerinhalten bitweise UND-verknüpft. Eine Weiterleitung des Interrupts geschieht nur, wenn diese Verknüpfung eine logische Eins ergibt. Sollten zwei Interrupts gleichzeitig anliegen dominiert ein *fint* gegenüber dem *nint*, außerdem entscheidet innerhalb der Interruptklassen die Wertigkeit des Bits über den Vorrang der Abarbeitung.

PIC-Modul	0x100001FF	Ungenutzt	0xFFFFFFFF	WISHBONE Brücke	0xFFFFFFFF	SWITCH_BOX					
	⋮				⋮	0x80000000	PACKET_MEM				
	0x10000108				FIER	0x7FFFFFFF	0x40000000	SRAM_WB			
	0x10000104				NIER	0x3FFFFFFF	0x30000000	UART_TOP			
Adressierung PIC-Modul			0x11000000	Ungenutzt	0x2FFFFFFF	0x20000000	NCORE_WB(15)				
TIMER-Modul	0x100002FF	Ungenutzt	0x10FFFFFF		Timer-Modul	0x1FFFFFFF		0x1F000000	NCORE_WB(n)		
	⋮		⋮	0x1nFFFFFF		⋮	NCORE_WB(2)				
	0x10000210		divider_reg	0x10000300		PIC-Modul		0x1n000000		NCORE_WB(1)	
	0x1000020C		count_reg	0x10000200				Reserviert			0x12FFFFFF
	0x10000208		modulo_reg	0x100001FF		Speicher (max. 256 MB)					0x12000000
	0x10000204		ctrl_reg	0x10000100				⋮			
Adressierung Timer-Modul			0x100000FF	⋮	0x11FFFFFF						
Lokale Module			0x10000000	0x10000000	0x11000000						
Prozessorsubsystem			0x0FFFFFFF	⋮	0x10FFFFFF						
Prozessorfeld			0x00000000	0x00000000	0x00000000						

Abbildung 4-18: Adressierungen und Speicherbereiche im GigaNetIC-System auf Clusterebene

Das Timer-Modul stellt die Funktionalität eines programmierbaren Zählers und Zeitgebers. So können z. B. Zeitstempel für Pakete im Anwendungsbereich der Netzwerkdatenverarbeitung etc. erzeugt werden. Außerdem kann das Modul als konfigurierbarer Taktzähler zur Performanzmessung genutzt werden.

Im Falle der Wishbone-Bus-Realisierung wird unter dem Adressraum 0x11000000 bis 0xFFFFFFFF die Wishbone-Brücke angesprochen. Sie übernimmt die Protokollumsetzung zwischen N-Core und Wishbone-Standard. Da die Wishbone-Spezifikation prinzipiell nur 32-Bit-breite Wortzugriffe gestattet, werden zusätzliche *Select*-Signale des Wishbone-Busses zur byteweisen Adressierung verwendet, damit der volle Funktionsumfang des N-Core-Prozessorsubsystems genutzt werden kann. Im Idealfall und ohne Arbitrierungsverluste benötigen Schreib- und Lesezugriffe über den Wishbone-Bus drei Takte. Detaillierte Analysen zur Performanz der Wishbone-Implementierung werden in [109][131] und Kapitel 7 gegeben.

Zur einfachen Möglichkeit der Interaktion und für Debuggingzwecke wurde außerdem eine serielle Schnittstelle (*UART*) als IP-Block integriert (vgl. Abbildung 8-3). Der Wishbone-basierte Cluster erlaubt eine maximale Anzahl von 15 N-Cores sowie weitere Verarbeitungseinheiten und Speicher, siehe Abbildung 4-18.

Für das Prozessorsubsystem des N-Core wurde zusätzlich ein multiprozessorfähiger Cache implementiert [113], der die Verarbeitung zahlreicher Anwendungen beschleunigt. Die Architektur des Caches wird in Abschnitt 4.4.2 vorgestellt, Resultate der Performanzsteigerung werden in Kapitel 6 und 7 dargelegt.

### 4.3.3 Hardwarebeschleuniger

Hardwarebeschleuniger sind neben den Prozessorkernen des GigaNetIC-Systems die wichtigsten Verarbeitungseinheiten. Sie übernehmen anwendungsspezifische Aufgaben, die sie effizienter bearbeiten können, als es den weniger spezialisierten Universalprozessoren des Systems möglich ist. Durch die Verlagerung besonders rechenintensiver Aufgaben auf diese Spezialeinheiten und die Verwendung der Prozessorkerne für deutlich mehr Flexibilität erfordernde Kontrollaufgaben wird eine besonders effiziente Symbiose von hoch-performanten und hoch-flexiblen Systementitäten geschaffen. Die GigaNetIC-Architektur erfordert zwar nicht zwingend den Einsatz von Hardwarebeschleunigern, da die N-Core-Prozessoren für eine Vielzahl von Problemen genügend Rechenleistung zur Verfügung stellen, und aufgrund der parallelen Struktur der Architektur ggf. eine zusätzliche Beschleunigung erreichbar ist (vgl. Kapitel 7). Sollte das Einsatzgebiet jedoch im Vorfeld der Implementierung genauer spezifiziert sein, werden dem Systemarchitekten eine Vielzahl von Integrationsmöglichkeiten zur Auswahl gegeben. Im weiteren Sinne sind auch integrierbare FPGA-Zellen als „flexible“ Hardwarebeschleuniger zu sehen, die während der Laufzeit, compilergestützt, konfiguriert werden können.

Die ITRS [2] gibt an, dass Hardwarebeschleuniger im Vergleich zu Universalprozessoren derzeit bis zu vier Größenordnungen effizienter (z. B. im Sinne von GOPS/mW) arbeiten. Zudem vergrößert sich diese Lücke zunehmend, so dass Universalprozessoren noch stärker einem Wettstreit mit anwendungsspezifischer oder auch rekonfigurierbarer Hardware ausgesetzt sein werden. In Kapitel 6.3 wird dieser Trend anhand von eigenen Implementierungen und Analysen von dedizierten Hardwarebeschleunigern für das GigaNetIC-System untermauert. Es gilt, je nach Einsatzgebiet und dessen Anforderungen im Hinblick auf die Ressourceneffizienz nach Definition 14, einen geeigneten Kompromiss zwischen Flexibilität und Leistungseffizienz zu finden (vgl. Kapitel 8.3).

Auch AMD setzt u. a. in der „Torrenza“-Initiative bei den zukünftigen Prozessorgenerationen verstärkt auf eine Kopplung von wenigen Universalprozessorkernen und anwendungsspezifischen Hardwarebeschleunigern, um sich so gegenüber den von Intel propagierten Architekturen, die schwerpunktmäßig auf homogene, parallele Prozessorfelder setzen, zu behaupten [93].

Das GigaNetIC-Architekturkonzept hingegen vereint beide Ansätze. Zur Anbindung der Hardwarebeschleuniger stellt das GigaNoC unterschiedliche Möglichkeiten zur Verfügung. Der jeweilige Anknüpfungspunkt im On-Chip-Netzwerk hängt von einer Vielzahl von Parametern ab, die es im Vorfeld einer ASIC- bzw. auch FPGA-Implementierung zu eruieren gilt. Hierzu zählen:

- die Beschleunigung des Hardwaremoduls
- der Funktionsumfang des Beschleunigers (Grad der Autonomie)
- die gewünschte Verfügbarkeit / Erreichbarkeit für andere SoC-Komponenten
- der Speicherbandbreitebedarf und die benötigte Speichermenge
- etwaige Flächenrestriktionen (z. B. gemeinsamer vs. lokaler Speicher)



Abbildung 4-19 zeigt die verschiedenen Anbindungsmöglichkeiten für Hardwarebeschleuniger und IP-Blöcke im GigaNetIC-System. Dabei können die einzelnen Wrapper<sup>21</sup> entweder am lokalen Bus auf Clusterebene oder aber an einen beliebigen freien Port einer Switch-Box unter Zuhilfenahme eines angepassten Communication-Controllers angeschlossen werden.

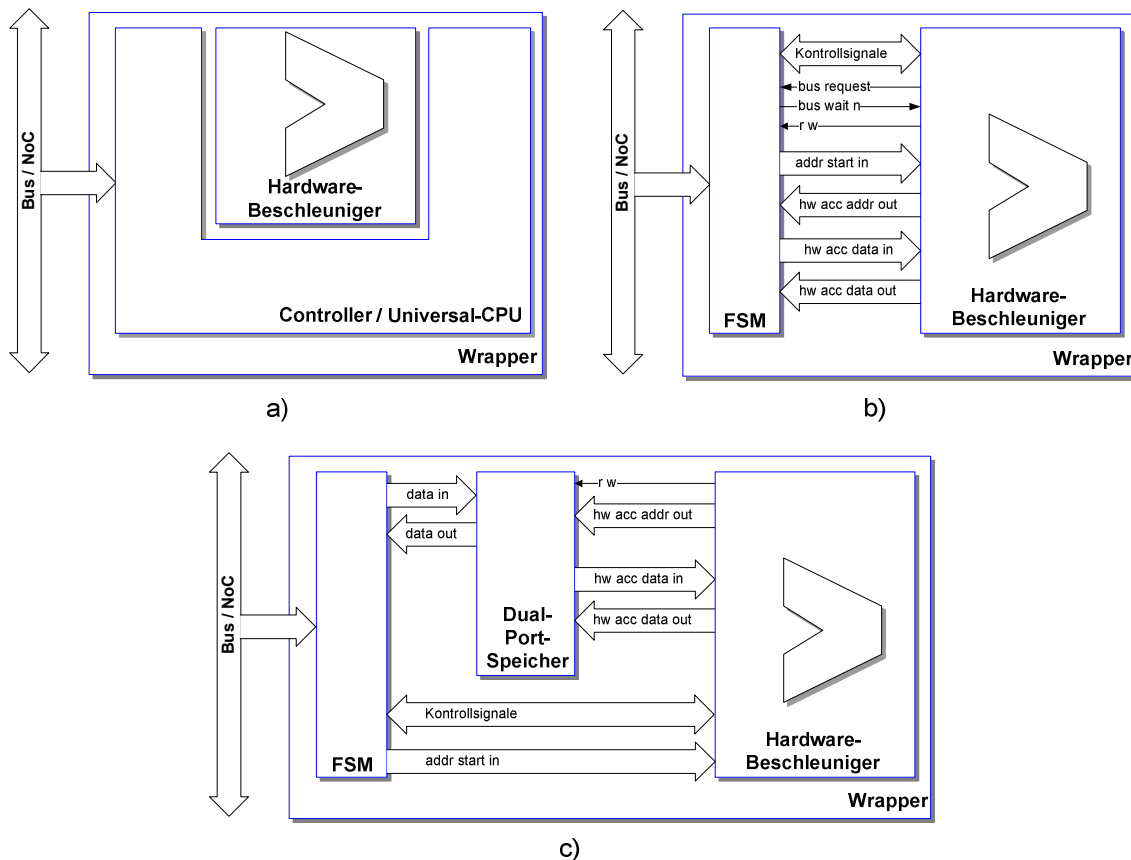


Abbildung 4-19: Unterschiedliche Anbindungsmöglichkeiten von Hardwarebeschleunigern

Abbildung 4-19 a) stellt eine *eng-gekoppelte* Integration eines Hardwarebeschleunigers bzw. IP-Blocks dar, der direkt mit einem Prozessorkern verbunden ist und ggf. direkt über dessen Coprozessorschnittstelle angesteuert wird. Bei dieser Variante ist nicht zwangsläufig dedizierter Speicher notwendig. Abbildung 4-19 b) zeigt einen *semi-eng-gekoppelten* Hardwarebeschleuniger bzw. IP-Block, der auf Cluster-Ebene als eigenständiger Busteilnehmer angeschlossen ist. Er hat Zugriff auf den gemeinsamen Speicher des Clusters und kann mit Hilfe von Kontrollregistern und über einen dedizierten Adressraum von anderen Teilnehmern angesprochen werden. Die Abarbeitung des Problems erfolgt dann zumeist autonom und entkoppelt vom auftraggebenden Prozessorkern. Abbildung 4-19 c) veranschaulicht eine *lose Kopplung* eines Hardwarebeschleunigers. Bei dieser Art der Kopplung verfügt der vom GigaNoC zur Verfügung gestellte Wrapper sowohl über einen Zustandsautomaten (*Finite State Machine / FSM*), der die notwendigen Kontrollfunktionen übernimmt, als auch über eine parametrisierbare Menge lokalen Speichers, der in der Regel als Dual-Port-RAM ausgelegt ist. Diese Variante der lose gekoppelten Hardwarebeschleuniger wird vorwiegend an dedizierten Ports von ausgewählten Switch-Boxen eingesetzt. Sie eignet sich vor allem für IP-Blöcke, die über einen hohen Grad an Autonomie bei der Verarbeitung verfügen und ein ange-

<sup>21</sup> Unter dem Begriff *Wrapper* wird eine Umhüllung bzw. Einhüllung einer gegebenen Systementität verstanden, die diese in ein bestehendes System möglichst effizient und transparent für die weiteren Systementitäten integriert.

messenes Verhältnis zwischen Berechnungszeit und Kommunikation der Daten über das Netz aufzuzeigen. Besonders Verarbeitungseinheiten, die global im System zur Verfügung stehen sollen, aber zahlenmäßig nur gering eingesetzt werden (evtl. aufgrund einer nicht unerheblichen Fläche, oder aber weil sie eine überaus hohe Verarbeitungsgeschwindigkeit aufweisen), bieten sich für diese Kopplung besonders an. Detaillierte Analysen der Kopplungsarten für gegebene Hardwarebeschleuniger werden in [109][131] und in Kapitel 7 vorgestellt.

Natürlich lassen sich die vorgestellten Kopplungen und die entwickelten Wrapper nicht nur für Hardwarebeschleuniger einsetzen, sondern erlauben auch die Integration beliebiger IP-Blöcke, die zwar keine Beschleunigerfunktionalität zur Verfügung stellen, aber andere benötigte Dienste, wie z. B. Ethernetschnittstellen, integrieren (vgl. Abschnitt 4.3.4). Der sich ergebende Flächenbedarf der einzelnen GigaNoC-Wrapper wird in Tabelle 4-5 aufgezeigt<sup>22</sup>.

**Tabelle 4-5: Flächenbedarf der GigaNoC-Wrapper zur Ankopplung beliebiger 32-Bit-IP-Blöcke @250MHz**

Technologie	Fläche [mm <sup>2</sup> ]		
	PE-Ebene	Cluster-Ebene	SoC-Ebene
130 nm	0,0097	0,0039	0,6389
90 nm	0,0068	0,0036	0,5449

Die derzeit für das GigaNetIC-System realisierten Hardwarebeschleuniger finden im Anwendungsbereich der Netzwerkverarbeitung Einsatz und werden in Kapitel 6 und 7 detaillierter vorgestellt.

#### 4.3.4 Sonstige IP-Blöcke

Neben Hardwarebeschleunigern können auch beliebige I/O-Kontrolleinheiten, wie z. B. Speichercontroller oder Ethernetschnittstellen an die oben beschriebenen Schnittstellen des GigaNoCs angeschlossen werden. Auch hier entscheiden letztlich die Anforderungen der Anwendung in Form von Durchsatz und Verfügbarkeit der entsprechenden I/O-Funktionalität für das gesamte SoC, welche Kopplung verwendet werden sollte. Aufgrund des hohen Transportvermögens der Switch-Boxen und der einheitlichen Schnittstelle bietet sich für hochperformante Einheiten in vielen Fällen der Anschluss über einen freien Port einer Switch-Box an (vgl. Abbildung 4-19 c)). Dies ist z. B. bei den im Rahmen des GigaNetIC-Projektes entwickelten Ethernetschnittstellen [110][109][131] der Fall. Für Debugging- und Interaktionszwecke wurde eine serielle Schnittstelle realisiert, die derzeit als Wishbonebusteilnehmer auf Clusterebene integriert ist [109]. Der lokale Anschluss ermöglicht eine flächeneffiziente Integration, die den relativ geringen Bandbreiteansprüchen dieser I/O-Schnittstellen mehr als genügt.

Speicherschnittstellen sind hingegen auf Cluster- und auf SoC-Ebene für das GigaNetIC-System verfügbar. Die Kopplung hängt hier sehr stark vom Zweck und von der Lokalität der Daten ab. So werden die N-Core-Programmabbilder z. B. in einem global erreichbaren externen oder auch internen EEPROM abgelegt und bei Inbetriebnahme des Chips mit Hilfe des On-Chip-Netzwerks (vgl. Abschnitt 4.2.2) zu den einzelnen lokalen Speichern des N-Cores transportiert.

<sup>22</sup> Die zugrunde liegenden Implementierungen erlauben Taktfrequenzen die sich deutlich über der hier zugrunde liegenden 250-MHz-Synthese-Einstellung bewegen. Der Wrapper ist aufgrund seiner geringen Logiktiefe nicht als Flaschenhals zu sehen. Die Betriebsfrequenz bestimmt letztendlich der Hardwarebeschleuniger bzw. das On-Chip-Netzwerk. Die SoC-Ebenen-Anbindung verfügt in der angegebenen Variante bereits über 16 KByte Dual-Port-Speicher. Die anderen beiden Wrapper greifen standardmäßig auf gemeinsamen Speicher zu, der nicht in die Flächenangabe einfließt.

Details zur Implementierung der Ethernetcontroller und zu den Möglichkeiten der Interaktion mit dem GigaNetIC-Prototypen, die durch die seriellen Schnittstellen und die angeschlossenen berührungssensitiven Anzeigen gegeben sind, werden in Kapitel 8 dargestellt.

## 4.4 Speicher

Neben einer leistungsfähigen Kommunikationsinfrastruktur sowie flexiblen und leistungsfähigen Verarbeitungseinheiten gehört der Speicher zu den wesentlichen Komponenten eines Chip-Multiprozessors. Die GigaNetIC-Architektur unterstützt das in Abschnitt 2.5 vorgestellte Konzept einer mehrschichtigen Speicherhierarchie. Die Klassifizierung der einzelnen Speicherstufen für das GigaNetIC-System in der momentanen Ausbaustufe werden in Tabelle 4-6 gezeigt. Zu beachten ist, dass bei Zugriffen auf entfernten Speicher die Latenz aufgrund der Flitkonfiguration für 64 Bit Daten anstatt 32 Bit angegeben wird.

**Tabelle 4-6: Speicherhierarchien der GigaNetIC-Architektur**

Hierarchie	Speicher	Zugriffszeit [Takte]
<b>CPU</b>	Register	1
<b>L1</b>	Lokaler Prozessorspeicher bzw. Cache	2
<b>L2</b>	Gemeinsamer Speicher auf Clusterebene	Minimum: 4 bis 5 bei freier Ressource (je nach Arbitr-Zustand), Maximum abhängig von Anzahl der Busteilnehmer (min + (n-1)) und / oder der maximal zulässigen Burstlänge
<b>L3</b>	Entfernter Speicher anderer Cluster am lokalen Bus	2 x Paketlatenz, nach Formel (4.9) + L2-Latenz
<b>L4</b>	Externer Speicher DRAM / SRAM etc.	2 x Paketlatenz, nach Formel (4.9) + Latenz des Speichercontrollers + Latenz des externen Speichers

Für den Anschluss der Speicher stehen u. a. die in Abschnitt 4.3.3 vorgestellten Kopplungsmöglichkeiten zur Verfügung. Für den Anschluss externer Speicherbausteine kommen modifizierte Instanzen des Communication-Controllers (vgl. Abschnitt 4.2.1.1), die über die benötigte Kontrolllogik zur Ansteuerung des jeweiligen Speichertyps verfügen, zum Einsatz. Für lokalen, SRAM-basierten Speicher stehen Wishbone- bzw. AMBA-Schnittstellen zur Verfügung. Für andere Speichertechnologien kann entweder auf standardisierte IP-Blöcke zurückgegriffen oder es können wahlweise eigene Lösungen integriert werden.

### 4.4.1 Lokaler Speicher auf Cluster-Ebene

Zum lokalen Speicher des GigaNetIC-Systems gehört der L1-Speicher des einzelnen Prozessorkerns (Prozessorspeicher). Dieser ist zunächst nur vom jeweiligen Prozessor adressierbar und stellt Instruktionen und Daten zur Verfügung. Hierbei kann es sich um normalen SRAM handeln oder aber um Cache- bzw. multiprozessorfähigen Cache-Speicher, der im folgenden Abschnitt näher vorgestellt wird. Außerdem ist der gemeinsame L2-Speicher auf Cluster-Ebene noch zum lokalen Speicher zu zählen. Seine Zugriffslatenz liegt zwar über der des eng-gekoppelten Prozessorspeichers, ist aber verglichen mit den Latenzen der entfernteren Speicher immer noch gering.

Die Größe des Prozessorspeichers ist, wie auch die des gemeinsamen Cluster-Speichers, parametrisierbar und derzeit bei der Wishbone-basierten Realisierung mit je 32 KByte vorgesehen. Es handelt sich in beiden Fällen um Dualport-Speicher, der im Falle des Prozessorspeichers zum einen vom Prozessor über den Prozessorbus gelesen und beschrieben werden kann. Zum anderen ist dieser Speicher über eine Wishbone-Slave-Schnittstelle zur Initialisierung oder zum Austausch gemeinsamer Variablen von anderen Wishbone-Bus-Teilnehmern adressierbar (vgl. Abbildung 4-17). Der L2-Speicher auf Clusterebene dient u. a. als Paketspeicher zur Terminierung von GigaNoC-

basierten Paketen bzw. zu deren Inauftraggabe und Injektion über den Communication-Controller ins GigaNoC. Außerdem können über ihn Daten mit anderen Prozessoren des Clusters ausgetauscht werden, was je nach Programmiermodell (vgl. Abschnitt 4.5) von Bedeutung sein kann. Zusätzlich zu diesem Dualport-L2-Speicher ist eine Schnittstelle für ein ggf. externes SRAM-Modul vorgesehen, welches größere Datenmengen zur gemeinsamen Datennutzung halten kann [109]. Abbildung 4-20 zeigt die Wishbone-Bus-Realisierung der GigaNetIC-Architektur auf Cluster-Ebene und die unterschiedlichen Speichermodule auf L1- und L2-Ebene.

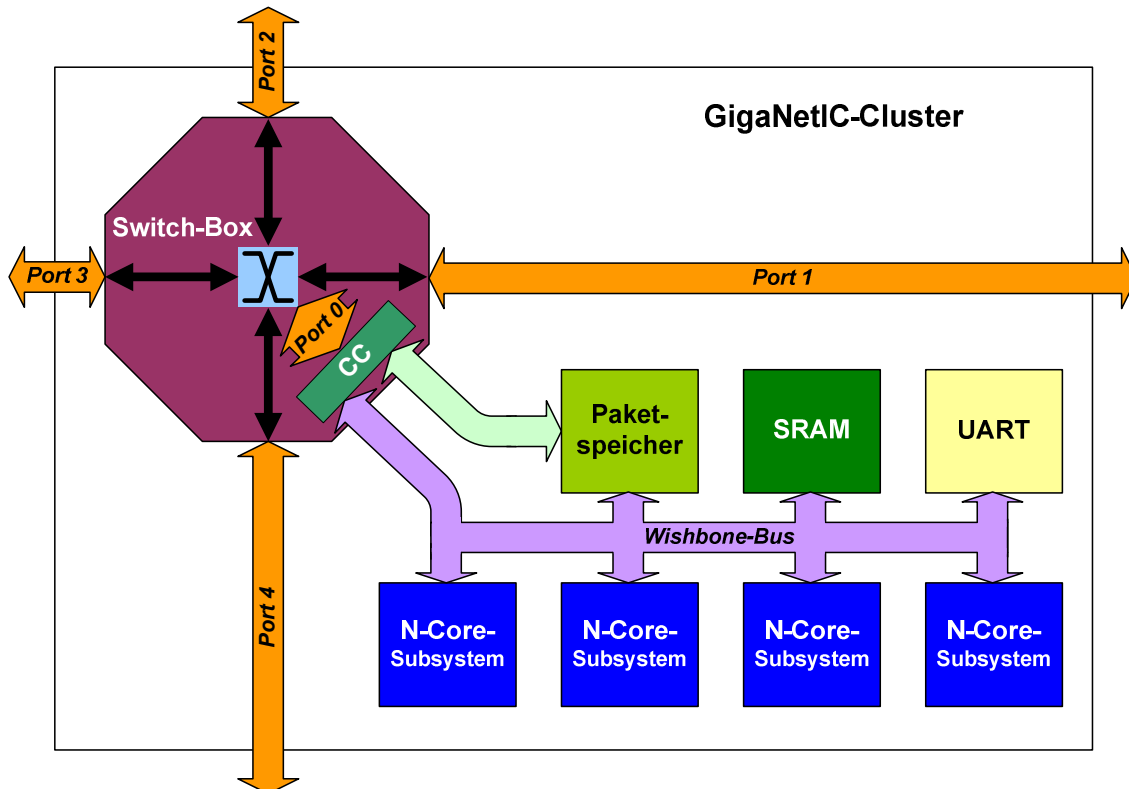


Abbildung 4-20: Wishbone-Bus-basierte GigaNetIC-Architektur auf Cluster-Ebene

#### 4.4.2 Cache-Speicher auf Cluster-Ebene

Für viele Anwendungen empfiehlt sich der Einsatz von Cache-Speichern, die nach dem Lokalitätsprinzip die Daten puffern und so die Ausführung zahlreicher Programme beschleunigen, da Daten bzw. Speicherseiten, auf die häufig zugegriffen wird, nicht jedes Mal neu, zeitaufwändig aus dem Hauptspeicher geholt werden müssen.

Für das GigaNetIC-System wurde ein spezieller Multiprozessor-Cache entwickelt [113], der die Systemleistung für eine Vielzahl von Anwendungen deutlich steigern kann (vgl. Abschnitt 6.7). Er ist ebenso wie die GigaNetIC-Architektur in vielerlei Hinsicht parametrisierbar und somit flexibel an die Anforderungen des entsprechenden Anwendungsszenarios anpassbar. Außerdem lässt er sich aufgrund seiner flexiblen Struktur mit anderen Prozessoren kombinieren und in andere Multiprozessorsysteme integrieren. Abbildung 4-21 zeigt die prinzipielle Realisierung des Multiprozessorcaches am Beispiel eines GigaNetIC-Clusters mit vier N-Core-Prozessoren. Die Prozessoren sind mittels einer AMBA-Switchmatrix (vgl. Abschnitt 4.2.4) und über die Caches miteinander und mit der Switch-Box verbunden. Zusätzlich ist ein so genannter Snooping-Bus integriert worden, der die

Cache-Kontrolllogik über die Transaktionen der einzelnen Caches in Kenntnis setzt und für Datenkohärenz sorgt. Der GigaNetIC-Multiprocessorcache verwendet das MOESI-Kohärenzmodell<sup>23</sup> zur Sicherstellung der Datenintegrität [132]. In [133] wird eine Teilnehmerzahl von vier für MESI-protokollbasierte *Snooping*-Busse bzw. Realisierungen mit ähnlichen Protokollen als leistungsfähige Busstruktur charakterisiert und empfohlen. Die in [113] und im Folgenden erläuterten zusätzlichen Merkmale der AHB-Switchmatrix-basierten und durch den dedizierten Snooping-Bus erweiterten GigaNetIC-Multiprocessorsysteme erhöhen die in [133] beschriebene Leistungsfähigkeit nochmals.

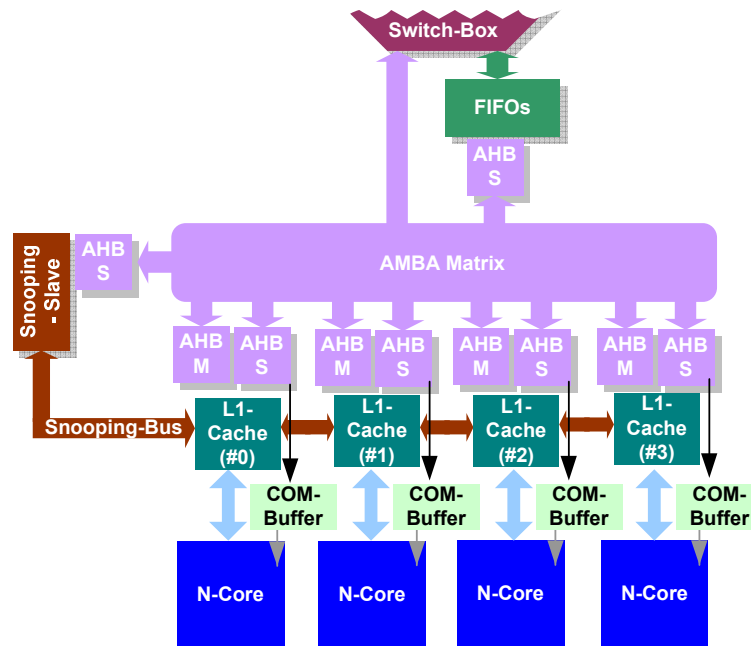


Abbildung 4-21: Integration des Multiprocessor-Caches auf Cluster-Ebene

Der GigaNetIC-Multiprocessor-Cache bietet viele, durch die VHDL-Beschreibung gegebene Freiheitsgrade der Parametrisierung. Der Cache ist nicht nur als Multiprocessorvariante für bis zu acht Prozessoren nutzbar, sondern auch als Uniprocessorcache implementierbar. Es kann zwischen einer *Split*- oder *Unified*-Architektur gewählt werden, so dass optional ist, ob für Instruktionen und Daten separate Speicher verwendet werden oder nicht. Die Assoziativität ist zwischen 2 bis 32 wählbar, wobei pro Weg die Anzahl der *Cachelines* zwischen 8 bis theoretisch  $2^{20}$  *Lines* mit einer Weite von 4 bis 128 Byte eingestellt werden kann. Ebenso ist die Systembusschnittstellenweite zwischen 32 und 1024 Bit Breite parametrisierbar, so dass die Cache-Struktur auf andere Bussysteme und Prozessortypen leicht adaptiert werden kann. Abbildung 4-22 zeigt die wesentlichen Freiheitsgrade der Parametrisierung auf. Viele dieser Parametrisierungen wirken sich sowohl auf die benötigten Flächenressourcen als auch auf die Leistungsfähigkeit aus. Häufig steht der Performanzgewinn in direkter Abhängigkeit zu den jeweiligen Anwendungen, außerdem spielen die Zugriffszeiten zu den verwendeten L3- und L4-Speichern eine nicht unwesentliche Rolle. Detaillierte Aussagen hierzu sind in [113] veröffentlicht. Aufgrund dieser Umstände sollten im Vorfeld, falls die Anwendungen

<sup>23</sup> Hierbei bezeichnet der Begriff MOESI die einzelnen Zustände, die eine Cacheline innehaben kann: *Modified*, *Owned*, *Exclusive*, *Shared* und *Invalid*. Das MOESI-Kohärenzprotokoll findet u. a. in der AMD64-Architektur Verwendung [134].

bereits bekannt sind, tiefgehende Analysen bzgl. der Parameterwahl angestellt werden. Die entwickelte Werkzeugkette stellt hierzu eine umfassende *Profiling*-Umgebung zur Verfügung (vgl. Abschnitt 6.7).

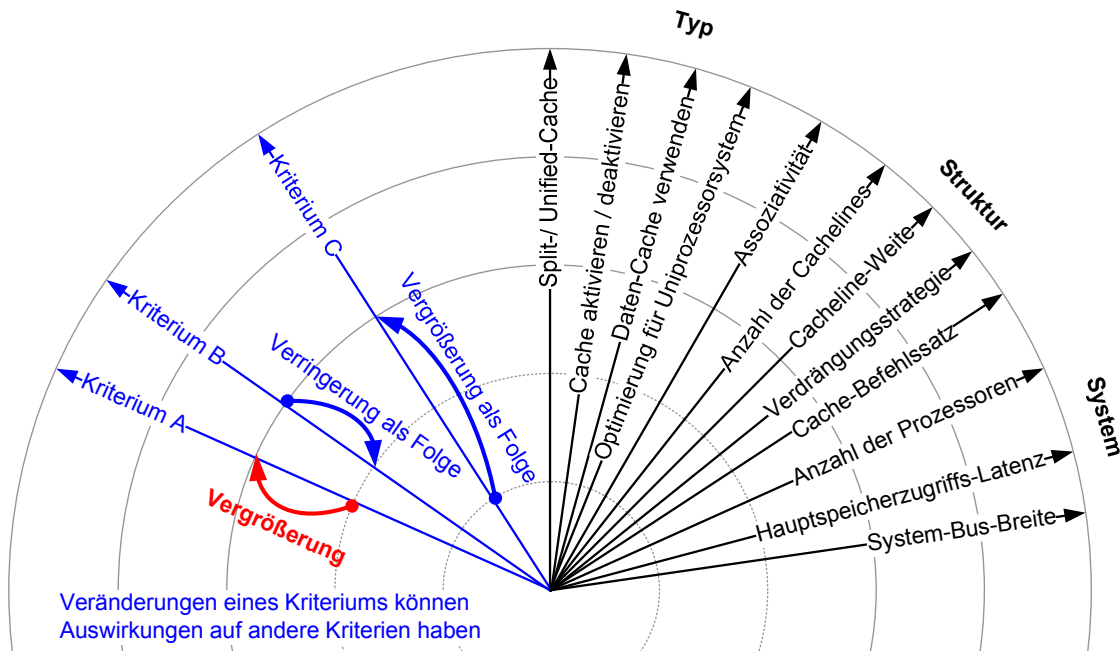


Abbildung 4-22: Wesentliche Freiheitsgrade beim GigaNetIC-Multiprozessorcache

Abbildung 4-23 zeigt die innere Struktur des GigaNetIC-Multiprozessorcaches. In der Standardrealisierung wird eine *Split-Cache*-Struktur mit einem relativ einfachen Block für den Instruktionscache und mit einem komplexeren Teil für den Datencache verwendet. Über den *Communication-Buffer* können innerhalb weniger Takte und unter Umgehung des Caches Daten mit anderen Prozessorsubsystemen oder der Switch-Box des Clusters ausgetauscht werden. Es handelt sich um eine *Write-Back*-Architektur, die die Skalierbarkeit des Clusters erhöht und die Prozessoren vom Systembus entkoppelt. Die parametrisierbare satzassoziative (*set-associative*) Struktur erleichtert die Realisierung geeigneter Kompromisse (bzw. pareto-optimaler Punkte, nach Definition 12 und 13) zwischen Flächenbedarf und Trefferrate (*hit-rate*).

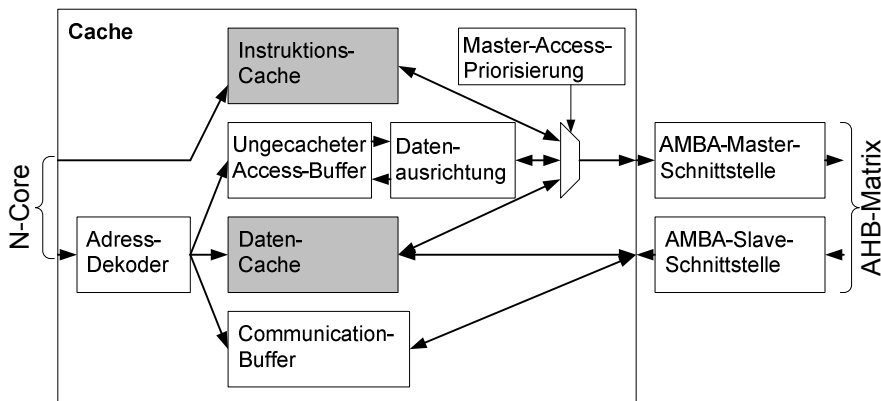


Abbildung 4-23: Prinzipieller Aufbau des multiprozessorfähigen GigaNetIC-Caches

Der derzeit eingestellte *True-LRU*-Verdrängungsmechanismus erlaubt eine hohe Trefferrate und ist zudem mit überschaubarem Aufwand in Hardware zu realisieren. Bei Verwendung des BSP-Programmiermodells (vgl. Abschnitt 4.5.2) basiert das System auf schwacher Datenkonsistenz, weshalb der Cache softwareinitiierte Barrierensynchronisationen unterstützt, die z. B. automatisch

vom Compiler hinzugefügt werden können. Hierdurch wird ein hoher Grad an Programmierbarkeit mit geringer Komplexität ermöglicht. In diesem Fall kann der Communication-Buffer zur schnellen Nachrichtenübermittlung (*Message Passing*) eingesetzt werden. Ein zusätzlicher *Uncached Access Buffer* dient als Zwischenspeicher für optionale, „ungecachte“ Zugriffe.

Der Cache unterstützt eine Vielzahl von speziellen Cachebefehlen, die entweder manuell, z. B. Software-basiert durch den Prozessor, oder aber schon im Vorfeld vom Compiler angestoßen werden können. Hierzu zählen u. a. *Prefetching*, also das Laden von Speicherbereichen, bevor diese vom Prozessor für die Verarbeitung benötigt werden, Festschreiben (*Locking*) oder Freigeben (*Unlocking*) einer Cacheline, Invalidierung (*Invalidation*) einer Cacheline, Aufheben der Kohärenz durch Exkludieren von Zeilen aus der Kohärenzverwaltung etc. Weitere Details zu den Merkmalen des Caches sind [113] zu entnehmen.

Der GigaNetIC-Multiprozessorcache unterstützt mit Hilfe des MOESI-Protokolls *Direct-Data-Intervention* (direkten Daten-Eingriff). Dies bezeichnet die Möglichkeit, Daten von einem Cache zu einem anderen Cache des Clusters transportieren zu können (ohne einen Zwischenschritt über den Hauptspeicher). Dies bedeutet einen deutlichen Performanzvorteil für den Fall, dass andere Caches Daten schneller liefern können als der Hauptspeicher.

Da der lokale Speicher des Clusters für einige Anwendungsklassen verhältnismäßig klein konzipiert sein wird, und die Hauptspeicheranbindung durch ein recht großes globales Kommunikationsnetzwerk bzw. durch die verwendete Speichertechnologie ggf. eine nicht zu vernachlässigende Latenz aufweisen wird, ist Direct-Data-Intervention eine gute Möglichkeit, solche Latenzen zu vermeiden. Durch Direct-Data-Intervention wird so die effektiv zur Verfügung stehende Speichermenge des Clusters ggf. erhöht, was bei der ansonsten relativ geringen L1- und L2-Speichergröße einen weiteren positiven Effekt bedeutet.

**Tabelle 4-7: Synthesewerte für Varianten des GigaNetIC-Multiprozessorcaches in 90-nm-Standardzellentechnologie**

Konfiguration	Line-Größe [Bit]	Tiefe	Assoziativität	Split-Cache	Fläche [ $\mu\text{m}^2$ ]	Taktperiode [ns]	Leistungsaufnahme @250MHz [mW]
kurze Cachelines	64	256	2	Nein	0,61	4,16	136,79
Split Cache	128	256	2	Ja	1,11	4,08	276,13
hohe Assoziativität	128	256	4	Nein	1,31	4,48	376,18
weniger Cachelines	128	128	2	Nein	0,58	3,92	164,87
Standard	128	256	2	Nein	0,73	4,1	180,99

Tabelle 4-7 zeigt die wesentlichen Synthesergebnisse für ausgewählte Cachevarianten in einer 90-nm-Standardzellentechnologie. Hierbei werden jeweils die Daten für einen Cache angegeben, so dass Fläche und Leistungsaufnahme für die Clusterimplementierung mit der Anzahl der instanziierten Prozessoren pro Cluster multipliziert werden müssen, um die Fläche bzw. die Leistungsaufnahme des „eigentlichen“ Multiprozessorcaches zu erhalten (vgl. Tabelle 8-4). Bzgl. der Leistungsaufnahme sei bemerkt, dass es sich hier um sehr konservative Abschätzungen seitens des Synthesewerkzeugs handelt, das eine Schaltwahrscheinlichkeit von 50 % annimmt und nicht die als deutlich geringer anzunehmenden Werte des realen Verhaltens der Schaltung ansetzt. Hierzu kann die erstellte Werkzeugkette deutlich genauere Werte liefern, die auf den tatsächlichen Schaltaktivitäten der Komponenten während der Verarbeitung basieren [116][111] (vgl. Kapitel 5 und 6).

Aus Tabelle 4-7 wird ersichtlich, dass der GigaNetIC-Cache in der Standardkonfiguration mit 8 KByte und einer Fläche von 0,73  $\text{mm}^2$  verglichen mit dem normalen L1-Speicher der Wishbone-

Realisierung (32 KByte Dual-Port-Speicher, 0,875 mm<sup>2</sup>) ca. 3,3 mal so viel Fläche pro KByte benötigt. Zudem erlaubt er derzeit eine nur halb so hohe maximale Betriebsfrequenz, was allerdings durch den langsameren N-Core nicht ins Gewicht fällt. Trotz der deutlich höheren Kosten des Caches im Sinne von Flächenbedarf bzw. Leistungsaufnahme ist von Fall zu Fall, d. h. respektive des Anwendungsszenarios und der definierten Randbedingungen, abzuwägen, ob sich sein Einsatz dennoch rentiert – gerade vor dem Hintergrund der bereits erwähnten, Komplexitätssprünge von zukünftigen SoCs aufgrund der immens wachsenden Transistorzahlen. Wir konnten in [113] zeigen, dass unsere Cache-Implementierung für ausgewählte Anwendungen Performanzsteigerungen von Faktor 23 bzw. sogar eine Reduzierung der benötigten Energie von bis zu 89 %, verglichen mit einer Implementierung mit normalem lokalen Speicher, ermöglicht, vgl. auch Abschnitt 6.7.

Die nächste eigenständige Speicherhierarchie ist der L4-Speicher oder auch Hauptspeicher, der im folgenden Abschnitt diskutiert wird.

### 4.4.3 Hauptspeicher

Der Hauptspeicher des GigaNetIC-Systems kann, je nach Anwendungsgebiet, als Pufferspeicher für Netzwerkdaten oder Anwendungsdaten eingesetzt werden. Er kann je nach Chipgröße und Technologie direkt auf dem Die integriert werden und lässt sich, wie oben beschrieben, über Switch-Box-Ports oder clusterbasiert adressieren. Die Adressvergabe und der Adressraum können hierbei von den Kontrolleinheiten der modifizierten Communication-Controller übernommen werden. Sollen standardisierte Off-Chip-Speichermodule eingesetzt werden, stellen die Communication-Controller die Schnittstelle nach außen zur Verfügung. Durch diese Kopplungsart, unabhängig ob der Speicher *on-* oder *off-chip* positioniert ist, erlaubt die GigaNetIC-Architektur eine gute Skalierbarkeit der Speichergröße. Zur Ansteuerung von SDRAM (*Synchronous Dynamic Random Access Memory*) kann auf einen im Fachgebiet Schaltungstechnik entworfenen IP-Block zurückgegriffen werden.

Die etwaige Umsynchronisierung auf die Taktfrequenz des Chip-Multiprozessors kann von den angepassten Communication-Controllern durchgeführt werden, da nicht grundsätzlich von gleichen Taktraten auf Speicher- und CMP-Seite auszugehen ist. Die Bandbreite, die seitens der GigaNetIC-Architektur zur Verfügung gestellt werden muss, um heutigen Speichermodulen gerecht werden zu können, beträgt bei den derzeit schnellsten PC-Speichern, den DDR3-1600-PC3-12800-Speichermodulen, 12,8 GB/s bei 800 MHz, und bei derzeit üblichen PC-Speicherriegeln, den DDR2-667 PC2-5300 mit 333 MHz, 5,3 GB/s. Diese Speicherriegel sind mit einer 64-Bit-breiten Datenanbindung bereits passend für die derzeit eingestellte Flitdatenbreite eines Switch-Box-Ports. Wie in Abbildung 4-15 gezeigt, ermöglicht ein Port einer Switch-Box in der derzeitigen Realisierung bereits bis zu 5,7 GB/s Netto-Datendurchsatz und damit schon heute die volle Bandbreitenausnutzung gängiger externer Speichermodule. Im Falle von höheren Speicherbandbreiten kann ggf. mit Hilfe einer Portbündelung der Switch-Boxen die gewünschte Performanz erzielt werden.

Um die in den letzten Abschnitten vorgestellten Kernkomponenten eines Chip-Multiprozessor-systems wie dem GigaNetIC effizient einsetzen zu können, bedarf es zusätzlich zu den Hardware-Komponenten noch eines angepassten Programmiermodells und einer leistungsfähigen Werkzeugkette. In Abschnitt 4.5 werden zunächst anwendbare Programmiermodelle für das GigaNetIC-System vorgestellt; die im Rahmen des Projekts entstandene Werkzeugkette wird ausführlich in den Kapiteln 5 und 6 behandelt.



## 4.5 Programmiermodell

Außer den physikalisch greifbaren Systementitäten der GigaNetIC-Chip-Multiprozessorarchitektur gibt es noch die bereits erwähnten immateriellen Bestandteile, die eine Multiprozessorarchitektur ausmachen. Hierzu zählt das Programmiermodell, das letztendlich festlegt, nach welchen Regeln die einzelnen Komponenten des Systems ihre Arbeit verrichten und wie ihnen diese zugeteilt wird, und damit verbunden der systemweite Austausch von Daten und Zustandsinformationen.

Programmiermodelle für parallele Systeme müssen eine Vielzahl von Faktoren berücksichtigen, zu deren wichtigsten Merkmalen zählen:

- **Kontrolle:** Wie wird die Parallelität zur Verfügung gestellt und wie werden die Einheiten synchronisiert?
- **Daten:** Welche Daten sind lokale und welche sind gemeinsame Daten? Wie können gemeinsame Daten erreicht bzw. übermittelt werden?
- **Operationen:** Welche atomaren Operationen werden vom System zur Verfügung gestellt?
- **Kosten:** Mit welchen Kosten können die obigen Faktoren belegt werden?

Es gibt eine Vielzahl von Programmiermodellen für parallele Systeme, wie *Shared Memory*, *Message Passing* oder *Data Parallel*. Allen Modellen gemeinsam sind die vier Phasen der Anwendungsabbildung vgl. Abbildung 4-24, in denen die Anwendung zunächst in passende Aufgaben bzw. Tasks genügender Granularität aufgeteilt wird. Hierbei ist ein Kompromiss zwischen genügend Nebenläufigkeit und dem daraus resultierenden Verwaltungsaufwand zu finden. Im Anschluss müssen diese Tasks zu geeigneten Prozessen bzw. Threads zugeordnet werden. Dabei ist auf eine möglichst ausgewogene Verteilung der Aufgaben auf die einzelnen Verarbeitungseinheiten zu achten. Dieser Vorgang muss in der Art geschehen, dass die Prozesse möglichst effizient und auf die architektur-spezifischen Gegebenheiten angepasst miteinander kommunizieren können. Ziel hierbei ist eine korrekte Abbildung der parallelen Verarbeitung auch im Hinblick auf inhärente Datenabhängigkeiten des Algorithmus. In Abhängigkeit von der jeweiligen Kostenfunktion sind die einzelnen Maße wie Kommunikation, Häufigkeit der Synchronisationen und Verwaltungsaufwand durch die parallele Lösung gegeneinander abzuwägen.

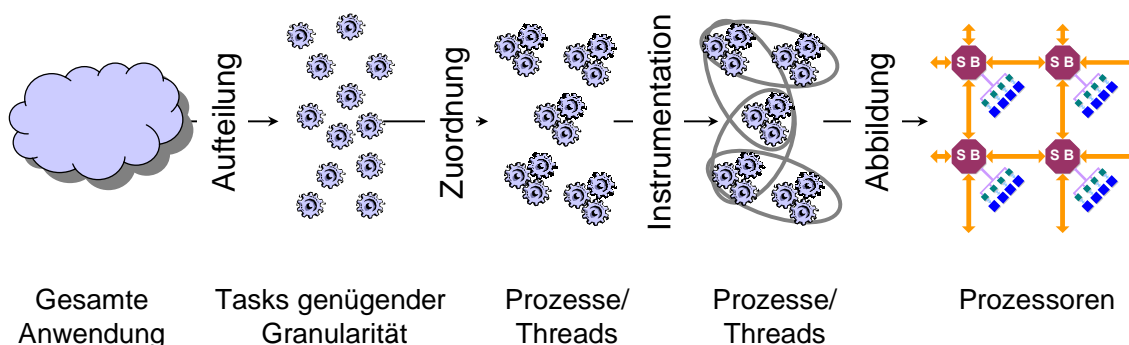
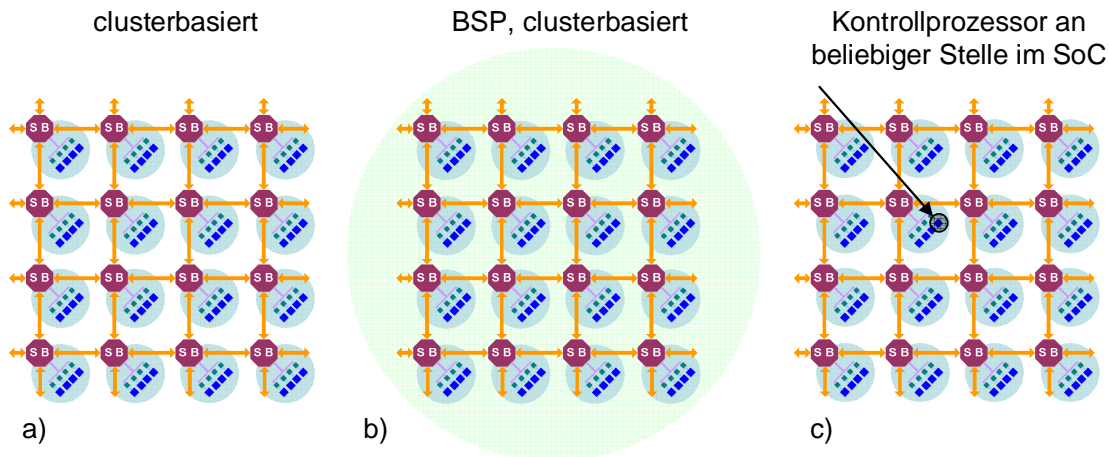


Abbildung 4-24: Schritte der Anwendungsabbildung

Die für die GigaNetIC-Architektur vorgesehenen Programmiermodelle basieren, ebenso wie die GigaNetIC-Hardware, auf einem hierarchischen Ansatz. Auf Clusterebene wird ein speziell auf die eingesetzten Verarbeitungseinheiten optimierter, aus einer eigens für das GigaNetIC-System entwickelten Werkzeugkette generierter Compiler eingesetzt [134][135][6][111][112][136]. Programme

werden in der Hochsprache C verfasst und auf die Prozessoren des Clusters abgebildet (vgl. Abschnitt 4.5.1). Sollten keine übergeordneten Kontrollmechanismen zur Ausführung benötigt werden, so kann dieser einfache Ansatz bereits ausreichen, um das System die gestellten Aufgaben effektiv bearbeiten zu lassen. Andernfalls dient das clusterbasierte Programmiermodell als untere Hierarchie, auf der eines der beiden Programmiermodelle der SoC-Hierarchie aufsetzt (vgl. Abschnitt 4.5.2 und 4.5.3).



**Abbildung 4-25: Drei wesentliche Programmiermodelle des GigaNetIC-Systems:**

**a) dezentrales Cluster-Modell, b) globales SoC-BSP-Modell und c) zentrales SoC-Modell**

Auf Systemebene kann auf das etablierte *Bulk-Synchronous-Parallel-Programmiermodell* nach VALIANT [18] für Parallelrechner zurückgegriffen werden. Für Anwendungsklassen geringerer Komplexität kann das leicht zu implementierende zentrale SoC-Programmiermodell eingesetzt werden, das einen zentralen Kontrollprozessor zur Ablaufsteuerung einsetzt. Beiden Programmiermodellen gemein ist die Möglichkeit der komfortablen Nutzung der GigaNoC-Software-Systembibliothek (vgl. Abschnitt 4.2.2), in der alle Funktionalitäten, die das On-Chip-Netzwerk zur Verfügung stellt, enthalten sind. Abbildung 4-25 zeigt die drei derzeit eingesetzten Programmiermodelle, die im Folgenden näher erläutert werden.

#### 4.5.1 Programmiermodell auf Clusterebene

Der vom Fachgebiet Kastens zur Verfügung gestellte Compiler kann automatisch Befehlsatzerweiterungen und eng-gekoppelte Hardwarebeschleuniger des Clusters berücksichtigen und mit in die Code-Abbildung einbeziehen. Für Anwendungen, die sich besonders für eine feingranulare Parallelisierung auf Instruktionsebene (*ILP*) (vgl. Abschnitt 2.4.4) eignen, kann zusätzlich eine Kompilierung des Programms für mehrere oder alle Prozessoren des Clusters angestoßen werden. Für diesen parallelisierenden Compiler wurde das GigaNetIC-System durch zusätzliche konfigurierbare Hardwareblöcke erweitert, die es ermöglichen, die für die Synchronisierung der Prozessoren erforderlichen Barrieremechanismen innerhalb eines Taktes umzusetzen [6]. Abbildung 4-26 zeigt den Ablauf einer solchen Synchronisierung, die immer dann notwendig ist, wenn auf Variablen bzw. Daten zugegriffen wird, die von anderen Verarbeitungseinheiten modifiziert wurden. Die Barrieren werden vom Compiler als eigenständiger, parametrisierter Befehl dahin gehend, welche PEs synchronisiert werden müssen, in den Programmablauf integriert. Die Hardware übernimmt die Synchronisation, also das Anhalten und Fortsetzen der Befehlsausführung der betreffenden Prozessoren. Für welche Anwendungsszenarien sich diese feingranulare Parallelisierung besonders effizient einsetzen lässt ist noch Bestandteil aktueller Forschungen.

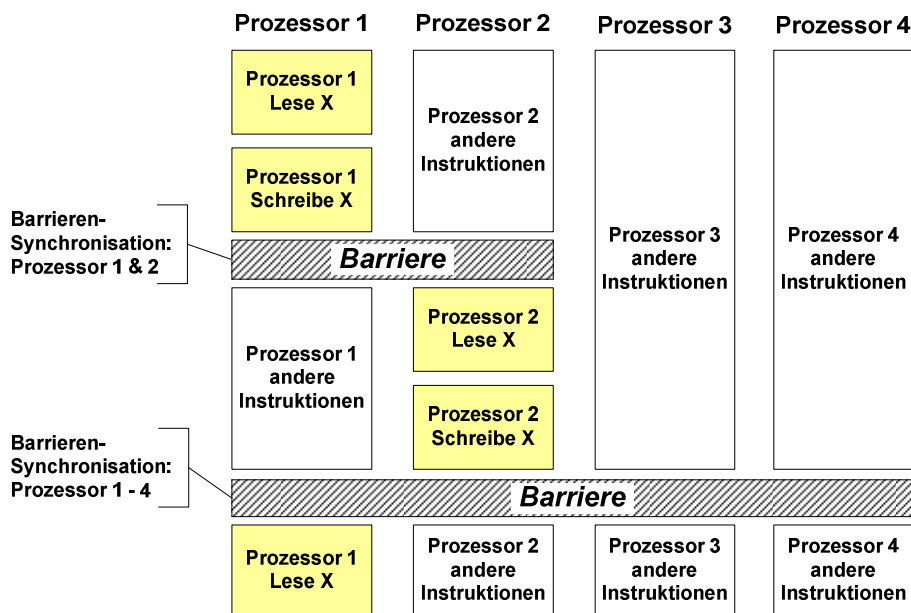


Abbildung 4-26: Synchronisierungsmechanismus auf Cluster-Ebene

Das Zusammenspiel der Compiler-Werkzeugkette mit der Hardware-Entwicklungsumgebung für den am Fachgebiet entwickelten N-Core-RISC-Prozessorkern [108][111] (vgl. Abschnitt 4.3.1) wird in Kapitel 5 und 6 detailliert beschrieben.

Wird dieses Programmiermodell ohne Zuhilfenahme eines der beiden für die SoC-Ebene konzipierten Ansätze eingesetzt, dann sind die Cluster bzw. sogar die einzelnen Prozessoren allein für die Ausführung der Anwendung zuständig. Eine Synchronisierung mit anderen Blöcken des SoCs kann nur über vorbestimmte Nachrichten geschehen. Diese Art der Programmierung eignet sich für relativ einfache Anwendungsklassen, die vollständig auf einzelne Cluster bzw. Prozessorkerne abgebildet werden können. Dies können z. B. einfache Paketverarbeitungsprozesse sein, bei denen die parallele Architektur auf unkorrelierten parallelen Datenströmen arbeitet (vgl. Abschnitt 8.1). Für komplexere Anwendungsklassen stehen übergeordnete Programmiermodelle zur Verfügung, die in den nächsten beiden Abschnitten vorgestellt werden.

#### 4.5.2 Programmiermodell auf SoC-Ebene – *Bulk Synchronous Parallel*

Das BSP-Modell von VALIANT [18][6] ist ein Ansatz, der versucht die beiden Seiten eines Multiprozessorsystems, Software und Hardware, kombiniert zu modellieren. Zum einen dient es dem Systemarchitekten als Modell für die parallele Verarbeitung und die Auswirkungen der Hardwarearchitektur auf die Systemleistung. Zum anderen ist es ein Programmiermodell für die Algorithmen-Entwickler. Es soll als gemeinsamer Standard dienen, um möglichst effiziente Systeme zu entwickeln, bei denen das Zusammenspiel zwischen Hard- und Software gut aufeinander abgestimmt ist.

Das Modell besteht aus drei Teilen: Die Charakterisierung der Hardware, das Programmiermodell als solches und das Kostenmodell zur Abschätzung der Laufzeit der BSP-Algorithmen. Ein Computer bzw. ein Chip-Multiprozessor besteht aus Sicht des BSP-Modells aus  $P$  Prozessoren, die über lokalen Speicher verfügen und über ein beliebiges Netzwerk miteinander über Punkt-zu-Punkt-Verbindungen kommunizieren können. Zudem muss das System eine Barriersynchronisation (ähnlich Abbildung 4-26) unterstützen. BSP macht zunächst keine Unterscheidung bzgl. der Topologie oder Lokalität der Daten. Das auszuführende Programm wird in mehrere sequentiell aufeinander folgende Teile, so genannte *Supersteps* zerlegt (vgl. Abbildung 4-27). Während der Supersteps

kann jeder Prozessor lokale Berechnungen durchführen und Nachrichten zu anderen Prozessoren schicken. Am Ende des Supersteps wird eine Synchronisierungsfunktion aufgerufen. Haben alle Prozessoren diese „Barriere“ erreicht, werden die Nachrichten innerhalb des folgenden Supersteps ausgewertet bzw. verarbeitet. Die Art und Weise, Algorithmen so zu beschreiben hat u. a. folgende Vorteile: Aufgrund der einseitigen Kommunikation, ohne Anforderungs-/Bestätigungsmechanismus, vermeidet man Blockaden, die z. B. durch falsch initiierte Anforderungsnachrichten verursacht werden könnten. Ein weiterer Vorteil ist der, dass der Ablauf deterministisch und unabhängig von der Latenz des Netzwerks ist, bzw. die Reihenfolge, in der die Nachrichten bei ihren Empfängern eintreffen, ist irrelevant. Abbildung 4-27 zeigt den prinzipiellen Ablauf eines Supersteps im BSP-Modell und die damit verbundenen Kosten auf.

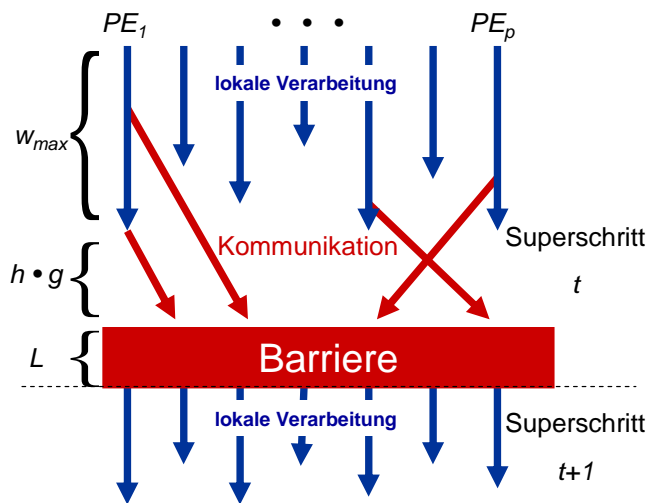


Abbildung 4-27: Ablauf und Kosten beim BSP-Modell

Die Gesamtkosten eines Supersteps  $w_{max}$  setzen sich aus dem Maximum der gesendeten bzw. empfangenen Daten  $h$  der Prozessoren multipliziert mit  $g$ , der Lücke  $gap$  die durch die Übertragung durch das Netzwerk entsteht, und der Latenz  $L$  für die Barrierensynchronisation zusammen.  $g$  ist von mehreren Parametern abhängig wie z. B. von der Art der verwendeten Netzwerkprotokolle und von der Art und Weise, wie Nachrichten ins Netz injiziert werden können. Die Routingstrategie beeinflusst das Zeitverhalten der Kommunikation sowie das Maß wie effektiv das Speichermanagement im Prozessor und im Netzwerk ist. Zuletzt ist noch der Overhead durch die BSP-Implementierung zu berücksichtigen.  $g$  wird bei realen Systemen meist durch Messungen bestimmt und bezieht sich auf die Übertragungszeit für eine Nachricht einfacher Länge unter kontinuierlichem Netzwerkverkehr des Systems.

$$\begin{aligned}
 T_{Superstep} &= \max \text{ Berechnungszeit} + \max \text{ Kommunikation} + \text{Synchronisierungszeit} \\
 &= \max_{i=1}^P (w_i) \quad + \max_{i=1}^P (h \cdot g) \quad + L
 \end{aligned} \tag{4.10}$$

Die Kosten für den gesamten Algorithmus mit  $P$  Prozessoren und  $S$  Supersteps lassen sich somit wie folgt berechnen:

$$T_{gesamt} = W + Hg + SL = \sum_{s=1}^S w_s + g \cdot \sum_{s=1}^S h_s + SL \tag{4.11}$$

Die existierende Paderborner BSP-Bibliothek für diskrete Multiprozessoren [137] wurde vom Fachgebiet von Prof. Friedhelm Meyer auf der Heide auf die N-Core-Architektur portiert und für diese optimiert. Die BSP-Bibliothek ist in C implementiert und setzt hierarchisch gesehen oberhalb

des N-Core-Compilers an und kann somit von allen speziellen Funktionen (ILP, spezielle Instruktionen, Hardwarebeschleunigeransteuerung etc.) des Compilers profitieren. Besonderheiten der GigaNetIC-Architektur in Bezug auf das BSP-Modell sind spezifische, sehr schnelle Synchronisierungsmechanismen, die zu sehr kurzen Latenzwerten für  $L$  führen, sowie die sehr geringen Kommunikationskosten, verglichen mit Grid- oder Cluster-Multiprozessorsystemen. Daher kann das sonst eher für grobgranulare Parallelität verwendete Programmiermodell auch in Anwendungen eingesetzt werden, die stärker durch feingranulare Parallelität profitieren.

In [115] wurden von mir folgende Werte für die GigaNetIC-Architektur im Hinblick auf die BSP-Kosten ermittelt:

$$Takte_{\text{besten Fall}} / \text{Paket} = 3 \cdot (h+1) \cdot \left\lceil \frac{m}{m_f} \right\rceil \quad (4.12)$$

$$Takte_{\text{schlechtester Fall}} / \text{Paket} = 3 \cdot (h+1) \cdot (n-1) \cdot \text{FIFOTiefe} \cdot \left\lceil \frac{m}{m_f} \right\rceil \quad (4.13)$$

In (4.12) sind die Kosten bzw. Takte angegeben, die ein Paket  $M$ , bestehend aus  $m$  Bytes benötigt, um von einem Cluster zu einem  $h$  Hops entfernten Cluster im GigaNetIC-System zurückzulegen. Die Anzahl der Datenbyte pro Flit wird mit  $m_f$  angegeben. Im besten Fall sind alle FIFO-Ketten des betreffenden Pfades leer, dann ergibt sich die Anzahl der benötigten Takte zur Übertragung zu (4.12). Als schlechtester Fall wird hingegen angenommen, dass alle FIFO-Ketten entlang des Übertragungsweges gefüllt und die anderen  $(n-1)$  Ports der entsprechenden Switch-Boxen ebenfalls um die betreffenden Ausgangswarteschlangen konkurrieren, dann ergibt sich die Anzahl der benötigten Takte zu (4.13).

Die Takte der entsprechenden Funktionen, die aufgewendet werden müssen, um ein Paket seitens eines Prozessors ins Netzwerk zu injizieren sind in (4.14) aufgeführt. Sie resultieren aus den Ausführungszeiten der entsprechenden Funktionen der GigaNoC-Softwarebibliothek.

$$PE_{\text{Takte}} / \text{Paketinjektion} = \overbrace{\text{net\_send\_data}}^{96} + \overbrace{\text{net\_get\_ack}}^{42} + \overbrace{\text{net\_free\_packet}}^9 \quad (4.14)$$

Zusätzlich gibt es eine Funktion *is\_bsp\_synchronization*, die innerhalb von 50 Takten feststellt, ob ein eingetroffenes Paket eine BSP-Barrieren-Synchronisations-Nachricht eines anderen Clusters ist.

In [115] wurden, mit Hilfe der zur GigaNetIC-Architektur gewonnenen BSP-Parameter, Analysen zu einem erweiterten BSP-Modell vorgestellt, auf die hier nicht näher eingegangen werden kann.

### 4.5.3 Programmiermodell auf SoC-Ebene – Zentraler Kontrollprozessor

Für weniger komplexe Algorithmen, die besondere Anforderungen an den Ablauf oder das Zeitverhalten der Ausführung, aber weniger Inter-Prozesskommunikation beinhalten, wie es z. B. bei einigen Netzwerkanwendungsszenarien der Fall ist, kann ein weiteres Programmiermodell auf SoC-Ebene zum Einsatz kommen. In diesem Fall wird nicht, wie beim BSP-Modell (vgl. Abschnitt 4.5.2), versucht, das Programm durch Partitionierung auf mehrere Verarbeitungseinheiten aufzuteilen und die Synchronisierung durch Barrieren, wie in Abbildung 4-27 gezeigt, zu realisieren. Statt dessen wird die globale Kontrolle von einem zentralen Prozessor übernommen. Dabei kann im Prinzip

ein beliebiger Prozessor eines beliebigen Clusters des Systems diese Aufgabe zugeteilt bekommen. Allerdings sind die anwendungsspezifischen Lastaufkommen zu berücksichtigen, die ggf. eine zentrale Position im Gitter oder an einer der Kanten, und damit nah an den externen Schnittstellen des Chips, begünstigen können. Dieser Kontrollprozessor überwacht die Zustände der einzelnen Cluster und Prozessorkerne und steuert das Verhalten des Gesamtsystems. Er kann u. a. als Lastverteiler (*Load balancer*) fungieren, indem er die eintreffenden parallelen Paketströme, je nach Auslastungsgrad, den einzelnen Clustern und damit den zugehörigen Switch-Boxen zuweist.

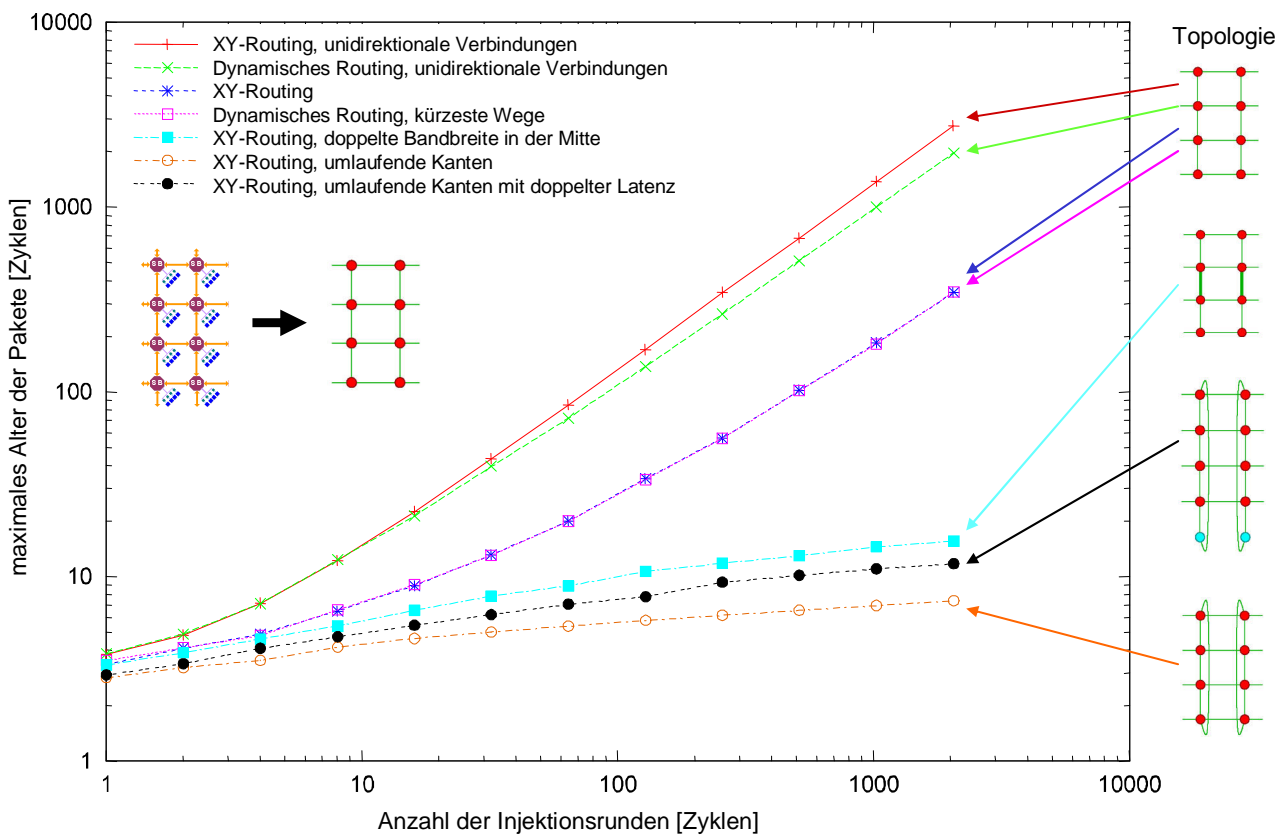
Dieses Modell ist relativ einfach zu implementieren und eignet sich für Prozesse, die aufgrund ihrer Komplexität nicht auf mehrere Prozessoren aufgeteilt werden müssen, sondern bei denen die Vielzahl der Prozessoren zur Bearbeitung gleichartiger Aufgaben auf verschiedenen Daten genutzt wird.

In [102][103][104] stellen wir zusätzlich eine Methode und die dazugehörige Werkzeugkette *NetAMap (Network Application Mapper)* vor, mit der wir in der Lage sind, taskbasierte Anwendungen auf Multiprozessorsystemen unterschiedlicher Art abzubilden. Hierbei können Applikationen abgebildet werden, deren Bandbreitebedarf, Rechenlast bzw. Verarbeitungsschritte während der Konzeption bereits bekannt sind und sich während des Betriebs deterministisch verhalten. Diese Eigenschaft ist allerdings eine starke Einschränkung und trifft nur für spezielle Anwendungsszenarien zu und gilt nicht für Netzwerkanwendungen allgemein. Dies können z. B. Protokollverarbeitungsprozesse für selbst konzipierte mobile Ad-Hoc-Netzwerke sein, z. B. für autonome Miniroboter [138], deren Struktur speziell auf die Abbildungsmethode angepasst ist. Diese Art der Anwendungsszenarien liegt nicht im Mittelpunkt der in Kapitel 7 untersuchten Einsatzgebiete der GigaNetIC-Architektur, weshalb an dieser Stelle, um den Rahmen dieser Arbeit nicht zu sprengen, auf die genannten Veröffentlichungen lediglich verwiesen sei.

## 4.6 Diskussion von Topologie und Routingverfahren

In [6] wurden bereits verschiedene 2D-Topologien bezüglich ihrer Routingeigenschaften untersucht. Wesentliche Kriterien wie hoher Durchsatz gekoppelt mit möglichst geringer Latenz sind für die GigaNetIC-Architektur von großer Bedeutung. Dies soll möglichst nicht auf Kosten einer zu großen Fläche bzw. einer zu hohen Leistungsaufnahme gelöst werden.

Abbildung 4-28 stellt das zeitliche Verhalten und die Aufenthaltsdauer im NoC von Paketen für zwei unterschiedliche Routingverfahren und mehrere Gittervarianten dar. Die dargestellten Werte wurden durch Simulationen ermittelt. Das zugrunde liegende Chip-Multiprozessorsystem entspricht dem in Abschnitt 8.2 realisierten ersten ASIC-Prototypen des GigaNetIC-Systems mit 32 Prozessoren und 8 Switch-Boxen als On-Chip-Routingknoten. Als Routingverfahren wurde das XY-Routing und eine Form des dynamischen Routings eingesetzt, die in Abschnitt 4.2.1.4 bereits vorgestellt wurden. Der Simulation zugrunde liegt ein Paketinjektionsalgorithmus mit zufallsbasiertem Ziel im NoC. Jeder Prozessor eines Clusters verschickt in jeder Runde ein Paket an einen wahllos ausgesuchten Zielknoten im GigaNoC. Die Simulation ist beendet, wenn nach Ablauf der Injektionsrunden alle Pakete ihr Ziel erreicht haben.



**Abbildung 4-28: Performanzanalyse verschiedener Routingverfahren  
in Verbindung mit unterschiedlichen Gitter-Topologien**

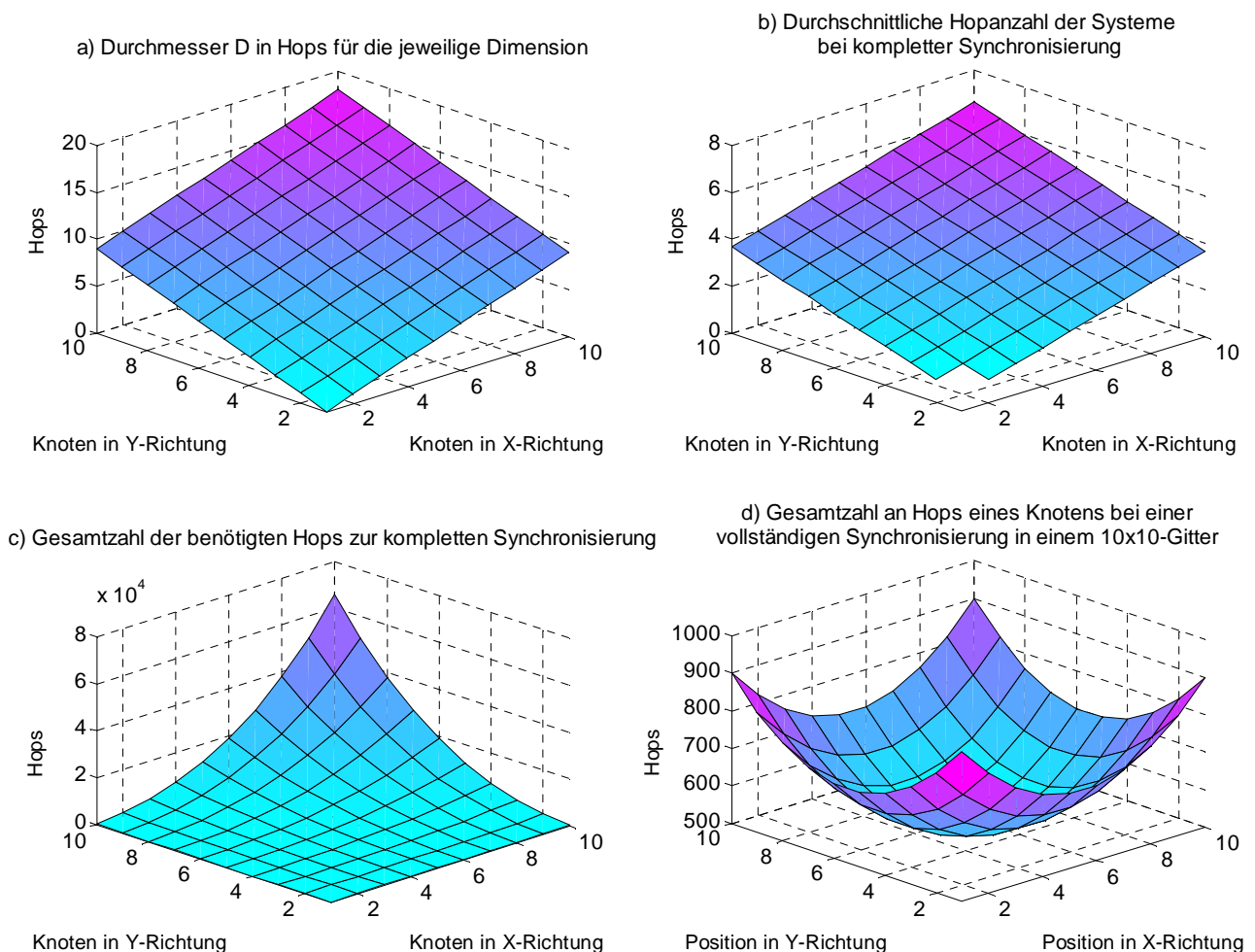
Bei der unidirektionalen Variante kann jeder Inter-Switch-Box-Link nur ein Paket in eine Richtung weiterleiten. Dies könnte z. B. der Fall sein, wenn nicht genügend Fläche für bidirektionale Verbindungen zwischen den Switch-Boxen vorhanden wäre. In diesem Fall verhält sich das dynamische Routing vorteilhafter als das XY-Routing. Im Gesamtvergleich benötigen jedoch beide Verfahren deutlich mehr Zeit zur Bewältigung des Datenaufkommens als die anderen Varianten. Verwendet man hingegen bidirektionale Verbindungen, entfällt der Vorteil des dynamischen Routings und die Performanz ist nahezu um eine Größenordnung besser.

Ein Problem bei dem  $4 \times 2$ -Gitter ist die geringe Bisektionsbandbreite  $B_B$  (vgl. Abschnitt 2.3.1). *Congestion* auf den beiden zentralen Verbindungskanälen des NoCs reduziert die Leistungsfähigkeit des Systems deutlich. Neben den in Abschnitt 2.3.2 genannten Routingverfahren, die z. T. erheblichen Hardwareaufwand bedeuten, lassen sich Verbesserungen um mehr als eine Größenordnung in diesem Fall z. B. durch einfache Topologiemodifikationen erreichen:

Ist mit sehr hohem Datenaufkommen innerhalb des On-Chip-Netzwerks zu rechnen, so können Verbindungen doppelter Bandbreite, z. B. durch entsprechend angepasste Switch-Box-Ports einen deutlichen Geschwindigkeitsvorteil liefern. Die leistungsfähigste Variante stellt so genannte *Wrap-around*-Verbindungen, also umlaufende Verbindungen zur Verfügung und bildet damit die Struktur eines halbverbundenen Torus (vgl. Abbildung 2-8) auf der oberen Hierarchie nach. Sie bedeuten grundsätzlich eine Performanzsteigerung, verringern sie doch den Durchmesser des On-Chip-Netzwerks. Allerdings ist die Realisierung einer solchen Topologie, mit derzeitigen CMOS-Prozessen, mit Schwierigkeiten verbunden, da die umlaufenden Kanten deutlich länger als die übrigen Verbindungen sind. Dies führt zu Laufzeitunterschieden und damit zur Leistungsreduktion des Gesamtsystems. Diesem Phänomen wird mit einer leicht abgewandelten Topologie entgegengetre-



ten, bei dem *Repeater* zwischengeschaltet sind, die die Laufzeitunterschiede egalisieren und so die längeren Verbindungen zwischen Nord- und Südknoten transparent erscheinen lassen. Allerdings wird hierdurch eine zusätzliche Latenz eingebracht, der mit einem „virtuellen Knoten“ Rechnung getragen wird. Auffällig ist die dennoch gute Performanz trotz dieser zusätzlichen Latenz. Je nach Betriebsfrequenz eignet sich eine solche Topologiemodifikation auch für fertige ASICs des GigaNetIC-Systems, die auf *Board*-Ebene dann mit solchen Verbindungen ggf. zu einem Torus erweitert werden können.



**Abbildung 4-29: Anzahl benötigter Hops bei Multicastszenarien,  
in Abhängigkeit von der Gittergröße und der Lage des Quellknotens**

Je nach Bandbreitebedarf und Verkehrsmuster der zukünftigen Anwendungen sollte abgewogen werden, ob zusätzliche Modifikationen der 2D-Gittertopologie erforderlich sind oder zusätzliche optimierte Routingverfahren benötigt werden. Ein weiteres Kriterium, mit besonderer Bedeutung für nachrichtenbasierte Programmiermodelle, ist die Anzahl der benötigten Zyklen bzw. Hops, um von einem Knoten alle weiteren Knoten des Netzwerks zu kontaktieren. Handelt es sich hierbei stets um die gleiche Nachricht, so könnte ein *Multicast*-Mechanismus der Switch-Boxen zur Vervielfältigung der Nachricht herangezogen werden, so dass sich der Aufwand des Senderknotens auf ein Minimum beschränken lässt. Hierbei würde es seitens des Quellknotens ausreichen, eine *Multicast*-Nachricht an die lokale Switch-Box auszusenden, die ggf. die beabsichtigte Reichweite des *Multicasts* beinhaltet. Die maximale Latenz einer Nachricht, die den entferntesten Knoten innerhalb



eines Netzwerks erreicht, ergibt sich aus dem Durchmesser  $D$  der Topologie (vgl. Definitionen aus Abschnitt 2.3.1) und der Lage des Quellknotens.

Abbildung 4-29 a) veranschaulicht den linearen Zusammenhang dieses Sachverhalts für gitterförmige GigaNetIC-Systeme von bis zu zehn Knoten pro Kante. Bei einer Topologie mit  $10 \times 10$  Knoten benötigen die Eckknoten 18 Hops für eine Datenübertragung. Nach (4.9) bedeutet dies im besten Falle 65 Takte für die Übertragung eines Kommandoflits mit acht Datenbyte, 66 Takte für 16 Datenbyte etc. Handelt es sich hingegen um disparate Nachrichten, sozusagen eine vollständige Synchronisierung aller Prozessoren untereinander, so bedeutet dies, dass eine zur Gesamtzahl der Prozessoren bzw. Cluster proportionale Latenz für die Gesamtheit der Übertragung aller Nachrichten angesetzt werden kann. Die resultierende, durchschnittliche Hopanzahl für einen solchen Multicast für gegebene Gittergrößen zeigt Abbildung 4-29 b) auf. Für zweidimensionale Gitter ergibt sich diese zu  $\frac{2}{3} \cdot \sqrt{N}$ , mit  $N$  Knoten im Gitter.

Die Gesamtheit der zu bewältigenden Hops für derartige disparate Multicasts stellt Abbildung 4-29 c) dar. Für ein  $10 \times 10$ -Gitter werden 66.000 Hops für eine vollständige Punkt-zu-Punkt-Kommunikation aller Knoten benötigt. Zu beachten ist, dass diese Zahl nicht mit der für die Synchronisation benötigten Zeit gleichzusetzen ist, da ein Großteil der Übertragungen parallel abläuft. Unter der Voraussetzung, dass jeder Cluster nur eine Nachricht pro Zeiteinheit absetzen kann, ergibt sich hieraus für ein  $10 \times 10$ -Gitter eine Zeitdauer von mindestens 18 Hops bzw. 65 Takten im besten Fall, wenn das Netzwerk nicht aus- bzw. überlastet ist. Im schlechtesten Fall, wenn z. B. immer nur ein Knoten die Synchronisierung durchführte, um so Stauungen im Netzwerk vollständig zu vermeiden, ergäben sich  $100 \cdot 99 = 9.900$  Übertragungen. Dies resultierte in einer Synchronisationszeit von 9.914 Takten, wenn die Übertragungen zu den entfernteren Knoten zu Anfang gestartet würden. Bei einer Taktfrequenz von 714 MHz (vgl. Abschnitt 4.2.3) wäre eine vollständige Synchronisation von 100 Knoten bzw. maximal 800 Prozessoren in  $13,9 \mu\text{s}$  abgeschlossen. Dies ist allerdings eine äußerst konservative Abschätzung, die in der Realität zweifelsohne deutlich geringer ausfallen wird, da die parallelen Transfers in vielen Fällen blockadefrei ablaufen. Abbildung 4-29 d) zeigt die Hopanzahl, die sich ergibt, wenn ein Knoten disparate Nachrichten an alle anderen Knoten im Netzwerk schickt. Hierbei variiert die Zahl in Abhängigkeit von der Position im Gitter. Für zentrale Knoten beläuft sich der Wert für einen derartigen Multicast auf 500 akkumulierte Hops, wohingegen die Eckknoten mit 900 Hops 80 % mehr benötigen.

## 4.7 Skalierung des Systems durch Variation von Systemparametern

Da Flexibilität und Skalierbarkeit eine besonders wichtige Rolle beim GigaNetIC-System und beim GigaNoC-On-Chip-Netzwerk einnehmen, wurden alle relevanten Parameter der Hardwarebeschreibung generisch implementiert und in zentralen *Designpackages* gehalten, so dass durch geringsten Aufwand neue Varianten des On-Chip-Netzwerks bzw. des gesamten GigaNetIC-Systems erzeugt werden können. Abbildung 4-30 zeigt wesentliche Freiheitsgrade und die sich daraus ergebende Flexibilität der Architektur, die der Systemarchitekt bei der Realisierung eines GigaNetIC-Systems in der Entwurfsphase hat. Die wesentlichen Parameter, die das generische GigaNetIC-Design ausmachen, sind in Anhang B aufgelistet und werden dort kurz erläutert. Diese Parametrisierung ist ebenfalls für die im folgenden Kapitel vorgestellten Simulatoren und Rapid-Prototyping-Modelle konsequent im jeweiligen Rahmen berücksichtigt worden.

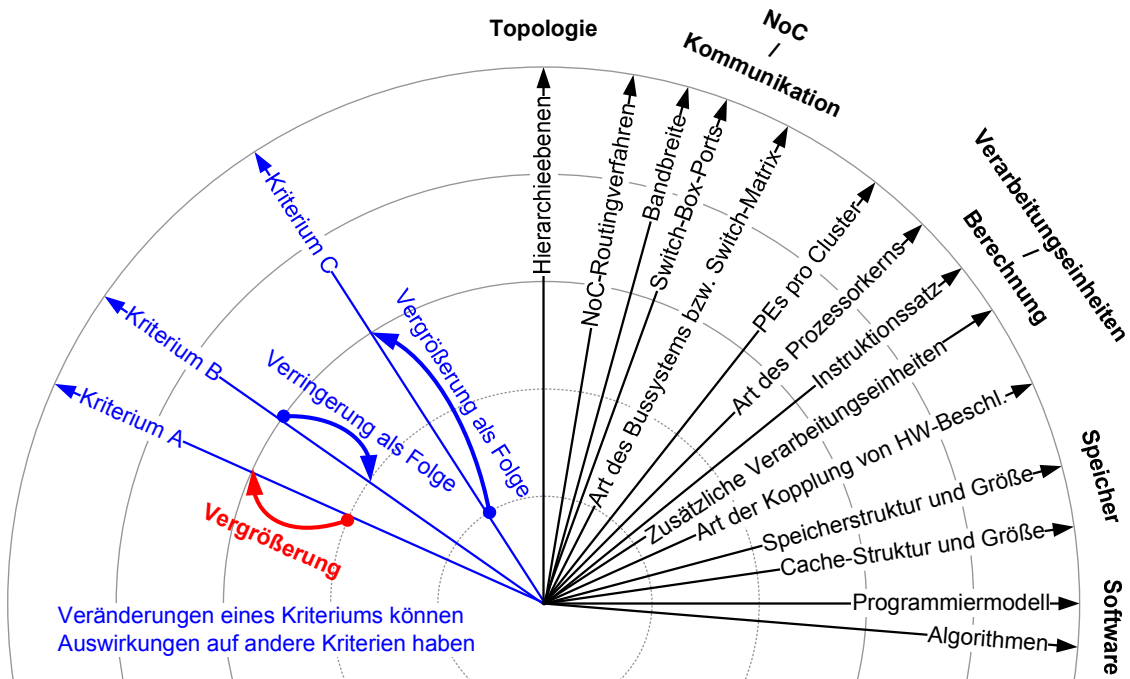


Abbildung 4-30: Freiheitsgrade bei der Skalierung der GigaNetIC-Systemarchitektur

Zukünftige Realisierungen, die auf Weiterentwicklungen der GigaNetIC-Architektur beruhen, werden zusätzlich über Mechanismen verfügen, die während der Laufzeit Veränderungen der Hardware erlauben. Hierzu zählen sowohl Adaption der Bandbreite und der aktiven Routingkanäle an die jeweiligen Erfordernisse, Maßnahmen zur aktiven Leistungsaufnahmereduktion wie z. B. *Clockgating* für nichtbenutzte Einheiten, als auch Möglichkeiten der Ausschöpfung von *Voltage-Scaling*-Techniken bei der Reduktion der Taktrate. Hardwarebeschleuniger könnten während der Laufzeit durch Integration rekonfigurierbarer Strukturen nach Bedarf geladen werden. Genauso wäre das Nachladen von speziellen Instruktionssätzen bzw. die Variation von CPU-Ressourcen wie z. B. der Registerbreite und der Anzahl auf Prozessorebene denkbar. Die GigaNetIC-Architektur ist grundsätzlich vorbereitet für diese neuen Möglichkeiten.

## 4.8 Zusammenfassung

In diesem Kapitel wurde eine neuartige, skalierbare Chip-Multiprozessor-Architektur vorgestellt, die aufgrund einer sehr flexibel gestalteten, parametrisierbaren Hardwarestruktur an verschiedenste Anforderungen angepasst werden kann, um so eine nach Definition 14 möglichst ressourceneffiziente Lösung zu erhalten. Alle Bestandteile, die einen CMP ausmachen, wurden eigens für diese Architektur implementiert bzw. angepasst. So wurde ein neuartiges, hierarchisches On-Chip-Netzwerk namens GigaNoC zusammen mit einem umfassenden Konzept zur Kopplung unterschiedlichster Verarbeitungseinheiten an den verschiedenen SoC-Ebenen entworfen. Bei der Speicherwahl kann zwischen normalem SRAM oder einem speziell entwickelten Multiprozessorcache gewählt werden. Durch die spezielle Konstruktion der On-Chip-Routingknoten der Switch-Boxen ist nicht nur eine gute Skalierbarkeit auf Chip-Ebene während des Entwurfs gegeben, sondern auch die Möglichkeit einer späteren Kombination von GigaNetIC-basierten CMPs auf Chip-Ebene, indem Switch-Box-Ports nach außen geführt und mit den Nachbarchips verbunden werden. Dies erlaubt auf Board-Ebene eine deutlich größere Anzahl von verbundenen Knoten, als es derzeit auf einem Chip möglich wäre.

Das in diesem Kapitel vorgestellte Hardwaremodell der GigaNetIC-Systemarchitektur wurde in der Hardwarebeschreibungssprache VHDL realisiert und umfasst über 161.000 Zeilen kommentierten VHDL-Code in insgesamt mehr als 470 Dateien.

Da das GigaNetIC-System als Basis für weitere Forschungsvorhaben der Universität Paderborn dient, wurden zur weiteren Nutzung und Erweiterung in den erfolgreich beantragten Folgeprojekten (wie z. B. PlaNetS<sup>24</sup>, MxMobile<sup>25</sup>, EasyC<sup>26</sup> oder auch DFG sowie in den DFG-Sonderforschungsbereichen SFB 376 „Massive Parallelität: Algorithmen, Entwurfsmethoden, Anwendungen“ und SFB 614 "Selbstoptimierende Systeme des Maschinenbaus") alle projektrelevanten Quellen und Dokumentationen in ein eigens angelegtes Versionsverwaltungssystem eingepflegt. Außerdem wurde eine GigaNetIC-Linux-Live-CD, auf der alle relevanten Daten enthalten sind, erstellt. Sie dient als eigenständige, bootfähige linuxbasierte Softwareplattform, die nahezu auf jedem PC-System dem Anwender eine vorkonfigurierte GigaNetIC-Entwicklungsumgebung zur Verfügung stellt.

Zusammen mit der GigaNetIC-Architektur entstand eine leistungsstarke Werkzeugkette, die von der Projektierung und Systemdimensionierung über Simulatoren auf verschiedener Abstraktionsebene bis hin zu einer hierarchisch gerichteten Optimierungsmethode zur Verbesserung der Systemeigenschaften für beliebige Anwendungen alle wesentlichen Komponenten beinhaltet. Diese Werkzeugkette und deren Einsatz sowie die Resultate werden in den Folgekapiteln 5 und 6 beschrieben.

Der Einsatz der GigaNetIC-Architektur und Leistungsbewertungen für dedizierte Anwendungen finden ausführliche Behandlung in im Kapitel 7. Die prototypische Realisierung auf FPGA- und Standardzellentechnologien bildet mit Kapitel 8 den Abschluss der Vorstellung dieser massiv parallelen, skalierbaren, ressourceneffizienten Chip-Multiprozessor-Architektur.

---

<sup>24</sup> Das vom Bundesministerium für Bildung und Forschung (BMBF) finanzierte Projekt "NGN-Platforms for Networked Services" (NGN-PlaNetS) soll dazu beitragen, dass alle europäischen Bürger einen leistungsfähigen Breitband-Internet-Zugang bekommen.

<sup>25</sup> Das BMBF-Projekt MxMobile ist eine Kooperation zwischen verschiedenen Industrieunternehmen und Universitäten und erforscht, entwickelt und demonstriert Schlüsselkomponenten von programmierbaren Plattformen für den Multi-band-Multistandard-Betrieb von Terminals und Basisstationen. Die Arbeiten der Universität Paderborn ordnen sich in den Teilbereich „Modellierung und Verifikation der Systemarchitektur“ ein.

<sup>26</sup> Das BMBF charakterisiert das Projekt wie folgt: „Ziel der Forschungsaktivitäten im EASY-C-Verbundvorhaben ist es, Schlüsseltechnologien für die nächste Generation von zellularen Mobilfunknetzen voranzutreiben, um die Entwicklung von neuen Echtzeit-Applikationen mit hohen Datenraten zu unterstützen. Diese Applikationen wie z. B. Video-streaming und Lokalisierungsdienste stellen enorme Anforderungen an die gesamte Netzinfrastruktur hinsichtlich Bandbreite, Latenz, spektraler Effizienz und Fairness gegenüber allen mobilen Teilnehmern, insbesondere an den Zellrändern.“ (<http://www.pt-it.pt-dlr.de/de/1772.php>, Stand: November 2007)



## 5 Analyse und funktionale Verifikation des Chip-Multiprozessorsystems

Den Erfolg einer Prozessor- bzw. Multiprozessorarchitektur bestimmt nicht nur die reine Hardware, sondern ebenfalls zu einem großen Teil die begleitende Werkzeugkette. Schon per Definition (vgl. Definition 3) versteht sich die GigaNetIC-Architektur als mehr als nur das im vorigen Kapitel vorgestellte Hardware-System. Ohne einen leistungsfähigen Compiler, der aus einer Hochsprache, wie z. B. C oder C++, effizienten Maschinencode erzeugt, stünde die ansonsten sehr umständliche und zeitaufwändige Softwareentwicklung dem Erfolg des Chip-Multiprozessors stark im Wege. Eine Wiederverwendbarkeit des Codes für Weiterentwicklungen der Prozessorkerne oder gar andere Prozessoren wäre nahezu ausgeschlossen. Zudem sollte eine leistungsfähige Werkzeugkette möglichst flexibel gehalten sein und auf Modifikationen und Erweiterungen der Hardware schnell reagieren können. Weiterhin ist eine Reihe auf unterschiedliche Anforderungen angepasste Simulatoren wünschenswert, die eine schnelle respektive sehr akkurate Simulation des Gesamtsystems oder einzelner Systementitäten ermöglichen. Diese Simulationen dienen im Vorfeld der ASIC-Realisierung eines GigaNetIC-Chip-Multiprozessors zum einen zur Verifikation der Funktion, aber auch zum anderen der Evaluierung der Leistungsfähigkeit und zur Parameterfindung (vgl. Abschnitt 4.7), um ein möglichst ressourceneffizientes System zu verwirklichen.

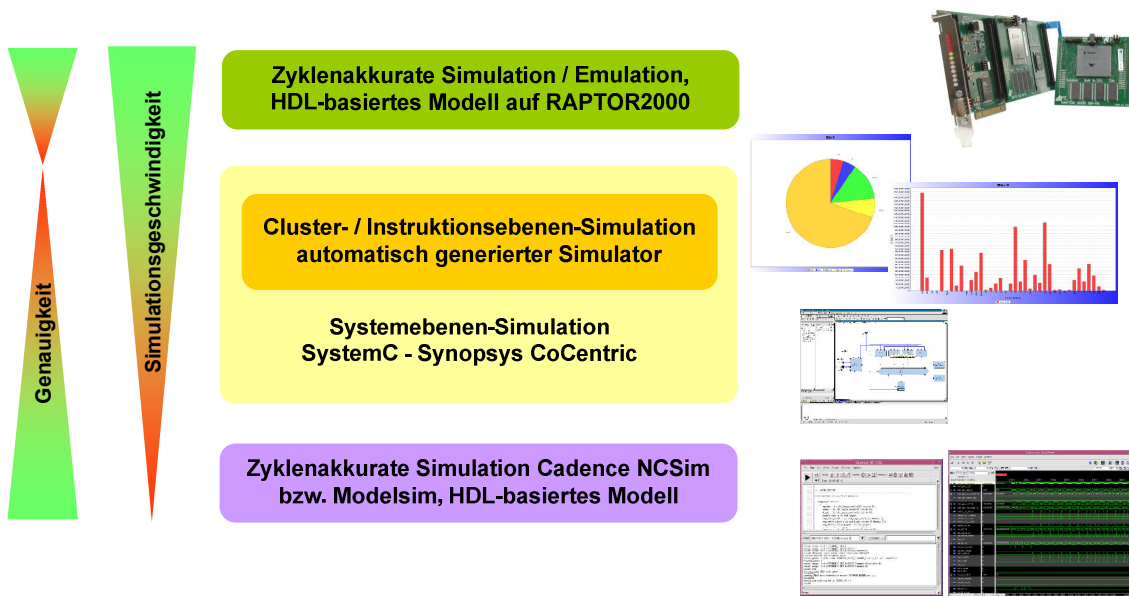



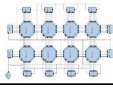
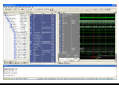
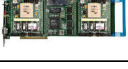
Abbildung 5-1: Möglichkeiten der Systemsimulation und - Verifikation auf unterschiedlichen Abstraktionsebenen

Zusammen mit den Fachgebieten Programmiersprachen und Übersetzer, Prof. Dr. Uwe Kastens sowie Algorithmen und Komplexität, Prof. Dr. math. Friedhelm Meyer auf der Heide, wurde eine mehrschichtige, auf verschiedenen Abstraktionsebenen greifende Werkzeugkette entworfen [6][116][117]. Abbildung 5-1 zeigt im Überblick die unterschiedlichen Abstraktionsebenen der Systemsimulation bzw. -emulation auf. Jede Abstraktionsebene hat ihre Berechtigung und Vorteile gegenüber den anderen. Zu diesen Kriterien zählen u. a. die Simulationsgeschwindigkeit  $v_{sim}$ , gemessen in benötigter Zeit pro simuliertem Takt des Zielsystems

$$v_{sim} \left[ \frac{\text{Takte}}{s} \right] = \frac{T_{\text{simuliertes System}}}{t_{\text{simulierendes System}}}, \text{ die Ge-}$$

naugigkeit bzw. Detailliertheit der Simulation, oder auch der Aufwand, den Änderungen im Simulator bedeuten. Tabelle 5-1 gibt Aufschluss über die einzelnen Charakteristika des jeweiligen Simulations- / Emulationswerkzeugs.

**Tabelle 5-1: Merkmale der einzelnen Simulations- bzw. Emulationsmöglichkeiten**

	<b>C-Simulation</b>	<b>SystemC-Simulation</b>	<b>VHDL-Simulation</b>	<b>FPGA-Emulation</b>
				
<b>Simulationsgeschwindigkeit</b>	~10 MHz	~ 100 kHz	~ 100 Hz	~ 20 MHz
<b>Detaillierungsgrad</b>	relativ gering, taktgenau auf Prozessorebene	mittel	hoch	hoch
<b>Architekturelle Einschränkungen der Modelle</b>	einfaches Busmodell	nur Unified-Multiprozessor-Cache	keine	keine
<b>Modifizierbarkeit</b>	sehr schnell	sehr schnell	weniger schnell	zeitaufwändig
<b>Simulationsumfang</b>	parametrisierbar, auf Clusterebene	parametrisierbar, auf SoC-Ebene	parametrisierbar, auf SoC-Ebene	parametrisierbar, auf SoC-Ebene (abhängig von der FPGA-Größe)

Die GigaNetIC-Architektur stellt einen automatisch generierten, C-basierten Simulator für die Cluster-Ebene zur Verfügung [6], auf dem aufbauend der SystemC-Simulator *SiMPLE* (*SystemC integrated Multiprocessor-Level Environment*) [117] von uns entworfen wurde, der zur Simulation des gesamten Chip-Multiprozessors herangezogen werden kann. Zur genauen Simulation des Gesamtsystems oder einzelner Hardwareentitäten mit höchstem Detaillierungsgrad, wird die zumeist sehr zeitaufwändige HDL-basierte Simulation unter Verwendung modernster Simulationswerkzeuge eingesetzt [6][116]. Letztendlich steht mit dem am Fachgebiet Schaltungstechnik entwickelten Rapid-Prototyping-System RAPTOR2000 [139][9] eine umfangreiche FPGA-basierte Entwicklungsumgebung zur Verfügung, die in Forschung und Lehre zugleich eingesetzt wird [140]. Das RAPTOR2000-System konnte bereits für andere ASIC-Implementierungen als Evaluierungsplattform erfolgreich genutzt werden [10]. Mit ihrer Hilfe lassen sich komplexe Chipentwürfe im Zusammenspiel mit realer Hardware testen<sup>27</sup>.

Die grundsätzliche Vorgehensweise während der Entwicklung und Optimierung eines GigaNetIC-Systementwurfs unter Zuhilfenahme der erstellten Entwicklungsumgebungen sieht in der Regel wie folgt aus:

Zunächst werden grundlegende Veränderungen in den leicht zu modifizierenden, schnellen Simulatoren implementiert. Hierzu zählt der Cluster-Simulator (vgl. Abschnitt 5.1) auf Prozessor- und Cluster-Ebene, während die SystemC-basierte Simulationsumgebung *SiMPLE* (vgl. Abschnitt 5.2) auf allen drei Ebenen, also auch auf der SoC-Ebene greift. Diese Simulationen können zum Teil sogar automatisiert durch die von mir erstellte MultiSim-Werkzeugkette ablaufen und ausgewertet werden, was einhergehend mit der relativ hohen Simulationsgeschwindigkeit eine hohe Abdeckung des Entwurfsraums gewährleistet. Die auf dieser abstraktesten Ebene gewonnenen Ergebnisse geben Impulse für die Gestaltung bzw. Parametrisierung der einzelnen Hardwareentitäten. Die hier

<sup>27</sup> Vgl. mit dem GigaNetIC-Demonstrator der Cebit 2005 und der Hannover-Messe 2005, Abschnitt 8.1.

anwendbaren Maßnahmen, z. B. die Optimierung der Algorithmen und die Konzeption der Struktur, besitzen in der Regel das größte Potential zur Verbesserung der Ressourceneffizienz.

Im Anschluss werden besonders vielversprechende Varianten des Multiprozessorsystems in einer synthetisierbaren Hardwarebeschreibung realisiert und mit Hilfe der weiterentwickelten PERFMON-Umgebung (vgl. Abschnitt 5.3) detailliert untersucht. Die hier gewonnenen Messergebnisse beinhalten nicht nur taktgenaue Laufzeiten sondern genaue Informationen über die zu erwartende Leistungsaufnahme und Fläche der jeweiligen Realisierung. Aufgrund der sehr geringen Simulationsgeschwindigkeit ist die Möglichkeit der Vorauswahl durch die zuvor genannten Simulatoren mehr als ratsam. Durch das in Abschnitt 5.4 beschriebene MultiSim-Entwurfswerkzeug können dann, als Folgeschritt, Modifikationen in enger abgesteckten Grenzen bzw. für einzelne, vielversprechende Punkte des Entwurfsraums erfolgen. Für die Optimierung des Systems greift dann die im folgenden Kapitel 6 vorgestellte Methodik der „hierarchisch gerichteten Optimierung“.

Potentielle Realisierungen werden dann im Anschluss auf dem speziell erweiterten, FPGA-basierten Rapid-Prototyping-System RAPTOR2000 (vgl. Abschnitt 5.5) im direkten Hardwareumfeld getestet. Diese Integration in ein „reales“ Anwendungsszenario ermöglicht eine chipübergreifende funktionale Verifikation des Systems und lässt erste Aussagen über die Leistungsfähigkeit der zukünftigen ASIC-Realisierung zu. Die durch Simulation erfolgte Verifikation der erstellten Hardware kann hier durch Hardwareemulation bestätigt werden. Durch die hohe Simulationsgeschwindigkeit sind zudem weitaus komplexere Tests möglich, als es die HDL-Simulationsumgebung zuließe.

Die vorgestellten Entwicklungsumgebungen bzw. Simulatoren lassen sich darüber hinaus auch unabhängig voneinander nutzen. So kann Software zunächst entkoppelt vom langsamen Hardware-simulator entwickelt, getestet sowie funktional optimiert und deren Laufzeit analysiert werden. Hardwareblöcke hingegen können z. B. aus verfügbaren IP-Blöcken integriert werden und zunächst in der HDL-basierten Entwicklungsumgebung evaluiert werden, ohne für alle Simulatoren funktional realisiert werden zu müssen. Eine einheitliche Übersetzer-Werkzeugkette komplettiert schließlich die GigaNetIC-Architektur und ermöglicht dem Entwickler eine unkomplizierte Nutzung aller Simulationswerkzeuge und Emulationsplattformen, vgl. Abschnitt 5.6.

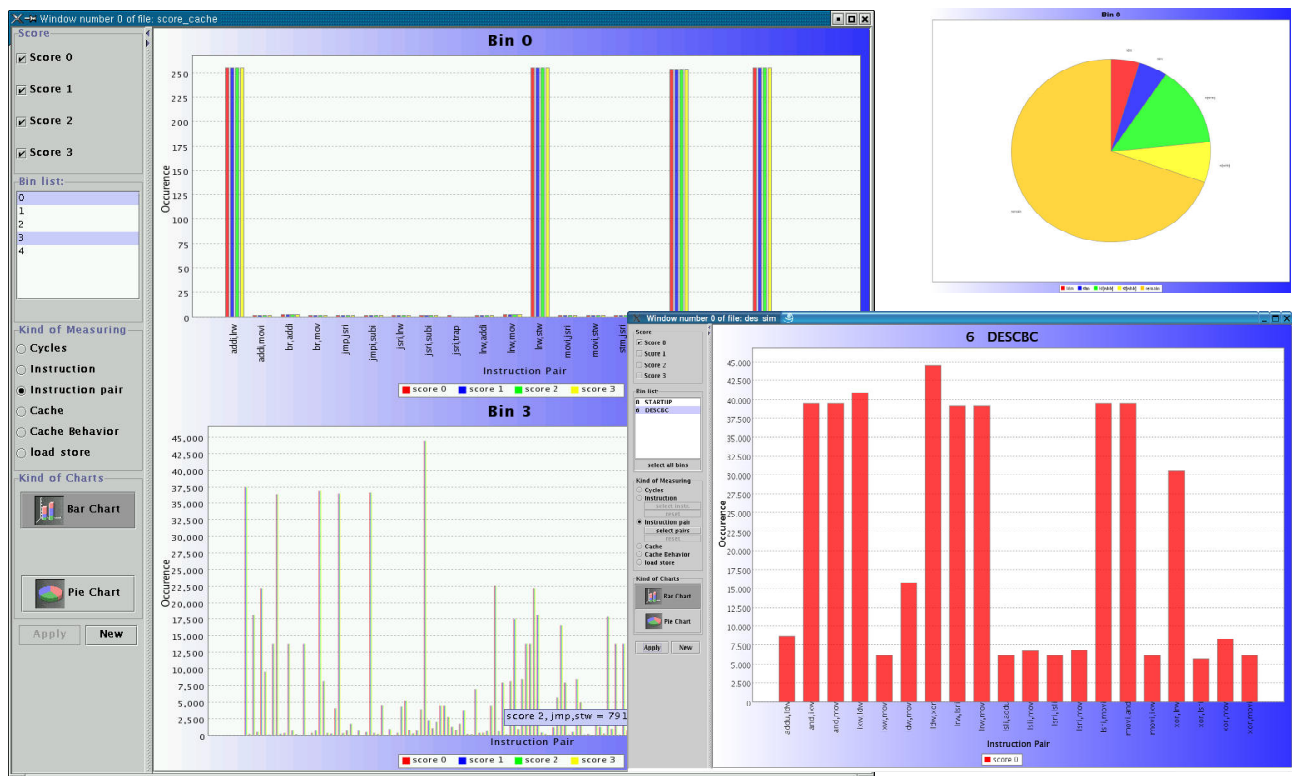
Detaillierte Beschreibungen der einzelnen Simulations- bzw. Emulationswerkzeuge und deren Einsatz werden in den folgenden Abschnitten gegeben.

## 5.1 C-basierter Cluster-Simulator zur Simulation und Profilierung

Ausgangspunkt der Simulatoren ist ein aus der Compiler-Werkzeugkette generierter, C-basierter Simulator, der vom Fachgebiet Programmiersprachen und Übersetzer, Prof. Dr. Uwe Kastens, entwickelt worden ist. Dieser zyklenakurate Simulator ist ein vergleichsweise schneller Simulator, der zur Simulation der Clusterebene des GigaNetIC-Systems (vgl. Abschnitt 4.1) eingesetzt wird. Er ermöglicht die taktgenaue Verhaltenssimulation eines N-Cores oder aber auch des gesamten N-Core-Clusters unter Verwendung eines einfachen Busmodells, das keine Matrixfähigkeit aufweist, und über einen *Round-Robin*-Mechanismus, ähnlich dem Wishbone-Bus (vgl. Abschnitt 4.2.4) die Busvergabe regelt. Eine Besonderheit des Simulators ist, dass er nicht manuell erstellt werden muss, sondern durch die entwickelte Werkzeugkette automatisch generiert wird [6], vgl. auch Abbildung 6-4. Mit ca. 300 Host-CPU-Takten pro simuliertem Takt des GigaNetIC-Clusters erreicht dieser Simulator eine beachtliche Simulationsgeschwindigkeit. Z. B. benötigt ein Intel Pentium 4m



(2,2 GHz) 314 Takte für einen simulierten N-Core-Takt, was einer effektiven Simulationsschwindigkeit von 7,63 MHz entspricht. Eine Simulation des C-basierten Simulators von einer Sekunde würde somit ca. 14 Stunden der hardwarenahen VHDL-Simulation mit einer effektiven Simulationsschwindigkeit von 150 Hz bedeuten. Dieser Sachverhalt zeigt sehr deutlich, dass zum einen abgewogen werden sollte, welcher Simulator zu welchem Zweck verwendet wird, und zum anderen auch die Notwendigkeit unterschiedlicher Simulationswerkzeuge bzw. -methoden. So bieten sich zur ersten Leistungsbewertung und Entwurfsraumexploration die schnellen Simulatoren an, die in der Lage sind, rechenaufwändige und damit zeitintensive Benchmarks bzw. Algorithmen in relativ kurzer Zeit abzuarbeiten. Sie liefern wichtige Ergebnisse zur Einstufung der Leistungsfähigkeit der zu evaluierenden Architektur mit genügender Genauigkeit. Im Anschluss kommen dann für detailliertere Analysen ggf. auch nur für Teilprobleme, die genauen, zeitaufwändigen Simulationsumgebungen zum Einsatz. Abschließend kann dann der Prototyp als FPGA-Realisierung im realen Hardwarezenario die Funktionalität des Systems unter Beweis stellen.



**Abbildung 5-2: Exemplarische Performanzvisualisierung mit JScore  
der mit dem Cluster-Simulator ermittelten Laufzeitdaten**

Komplettiert wird die clusterorientierte Entwicklungsumgebung durch ein leistungsfähiges Profilierungswerkzeug, *JScore*, das das *Profiling* auf Task- und Funktionsebene des Clusters ermöglicht [6]. So können zyklenakkurate Simulationswerte aufgrund der detaillierten Profilingmöglichkeit komfortabel zur Ergebnisvisualisierung der gewonnenen Laufzeitdaten herangezogen werden. Abbildung 5-2 zeigt beispielhaft die Ergebnisvisualisierung von Laufzeitdaten, die mit dem Cluster-Simulator gewonnen wurden. Die Entwicklungsumgebung unterstützt die Instrumentarisierung des Programmcodes, dadurch ist es möglich, Abschnitte einzuteilen und deren Profilingdaten separat voneinander, in so genannten *Bins* (also Körben), für die beteiligten Prozessoren des Clusters protokollieren zu lassen. Es werden die benötigten Takte, die Anzahl der jeweils ausgeführten Instruktionen und der datenabhängigen Instruktionspaare je Bin ausgewertet. Außerdem werden Lade- und Speicheroperationen protokolliert.



Der Cluster-Simulator unterstützt darüber hinaus eine einfache Einbindung von Hardwarebeschleunigern mit Annotation der durch HDL-Simulation gewonnenen exakten Laufzeitdaten. Eine in C beschriebene Funktion muss lediglich als die Funktionalität gekennzeichnet werden, die später im realen System von einem Hardwarebeschleuniger ausgeführt wird. Die Laufzeitdaten des realen Hardwarebeschleunigers werden dem Werkzeug übergeben. Während der Simulation führt ein Prozessor den Code aus, die Laufzeitdaten werden jedoch durch die zuvor annotierten ersetzt. Der Vorteil dieser Möglichkeit ist, zunächst die Funktionalität durch schnell realisierbare Software bereitzustellen, um dann potentielle Funktionsblöcke später durch Hardwarebeschleuniger ersetzen zu können. Die neuen, beschleunigten Laufzeiten können dann leicht mit in die Performanzbewertung durch den *Profiler* einbezogen werden.

Für die Cluster-Simulationswerkzeugkette wurde von mir eine skriptbasierte Evaluationsumgebung geschaffen, mit deren Hilfe eine schnelle Analyse des Entwurfsraums in Bezug auf die Softwareimplementierung und ggf. vorhandener Hardwarebeschleuniger möglich wird. Unter Verwendung dieser Erweiterung konnten umfangreiche Simulationen eines DSLAM-Benchmarks (vgl. Kapitel 7) in sehr kurzer Zeit durchgeführt werden, die eine Exploration des Entwurfsraums in zumutbarer Zeit erlaubten [141][118]. Hierbei galt es, sieben unterschiedliche Benchmarkszenarien mit je ca. 20 relevanten Tasks zu untersuchen. Es wurden nach dem iMix (vgl. Abschnitt 7.2.3) vier verschiedene Pakettypen verarbeitet. Beim Simulieren wurden über elf verschiedene Instruktionssatzvarianten des N-Cores (vgl. Abschnitt 6.2) iteriert, so dass sich letztendlich insgesamt 308 Benchmarkkonstellationen ergaben. Jeder dieser Benchmarks benötigte ca. 26 Mio. N-Core-Taktzyklen, was in einer Gesamtzahl von 8,2 Milliarden simulierten Takten resultierte. Die gesamte Verarbeitung inklusive Kompilierung und Auswertung benötigte auf dem P4m-System mit 2,2 GHz nur ca. 45 Minuten Laufzeit. Abbildung 5-3 zeigt einen exemplarischen Auszug aus einem der automatisch erzeugten Ergebnisprotokolle.

```

3049600 ldwixw
3049600 ldwxorls18
3049600 xorldw_andshr_ixwandshr_orsh181624_ldwxorls18_ldwixw
3049600 xorldw_andshr_ixwandshr_orsh181624_ldwxorls18_ldwixw_ldwaddi
3049600 xorldw_ldwixw_ldwaddi
3201000 xorldw
3352400 andshr
3352400 ixwandshr
3352400 ldwaddi
3352400 none
3352400 orsh181624
1,09929 Gain Min 3049600 Max 3352400 up-BM1-CheckCRC.ext.txt.sorted

```

**Abbildung 5-3: Auszug aus einem der automatisch erzeugten Ergebnisprotokolle zur DSLAM-Benchmarkauswertung zu verschiedenen Instruktionssatzvarianten**

Die Zahlen geben die jeweils benötigten Taktzyklen für den entsprechenden Benchmark und den instrumentierten Codeabschnitt an. Die Kürzel stehen für den simulierten Instruktionssatz. Zusammenfassend wird in der letzten Zeile der maximale Gewinn (*Gain*) durch eine Instruktionssatzerweiterung notiert. In dem aufgeführten Beispiel ergibt sich eine maximale Beschleunigung um 9,92% durch die *ldwixw*-Befehlssatzvariante gegenüber dem unmodifizierten N-Core. Nach ausgiebigen Simulationen kann so dennoch relativ schnell entschieden werden, welche Hardwarevariante die besten Performanzsteigerungen erzielt. In einer weiteren Ausbaustufe dieser Werkzeugkette können zusätzlich die in der hardwarebasierten Werkzeugkette (vgl. Kapitel 6) ermittelten Daten für Fläche und Leistungsaufnahme der entsprechenden Hardwareblöcke mit berücksichtigt und ausgewertet werden.

## 5.2 Modellierung des GigaNetIC-Chip-Multiprozessors in SystemC

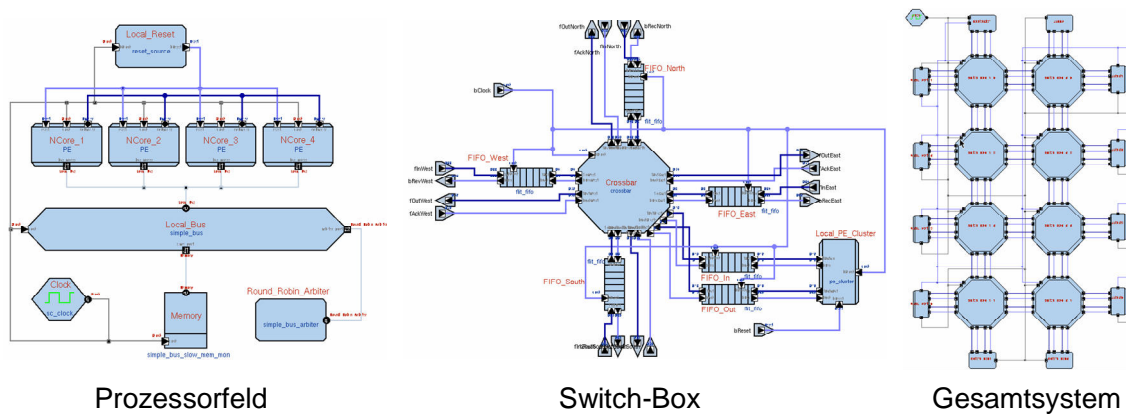
Um detaillierte Aussagen über die Leistungsfähigkeit der gesamten GigaNetIC-Multiprozessor-Architektur treffen zu können, die über die Ergebnisse zur Leistungsfähigkeit der einzelnen Prozessorkerne und Hardwarebeschleuniger hinausgehen, wurde ein zyklenakkurates Modell unseres Systems namens *SiMPLE* in SystemC modelliert. Dieses entspricht in seinem grundsätzlichen Aufbau der in Abbildung 4-2 gezeigten Struktur. Zunächst wurde ein Prozessor basierend auf dem automatisch generierten Cluster-Simulationsmodell (Abschnitt 5.1) in eine in SystemC modellierte Struktur eingebunden. Darauf aufbauend wurde ein Prozessorcluster mit vier N-Core-CPU's, Multiprozessor-Cache (vgl. Abschnitt 4.4.2) und lokalem Speicher erstellt, wobei die Kommunikation über einen *Round-Robin*-arbitrierten Bus abgewickelt wird. Letztendlich werden diese Prozessorfelder über Switch-Boxen, die unter anderem über parallele FIFO-Strukturen, Kontrolllogik zur Ansteuerung des lokalen Prozessorclusters, Routingmechanismen zur Weiterleitung der Pakete über das On-Chip-Netzwerk und einen Kreuzschienenverteiler zur Verbindung der Ein- und Ausgangsports verfügen, verbunden [130].

Die Simulation ist für viele Bereiche des Systems zyklenakkurat und weicht nur in einigen unwesentlichen Teilen geringfügig von der Hardware ab. Sie ist zudem deutlich schneller als eine vergleichbare Simulation auf Basis der VHDL-Beschreibung der Einzelkomponenten. Falls erforderlich, können die einzelnen Hardwarebeschleuniger zur Verifikation des Zusammenspiels mit der Software in einer Co-Simulation als VHDL-Beschreibung mit eingebunden werden. Um eine höhere Simulationsgeschwindigkeit zu erreichen, werden diese ansonsten durch funktionale Beschreibungen in C++ bzw. SystemC ersetzt. Funktional verhält sich *SiMPLE* weitestgehend identisch zur Hardware, so dass auch hier des Postulats eines hohen Übereinstimmungsgrads der unterschiedlichen Simulationswerkzeuge Folge geleistet wird.

Das System ist so konstruiert, dass beim Start zunächst die Instruktionsspeicher der einzelnen CPUs mit den zuvor durch den Compiler erstellten Binär-Dateien geladen werden, wobei diese Daten bereits über das GigaNoC-On-Chip-Netzwerk von einem zentralen *Bootloader* (vgl. Abschnitt 4.2.2) aus an die entsprechenden Speicheradressen gesendet werden. Im Anschluss wird das umgebende System aktiviert, also z. B. Paketgeneratoren, die die eigentlichen Nutzdaten an die Eingangsports des Systems liefern. Dieser Prozess kann bei Bedarf beschleunigt werden, indem die Instruktionsspeicher direkt, unter Umgehung des normalen Bootvorgangs, bei Beginn der Simulation mit den entsprechenden Daten initialisiert werden.

Zunächst wurde *SiMPLE* in der Synopsys-CoCentric-Entwicklungsumgebung für SystemC-Modelle realisiert (vgl. Abbildung 5-4). Die CoCentric-Entwicklungsumgebung stellt eine Vielzahl von Bibliothekselementen zur Verfügung, so z. B. auch ADSL-Kanäle, die zur Modellierung der Übertragungsstrecke der Endkunden hin zum DSLAM (vgl. Abschnitt 7.1) dienen und in einigen Analysen verwendet wurden. Mittlerweile ist *SiMPLE* auch als alleinstehende Anwendung einsetzbar, so dass eine Lizenz für die Werkzeuge von Synopsys nicht zwingend erforderlich ist.

Mit Hilfe der *SiMPLE*-Systemmodellierung können beliebige Anwendungsszenarien detailliert und schnell evaluiert werden. Hard- und Software für das gesamte System lassen sich bereits im Vorfeld, vor der ASIC-Implementierung testen und optimieren.



**Abbildung 5-4: SystemC-Modellierungsebenen,  
eingebunden in die Simulationsumgebung CoCentric von Synopsys**

Auch für die SiMPLE-Simulationsumgebung wurde die bereits für den Cluster-Simulator (vgl. Abschnitt 5.1) entwickelte skriptbasierte Steuersoftware zur Automatisierung der Entwurfsraumexploration weiterentwickelt. Mit ihrer Hilfe ist der Software- bzw. Hardwareentwickler in der Lage automatisch gesteuerte Simulationen über nahezu alle zur Verfügung stehenden Parameter der Hardware zu iterieren (vgl. Anhang B). So können Optimierungspotentiale seitens der Soft- und Hardware im frühen Entwicklungsstadium erkannt und in den Prototypen integriert werden. Fehler der Software oder funktionales Fehlverhalten der Hardwareabstraktion können so ohne großen Kostenaufwand beseitigt werden. Alle während der Simulationen gewonnenen Messergebnisse werden protokolliert und abschließend zur weiteren Analyse und Visualisierung für Tabellenkalkulationsprogramme als CSV<sup>28</sup>-Dateien aufbereitet. BONA ET AL. [142] verwenden ein erweitertes SystemC-Simulationsmodell zur schnellen Bestimmung der Verlustleistungsaufnahme eines Multiprozessor-systems und im Speziellen des eingesetzten On-Chip-Netzwerks. Bei diesem Simulationsmodell beträgt die Diskrepanz zwischen den ermittelten Werten der SystemC-Simulation und den Werten der *Gate-Level*-Synthese nach Aussage der Autoren maximal 9 %. Dies ist für erste Abschätzungen hinreichend genau und erlaubt so eine schnelle Entwurfsraumexploration, was auch für die SiMPLE-Entwicklungsumgebung eine lohnenswerte Erweiterung wäre.

### 5.3 HDL-basierte Simulation auf Register-Transfer-Ebene

Die HDL-basierte Simulation auf Register-Transfer-Level (*RTL*) stellt die genaueste und detaillierteste Simulationsumgebung des GigaNetIC-Systems dar. Sie ermöglicht zwar in Abhängigkeit von der Größe des zu analysierenden Chip-Multiprozessors nur eine sehr geringe Simulationsgeschwindigkeit (vgl. Tabelle 5-1), bietet aber dafür Beobachtbarkeit und Steuerbarkeit auf nahezu allen Ebenen des Hardwaresystems und ist für die Verifikation der Hardware unverzichtbar. Die von uns realisierte Umgebung erlaubt neben der *RTL*-Simulation außerdem *Post-Synthese*-Simulationen auf Gatterebene der verwendeten Standardzellentechnologie. Diese liefert mit Hilfe von Schaltaktivitätenannotationen während der Ausführung des Anwendungsprogramms sehr genaue Daten zur Leistungsaufnahme der Schaltungskomponenten.

<sup>28</sup> Das *CSV(Comma Separated Values)*-Format ist ein gebräuchliches Format für Datendateien.

Zur Bestimmung von Ausführungszeiten, Chipfläche und Energiebedarf von eingebetteten Systemen existiert bereits eine Vielzahl von Methoden. Allerdings analysieren viele der etablierten Werkzeugketten nur ein oder zwei der genannten Charakteristika. Einige experimentelle Arbeiten beschäftigen sich mit abstrakteren Modellen der Software-Energiebedarfsanalyse, z. B. auf Instruktionsebene [143][144][145], für Softwarefunktionen [146][147][148][149] und ebenso für funktionale Einheiten wie Controller und Prozessoren [150][151]. Viele der Verfahren bestimmen für die Ausführungszeit nur eine obere Schranke, die *Worst Case Execution Time (WCET)*, ohne mögliche Daten-, Pfad- und Zustandsabhängigkeiten zu berücksichtigen. Die im Folgenden vorgestellte Werkzeugkette ermöglicht weitaus umfangreichere Analysen und hilft dem Entwickler bei der Findung von Flaschenhälsen und zeigt Optimierungspotentiale auf.

Für die Umsetzung und Analyse der GigaNetIC-Hardwarebeschreibung wurde die PERFMON-Werkzeugkette [116] weiterentwickelt und an die Bedürfnisse eines Multiprozessorsystems angepasst. Diese Weiterentwicklung unterstützt den Hardwareentwickler bei der Bewertung der Ressourceneffizienz unterschiedlicher Architekturvarianten in mehrfacher Hinsicht.

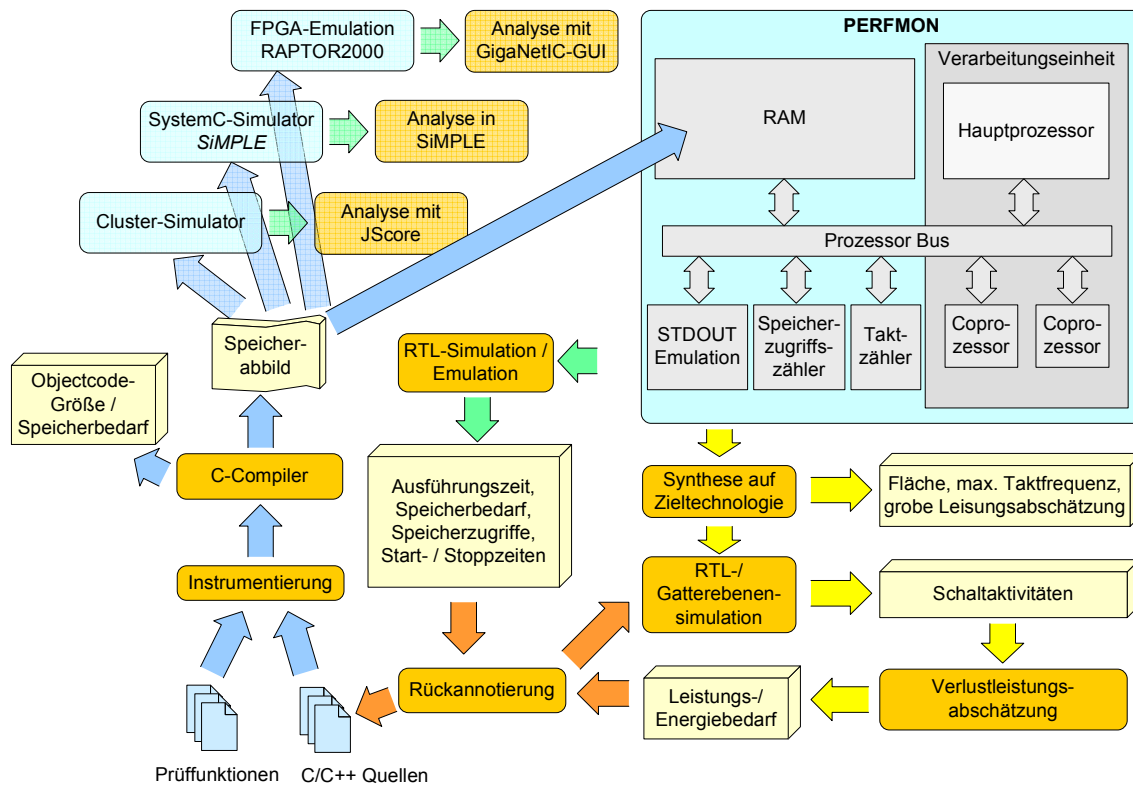


Abbildung 5-5: Performanz-Evaluierungsumgebung PERFMON, basierend auf HDL-Simulation

Die PERFMON-Umgebung (vgl. Abbildung 5-5) verwendet zunächst die in der Hardwarebeschreibungssprache VHDL erstellte Modellierung des Prozessors und des in Abbildung 4-17 gezeigten Prozessorsubsystems. Je nach Art und Umfang der Analysen beschränkt sich das verwendete Hardwaremodell aber nicht nur auf Prozessorebene, sondern wird bei Bedarf auf das gesamte zu charakterisierende GigaNetIC-Multiprozessorsystem ausgeweitet. Die dreidimensional dargestellten Elemente in der Grafik markieren die jeweils ermittelten Kennwerte des zu analysierenden Systems.

Als Eingang für die PERFMON-Umgebung dient die Anwendungssoftware in Form von Code in der Hochsprache<sup>29</sup> C oder C++. Diese wird durch spezielle Instrumentierungen annotiert (ähnlich der Vorgehensweise beim Cluster-Simulator, Abschnitt 5.1), um so die für die Analyse relevanten Codesegmente zu kapseln. Zusammen mit den von der Werkzeugkette benötigten Prüffunktionen werden die Quellen kompiliert und ein entsprechendes Speicherabbild erstellt. Je nach weiterer Verwendung wird es dem entsprechenden Simulator bzw. Emulator gerecht aufbereitet. Bei der Verwendung von PERFMON wird es in ein SRAM-Simulationsmodell des lokalen Prozessorspeichers geladen. Die Simulation im HDL-Simulator liefert dann ein umfangreiches Profil mit hohem Detaillierungsgrad im Hinblick auf die dynamischen Laufzeiten, Codegrößen, Speicherzugriffe und die Auslastung des Stapelspeichers (*Stacks*) der analysierten Anwendung. Bei Algorithmen, deren Laufzeit datenabhängig ist, werden zudem Minimum, Maximum und Mittelwert der instrumentierten Codesegmente ausgewertet (vgl. Abbildung 5-6).

Während der Simulation werden die Schaltaktivitäten zunächst auf *RT(Register Transfer)*-Ebene für die zu analysierenden Hardwarekomponenten protokolliert und abgespeichert. Diese werden im anschließenden, ebenfalls automatisierten Schritt, zu einer weiteren Abschätzung der dynamischen Verlustleistungsaufnahme herangezogen. Diese Abschätzung ist in nahezu allen Fällen weitaus genauer, als die rein statistische Abschätzung der Leistungsaufnahme seitens des Synthesewerkzeugs. Die hier gewonnenen Resultate bedeuten einen relativ guten Kompromiss zwischen Rechenaufwand und Genauigkeit. In der frühen Phase eines Designentwurfs sind für die Konzipierung des SoCs eher die qualitativen Zusammenhänge als quantitative entscheidend. Die RTL-basierte Analyse zeigt Trends auf, die in etwa in der zu erwartenden Größenordnung der Verlustleistungsaufnahme des späteren Chipdesigns liegen.

Der nächste Schritt ist die Synthese des RTL-Designs mit dem Synopsys Design Compiler, dessen Resultat, die Gatternetzliste der Zieltechnologie, als Eingabe für die nächstgenauere Verlustleistungsberechnung dient die Gatterebenessimulation ohne Berücksichtigung von Verzögerungszeiten. Anhand der protokollierten Schaltaktivitäten und der Charakterisierungsinformationen der Bibliothekselemente der verwendeten Standardzellentechnologie kann der *Synopsys Power Compiler* bereits eine deutlich genauere Abschätzung der zu erwartenden Leistungsaufnahme treffen. Außerdem ergeben sich erste Werte für die zu erwartenden Kenngrößen Fläche und Taktfrequenz der Hardwarekomponenten. Die Bestimmung der maximal möglichen Taktfrequenz der betreffenden Hardwarekomponenten übernehmen Funktionsaufrufe einer skriptbasierten Werkzeugkette der erweiterten GigaNetIC-PERFMON-Umgebung. Diese steuert und analysiert die Resultate der entsprechenden *Synopsys*-Werkzeuge. Hierdurch ist es möglich, ohne Interaktion sehr gute Syntheseresultate zu erzielen, die allerdings ggf. entsprechende Rechenressourcen bzw. Zeit erfordern<sup>30</sup>.

Als finale Möglichkeit der Berechnung der dynamischen Verlustleistung unterstützt die PERFMON-Umgebung die Gatterebenessimulation unter Verwendung der sich ergebenden Schaltverzö-

---

<sup>29</sup> In Abhängigkeit vom verwendeten Compiler, derzeit unterstützt der UPB-Compiler nur C-Code.

<sup>30</sup> Die Synthese der auf diese Weise gewonnenen Implementierung des N-Cores (vgl. Abschnitt 6.2) nahm auf einer leistungsfähigen Workstation mit 2 GHz Prozessortakt und 16 GByte Arbeitsspeicher ca. 24 Stunden in Anspruch.

gerungen<sup>31</sup>. Die Abschätzung kann, je nach Güte der Charakterisierung der verwendeten Gatterbibliotheken im einstelligen Prozentbereich<sup>32</sup> liegen [152][153]. Unabhängig von der zu erreichenden Genauigkeit bzgl. der Bestimmung der Leistungsaufnahme, sind die relativen Unterschiede verschiedener Implementierungsvarianten von entscheidender Bedeutung. Anhand der sich ergebenden Tendenzen können vielversprechende Realisierungen bestimmt werden.

Tabelle 5-2 gibt Aufschluss über die Merkmale der einzelnen Methoden der Schaltaktivitätenaufnahme und die Qualität der daraus gewonnenen Werte zur dynamischen Verlustleistungsaufnahme der untersuchten Hardwareblöcke.

**Tabelle 5-2: Vergleich der unterschiedlichen Methoden zur Aufnahme von Schaltaktivitäten und die Qualität der daraus gewonnenen Verlustleistungsabschätzung**

Simulationsebene	Erfasst	Nicht erfasst	Für und Wider
<b>Register Transfer</b>	syntheseinvariante Elemente	Interne Knoten; Korrelation, nicht Synthese invarianter Elemente; vorkommende <i>Glitches</i> ; Zustands- und Pfadabhängigkeiten	relativ schnelle Laufzeit, geringere Genauigkeit
<b>Null-Verzögerung, Gatterebene (Gate Level)</b>	syntheseinvariante Elemente; interne Knoten; Korrelation, nicht Synthese invarianter Elemente; gewisse Pfadabhängigkeiten	vorkommende <i>Glitches</i> , gewisse Pfadabhängigkeiten	genauer als die RTL-Simulation, aber deutlich längere Laufzeit
<b>Zeitinformationen-annotierte Gatterebene (Gate Level)</b>	Alle Designelemente; Korrelation, nicht Synthese Invarianter Elemente; Zustands- und Pfadabhängigkeiten	Korrelation gleichzeitig schaltender primärer Eingänge	höchste Genauigkeit, einhergehend mit sehr langer Laufzeit

Abschließend werden die erzielten Ergebnisse in Kopien der Quellcode-Dateien an die entsprechend instrumentierten Segmente annotiert und können ausgewertet werden.

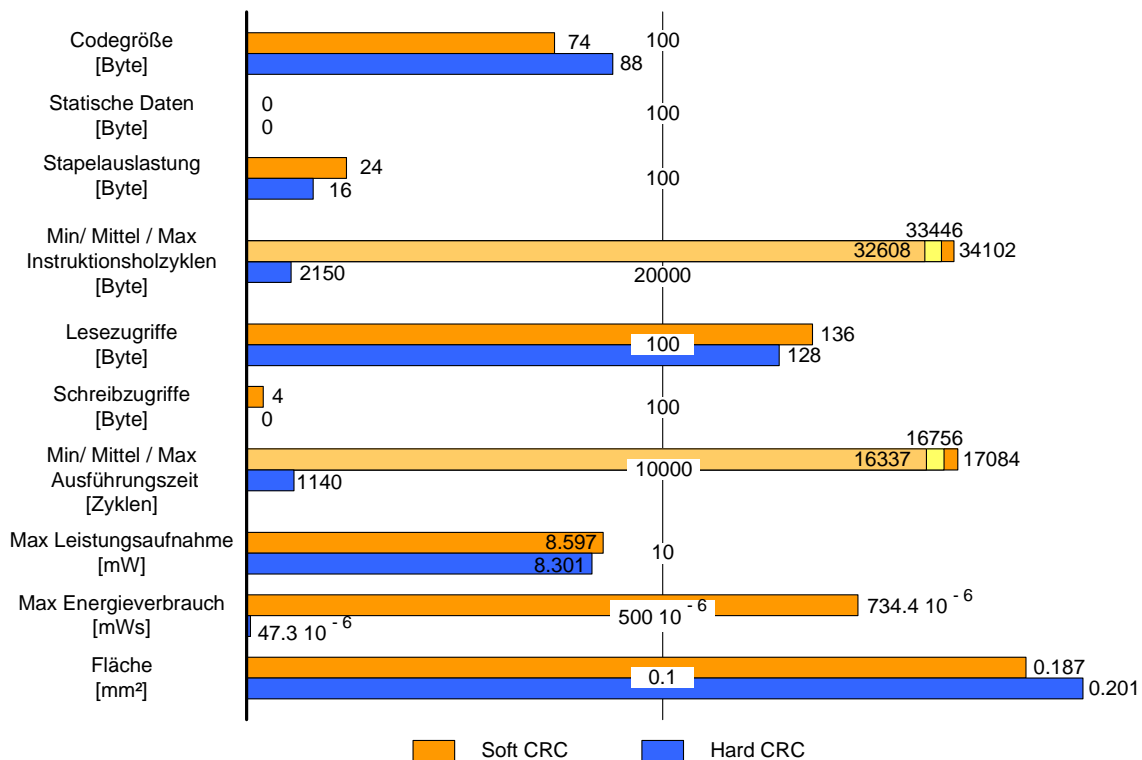
Abbildung 5-6 zeigt beispielhaft eine Gegenüberstellung ausgewählter Ergebnisse einer eigenen CRC-Implementierung in Hard- und Software, die mit Hilfe von PERFMON gewonnen wurden [116]. Die CRC-Berechnung ist speziell im Bereich der Netzwerkprozessoren von besonderer Bedeutung (vgl. Kapitel 7) und wird häufig zur Fehlererkennung und Absicherung in vielen Netzwerkprotokollen verwendet. Die Auswertung zeigt u. a. die datenabhängige Verarbeitungszeit bei der Softwarerealisierung des CRC-Verfahrens, so dass minimale, maximale und mittlere Werte für Instruktionsholzyklen und Ausführungszeit gegeben werden. Die ermittelten Werte basieren auf zufallsgenerierten Paketen mit 128 Byte Länge. Die hier dargestellte Analyse bezieht sich in diesem Fall auf das ProzessorSubsystem mit angeschlossenen, enggekoppelten Hardwarebeschleunigern. Die Verarbeitung erledigt der hier implementierte Hardwarebeschleuniger ca. 15mal schneller, wobei die maximale Leistungsaufnahme des Systems marginal geringer ausfällt, als die des rein Soft-

<sup>31</sup> In diesem Fall werden bei der Simulation des Verhaltens der Gatternetzliste die jeweiligen Verzögerungszeiten der Gatter mitberücksichtigt. Diese werden nach der Synthese in einer *Standard-Delay-Format(SDF)*-Datei abgespeichert und während der folgenden Simulation mit berücksichtigt.

<sup>32</sup> Bei der referenzierten Studie des *Oki Techno Centers* lag die Abweichung zwischen der zeitinformationsannotierten Gatterebenen-Simulation unter Zuhilfenahme der annotierten Schaltaktivitäten und der Messung am gefertigten Chip bei 5%. Synopsys gibt Werte zwischen 10 % bis 25 % Abweichung im Vergleich zu einer *SPICE*-Simulation an.

ware-basierten Systems. Der Energiebedarf des hardwarebasierten Systems ist 94 % geringer als bei der Software-basierten Variante.

Die erweiterte PERFMON-Umgebung dient u. a. dazu, häufiger genutzte Software-Funktionen oder aber auch Hardwareblöcke bzgl. ihrer Eigenschaften zu charakterisieren. Die jeweiligen gewonnenen Informationen werden, wie z. B. auch die des oben gezeigten CRC-Beispiels, in einer Bibliothek abgelegt. Automatisierte Werkzeuge wie NetAMap [102][103][104] können dann anhand dieser Daten eine Partitionierung der Algorithmen und Allokierung der Hardwareblöcke vornehmen, die eine möglichst ressourceneffiziente Realisierung der Gesamtanwendung erzielt.



**Abbildung 5-6: Beispielhafte Ergebnisse für eine *Cyclic-Redundancy-Check*(CRC)-Realisierung in Software und Hardware**

Mit Hilfe der erweiterten PERFMON-Umgebung konnte an einer Reihe von Beispielen gezeigt werden, dass durch geeignete Maßnahmen im Software- bzw. RTL-Design und in der Synthese ein nicht zu vernachlässigender Anteil der dynamischen Verlustleistung für den aktiven Betrieb eingespart werden kann [116][102][103][111][130][117][118][104][109][131].

Es gibt zahlreiche Methoden zur Reduktion der Leistungsaufnahme von Digitalschaltungen. U. a. kann durch eine angepasste Feinarchitektur dafür gesorgt werden, dass temporär ungenutzte Schaltungsteile von aktiven Eingangssignalen entkoppelt werden können, wodurch auch die internen Signale dieser Blöcke unverändert bleiben, und so die Verlustleistung dieser Einheiten merklich reduziert werden kann. Ein sehr ähnliches Beispiel ist die Anwendung von *Clock-Gating*, das zum Teil vom Synthesewerkzeug selbständig eingebracht werden kann. Hierbei wird das Taktsignal von ganzen Registerbänken oder Funktionsgruppen abgeschaltet, wenn diese ihren Zustand nicht ändern müssen. Das Einführen von Variationen der Taktfrequenz bzw. von Bereichen unterschiedlicher Versorgungsspannung (*Voltage Islands*) steigert ebenfalls die Ressourceneffizienz einer Schaltung, die Möglichkeit der Anwendung ist allerdings abhängig von der verwendeten Technologie.



Auch eine Reduzierung der Chipfläche und somit der Kosten liegt als teilweise konkurrierender Faktor mit im Blickfeld der Architekturoptimierungen. Die entwickelten Werkzeuge erlauben das Abwägen der systemrelevanten Parameter und das Finden von pareto-optimalen Entwürfen.

## 5.4 MultiSim – Parametervariation zur gezielten Entwurfsraumexploration

Für die in den vorangegangenen Abschnitten vorgestellten Simulatoren und Entwicklungsumgebungen wurde die *MultiSim*-Werkzeugkette entwickelt [131][113], eine spezielle Kombination aus Skripten und *Tcl* (*Tool command language*)-Anwendungen, die interagieren, und den Ablauf der Simulationen steuern. Durch sie werden, einer Stapelverarbeitung ähnlich, Simulationen eigenständig gestartet, analysiert im Anschluss modifiziert und, den gegebenen Parametern Folge leistend automatisiert weitere Simulationen angestoßen. Die Ergebnisse werden anschließend in *CSV*-Dateien zusammengestellt und können mit Tabellenkalkulationsanwendungen weiterverarbeitet und visualisiert werden.

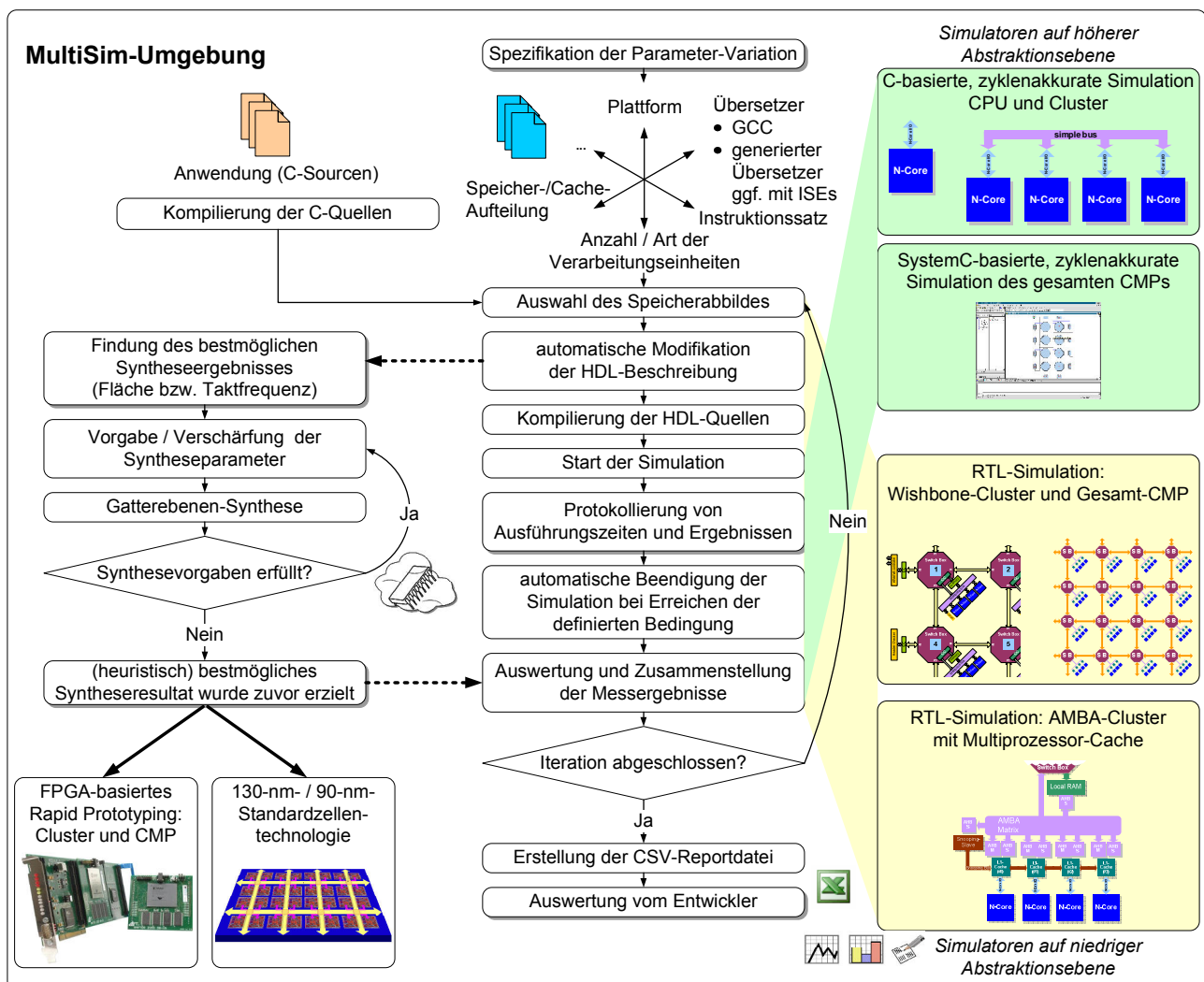


Abbildung 5-7: Funktionsweise der MultiSim-Entwicklungsumgebung

zur automatisierten Entwurfsraumexploration für die Hardware- und Software-Entwicklung

MultiSim ist für die Parametervariation in einem vorgegebenen Rahmen gedacht, aber auch evolutionäres Verhalten wäre in der Werkzeugkette modellierbar und könnte so ggf. neue Impulse bzgl. der Findung pareto-optimaler Punkte im Entwurfsraum setzen. Dies setzt natürlich je nach Simula-



tor und Umfang der Parametervariation entsprechend leistungsfähige Rechner voraus. Abbildung 5-7 zeigt den Ablauf bei der Verwendung der MultiSim-Werkzeugkette für die geschilderten Simulationsumgebungen auf.

In [109][131] wurde der Multi-Sim-Entwurfsprozess genutzt, um die Wishbonebus-basierte Giga-NetIC-Chip-Multiprozessorarchitektur automatisiert für unterschiedliche Benchmarkszenarien mit einhergehender Variation der Anzahl der lokalen Verarbeitungseinheiten zu analysieren. Ebenfalls wurde die Art der Kopplung spezieller Hardwarebeschleuniger mit Hilfe von Multi-Sim untersucht. In [113] wurde Multi-Sim eingesetzt um die Leistungsfähigkeit verschiedener Multiprozessor-Cache-Implementierungen zu analysieren. Hierbei wurden insgesamt 2.716 Simulationsdurchgänge von vier Rechnern<sup>33</sup> über eine Dauer von ca. vier Wochen ausgeführt. Variiert wurden seitens der Hardware u. a. die Assoziativität der Caches, die Größe der Cachelines und deren Anzahl, die Hauptspeicherlatenz sowie die Struktur des Caches zwischen *Unified*- oder *Split*-Cache (vgl. Abschnitt 4.4.2). Auf Softwareseite fand eine Variation der Anwendungssoftware, also der verwendeten Benchmarks statt. Die gewonnenen Ergebnisse lieferten detaillierte Aussagen darüber, welche der untersuchten Cachevarianten besonders effektiv für die jeweiligen Anwendungen sind.

## 5.5 Systememulation mit einem Rapid-Prototyping-System

Das am Fachgebiet Schaltungstechnik entwickelte FPGA-basierte Rapid-Prototyping-System RAPTOR2000 [9][154] erlaubt eine sehr hohe Emulationsgeschwindigkeit des späteren ASIC-Designs oder dient als Hardwarebeschleuniger [10]. Die Emulation ist bis zu sechs Größenordnungen schneller als die VHDL-Simulation. Verglichen mit der SystemC-Simulation liegt die hier erzielte Beschleunigung bei über zwei Größenordnungen höher, gegenüber der C-basierten Cluster-Simulation wird noch ein Beschleunigungsfaktor von zwei erzielt, vgl. Tabelle 5-1.



**Abbildung 5-8: FPGA-basiertes Rapid-Prototyping-System RAPTOR2000 zur schnellen Systememulation**

Das modulare, als PCI-Bus-Teilnehmer realisierte RAPTOR2000-Board (vgl. Abbildung 5-8) kann mit bis zu sechs steckbaren Tochterplatinen erweitert werden. Die Tochterplatinen werden über je zwei, 128 Kontakte umfassende Steckverbindungen mit der Basisplatine verbunden. Die Integration

<sup>33</sup> Pentium 4 3000 HT mit 3 GHz und 1 GByte Arbeitsspeicher.

in ein PC-System stellt zum einen eine preiswerte Lösung und zum anderen eine komfortable Testumgebung dar. Aufgrund der umfangreichen Softwarebibliothek, die für die Einbindung des RAPTOR2000-Systems erstellt wurde, lassen sich alle Funktionen des Systems in eigene Anwendungen leicht integrieren. Für den FPGA-basierten GigaNetIC-Prototypen wurde ebenfalls eine graphische Benutzerumgebung zur Systemüberwachung und Steuerung entworfen [131][113][110], vgl. Abschnitt 8.1.

Aufgrund seiner Modularität ermöglicht das RAPTOR2000-System aber nicht nur eine hohe Simulations- bzw. Emulationsgeschwindigkeit, sondern erlaubt auch die Integration des HDL-basierten Prototypen in eine "reale" Hardware-Umgebung unter Nutzung einer Vielzahl von Schnittstellen, die das RAPTOR2000-System mittlerweile zur Verfügung stellt. So konnte im Falle des GigaNetIC-Chip-Multiprozessors mit Hilfe dieser Hardwarekonfiguration die spätere netzwerkspezifische Anwendungssoftware auf realer Hardware mit tatsächlichen Daten bei einer hohen Taktrate getestet werden (vgl. Abschnitt 8.1). Schon während der Prototypenentwicklung können die Anforderungen der später eingesetzten Peripheriehardware in die Verifikation des Entwurfs einbezogen werden. Fehler bei der Schnittstellenimplementierung können so auf ein Minimum begrenzt werden. Im weiteren Verlauf der Entwicklung kann dann der standardzellenbasierte ASIC-Prototyp schnell in eine bereits erprobte Umgebung integriert werden [108]. Aufgrund vorbereiteter Vorlagen für Erweiterungsplatinen des RAPTOR2000-Systems entfallen lange Systementwicklungszeiten.

## 5.6 Einheitliche Übersetzer-Werkzeugkette

Für alle Simulatoren und Hardwareplattformen (Wishbone-Bus oder AMBA-basierter Cluster) steht eine einheitliche Übersetzer-Werkzeugkette zur Verfügung. Beim Wechsel von einer Simulationsumgebung der abstrakteren Ebene zur hardwarenahen Simulation oder umgekehrt, oder aber auch bei der Erstellung der Speicherabbilder für den FPGA- oder ASIC-Prototypen muss dem Werkzeug nur ein anderes Zielsystem angegeben werden. Eine leichte Portierbarkeit des Quellcodes auf die jeweiligen Simulatoren bzw. Emulatoren ist gewährleistet. Die Kompilierung und die Erstellung der entsprechenden Speicherabbilder unter Berücksichtigung der jeweiligen Adressräume und der Modelleigenschaften erfolgt für den Entwickler transparent.<sup>34</sup>

Die GigaNetIC-Bibliotheksfunktionen (vgl. Anhang A) stehen bis auf einige systembedingte Ausnahmen<sup>35</sup> in jeder Simulationsumgebung zur Verfügung. Bibliotheksfunktionen können sich zwar in ihrer Realisierung je nach Simulationsumgebung unterscheiden, die grundsätzliche Funktionalität wird aber eingehalten. Bei der Angabe der Zielarchitektur bzw. des Zielsimulators wählt die von uns gestaltete Werkzeugkette die jeweils passende Implementierung aus.

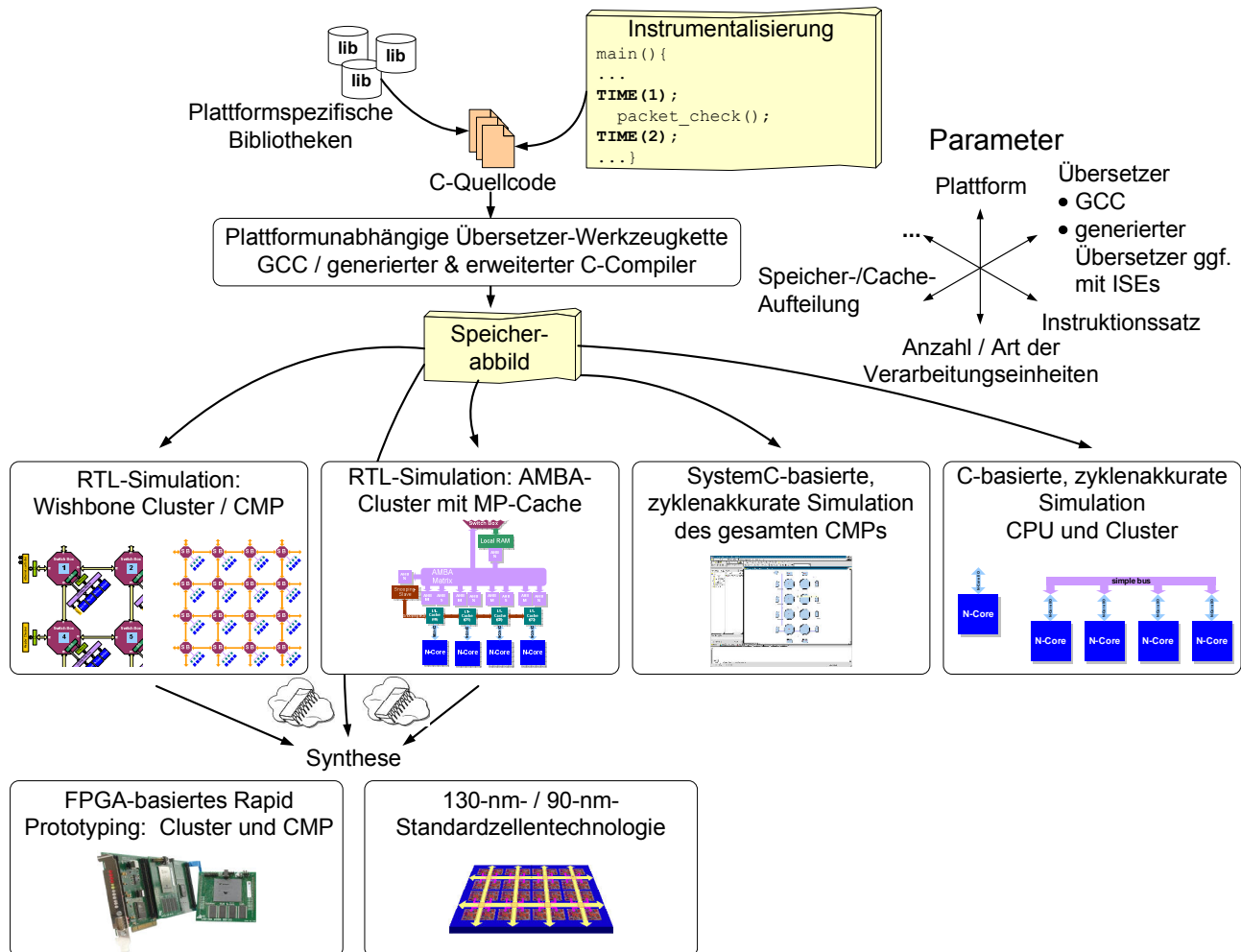
Die einheitliche Übersetzer-Werkzeugkette wurde im Laufe des GigaNetIC-Projekts fortwährend optimiert und erweitert, da ein nicht unwesentlicher Anteil der Gesamtperformanz durch die Soft-

---

<sup>34</sup> Die Übersetzer-Werkzeugkette wurde in Kooperation mit den Fachgebieten Programmiersprachen und Übersetzer, Prof. Dr. Uwe Kastens sowie Algorithmen und Komplexität, Prof. Dr. math. Friedhelm Meyer auf der Heide entwickelt.

<sup>35</sup> Z. B. ist der Cluster-Simulator derzeit nicht in der Lage, Befehle für den Communication-Controller korrekt zu interpretieren, da es diesen in dieser Simulationsumgebung nicht gibt.

ware-basierten Systementitäten geprägt werden kann. Auch die Benutzerfreundlichkeit und der Funktionsumfang wurden im Laufe der Entwicklung stetig vorangetrieben. Aus dem anfänglichen *GigaMake*, einem *Makefile*-basierten Ansatz, der die Wishbonebus- und die AMBA-Realisierung unterstützte entstand das umfassendere *SCC*-Compileskript. Es unterstützt alle Varianten und Simulatoren der GigaNetIC-Architektur. Aufgrund der Tatsache, dass es sich hierbei um ein skriptbasiertes Steuerungswerkzeug für den Prozess des Übersetzens und Erstellens der Speicherabbilder handelt, eignet es sich zudem hervorragend für das zuvor vorgestellte Multi-Sim-Entwurfsraum-explorationswerkzeug, vgl. Abschnitt 5.4. Abbildung 5-9 veranschaulicht den Ablauf und stellt die Zielarchitekturen für den *SCC*-basierten Übersetzeraufruf dar.



**Abbildung 5-9: Einheitliche Übersetzerwerkzeugkette für alle Simulatoren und Hardwareplattformen der GigaNetIC-Architektur**

Abbildung 5-10 bildet den kommandozeilenbasierten Aufruf des *SCC*-Werkzeugs ab. Dem Entwickler werden zahlreiche Optionen gegeben. So kann er zwischen dem *GCC*-Übersetzer und dem *UPB*-Compiler auswählen. Durch optionale Schalter können beim *UPB*-Compiler zusätzlich vorhandene Befehlssatzerweiterungen berücksichtigt werden.

Als Zielarchitekturen (*targets*) können alle vorgestellten Simulations- und Emulationsumgebungen ausgewählt werden. Bei Verwendung des GigaNetIC-Multiprozessorcaches können spezielle Anweisungen mit übergeben werden. Soll ein Multiprozessorfeld simuliert werden, wird dessen Ausdehnung durch Angabe von Weite (*width*) und Höhe (*height*) in *X*- und *Y*-Richtung definiert. Die Anzahl der vorgesehenen lokalen Prozessoren bestimmt der Parameter *no-pes*.

Nach erfolgter Übersetzung werden die Speicherabbilder automatisch in einer entsprechenden Verzeichnisstruktur angelegt, die sich nahtlos in die Pfade der jeweiligen Simulatoren eingliedern lässt. Zur detaillierten Analyse dieser Speicherabbilder können die Werkzeuge der *GCC-M-Core-Binutils* verwendet werden.

```

niemam@linux:ip_pktcheck> scc --help
Compile script for S-Core architectures
SYNTAX: /opt/mcore/bin/scc [architecture args] [additional compiler args]
--help                this text
--target=<target>
    clustersim        simulator WG Kastens
    sbarraysim        systemc simulator
    wbcluster         wishbone VHDL cluster
    wbcluster-fpga    wishbone FPGA cluster
    ambacluster       ambacluster with cache
--output-format=<format>
--width=<w> --height=<h>    size of the array
--no-pes=<n>              number of PEs per switchbox
--pcscore                use pcscorec instead of gcc
--stack-size=<size>      stack size in bytes
--cache-size=<size>      cache size in bytes

```

Abbildung 5-10: Aufrufmöglichkeiten und Zielarchitekturen für das Compile-Skript *SCC*

## 5.7 Zusammenfassung

Im Verlauf dieser Arbeit ist nicht nur eine leistungsfähige Chip-Multiprozessor-Architektur entstanden, sondern außerdem eine in sich geschlossene und ineinander verzahnte Werkzeugkette: angefangen beim Prozessorentwurf über die automatische Generierung des Compilers und eines C-basierten zyklenakkuraten Instruktionssatzsimulators bis hin zu rückannotierten *RTL(Register Transfer Level)*-Beschreibungen. Letztere liefern detaillierte Informationen über Leistungsaufnahme, Flächenbedarf und Leistungsfähigkeit der integrierten Schaltung und geben Impulse für Instruktionssatzerweiterungen und Hardwarebeschleuniger sowie für Systemoptimierungen allgemeiner Natur.

Die einheitliche GigaNetIC-Übersetzer-Werkzeugkette ermöglicht einen reibungslosen Übergang zwischen den einzelnen Plattformen und garantiert ein funktional gleiches Verhalten des GigaNetIC-Systems in allen Simulatoren. Etwaige irrelevante Unterschiede der einzelnen Plattformen bleiben für den Systementwickler transparent.

Durch die in sich geschlossene Entwicklungsumgebung mit ihren an die jeweiligen Erfordernisse angepassten Simulatoren ermöglicht die GigaNetIC-Architektur schnelle Hard- und Software-Entwicklungszyklen, einhergehend mit umfangreichen Verifikationsmöglichkeiten, so dass kurze *Time-to-Market*-Spannen für die zu realisierenden Multiprozessorsysteme garantiert werden können. Durch den im Folgenden vorgestellten hierarchisch gerichteten Optimierungsansatz schließt sich die Werkzeugkette und ermöglicht dem Softwareentwickler sowie dem Systemarchitekten eine anwendungsspezifische Anpassung und Optimierung der einzelnen Komponenten, so dass die Ressourceneffizienz des Chip-Multiprozessors im Bezug auf die jeweiligen Anforderungen und Randbedingungen und Schranken ggf. noch gesteigert werden kann.

## 6 Optimierung der Multiprozessorarchitektur

Bei der GigaNetIC-Architektur handelt es sich zunächst um eine universell einsetzbare Multiprozessorarchitektur, die durch zahlreiche Mechanismen auf die unterschiedlichsten Anwendungsgebiete angepasst bzw. optimiert werden kann. Um dies für den Softwareentwickler und den Hardwarearchitekten so effizient wie möglich zu gestalten, wurde für die GigaNetIC-Architektur darüber hinaus die Methodik der hierarchisch gerichteten Optimierung entwickelt und eine entsprechende Werkzeugkette erstellt. Die entworfene Methodik und exemplarische Analysen der jeweiligen Optimierungsansätze werden im Folgenden präsentiert.

### 6.1 Optimierungsmethodik

Prinzipiell ist zwischen der Hardware- und der Software-Optimierung zu unterscheiden, wobei sich zum Teil beides gegenseitig bedingt. Die GigaNetIC-Entwicklungsumgebung kann bei der Hardware-Optimierung nur greifen, falls noch eine Veränderungsmöglichkeit besteht, also im Falle einer FPGA-Zieltechnologie oder falls das *Tapeout*, d. h. die finale Fertigstellung des standardzellenbasierten ASICs, noch aussteht. Sollte letzteres bereits geschehen sein, lassen sich Hardwarebeschleuniger zur Beschleunigung des GigaNetIC-Systems nur noch extern, z. B. über nach außen geführte Switch-Box-Ports anschließen. Die Software lässt sich dagegen auch nach Fertigstellung des Chips ändern und ggf. optimieren, da es sich beim Instruktionsspeicher der Verarbeitungseinheiten in jedem Fall um wiederbeschreibbaren Speicher handelt.

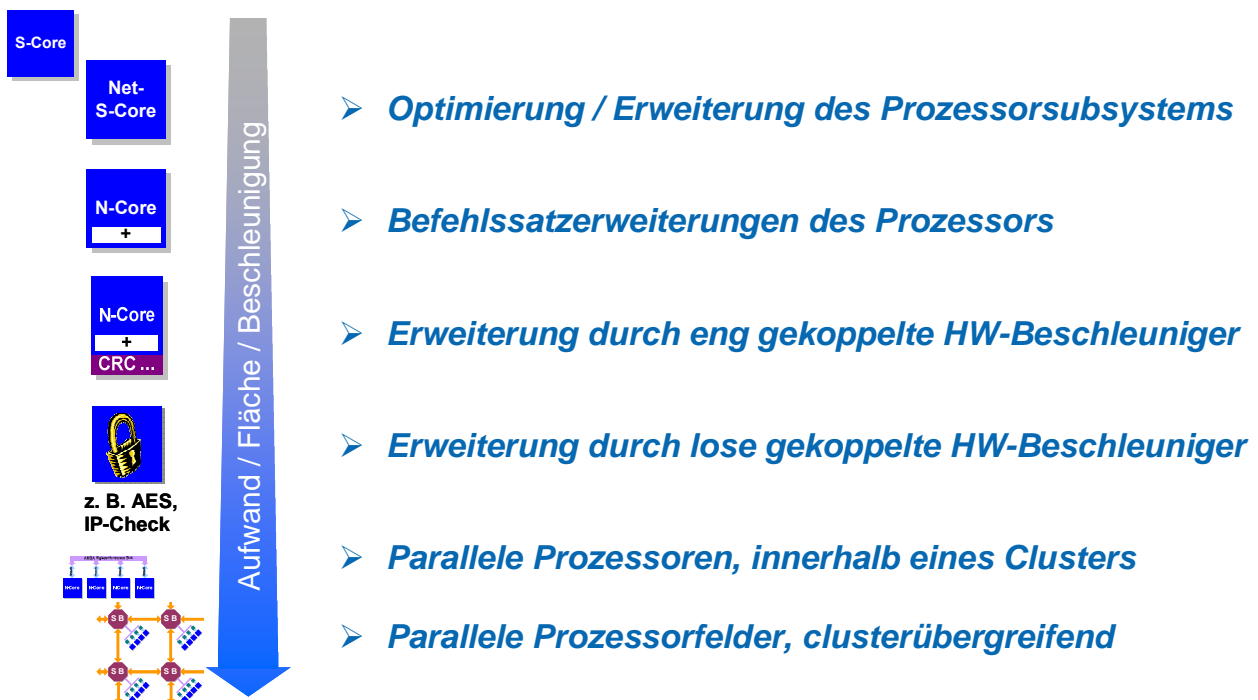


Abbildung 6-1: Entwurfsraumexploration – hierarchisch gerichteter Optimierungsansatz

Abbildung 6-1 zeigt die potentiellen Ansatzpunkte der Systemoptimierung und den gerichteten Ablauf, in dessen Verlauf der Entwurfsaufwand, die benötigte Fläche, aber auch die erzielte Beschleunigung deutlich zunehmen.

Die Maßnahmen zur Optimierung der Hardware der GigaNetIC-Architektur setzen auf Prozessorebene an, indem zusätzlich benötigte Einheiten, wie z. B. *Interruptcontroller* etc. (vgl. Abschnitt 4.3.2) in das Prozessorsubsystem eingefügt werden, um so die Systemleistung zu erhöhen bzw. anwendungsspezifische Anforderungen zu erfüllen. Der nächste Schritt ist die Optimierung des Prozessorkerns selbst, durch Erweiterung des Instruktionssatzes oder architekturelle Erweiterungen. Darüber hinaus kann der Einsatz von eng an den Prozessor gekoppelten Hardwarebeschleunigern erfolgen, entweder direkt über die Coprozessorschnittstelle des Prozessors und angesteuert über spezielle Befehle, oder aber ins Prozessorsubsystem integriert über den eigens erweiterten Prozessorsystembus (vgl. Abschnitt 4.3). Erste Optimierungen des Prozessors wurden bereits in [108] durchgeführt, was jedoch noch ohne die heutige Entwicklungsumgebung vonstatten gehen musste und sich so deutlich aufwändiger gestaltete.

Eine weitere Steigerung der Beschleunigung bieten in der Regel lose gekoppelte Hardwarebeschleuniger, die ggf. auch bei einer deutlich höheren Taktfrequenz arbeiten können als der Prozessorkern. Während eng-gekoppelte Hardwarebeschleuniger von der CPU angesteuert werden, übernehmen die lose gekoppelten Hardwarebeschleuniger unabhängig vom Prozessor eine Datenvorverarbeitung bzw. Nachbearbeitung und können mit Hilfe des bereits vorgestellten Communication-Controllers (vgl. Abschnitt 4.2.1.1) an einen beliebigen Port einer Switch-Box im SoC angeschlossen werden.

Letztendlich eröffnet sich dem Entwickler die Option des Einsatzes mehrerer Prozessoren bzw. paralleler Prozessorfelder zur Bearbeitung der Algorithmen. Dies erfordert in Abhängigkeit vom Anwendungsfall sowohl großen Aufwand auf Seiten der Software als auch den entsprechenden Aufwand an Chipfläche.

Die Werkzeugkette hilft dem Entwickler bereits in der Anfangsphase des Hardwareentwurfs bei der Abschätzung der zur Erfüllung der Spezifikation notwendigen Maßnahmen. So unterstützt sie bei der Entscheidung auf welcher Ebene der Optimierung angesetzt werden muss, um eine möglichst ressourceneffiziente Realisierung zu erzielen. Dies kann bereits mit Hilfe der sehr schnellen, auf abstrakterer Ebene arbeitenden Umgebungen wie dem Cluster-Simulator oder der SiMPLE-SystemC-Simulationsumgebung geschehen, vgl. Abschnitte 5.1 und 5.2.

Analysen im Bereich von Netzwerkanwendungen haben gezeigt, dass häufig eine ausgewogene Mischung all dieser Möglichkeiten den besten Kompromiss im Hinblick auf Ressourceneffizienz nach Definition 14 bietet [111][141][130][118][109][131], vgl. auch Kapitel 7. Die einfache Skalierbarkeit des GigaNetIC-Systems erweist sich hier ebenfalls als besonders förderlich, da sich für viele Anwendungen die Parallelisierbarkeit anbietet, und man deshalb gleichförmige Strukturen relativ gut einsetzen kann.

Der hierarchische Ansatz erweist sich als besonders vorteilhaft, wenn es darum geht, die bestehende Grundstruktur des GigaNetIC-Systems mit möglichst geringem Aufwand so zu erweitern, dass die vorgegebene Spezifikation erfüllt werden kann. Nachteilig an einem solch „maßgeschneiderten“ Chip ist allerdings die relativ aufwändige Analysephase, die sich ohne geeignete Entwicklungsumgebung als große Herausforderung gestalten kann. Bei der GigaNetIC-Architektur wird durch die Multi-Sim-Werkzeugkette und die leistungsfähigen Simulatoren (vgl. Kapitel 5) eine automatisierte und schnelle Durchführung dieser Analysen in vertretbarer Zeit ermöglicht. In Kapitel 7 werden Ergebnisse des Ansatzes der hierarchisch gerichteten Optimierung am Beispiel von dedizierten Netzwerkanwendungen dargestellt.



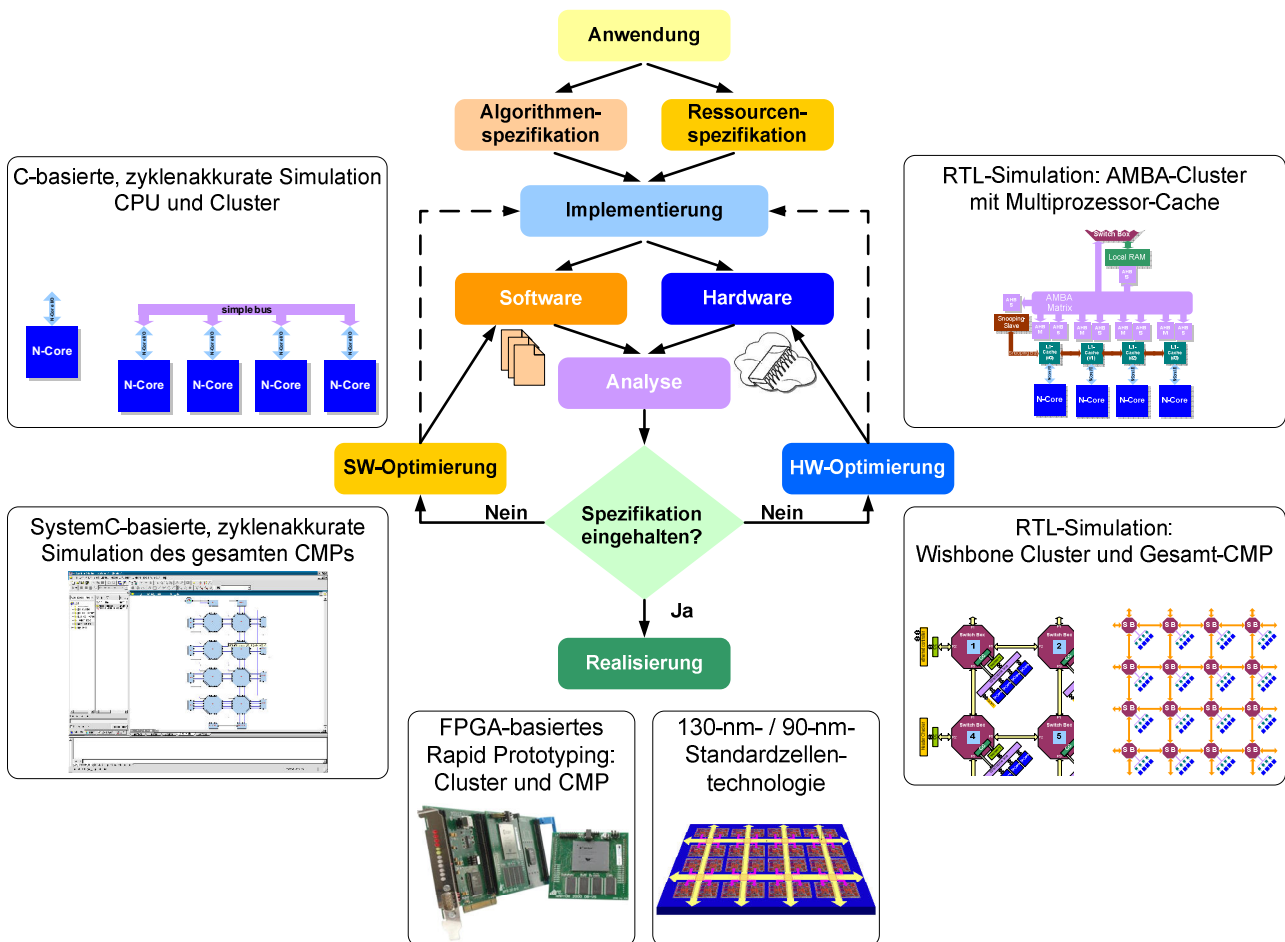


Abbildung 6-2: Anwendungsorientierte Realisierung und Optimierung der GigaNetIC-Architektur

Abbildung 6-2 zeigt das Vorgehen bei der anwendungsorientierten Realisierung und Optimierung der GigaNetIC-Architektur. Steht das Anwendungsszenario für den zu realisierenden Chip fest, werden dadurch die anzuwendenden Algorithmen spezifiziert. Für die Hardware werden Randbedingungen aufgestellt und damit die möglichen Ressourcen festgelegt. Bei der Implementierung wird zunächst auf die bestehenden IP-Blöcke der GigaNetIC-Architektur zurückgegriffen und unter Verwendung der Standardhardware und einer ersten lauffähigen Software das Laufzeitverhalten mit Hilfe der in Kapitel 5 vorgestellten Simulationsumgebungen analysiert. Je nach Art der Randbedingungen und Analyseergebnisse erfolgen dann die weiteren Schritte. Hierunter fallen die iterative Optimierung von Soft- und/oder Hardware entsprechend dem oben vorgestellten Modell mit eingehenden Simulationsläufen. Der Erfolg bei der Softwareoptimierung ist durch die schnellen Simulatoren auf abstrakterer Modellierungsebene besonders zeitnah erkennbar. Bei Modifikationen auf Hardwareseite hingegen fallen abschließend zwangsläufig die zeitintensiveren Simulationsläufe in der detaillierten hardwarenahen Simulationsumgebung an.

Abbildung 6-3 stellt Entwurfsschritte und Methoden der Verlustleistungsanalyse und Optimierung, die während der Realisierung eines ASIC-Entwurfs durchlaufen werden, dar und zeigt den zu erwartenden Einfluss der jeweiligen Optimierungsmaßnahmen auf die Reduktion der Verlustleistung auf. Ein guter Systementwurf hat den größten Einfluss auf die Optimierung der Leistungsaufnahme heutiger ASICs, deshalb wurde bei der Entwicklung der GigaNetIC-Entwicklungsumgebung gesteigerter Wert auf geeignete Simulations- und Analysemethoden für diesen Entwurfsschritt gelegt, vgl. Kapitel 5 [116][102][103][117][104][115][113]. Die hierdurch erzielten Reduktionen können im Folgenden durch den sich anschließenden RTL-Entwurf und dessen Optimierungspotential verbes-

sert werden. Allerdings sind die Resultate auf dieser Ebene, absolut gesehen, bereits deutlich geringer, aber dennoch nennenswert. Hier greifen die erweiterte PERFMON und Multi-Sim-Umgebung, vgl. Abschnitte 5.3 und 5.4, sowie die Werkzeugkette zur Instruktionssatzerweiterung des Prozessorkerns der GigaNetIC-Architektur [112][111][131][113]. Diese setzt sich deshalb aus zwei aufeinander abgestimmten Stufen zusammen, die im folgenden Abschnitt näher erläutert werden. Die weiteren Schritte versprechen ebenfalls Optimierungspotential, das allerdings im Allgemeinen deutlich geringer ausfällt als das der bereits genannten Entwurfsschritte. Aufgrund dieser Tatsache werden die Folgeschritte in dieser Arbeit nur am Rande erwähnt und das Hauptaugenmerk auf die wesentlichen Einflussquellen gesetzt.

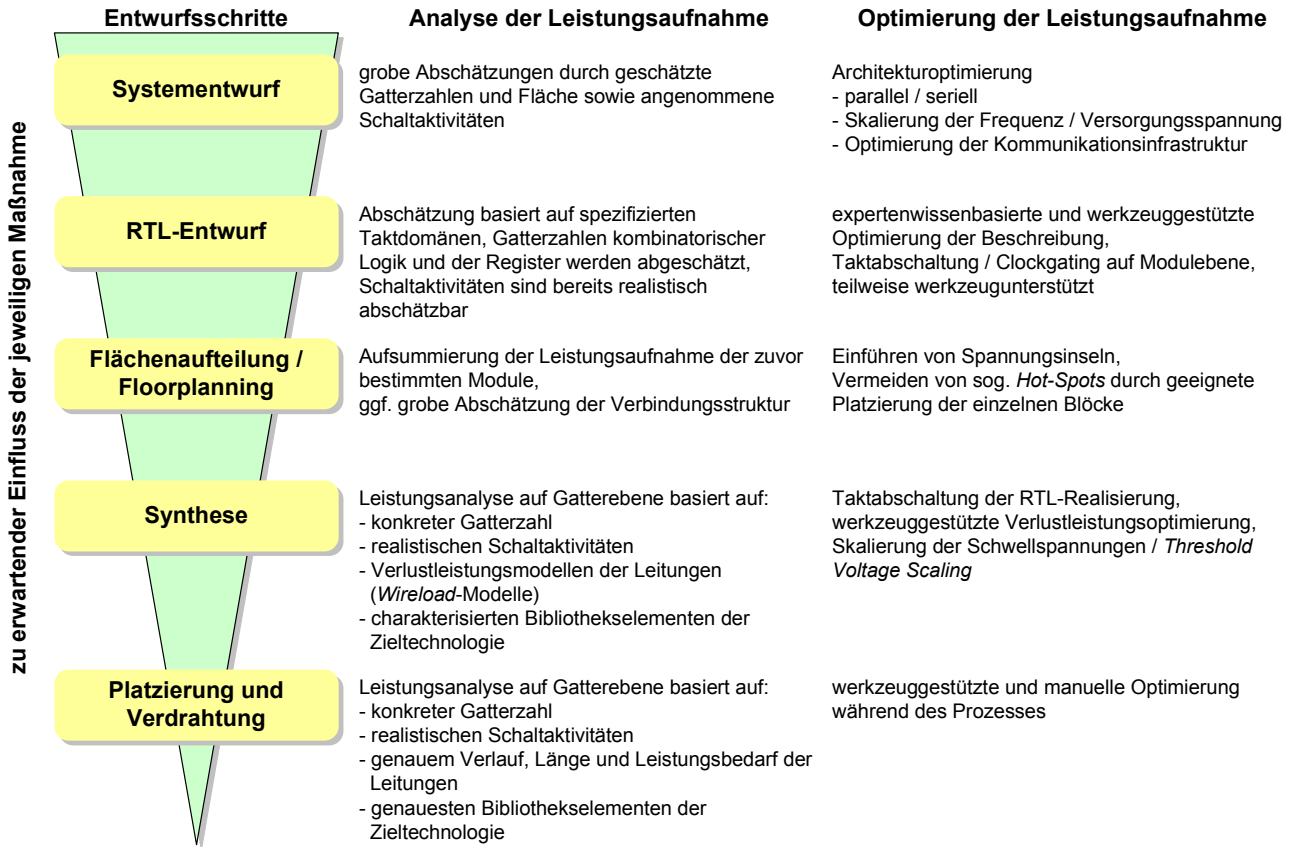


Abbildung 6-3: Entwurfsschritte und Methoden der Verlustleistungsanalyse und Optimierung

## 6.2 Optimierung auf Prozessebene

Um den immer höher werdenden Anforderungen, die an heutige und zukünftige eingebettete Systeme gestellt werden, gerecht werden zu können, sind viele Prozessoranbieter bzw. Anbieter von Übersetzerwerkzeugketten mit angeschlossenem Generator einer synthetisierbaren Architekturbeschreibung dazu übergegangen, dem Kunden Möglichkeiten an die Hand zu geben, den einzusetzenden Prozessor auf das zukünftige Anwendungsgebiet hin kundenspezifisch zu optimieren. Auch die GigaNetIC-Architektur bietet dem Systemarchitekten umfangreiche Möglichkeiten zur anwendungsspezifischen Optimierung des Befehlssatzes des N-Core-Prozessorkerns.

In Tabelle 6-1 wird ein Überblick über einige der derzeit verfügbaren Ansätze und deren Merkmale in Bezug auf Umfang und Gestalt der Entwicklungswerkzeuge gegeben. So sind sowohl Ansätze vertreten, die auf Beschreibungssprachen wie *Lisa* von CoWare basieren, die mit der *CoWare-Processor-Designer*-Werkzeugkette in Hardware umgesetzt werden kann [155]. Bei *nML*, der Rea-



lisierung von Target Compiler Technologies [156], steht die *Chess/Checkers*-Entwicklungs-umgebung zur Verfügung. Ein ähnlicher Ansatz wird mit der abstrakten Spezifikations-sprache *UPSLA (Unified Processor Specification Language)* [135] bei der GigaNetIC-Architektur verfolgt. ARC [70] hingegen verwendet mit dem *ARChitect Processor Configurator* eine graphische Ober-fläche zur Konfiguration seiner anwendungsspezifischen Prozessorkerne, wobei ggf. weniger komplexe Mechanismen angewendet werden können, als es durch den Einsatz von Beschreibungs-sprachen möglich wäre. Tensilica [157] erlaubt die Beschreibung des Prozessormodells in C/C++ und stellt zur Umwandlung in Hardware den *XPRES (Xtensa PROcessor Extension Synthesis) Com-piler* für automatisch generierte Prozessoren zur Verfügung, der eine besonders schnelle Umsetzung der Beschreibung in Hardware gewährleisten soll. Die Prozessormodelle sind kundenspezifische Architekturen, wobei bei ARC die ARC-Architektur als Ausgangspunkt dient und Tensilica den *Xtensa*-Prozessorkern verwendet. Alle Entwicklungsumgebungen bieten eine automatisierte Gene-rierung von Compiler und zugehörigen Instruktionssatz-Simulator, wobei die Simulation bei eben-falls allen Ansätzen zyklengenau ausfällt. Bei allen Vergleichsansätzen erfolgt zu dem eine Gene-rierung der spezifizierten Hardware. Dies ist bei der GigaNetIC-Architektur zwar vorgesehen und wird im Rahmen aktueller Forschungsprojekte der Universität Paderborn weiter vorangetrieben, ist allerdings noch nicht vergleichbar mit der Leistungsfähigkeit der anderen Ansätze. Deshalb bauen derzeit alle Befehlssatzerweiterungen der GigaNetIC-Architektur auf dem bewährten, relativ ein-fach gehaltenen N-Core-Prozessorkern auf, um so nicht stetig vollkommen neue Prozessorstrukt-uren entwerfen und optimieren zu müssen. Nur insgesamt drei der verglichenen Ansätze erlauben den iterativen Prozess der Erweiterung des bestehenden Befehlssatzes, wie es auch durch die Giga-NetIC-Werkzeugkette der Fall ist.

**Tabelle 6-1: Vergleich ausgewählter Werkzeugketten zur Optimierung konfigurierbarer Prozessorkerne**

Entwicklungsumgebung	Organisation	Beschreibungssprache	Prozessormodell	Compiler-Generierung	Simulator-Generierung	Zyklengenau Simulation	Hardware-Generierung	Unterstützung für ISE
LisaTek	CoWare	Lisa	kundenspezifisch	x	x	x	x	(x)
Chess/Checkers	Target Compiler Technologies	nML	kundenspezifisch	x	x	x	x	(x)
Express	Univ. of California, Irvine	Expression	kundenspezifisch	x	x	x	x	(x)
ARChitect	ARC	GUI-basiert	ARC	x	x	x	x	x
XPRES-Compiler	Tensilica	GUI-basiert	Xtensa	x	x	x	x	x
<b>GigaNetIC</b>	<b>Universität Paderborn</b>	<b>UPSLA</b>	<b>kundenspezifisch</b>	<b>x</b>	<b>x</b>	<b>x</b>	<b>(x)</b>	<b>x</b>

Der im Verlauf des GigaNetIC-Projekts entwickelte, zweistufige Entwurfsprozess zur anwendungs-spezifischen Systemoptimierung auf Prozessorebene durch Instruktionssatzerweiterung (auch *In-struction Set Extension*, bzw. *ISE*) wird in den folgenden Abschnitten näher erläutert, er teilt sich in einen compilerbasierten und einen sich anschließenden hardwarebasierten Schritt auf.

### 6.2.1 Compilerbasierter Entwurfsprozess zur Prozessoroptimierung

Der compilerbasierte Entwurfsprozess ermöglicht eine Exploration des Entwurfsraums in zweierlei Hinsicht: Zum einen erlaubt er eine sehr schnelle Suche von vielversprechenden Instruktionssatz-erweiterungen auf Prozessorebene. Zum anderen ist es möglich, unterschiedliche Compiler mit un-

terschiedlichen Optimierungsmethoden und Registerzuweisungsmechanismen zu evaluieren.<sup>36</sup> Diesem Teil des Entwurfsprozesses schließt sich der in Abschnitt 6.2.2 vorgestellte hardwarebasierte Entwurfsprozess nahtlos an.

Zur Evaluierung des Entwurfsraums auf Prozessorebene werden zunächst die Ressourcen des Prozessors abstrakt in der Beschreibungssprache *UPSLA (Unified Processor Specification Language)* [135] spezifiziert. Dies sind im Einzelnen: Art und Anzahl der Register, Funktionseinheiten, Instruktionen, prozessorinterne Parallelität und Verbindungsstrukturen. Im Anschluss wird mit Hilfe dieser Spezifikation automatisch ein Compiler nebst hoch-performantem, zyklenakkuratem Simulator (vgl. Abschnitt 5.1) generiert. Abbildung 6-4 zeigt den Ablauf und die wesentlichen Einflussfaktoren der compilerbasierten Werkzeugkette.

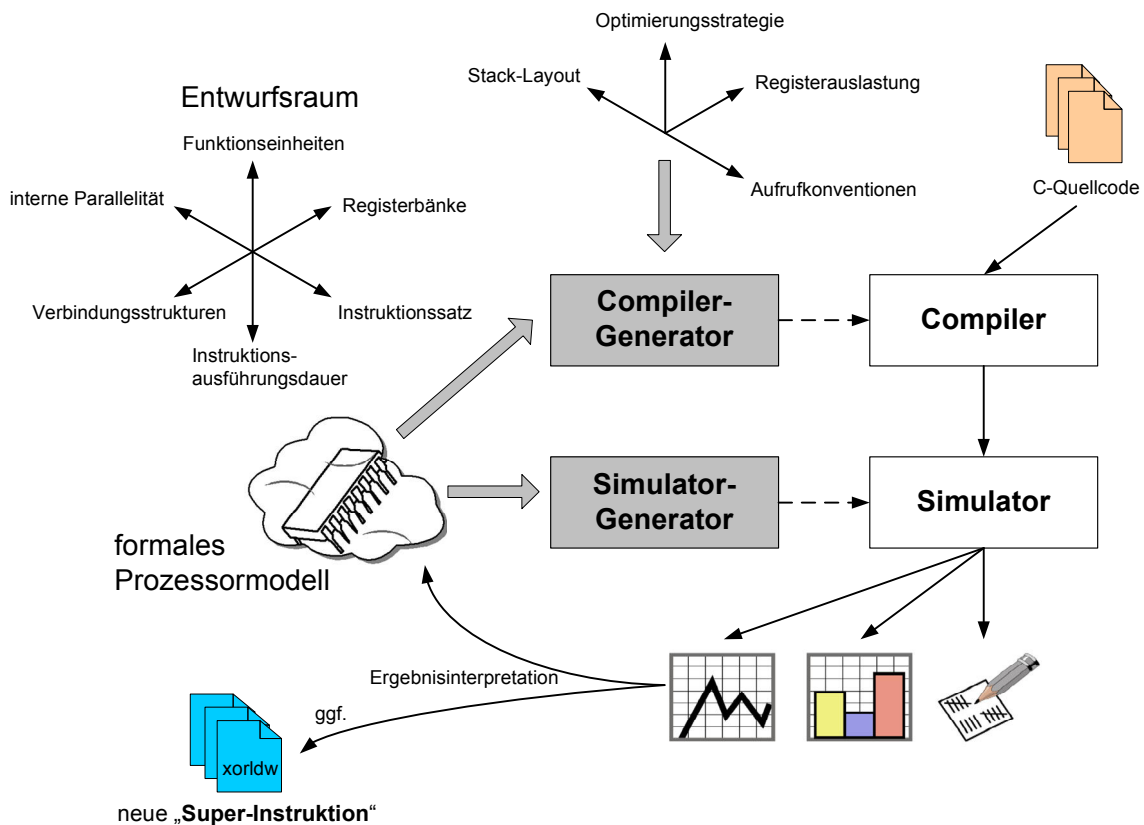


Abbildung 6-4: Schritt 1 des zweistufigen Optimierungsprozesses, compilerbasierter Entwurfsprozess

Der compilerbasierte Entwurfsschritt lässt sich äußerst schnell<sup>37</sup>, mit veränderten Ressourcen des Prozessors wiederholen, so dass in kurzer Zeit eine Vielzahl von Möglichkeiten untersucht werden kann. In unserer Werkzeugkette werden die Befehlssatzerweiterungen für den Anwender transparent vom Compiler ausgenutzt. Die zukünftigen, in der Hochsprache C vorliegenden Anwendungen werden zunächst nach der Erzeugung des Compilers kompiliert und mit dem korrespondierenden Simulator ausgeführt. Während der Ausführung werden alle relevanten, laufzeitabhängigen Daten vom Simulator in entsprechende Statistiken übertragen, die im Anschluss mit dem leistungsfähigen

<sup>36</sup> Dieser Teil des Entwurfsprozesses wurde vom Fachgebiet Programmiersprachen und Übersetzer von Herrn Prof. Dr. Uwe Kastens der Universität Paderborn entwickelt.

<sup>37</sup> Die Spezifikation und die Umsetzung der unter Abschnitt 6.2.3 vorgestellten Befehlssatzerweiterungen beanspruchen zusammen mit der Realisierung der neuen Compiler-Simulator-Werkzeugkette pro Befehl ca. zehn Minuten.

Profilierungswerkzeug *jScore* u. a. auf oft wiederkehrende, datenabhängige Befehlspaare bzw. -tripel untersucht werden können [112][6]. Die Befehlspaare mit dem höchsten Beschleunigungspotential werden anschließend zu so genannten „Superinstruktionen“<sup>38</sup> zusammengefasst und für die Realisierung in Hardware vorgeschlagen [111]. Weitere Details zur Compiler-Werkzeugkette sind u. a. [135][112][6] zu entnehmen.

Abschließend ist festzuhalten, dass der compilerbasierte Entwurfsschritt somit eine Leistungsabschätzung des späteren Prozessors in Kombination mit dem jeweiligen Compiler ermöglicht. Die Möglichkeit der Veränderung und Eruiierung unterschiedlicher Compilervarianten im Zusammenspiel mit der jeweiligen Prozessorarchitektur stellt einen großen Vorteil bzgl. des zu erwartenden Resultats dar. Würde nur eine der beiden Optimierungsmöglichkeiten ausgeschöpft, so könnten wesentliche Aspekte aus Sicht der Softwareoptimierung nicht betrachtet werden. Aufgrund des schaltungstechnischen Hintergrundes dieser Arbeit beziehen sich allerdings die weiteren Analysen auf den zweiten Schritt der Optimierung auf Prozessorebene, den hardwarebasierten Entwurfsprozess, dessen prinzipieller Ablauf im folgenden Abschnitt näher erläutert wird.

### 6.2.2 Hardwarebasierter Entwurfsprozess zur Prozessoroptimierung

Der hardwarebasierte Entwurfsprozess schließt sich dem compilerbasierten Entwurfsprozess an. Die zuvor durch den ersten Schritt der GigaNetIC-Prozessoroptimierung spezifizierten Instruktionssatzerweiterungen werden bei dem Folgeschritt auf die Möglichkeit einer ressourceneffizienten Hardwareimplementierung in den bestehenden Prozessorkern überprüft. Die Implementierung in Hardware und die sich anschließenden Analysen benötigen einen deutlich höheren Zeitaufwand als der übersetzerbasierte Entwurfsprozess<sup>39</sup>. Der erweiterte Prozessorkern wird im Anschluss in der Analyseumgebung PERFMON (vgl. Abschnitt 5.3) detailliert untersucht [116][111].

**Tabelle 6-2: Freie Opcodebereiche des S-Cores**

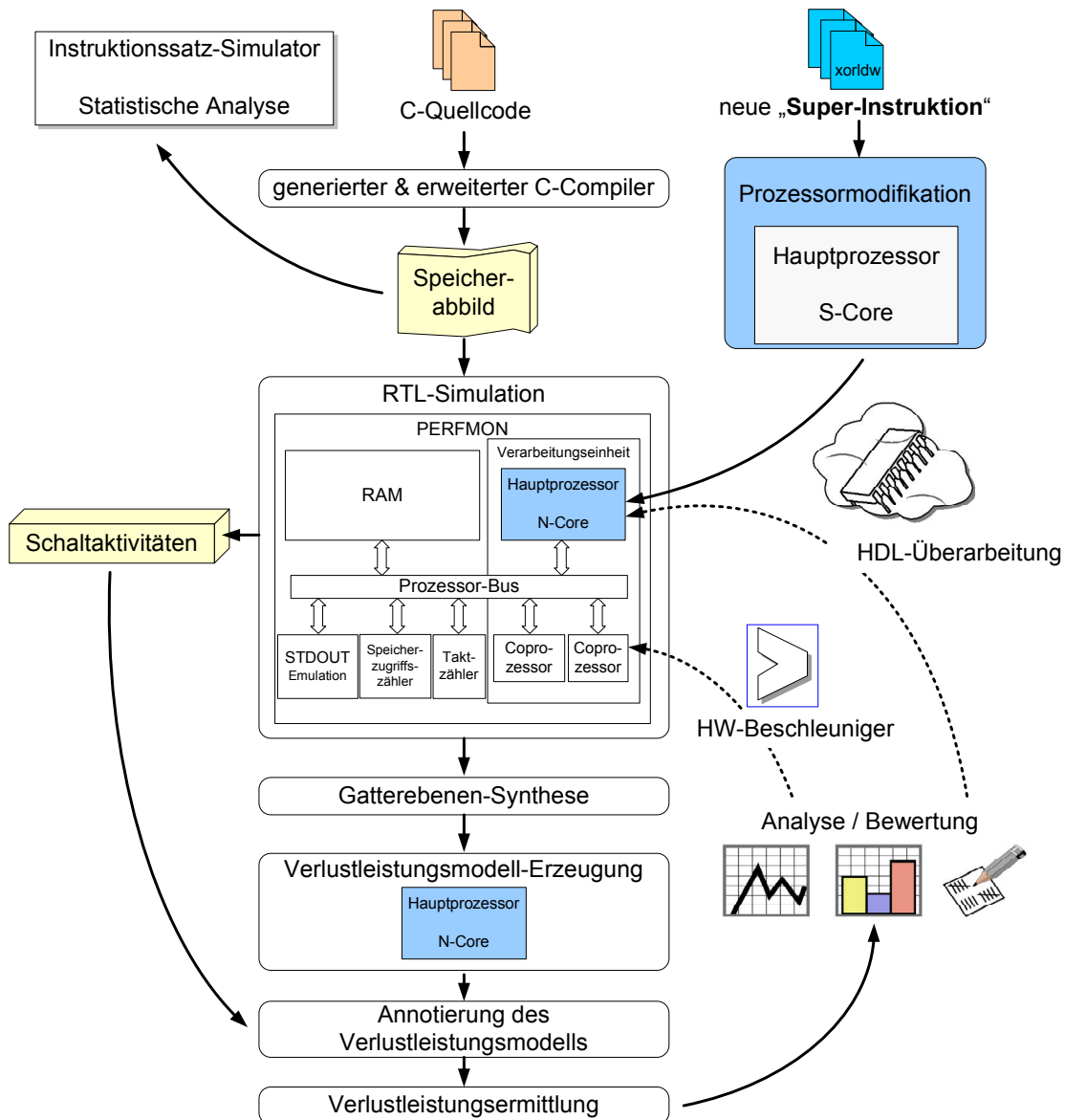
freie Opcodes				min	max	#Stellen
0000	0000	0000	0111	1	1	0
0000	0000	0000	11xx	1	4	2
0000	0000	0010	xxxx	1	16	4
0000	10xx	xxxx	xxxx	1	1024	10
0010	1100	001x	xxxx	1	32	5
0010	1100	01xx	xxxx	1	64	6
0011	0010	0000	xxxx	1	16	4
0011	0010	001x	xxxx	1	32	5
0011	0010	010x	xxxx	1	32	5
0011	0010	0110	xxxx	1	16	4
010x	xxxx	xxxx	xxxx	1	8192	13
0110	1xxx	xxxx	xxxx	1	2048	11
<b>Total</b>				<b>12</b>	<b>11477</b>	<b>69</b>

Der den Optimierungen zugrundeliegende S-Core-Prozessorkern (vgl. Abschnitt 4.3.1) lässt durch seine unbenutzten Opcodebereiche maximal 11.477 zusätzliche einfache Instruktionen ohne Para-

<sup>38</sup> Hierunter wird die Realisierung einer effizienteren, bisher nicht verfügbaren Einzelinstruktion basierend auf der Funktionalität mehrerer, bisher einzeln aufzurufenden Instruktionen verstanden.

<sup>39</sup> Die aufzuwendende Zeit bei einem erfahrenen Entwickler beläuft sich auf einige Stunden bis hin zu mehreren Tagen oder Wochen, je nach Komplexität und Art der Instruktionssatzerweiterung.

meterkodierung zu. Tabelle 6-2 gibt einen Überblick über die verfügbaren Bitkombinationen der noch verfügbaren Befehlswoorte (verfügbare Bitpositionen sind mit „x“ gekennzeichnet). „min“ gibt die möglichen, komplexen, parameterbehafteten Befehlswoorte und „max“ die Anzahl einfacher Befehle, die in dem aufgeführten Bereich realisierbar sind, an. „#Stellen“ beziffert die aufeinanderfolgenden verfügbaren Binärstellen, die ggf. zur Parameterübergabe des Befehlswortes genutzt werden können. Eine detaillierte Auflistung aller Befehle des S-Cores und deren Aufbau werden in Anhang D dokumentiert.



**Abbildung 6-5: Schritt 2 des zweistufigen Optimierungsprozesses, hardwarebezogener Entwurfsprozess**

Abbildung 6-5 zeigt die prinzipielle Vorgehensweise des hardwarebasierten Entwurfsprozesses zur Prozessoroptimierung. Zunächst müssen die neuen, abstrakt spezifizierten Veränderungen der bestehenden Prozessorbeschreibung in eine entsprechende Hardwarebeschreibung umgesetzt werden. Dies kann sich z. B. auf die Register, Funktionseinheiten, Instruktionen, und prozessorinterne Verbindungsstrukturen auswirken. Die um die neue Superinstruktion erweiterte Hardwarebeschreibung des Prozessors wird dann in die bestehende Simulationsumgebung integriert. Da sich im Allgemeinen die äußeren Schnittstellen des Prozessors bei einer solchen Erweiterung nicht verändern, ist der sich hieraus ergebende Aufwand zu vernachlässigen. Zur Simulation werden die gleichen Software-

quellen mit Hilfe des generierten Compilers übersetzt und das entsprechende Speicherabbild in den Speicher des Simulationsmodells geladen. Anschließend wird der ggf. instrumentierte Code (vgl. Abschnitt 5.3) auf dem erweiterten Prozessor abgearbeitet und durch die Werkzeugkette analysiert.

Die Synthese auf Standardzellen liefert u. a. wichtige Informationen über den kritischen Pfad und den Flächenbedarf des neuen Prozessorkerns. Es ist darauf zu achten, dass der kritische Pfad der Logik möglichst nicht verlängert wird, da es sonst zu einer Herabsetzung der Gesamtperformanz des Prozessors käme, und die Leistungsfähigkeit ggf. sogar durch die implementierte Befehlssatzerweiterung insgesamt reduziert würde. Anhand der Annotierung der Schaltaktivitäten bei der Abarbeitung des Anwendungsprogramms können im Folgenden detaillierte Aussagen über die Veränderung der Leistungsaufnahme und den resultierenden Energiebedarf der Schaltung getroffen werden.

Sollte das erzielte Ergebnis nicht zufriedenstellend sein, würde eine erneute Iteration mit Verfeinerung / Optimierung des VHDL-Entwurfs notwendig. Zeigt sich, dass durch die ausgewählte Instruktionssatzerweiterung ein zu langer kritischer Pfad entsteht, der nicht durch Optimierung beseitigt werden kann, und alternative Instruktionssatzerweiterungen ebenfalls nicht genügend Leistungszuwachs versprechen, so ist zu überlegen, ob ein leistungsfähigerer Hardwarebeschleuniger für diese Aufgabe implementiert bzw. eingesetzt werden muss. Die Analyse und Charakterisierung von anwendungsspezifischen Hardwareblöcken wird ebenfalls von der Werkzeugkette unterstützt, siehe auch Kapitel 5. Detaillierte Ergebnisse zu diesen Untersuchungen werden exemplarisch im Abschnitt 6.3 und Kapitel 7 vorgestellt.

Im Allgemeinen profitieren vornehmlich verwandte Anwendungsklassen von einer Superinstruktion, so dass nicht grundsätzlich mit einer universellen Beschleunigung für alle Algorithmen zu rechnen ist. Deshalb sollte, wenn möglich, vor der Realisierung des Systems, mit Hilfe der zur Verfügung gestellten Simulations- und Analyseumgebungen der Prozessorkern auf vielversprechende Instruktionssatzerweiterungen für den zukünftigen Einsatzzweck untersucht und ggf. erweitert werden.

Im Folgenden werden an einem Beispiel die Möglichkeiten dieser Optimierungsmethode auf der untersten Hierarchieebene der GigaNetIC-Architektur aufgezeigt. Es wird deutlich, dass bei einem sehr geringen Mehraufwand an Fläche deutliche Geschwindigkeitszuwächse zu erzielen sind.

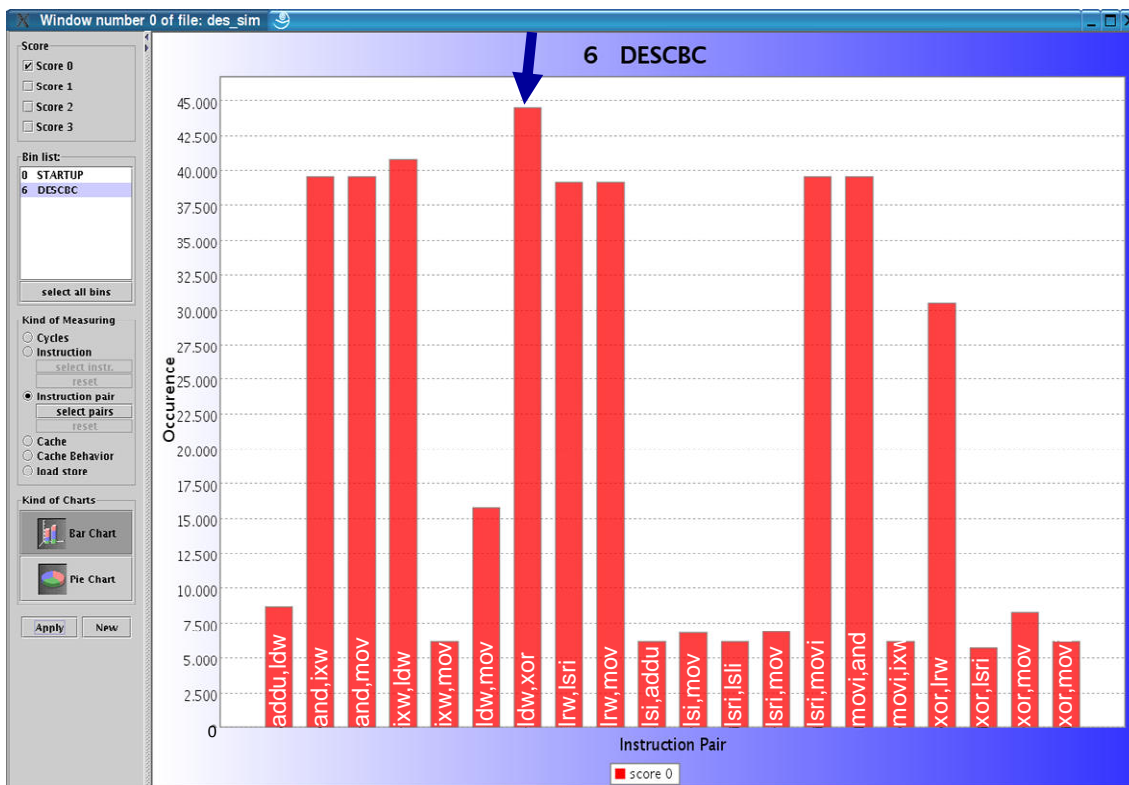
### 6.2.3 Optimierungspotential von Befehlssatzerweiterungen – ein Beispiel

Am Beispiel einer zuvor durch den compilerbasierten Entwurfsablauf ermittelten, für die *IP-Sec(Internet-Protocol-Security)*-Protokollsammlung [158] vielversprechenden Superinstruktion werden im Folgenden exemplarisch die Vorgehensweise und die aus der Instruktionssatzerweiterung resultierenden Verbesserungen vorgestellt.

*IPSec* ist eine Reihe von Protokollen, die die *Internet-Protokoll(IP)*-basierte Kommunikation mit zusätzlichen Sicherheitsmerkmalen ausstatten. Es sind u. a. Authentifizierungs- wie auch Verschlüsselungsverfahren integriert, die auf den einzelnen IP-Paketen angewendet werden. Dies geschieht im Gegensatz zu anderen *IP*-Sicherheitsprotokollen bereits auf der *OSI*-Schicht 3, was die Flexibilität erhöht, da u. a. sowohl *TCP*- als auch *UDP*-basierte Protokolle der vierten *OSI*-Schicht geschützt werden können. Dies geht allerdings mit einer erhöhten Komplexität und größerem Rechenleistungsbedarf einher. Aufgrund der in der Spezifikation vorgesehenen Erweiterungsoptionen dieses Standards sind Leistungsfähigkeit und Flexibilität gleichermaßen wichtig für eine effiziente

Verarbeitung. Aus diesem Grund sind besonders Universalprozessorelemente im Hinblick auf diese Protokollkategorie untersuchenswert.

In [112] wird u. a. die Compiler-seitige Auswahl von vielversprechenden Instruktionssatzerweiterungen für IPsec vorgestellt. Eine dieser Superinstruktionen ist der Befehl *XORLDW*, der die beiden Einzeloperationen *LDW* und *XOR* des Originalprozessorkerns zusammenfasst. D. h. es wird ein Wort aus dem Speicher geladen und direkt im Anschluss eine *XOR*-Verknüpfung mit einem weiteren Registerinhalt durchgeführt. Dieses Operationspaar konnte mit Hilfe der übersetzerbasierten Werkzeugkette als besonders vielversprechend für Prüfsummen- und Verschlüsselungsverfahren klassifiziert werden, vgl. Abbildung 6-6.

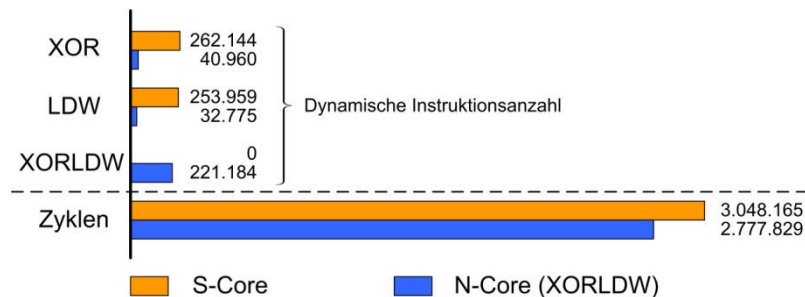


**Abbildung 6-6: Analyseergebnisse der IPsec-Protokollfunktion 3DES in Bezug auf vielversprechende Superinstruktionen, mit dem LDW-XOR-Paar als Favoriten**

Die schaltungstechnische Realisierung und Analyse dieser „Superinstruktion“ wird in [111] beschrieben. Zur Verwirklichung der Funktionalität waren in diesem Fall lediglich zusätzliche Kontrollstrukturen im Instruktionsdekoder des N-Cores einzufügen, da die benötigte Logik bereits im S-Core vorhanden war. Durch diese Superinstruktion kann eine Beschleunigung von ca. 10% bei Verschlüsselungsverfahren [111] und von ca. 25% bei Prüfsummenbildungen, wie z. B. beim *CRC* (*Cyclic Redundancy Check*) [130], erzielt werden. Abbildung 6-7 zeigt die Gesamtzahl der benötigten Zyklen für die Abarbeitung der 3DES-Verschlüsselungsfunktion<sup>40</sup> unter Verwendung des bestehenden Befehlsatzes, und alternativ dazu, mit der *XORLDW*-Superinstruktion, nebst den Häufig-

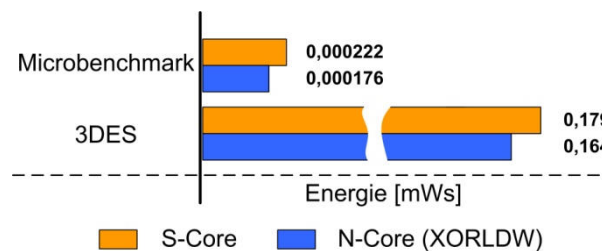
<sup>40</sup> Der 3DES(*Data Encryption Standard*)-Algorithmus hat mit 168 Bit eine dreimal größere Schlüssellänge als der ursprüngliche *DES* mit 56 Bit. *DES* und 3DES werden derzeit neben *AES* (*Advanced Encryption Standard*) in vielen sicherheits-relevanten Anwendungen zur Verschlüsselung eingesetzt.

keiten der Einzeloperationen. *3DES* findet z. B. bei *IP-Security*-Protokollen Verwendung<sup>41</sup>. Hierbei ist ein merklicher Rückgang der benötigten Zyklen zur Abarbeitung des Programms mit 9% zu beziffern. Bezogen auf die Gesamtfläche des Prozessorkerns beschränkt sich die zusätzlich benötigte Fläche zur Realisierung dieses Superbefehls auf 0,3 % bei 130-nm-Standardzellentechnologie bzw. 0,8 % bei 90-nm-Standardzellentechnologie, vgl. Abbildung 6-9.



**Abbildung 6-7: Zyklenanzahl des *3DES*-Algorithmus im *IPSec*-Protokoll vor und nach der *XORLDW*-Befehlssatzerweiterung**

In Abbildung 6-8 wird die Reduzierung des Energiebedarfs beider Prozessordesigns verdeutlicht. Betrachtet man zunächst einen *Mikrobenchmark*, bei dem nur die Leistungsfähigkeit der Superinstruktion gegenüber den ursprünglichen Einzeloperationen verglichen wird, so zeigt sich bei einer Laufzeitverkürzung von 33,3 % eine Energieersparnis<sup>42</sup> von 20,7 %. Beim *3DES*-Algorithmus lassen sich durch die Hinzunahme dieses einen Spezialbefehls neben der Laufzeitverkürzung um 9 % nahezu 9% an Energie einsparen.



**Abbildung 6-8: Energiebedarf des ursprünglichen S-Cores verglichen mit dem um die *XORLDW*-Superinstruktion erweiterten N-Core in Bezug auf zwei Benchmarkszenarien**

Ein weiterer positiver Nebeneffekt dieser Instruktionssatzerweiterung ist die Reduktion der Codegröße um 6% für die *3DES*-Anwendung. Waren es beim S-Core noch 5924 Bytes, so werden beim *XORLDW*-erweiterten N-Core nur noch 5572 Bytes an Instruktionsspeicher benötigt. Die eingesparten 352 Bytes On-Chip-SRAM entsprechen einer Fläche von ca. 9240µm<sup>2</sup>, was wiederum dem 75-fachen der Fläche für die Realisierung des Superbefehls (bezogen auf die 130-nm-Technologie) entspricht.

8 KB SRAM (0,21mm<sup>2</sup>) nehmen ca. 11,1 mW bei 250MHz auf, was in etwa vergleichbar mit der Leistungsaufnahme des N-Cores ist, so dass auch hier bei Einsparung von Instruktionsspeicher bzw.

<sup>41</sup> Das dem Benchmark zugrundeliegende Verkehrsmodell der IP-Pakete entspricht dem iMIX.

<sup>42</sup> Die Ergebnisse basieren auf der genauen Leistungsaufnahmeanalyse mittels Schaltaktivitätenannotation, wie sie in Abschnitt 5.3 vorgestellt wurde.



bei Reduktion der Anzahl der Speicherzugriffe die Ressourceneffizienz des Systems zusätzlich erhöht werden kann.

#### 6.2.4 Implementierte anwendungsspezifische Instruktionen

Im Rahmen der durchgeführten Instruktionssatzanalysen bzgl. der vorgestellten Netzwerkanwendungen wurden weitere Superinstruktionen für den N-Core realisiert. Dies betrifft neben der bereits detailliert beschriebenen *XORLDW*-Superinstruktion die Befehlssatzerweiterungen *LDWXORLSL8*, *ANDSHR*, [*ORSHL8*, *ORSHL16*, *ORSHL24*], *LDWADDI*, *IXWANDSHR* und *LDWIXW*, die ebenfalls aus [112] hervorgegangen sind und ursprünglich speziell für *IPSec*-Protokolle vorgesehen waren. Die jeweiligen Funktionsweisen der einzelnen Operationen werden in Anhang D erläutert. Zusätzlich sei auf die detaillierte Kommentierung der VHDL-Beschreibung verwiesen. Die Namen der Befehle wurden so gewählt, dass sie bereits einen ersten Eindruck von der jeweiligen Funktion geben. Die Auswertung der Performanzsteigerung durch diese zusätzlichen Instruktionen für Anwendungen aus dem Netzwerkbereich erfolgt in Kapitel 7.

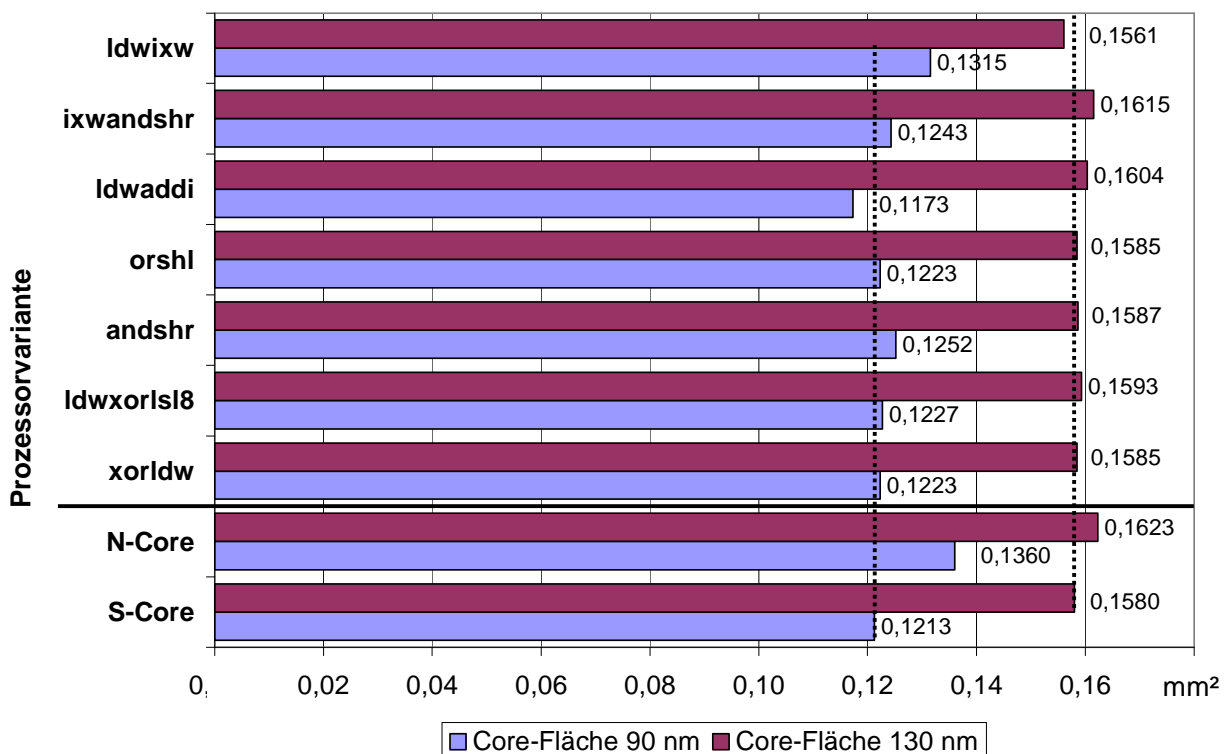


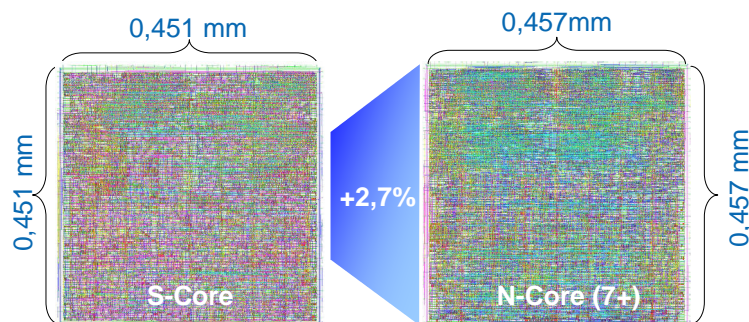
Abbildung 6-9: Gegenüberstellung des Flächenbedarfs in [mm²] von S-Core vs. N-Core in 130- und 90-nm-Standardzellentechnologie sowie für Prozessorvarianten einzelner Befehlssatzerweiterungen

Abbildung 6-9 liefert eine Gegenüberstellung des Flächenbedarfs von S-Core vs. N-Core (7+: umfasst die vorgestellten sieben Superinstruktionen) in 130- und 90-nm-Standardzellentechnologie sowie Werte für die Realisierungen der einzelnen Befehlssatzerweiterungen. Bei der Realisierung wurde sichergestellt, dass der kritische Pfad des Originalprozessorkerns nicht durch die jeweiligen Erweiterungen vergrößert wurde. Interessant ist die Tatsache, dass bei der 130-nm-Realisierung die *LDWIXW*-Variante und bei der 90-nm-Technologie die *LDWADDI*-Implementierung geringfügig kleiner als der unveränderte S-Core sind. Dies liegt in dem heuristischen Vorgehen des Synthesewerkzeugs begründet. Aufgrund der veränderten Struktur des Prozessorkerns schlägt der Synopsys-Design-Compiler eine teilweise vollkommen andere Struktur für einige Bereiche des Prozessors



vor, die dennoch die erforderliche Taktfrequenz von 257 MHz für 130 nm bzw. 270 MHz für 90 nm erfüllt. Aufgrund der unterschiedlichen Herangehensweise des Synthesewerkzeugs kann es so zu diesen Flächenunterschieden im einstelligen Prozentbereich kommen. Der prozentuale Flächenzuwachs für alle Instruktionssatzerweiterungen beträgt für die 90-nm-Technologie 12,1 % bzw. lediglich 2,7 % für die 130-nm-Technologie verglichen mit dem Original-S-Core.

Für die 130-nm-Standardzellentechnologie wurde für den um die genannten Superinstruktionen erweiterten N-Core exemplarisch eine Synthese mit anschließender Platzierung und Verdrahtung der erforderlichen Standardzellen sowie der notwendigen, vom Platzierungswerkzeug eingefügten *Filler Cells*, durchgeführt.



**Abbildung 6-10: Größenzuwachs des Prozessors durch die sieben zusätzlichen Instruktionen für eine 130nm-Standardzellentechnologie (Darstellung beruht auf dem erzeugten GDS-II-Plot<sup>43</sup>)**

Abbildung 6-10 zeigt einen Größenzuwachs von 2,7% des Prozessors bei Hinzunahme der sieben vorgestellten Instruktionssatzerweiterungen. Dies bedeutet durchschnittlich 0,39% Flächenzuwachs pro Superinstruktion. Der hier implementierte, N-Core-Prozessorkern besitzt eine Kantenlänge von 0,457 mm und eine Gesamtfläche von 0,209 mm<sup>2</sup> in der verwendeten 130-nm-Standardzellentechnologie.

Sollte ein anderer Prozessorkern als der N-Core als IP-Block zur Verfügung stehen und für das entsprechende Anwendungsgebiet deutlich leistungsfähiger sein, kann dieser aufgrund der offenen Schnittstellen und der Parametrisierbarkeit der GigaNetIC-Architektur ebenfalls leicht in das System integriert werden. Für besonders rechenintensive Aufgaben reichen Instruktionssatzerweiterungen allein meist nicht aus. Anwendungsspezifische Hardwarebeschleuniger können in diesem Fall den Prozessor bzw. die Prozessoren merklich entlasten. Für diese Art der Erweiterung/Optimierung ist die GigaNetIC-Architektur besonders vorbereitet, vgl. Abschnitt 4.3.3. Im Folgenden wird exemplarisch die Optimierung der GigaNetIC-Architektur durch anwendungsspezifische Hardwarebeschleuniger anhand eines Beispiels näher erläutert. Weitere realisierte Hardwarebeschleuniger werden kurz vorgestellt und bzgl. ihrer Leistungsfähigkeit charakterisiert.

### 6.3 Optimierung: Hardwarebeschleuniger auf Cluster- und SoC-Ebene

Bereits in Abschnitt 4.3.3 wurden die verschiedenen Möglichkeiten, anwendungsspezifische Hardwarebeschleuniger in das GigaNetIC-System zu integrieren, vorgestellt. Bevor ein spezieller Hardwarebeschleuniger implementiert wird, wird zunächst die entsprechende Anwendung, die den Ein-

<sup>43</sup> GDS II (Graphic Data System) Format, in 2004 der de Facto Standard für Layoutdaten der Chipfertigung.

satz einer solchen Spezialhardware ggf. erfordern könnte, mit einer der Simulationsumgebungen höherer Abstraktionsebene analysiert (vgl. Abbildung 6-2). Stellt sich heraus, dass die Verarbeitungseinheit die geforderte Leistung nicht erbringen kann, und zeigt sich weiterhin, dass Befehlsatzweiterungen ebenfalls nicht genügend Leistungszuwachs liefern, so kann eine Spezialhardware mit der benötigten Performanz implementiert werden. Hierbei wird darauf geachtet, dass diese Hardware ebenfalls möglichst flexibel gehalten ist, so dass sie auch für zukünftige Anwendungen bzw. Weiterentwicklungen bestehender Standards eingesetzt werden kann und somit ein Höchstmaß an Zukunftssicherheit garantiert. Mittels der in Kapitel 5 vorgestellten Entwicklungsumgebungen kann eruiert werden, an welcher Stelle im GigaNoC die entsprechende Hardwareerweiterung platziert werden sollte. Bandbreiten- und Rechenleistungsbedarf der Anwendung bestimmen Dimensionierung, Platzierung und Anzahl des jeweiligen Beschleunigers (vgl. Kapitel 7).

Im folgenden Abschnitt werden exemplarisch das Konzept und der Entwurf eines Hardwarebeschleunigers zur Protokollverarbeitung dargestellt. Der grundsätzliche Aufbau und die Funktionsweise zur Ansteuerung durch die Prozessoren des GigaNetIC-Systems, die auch die Steuerung der Lastverteilung übernehmen, werden hier verdeutlicht. Das vorgestellte Prinzip kann auf beliebige Hardwarebeschleuniger anderer Funktionalität übertragen werden. Die einheitlichen Schnittstellen und Mechanismen der GigaNetIC-Architektur verkürzen so die Entwicklungszeit, erhöhen die Wiederverwertbarkeit und steigern so letztendlich die Ressourceneffizienz.

### 6.3.1 Optimierungspotential von Hardwarebeschleunigern – ein Beispiel

In diesem Abschnitt soll, exemplarisch für alle implementierten Hardwarebeschleuniger, auf einen flexiblen Hardwarebeschleuniger zur Verarbeitung von Paketdaten und Prüfsummen zum Einsatz in kommunizierenden eingebetteten On-Chip-Systemen eingegangen werden. Solche Einheiten werden häufig auch als „TCP/IP-Offload Engine“ bezeichnet, da sie die Prozessoren von wesentlichen und besonders rechenintensiven Aufgaben der Protokollverarbeitung entlasten. Der hier vorgestellte Hardwarebeschleuniger und dessen Arbeitsweise wurde 2005 zusammen mit Infineon Technologies CPR ST zum Patent angemeldet [159].

In Kapitel 7 wird die Leistungsfähigkeit der GigaNetIC-Architektur im Hinblick auf Netzwerkverarbeitungsszenarien untersucht. Für besonders rechenintensive Algorithmen dieses Anwendungsbereichs wird die Architektur, dem hierarchischen Optimierungsansatz folgend, durch Instruktionssatzweiterungen und Hardwarebeschleuniger erweitert. Eine sehr häufig auftretende und zudem rechenintensive Funktion bei der Verarbeitung von Netzwerkpaketen ist die Generierung und Überprüfung von Prüfsummen nach dem Muster der „Internet Checksumme“ [160]. Bei der Evaluierung dieser Funktion im Rahmen der Definition des *DSLAM(Digital Subscriber Line Access Multiplexer)*-Referenzbenchmarks [141] für Infineon Technologies, die u. a. auch als Funktion im *EEMBC (Embedded Microprocessor Benchmark Consortium)*-Netzwerkbenchmark [161] vorkommt, zeigte sich, dass Bedarf für eine Beschleunigung dieser Funktionalität besteht. Nachdem Softwareoptimierung und Befehlsatzweiterungen allein nicht die benötigte Performanz liefern konnten, wurde die Implementierung eines Hardwarebeschleunigers notwendig [141][159][118].

#### 6.3.1.1 Hardwarebeschleuniger zur Protokollverarbeitung – Motivation und Funktion

Vernetzte Systeme kommunizieren auf der Basis von Netzwerkprotokollen über den Austausch von Paketen. Pakete bestehen aus Nutzdaten und zusätzlichen Protokollinformationen, die unter anderem die Weiterleitung (Quell- und Zieladressen) durchs Netzwerk ermöglichen und die Integrität

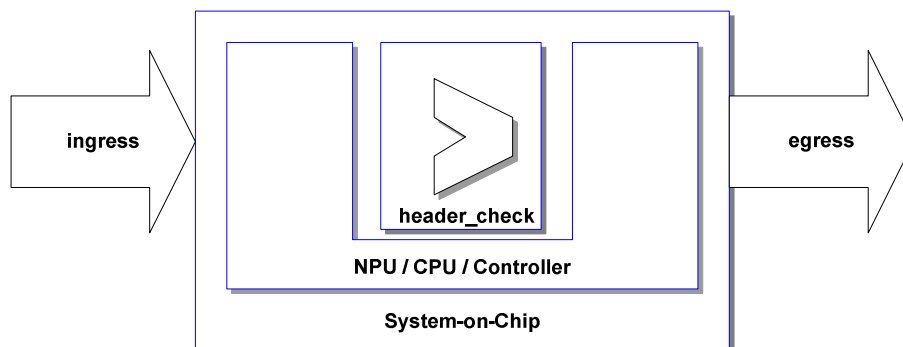
des Pakets gewährleisten (Prüfsummen, *Time-to-live*). Jeder Knoten im Netzwerk muss, wenn er die Pakete verarbeiten will, diese Informationen prüfen und ggf. vor der Weiterleitung modifizieren. Für die Verarbeitung von Netzwerkdaten kommt eine Vielzahl von Hardwarerealisierungen in Frage. So können anwendungsspezifische Bausteine (*ASICs*) auf Netzwerkkarten (*NICs*) die Verarbeitung übernehmen, aber auch Standard-CPU's werden hierzu teilweise verwendet. Im Laufe der letzten Jahre haben sich, vor allem für Einsatzgebiete mit hoher Datenlast, spezielle Netzwerk-Prozessoren (*NPU's*) etabliert. Dies sind besonders auf das Anwendungsgebiet Paketverarbeitung spezialisierte Hochleistungseinheiten. Diese programmierbaren Bausteine erlauben eine sehr flexible Gestaltung der notwendigen Verarbeitungsschritte. Dies ist im Hinblick auf den permanenten Wandel der Datenbeschaffenheit und der verwendeten Protokolle und der damit verbundenen Dienste von großem Vorteil. Mittlerweile gibt es eine Vielzahl etablierter Protokolle und Mechanismen, die aufgrund ihrer extrem häufigen Verwendung zu einem fixen Bestandteil der Netzwerktechnologie geworden sind. Hierzu zählen u. a. das *Internet-Protokoll Version 4 (IPv4)* auf der Netzwerkschicht (*Layer* 3) sowie viele der in diesem Protokoll gekapselten *Layer-4*-Protokolle (wie z. B.: *TCP, UDP, ICMP* etc.). All diesen Protokollen gemein ist die verwendete Prüfsumme, die auf dem 16-Bit-Einer-Komplement-Summen-Ansatz beruht [160].

Durch das Auslagern dieser Prüfsummenberechnung auf einen speziell für diese Aufgabe optimierten und im Rahmen des Notwendigen variabel gehaltenen Hardwarebeschleuniger lässt sich der Datendurchsatz eines Netzwerkknotens steigern bzw. die bisher eingesetzte Verarbeitungseinheit (*CPU, NPU*) entlasten oder sogar durch eine kostengünstigere, weniger leistungsstarke Verarbeitungseinheit ersetzen. Eingesetzt werden kann der im weiteren Verlauf beschriebene Beschleuniger also nahezu in jedem IP-verarbeitenden Netzwerkknoten, angefangen bei Endgeräten im *CPE(Customer Premises Equipment)*-Bereich (*Firewalls, NICs* etc.) bis hin zu Geräten, die im Kernnetzwerk angesiedelt sind, wie z. B. *Router*. Die Verarbeitungsleistung bei all diesen Geräten sollte möglichst hoch sein, um hohe Systemlasten aufgrund der ständig steigenden Datenraten im weltweiten *IP*-Verkehr über eine möglichst lange (*Time-in-Market*-)Zeitspanne zu unterstützen. Hierzu bietet es sich an, besonders rechenintensive Verarbeitungsschritte, wie z. B. Prüfsummenberechnungen, auf dedizierte Hardwareblöcke auszulagern. Dies erlaubt eine Verarbeitung der Pakete mit der erforderlichen „Leitungsgeschwindigkeit“ (*Linespeed*), auch in stark belasteten Netzwerkknoten. Ein weiterer Vorteil eines HW-Beschleunigers für diese Arbeitsschritte liegt in dem deutlich geringeren Flächenbedarf und der geringeren Leistungsaufnahme und führt somit zu einer Reduktion der Kosten (sowohl Initial- als auch Betriebskosten).

Abbildung 6-11 zeigt die prinzipielle Kopplung des Hardwarebeschleunigers mit der übergeordneten Kontrolleinheit (*NPU, CPU, Controller*). Hierbei ist es grundsätzlich erst einmal unerheblich, ob der Beschleuniger selbst über Speicher verfügt, oder dieser ihm über eine Bus- oder Netzwerk-on-Chip(*NoC*)-Anbindung zugänglich gemacht wird.

Ein Einsatzbereich des GigaNetIC-System-on-Chips liegt in der Verarbeitung und Weiterleitung von Netzwerkverkehr auf *Layer 3 (Internet Protocol)* und höher, vgl. Abschnitt 8.1.2. Um eintreffende Pakete (*Ingress*) verarbeiten bzw. weiterleiten zu können, müssen diese zunächst auf Korrektheit, also protokollkonformes Format überprüft werden. Dies geschieht durch den HW-Beschleuniger, der Pakete prüft, die mit einer auf dem 16-Bit-Einer-Komplement-Summen-Ansatz beruhenden Prüfsumme [160] gesichert sind. Im Anschluss übergibt er die Daten der übergeordneten Kontrolleinheit zur weiteren Bearbeitung. Soll ein Paket versendet werden (*Egress*), so wird

dem HW-Beschleuniger wiederum die Startadresse des zu bearbeitenden Pakets mitgeteilt, so dass im Folgenden der HW-Beschleuniger autonom per *Direct Memory Access (DMA)* die benötigten Paketdaten vom Speicher anfordern und die einzutragende Prüfsumme berechnen kann. Das Berechnen und Abspeichern der Prüfsumme erfolgt ohne weiteres Eingreifen der übergeordneten Kontrolleinheit, die so entlastet wird und so für die übergeordneten Kontrollaufgaben zur Verfügung steht. Außerdem erfolgt die Berechnung der Prüfsumme durch den HW-Beschleuniger in der Regel deutlich schneller, als dies eine Standard CPU zu leisten in der Lage wäre. Des Weiteren ist der HW-Beschleuniger in der Lage, die zulässige Verbleibenszeit *TTL (Time-To-Live)* im Netzwerk des Pakets zu prüfen und ggf. zu dekrementieren, was für Einsatzgebiete innerhalb von Routern etc. eine weitere Beschleunigung ermöglicht.



**Abbildung 6-11: Grobe Darstellung des Moduls und prinzipielle Kopplung mit Systemumgebung**

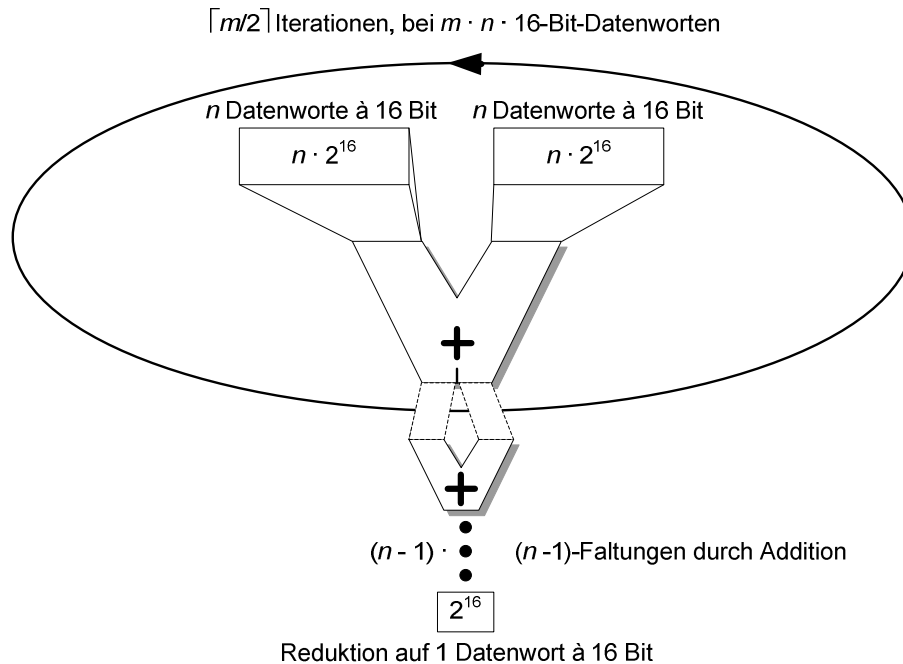
Der IP-Headercheck unterstützt sowohl *Little-* als auch *Big-Endian*-Systeme und kann im Betrieb auf die entsprechende Systemkonfiguration umgestellt werden. Dies erhöht die Interoperabilität, da er mit Prozessoren unterschiedlichster Hersteller einfach zu kombinieren ist. Er prüft die *IP-Version* und ist bereits vorbereitet auf *IPv6 (Internet Protocol Version 6)* [162]. Er unterstützt variable Header- und Paketlängen sowie eine gerade und ungerade Anzahl von Bytes im Paketrahmen. Aufgrund der einheitlichen und einfach gehaltenen Schnittstelle können unterschiedlichste Busprotokolle und proprietäre Schnittstellen unterstützt werden. Zur Reduktion des Chip-internen Kommunikationsaufwands kann bei der bloßen Überprüfung der Checksumme nur eine Ergebnisrückgabe der berechneten Checksumme erfolgen, ohne ggf. das gesamte Paket über das NoC zurückzuleiten.

Die Vorteile der vorgestellten Lösung im Vergleich zu existierenden Lösungen (vgl. z. B. [163]) liegen in einer sehr performanten Realisierung bei hinreichender Flexibilität. Diese beschleunigte Verarbeitung liegt zum einen an der Tatsache, dass es sich um anwendungsspezifisch optimierte Hardware handelt, die höhere Taktfrequenzen im Vergleich zu vielen Standard-Prozessoren erlaubt, zum anderen an dem speziellen, erweiterten Berechnungsverfahren zur Prüfsummenverarbeitung (vgl. Abschnitt 6.3.1.2). Weiterhin lässt sich die vorgestellte Lösung durch ein Steuerwort konfigurieren und parametrisieren, so dass verschiedene Verarbeitungsschritte (vgl. Abbildung 6-13) angestoßen werden können. Zur vereinfachten SoC-Integration ist das vorliegende Modul parametrisierbar in Bezug auf Datenbreite und die Art der Ankopplung an eine Kontrolleinheit. Die flexibel gehaltene Integration ins Gesamtsystem stellt eine universell einsetzbare Plattform zur schnellen Prüfsummenprüfung zur Verfügung (vgl. Abschnitt 6.3.1.4).

### 6.3.1.2 Performanzerhöhung mittels eines Algorithmus variabler Bitbreite

Der hier entwickelte Algorithmus weicht von der 16-Bit-Variante [160] ab und erlaubt die Variation der Additionsbreite zur zusätzlichen Beschleunigung der Verarbeitung. Die entwickelte Methode

zur Prüfsummenberechnung nutzt die Invarianz der Berechnungsvorschrift gegenüber der Größe des verwendeten Restklassenrings. Aufgrund der abschließenden Faltung durch fortgesetzte Addition von Teilergebnissen im Restklassenring  $2^{16}$  ist das Prüfsummenergebnis invariant gegenüber der Größe  $2^{16n}$  (mit  $n > 1$ ) des Restklassenrings, der zur Berechnung der Teilergebnisse verwendet wurde. Diese Teilberechnungen können somit nicht nur mit 16-Bit-breiten Datenwörtern sondern auch mit Vielfachen von 16-Bit-breiten Datenwörtern durchgeführt werden. Abbildung 6-12 zeigt schematisch die Funktionsweise des Algorithmus variabler Bitbreite.



**Abbildung 6-12:** Schematische Darstellung der Operation auf Restklassenringen von  $2^{16}$

Im Folgenden wird ein Realisierungsbeispiel für *IP/TCP/UDP*-Netzwerkprotokolle dargestellt, indem zunächst die Systemfunktion und im Anschluss daran die Anbindung an ein übergeordnetes System beschrieben wird. Die beschriebene Lösung gilt grundsätzlich auch für andere Protokolle, die auf dem 16-Bit-Einer-Komplement-Summen-Ansatz beruhen, d.h. z. B. auch für *ICMP*, *IGMP*, *ST-II*, *EGP*, *HMP*, *IRTP*, *OSPF*, *NETBLT*, *ENCAP*, *OSPFIGP* und ähnliche. Dies erhöht die Wiederverwendbarkeit des Hardwarebeschleunigers und damit auch die Ressourceneffizienz.

### 6.3.1.3 IP-Headercheck – Aufbau und Funktionsweise

Die wesentlichen Funktionsblöcke des Hardwarebeschleunigers für eine 32-Bit-breite Variante werden in Abbildung Anhang E-1 dargestellt. Der Hardwarebeschleuniger umfasst eine im Befehlsumfang und in Datenbreite parametrisierbare Arithmetisch/Logische Einheit (*ALU*), Ergebnisregister, Kontrollregister, Adressregister sowie diverse Register zum Zwischenspeichern temporärer Werte. Die Datenbreite dieser Einheiten kann je nach Ausführung des Hardwarebeschleunigers variieren, da die in Abschnitt 6.3.1.2 vorgestellte Berechnungsmethode auch andere Berechnungsbreiten zulässt, wobei sich die jeweiligen Realisierungen des Hardwarebeschleunigers im prinzipiellen Aufbau nicht von Abbildung Anhang E-1 unterscheiden.

Die Ansteuerung des Hardwarebeschleunigers geschieht zum einen über ein Kontrollregister (vgl. Abbildung 6-13), in dem die verschiedenen Operationsmodi über *Steuerflags* ausgewählt werden. Die Gültigkeit der anliegenden Steuerflags wird über eine logische Eins von *ctrl\_in\_en* signalisiert. Zum anderen muss die Verarbeitung der Beschleunigungseinheit noch durch das Signal

*check\_enable* aktiviert werden, das von außen durch eine übergeordnete Kontrolleinheit (CPU, NPU, Zustandsmaschine) angelegt wird. Ebenso muss dem HW-Beschleuniger noch die Startadresse (im Speicher) des zu bearbeitenden Pakets übermittelt werden, die im Adressregister abgelegt wird. Die für die Prüfsummenbildung benötigte Ablaufsteuerung ist durch einen generischen Zustandsautomaten (*Finite Statemachine / FSM*) realisiert. Dieser übernimmt die komplette Ansteuerung der einzelnen Komponenten des Hardwarebeschleunigers sowie die autonome Ansteuerung der Speicher- bzw. Busschnittstelle. Ist die Verarbeitung des Pakets gemäß der durch die gesetzten Kontrollflags vorgesehenen Arbeitsschritte abgeschlossen, werden die Ergebnisse an die dafür vorgesehenen Speicherstellen zurückgeschrieben. Im Kontrollregister werden die sich aus der Bearbeitung ergebenden Flags gesetzt. Das Signal *check\_ready* zeigt der übergeordneten Kontrolleinheit die Fertigstellung der Verarbeitung an, so dass ggf. ein neues Paket zur Verarbeitung in Auftrag gegeben werden kann.

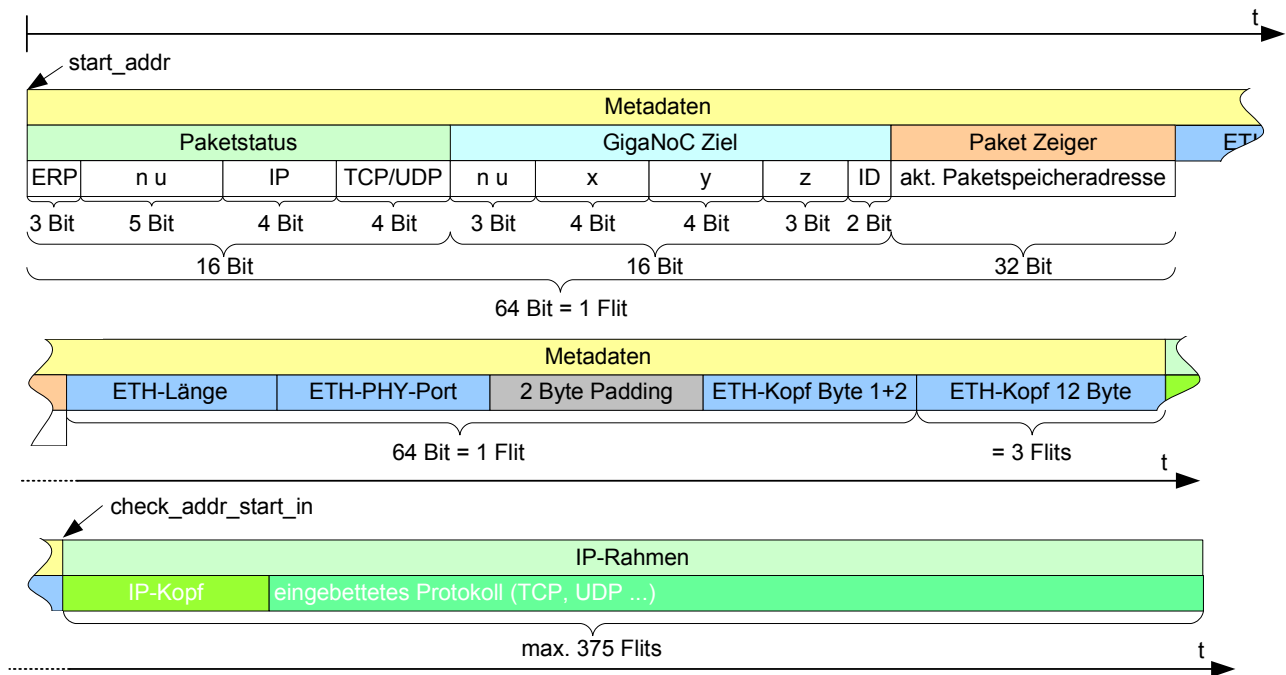
Kontrollregister																
Paketstatus																
x	x	x	nicht benutzt				IP			TCP/UDP						
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	
Big / Little Endian	Ergebnis / ges. Paket	Paket wird verarbeitet	nicht benutzt				dekrementiere TTL	korrekt / nicht korrekt	Check / Compute	Bearbeiten ja / nein	TCP / UDP	korrekt / nicht korrekt	Check / Compute	Bearbeiten ja / nein		
1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	
		x							x	x		x	x		x	

x = Modifikation durch Headercheck-Einheit

x = nur in Wishbone-Kopplung benutzt

**Abbildung 6-13: Implementierung des Headercheck-Kontrollregisters**

Der prinzipielle Ablauf einer Prüfsummenbildung für das Internet-Protokoll (*Layer 3*) und ggf. ein enkapsuliertes *Layer-4*-Protokoll ist in Abbildung Anhang E-2 dargestellt. Wesentlich ist die Unterscheidung zwischen Überprüfung (*Check*) und Neuberechnung (*Compute*) der Prüfsummen. Der Check-Modus kommt z. B. beim Empfang eines Pakets zum Einsatz, bei dem zunächst überprüft werden muss, ob es sich um ein gültiges, korrekt übermitteltes Paket handelt. Die Korrektheit eines eingegangenen Pakets wird durch eine logische Null am *packet\_error*-Ausgang angezeigt. Eine logische Eins würde ein fehlerbehaftetes Paket kennzeichnen, das ggf. verworfen wird. Beim Versenden käme die Neuberechnung der Prüfsummen zum Einsatz. Sollten Veränderungen an den Daten bzw. Headerinformationen wie z. B. das *TTL*-Feld, das ebenfalls bei Bedarf automatisch von der Einheit dekrementiert werden kann, vorgenommen worden sein, ist eine Neuberechnung der Prüfsumme unabdingbar.



**Abbildung 6-14: Protokollrahmen zur GigaNoC-basierten Ansteuerung des Headercheck-Hardwarebeschleunigers**

Abbildung 6-14 zeigt einen GigaNoC-Paketrahmen zur Ansteuerung eines Headercheck-Hardwarebeschleunigers, der lose-gekoppelt über einen Communication-Controller an einen Port einer Switch-Box angeschlossen ist. Prinzipiell kann der Ablauf auch auf andere Hardwarebeschleuniger übertragen werden. Der zu sendende Protokollrahmen setzt sich in diesem Fall aus sogenannten Metadaten und den eigentlichen Nutzdaten zur Verarbeitung zusammen. Die Metadaten beinhalten alle notwendigen Kontrolldaten zur Ansteuerung des Hardwarebeschleunigers, zur Wegewahl und zur Speicheransteuerung. Beim Headercheck-Hardwarebeschleuniger bestehen die Nutzdaten aus dem gesamten *IP*-Rahmen, der sowohl den *IP*-Kopf, als auch Daten von ggf. enkapsulierten Protokollen höherer Schichten beinhalten kann. Die ersten 16 Bit der Metadaten umfassen den Paketstatus, der direkt in das Kontrollregister (vgl. Abbildung 6-13) des Beschleunigers geschrieben wird. Die folgenden zwei Byte sind die Adressierungsdaten für die GigaNoC-Wegewahl. Anschließend folgt der Paketzeiger mit der jeweiligen Speicheradresse. Zusätzlich werden noch die Daten des Ethernetkopfes angehängt, so dass ein Paket, das vom Beschleuniger verarbeitet worden ist, nicht zwangsläufig zu einem übergeordneten Prozessor geleitet werden muss, sondern ggf. direkt zum Communication-Controller eines integrierten Ethernetcontrollers geleitet werden kann. Ebenso können die Communication-Controller der Ethernetcontroller die am GigaNoC angeschlossenen Beschleuniger direkt adressieren, ohne zwangsläufig Prozessoren mit einzubeziehen. Dies hängt von der jeweiligen Anwendung und der Funktion des Beschleunigers ab. Beim *IP*-Headercheck-Beschleuniger werden in diesem Fall Pakete dem Kontrollwort entsprechend verarbeitet und in Abhängigkeit des Ergebnisses verworfen oder weitergeleitet. Eine Erweiterung des Funktionsumfangs ist auch hier möglich und hängt letztendlich von der Anwendung ab.

#### 6.3.1.4 *IP*-Headercheck – Systemanbindung

Der *IP*-Headercheck-Hardwarebeschleuniger eignet sich für alle in Abschnitt 4.3.3 vorgestellten Kopplungsarten an das GigaNoC. In Abbildung 6-15 wird die Variante der Systemanbindung mit lokalem Speicher vorgestellt. Hierbei kann die Prüfsummeneinheit entweder über einen lokalen Bus



oder aber über das On-Chip-Netzwerk mit der übergeordneten Kontrolleinheit verbunden sein. In diesem Fall dient ein Dualport-Speicher als lokaler Paketspeicher. Für diese Kopplung wurde eine Anbindung des Prüfsummenmoduls sowohl über den Wishbone-Bus als auch über ein NoC (Giga-NoC) exemplarisch implementiert und verifiziert. Die Integration der Einheit erfolgt über einen Wrapper, der die Schnittstellenkonvertierung übernimmt.

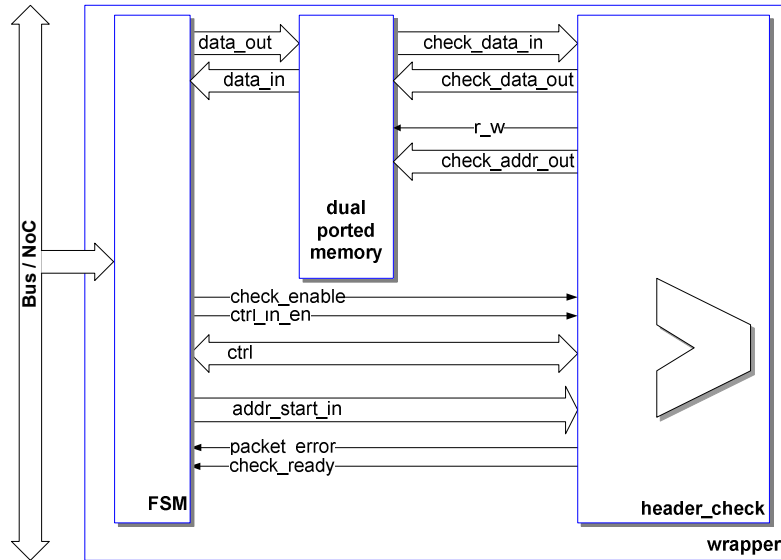


Abbildung 6-15: Systemanbindung über Bus oder NoC, mit lokalem Speicher

In der zweiten Variante, in Abbildung 6-16, erfolgt die Systemanbindung über einen Bus bzw. ein On-Chip-Netzwerk unter Verwendung eines gemeinsamen Speichers. Für die Anbindung an einen Bus bietet der HW-Beschleuniger die Option, die *Burstlänge* des Buszugriffs einzustellen, so dass für die Bearbeitungszeit auch andere Teilnehmer den Bus nutzen können und somit Blockierungen weitestgehend verhindert werden können.

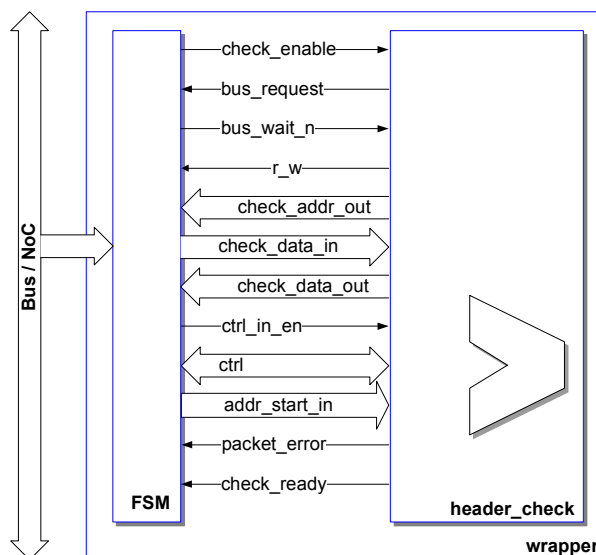


Abbildung 6-16: Systemanbindung über Bus oder NoC, mit gemeinsamem Speicher

Die dritte Variante in Abbildung 6-17 zeigt die Anbindung der Beschleunigereinheit an eine übergeordnete CPU/Kontrolleinheit unter Verwendung eines gemeinsamen Speichers. Die Steuerung des Datenflusses wird von der übergeordneten Kontrolleinheit übernommen.



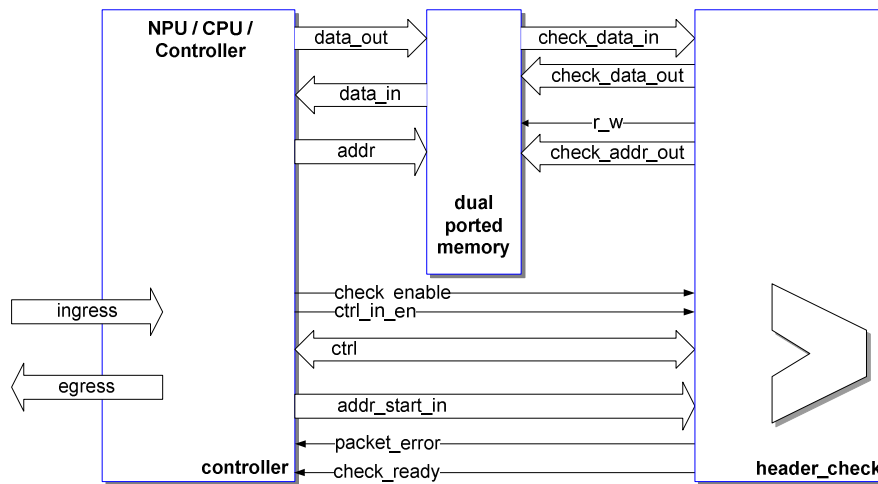


Abbildung 6-17: Anbindung des IP-Headercheckers über eine eng-gekoppelte Kontrolleinheit

Tabelle 6-3 zeigt den Flächenbedarf der einzelnen Implementierungsvarianten der 32-Bit-Variante inklusive der benötigten Schnittstellen zur Kopplung an das Gesamtsystem.

Tabelle 6-3: Flächenbedarf der 32-Bit-Variante des IP-Headercheck-Hardwarebeschleunigers in 130/90-nm-Standardzellentechnologie, inkl. NoC-Schnittstelle und 16 KB DPRAM auf SoC-Ebene

PE-Ebene		Fläche [mm <sup>2</sup> ] Cluster-Ebene		SoC-Ebene	
130 nm	90 nm	130 nm	90 nm	130 nm	90 nm
0,0357	0,0228	0,0299	0,0196	0,6649	0,5609

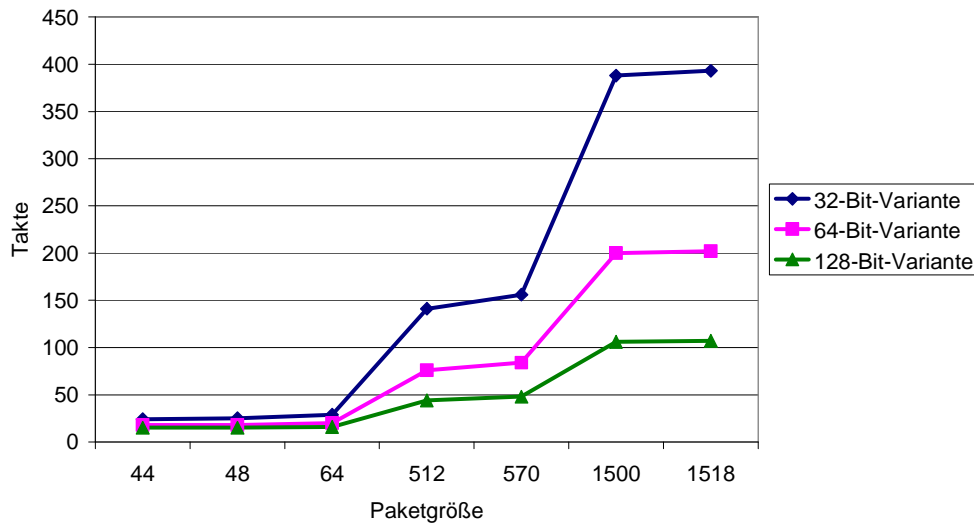
### 6.3.1.5 IP-Headercheck – Leistungsdaten

Die hier vorgestellte Ausnutzung der Tatsache, dass die Berechnungsvorschrift für die Prüfsummenbildung invariant zu Vielfachen von 16-Bit-Restklassenringen ist, erlaubt die Verarbeitung/Addition der Pakete in Vielfachen dieser 16-Bit-Halbworte und ermöglicht so einen höheren Durchsatz von Paketen bei gleich bleibender Taktrate der Verarbeitungseinheit (vgl. Abschnitt 6.3.1.2). Die Funktionalität des HW-Beschleunigers für größere Datenbreiten ist prinzipiell gleich, wobei sich die Ausführungszeit in erster Näherung (ggf. abhängig von der jeweiligen Hardwarerealisierung) reziprok zur Datenbreite der Hardware verhält<sup>44</sup>. Die Datenbreite des HW-Beschleunigers wird im Wesentlichen durch die Speicher- bzw. Bus-Anbindung bestimmt (vgl. Abschnitt 6.3.1.4).

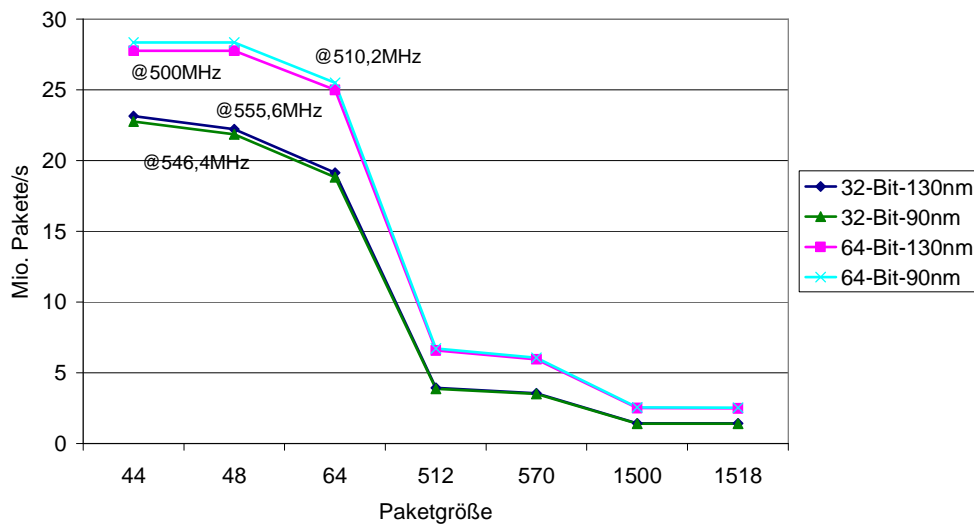
Abbildung 6-18 zeigt die Leistungsfähigkeit für die 32-, 64- und 128-Bit-Variante des IP-Headercheck-Hardwarebeschleunigers für enkapsulierte *Layer-4*-Protokolle in Abhängigkeit von der Paketgröße. Es zeigt sich bei großen Paketen deutlich die nahezu Verdoppelung bzw. Vervierfachung der Performanz gegenüber der 32-Bit-Variante bei Verwendung der 64- bzw. 128-Bit-breiten Implementierung. Bei minimal-großen Paketen mit einer Nutzdatenmenge von 44 Byte erfolgt lediglich eine Reduzierung von 24 auf 18 bzw. 15 Takten durch die breitere Verarbeitung. Be-

<sup>44</sup> **Implementierungsabhängige Berechnungsdauer:** Die 32-Bit-Variante benötigt  $\left\lceil \frac{B}{4} \right\rceil + 13$  Takte, mit  $B$  = Anzahl der zu verarbeitenden Bytes, die 64-Bit-Variante  $\left\lceil \frac{B}{8} \right\rceil + 12$  und die 128-Bit-Variante  $\left\lceil \frac{B}{16} \right\rceil + 12$  Takte pro *Layer-4*-Protokollrahmen.

zogen auf das spätere Anwendungsszenario ist von Fall zu Fall abzuwägen, ob der Einsatz einer leistungsfähigeren Variante des Hardwarebeschleunigers notwendig ist, oder ob die Beschleunigung, die durch die flächenmäßig kleinere 32-Bit-Variante (vgl. Abbildung 6-20) erreicht wird, bereits ausreicht. Speziell Anwendungen die hauptsächlich Pakete minimaler Länge einsetzen, wie *VoIP (Voice over IP)* etc., profitieren weniger von der Leistungsfähigkeit der IP-Headercheck-Hardwarebeschleuniger größerer Bitbreite. Im Sinne der Ressourceneffizienz wäre hier der 32-Bit-breite Beschleuniger ein pareto-optimaler Punkt im Sinne von Definition 12.



**Abbildung 6-18: Leistungsdaten des IP-Headercheck-Hardwarebeschleunigers in Abhängigkeit von der Verarbeitungsbreite**



**Abbildung 6-19: IP-Headercheck-Hardwarebeschleuniger – Performanz 32- vs. 64-Bit-Variante, bezogen auf die Synthesewerte**

Abbildung 6-19 zeigt die, sich aus den Synthesen ergebenden, realisierbaren Leistungswerte für eine 130-nm- und eine 90-nm-Standardzellentechnologie, in Abhängigkeit von der erreichbaren Maximalfrequenz in der jeweiligen Technologie. Es wird evident, dass die 64-Bit-Varianten deutlich leistungsfähiger als die 32-Bit-Implementierungen sind. Beide 64-Bit-Realisierungen erlauben Betriebsfrequenzen von über 500 MHz und verarbeiten 27,78 bzw. 28,34 Mio. Pakete à 44 Byte pro Sekunde. Das entspricht einer Nutzdatenmenge von mehr als 1220 MByte/s. Die 32-Bit-Varianten liegen fast 10 % höher bei den maximal erreichbaren Taktfrequenzen, als die der 64-Bit-Typen. So

können immerhin 22,77 bzw. 23,15 Mio. Pakete pro Sekunde verarbeitet werden. Die 64-Bit-Schaltungen haben aufgrund ihrer größeren ALU einen deutlich längeren kritischen Pfad, der diese Reduktion der maximal erreichbaren Taktfrequenz bewirkt. Bei der Paketgröße von 1518 Byte nivelliert sich der Anteil der fixen Berechnungskosten und die 64-Bit-Varianten erzielen eine deutlich höhere Effizienz als bei 44-Byte-Paketen. Hier liegt die pro Sekunde verarbeitbare Datenmenge bei 3757 bzw. 3834 MByte pro Sekunde, im Gegensatz zu 2146 bzw. 2110 MByte pro Sekunde, also ca. 80 % höher als bei der 32-Bit-Realisierung.

In [163] wird ebenfalls eine Verarbeitungseinheit zur Berechnung der *Internet Checksum* vorgestellt. Der beschriebene Funktionsumfang ähnelt dem des hier präsentierten Ansatzes allerdings nur zum Teil, da der GigaNetIC-Hardwarebeschleuniger parametrisierbar ist und so auch größere Datenbreiten als 32 Bit verarbeiten kann. Die Realisierung aus [163] liegt in einer 180-nm-Standardzellentechnologie vor und beansprucht eine Fläche von mindestens 0,171 mm<sup>2</sup> bei einer möglichen Taktfrequenz von 381 MHz. Skaliert man die Fläche mit Hilfe der S-Parameter (vgl. Kapitel 3, Definition 29) z. B. auf die 130-nm-Technologie des GigaNetIC-Hardwarebeschleunigers, so ergäbe sich eine wohlwollend abgeschätzte Fläche von:

$0,171\text{mm}^2 \cdot \left(\frac{180\text{nm}}{130\text{nm}}\right)^{-2} = 0,0892\text{mm}^2$ , was, verglichen mit den 0,0157 mm<sup>2</sup> [118] meiner Realisierung, mehr als dem 5,7-fachen der Fläche entspräche. Die Taktfrequenz ergäbe nach der Skalierung zu  $381\text{MHz} \cdot \left(\frac{180\text{nm}}{130\text{nm}}\right)^1 = 528\text{MHz}$ , was ebenfalls äußerst positiv geschätzt ist, betrachtet man z. B.

die Relationen zwischen den in dieser Arbeit erstellten 130-nm- und den 90-nm-Implementierungen. Gleichgesetzt mit dem Ansatz aus [163], erreicht der reine IP-Headercheck-Hardwarebeschleuniger des GigaNetIC-Systems ohne zusätzlichem *Wrapper* eine maximale Betriebsfrequenz von 1,69 GHz [118]. Dies stellt einen deutlichen Vorsprung um mehr als Faktor drei dar. Betrachtet man den erreichbaren Durchsatz beider Varianten, so liegt dieser nach der Notation aus [163] bei beiden Realisierungen bei 32 Bit pro Sekunde und ist damit lediglich abhängig von der maximalen Betriebsfrequenz. So resultierte daraus bei der GigaNetIC-Variante ein Geschwindigkeitsvorteil von ebenfalls Faktor 3,2. Angesichts dieser Werte wird deutlich, dass es sich bei dem GigaNetIC-IP-Headercheck-Hardwarebeschleuniger um eine äußerst ressourceneffiziente Implementierung handelt, deren spezieller Aufbau sie derart leistungsfähig macht. Ferner kann durch Verwendung der Varianten größerer Datenbreite der Durchsatz nochmals gesteigert werden.

Abbildung 6-20 zeigt die jeweils benötigte Fläche für die Realisierung des IP-Headercheck-Hardwarebeschleunigers in Abhängigkeit von der geforderten Taktfrequenz und Technologie. Es zeigt sich, wie zu erwarten war, dass, je höher die Anforderungen an die zu realisierende Taktfrequenz sind, desto mehr Fläche wird beansprucht. Bzgl. der 32-Bit-Variante in 90-nm-Technologie z. B. entspricht dies bei einer Steigerung der Frequenz um den Faktor 2,7 mehr als 68 % Flächenzuwachs. Die Realisierung der 32-Bit-breiten Variante des IP-Headercheck-Hardwarebeschleunigers entspricht in etwa einer Systemkomplexität von weniger als 6000 bzw. 14200 Gatteräquivalenten bei der 64-Bit-breiten Implementierung.

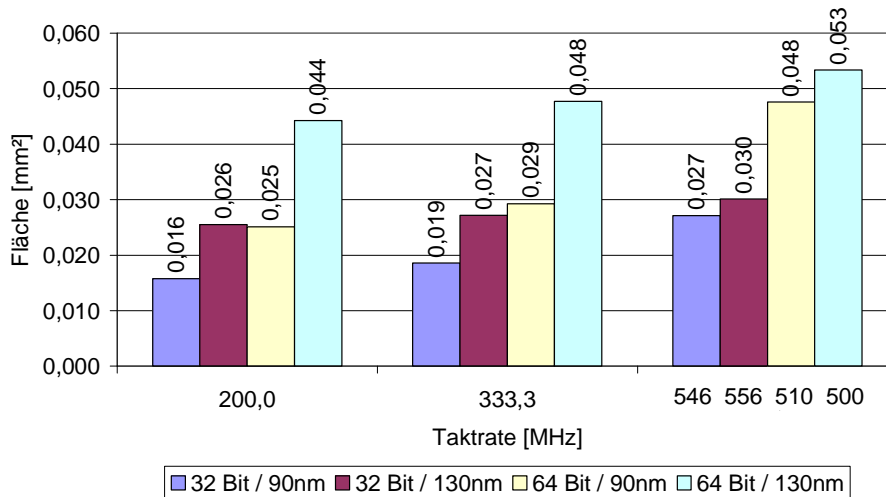


Abbildung 6-20: Flächenvergleich der verschiedenen IP-Headercheck-Implementierungen

Die hier präsentierten Werte zeigen, dass bei Integration der Beschleuniger in das GigaNetIC-System abgewogen werden sollte, welche Position für die Hardwarebeschleuniger gewählt wird. Bei einer genaueren Spezifikation der Zielapplikation können die zu erwartende Leistungsfähigkeit des Systems sowie die erforderliche Konfiguration mit Hilfe der vorgestellten Simulationsumgebungen (vgl. Kapitel 5) ermittelt werden. Sollte eine Kopplung an den lokalen Bus, der in der Regel eine geringere Frequenz aufgrund der angeschlossenen Prozessorelemente zulässt, genügen, so reicht die flächeneffizientere Implementierung aus. Erfordert die Spezifikation hingegen maximalen Durchsatz, so sollte eine lose Kopplung der leistungsfähigsten, aber auch flächenintensivsten Variante angestrebt werden. In diesem Fall ist der Betrieb mit einer deutlich höheren Taktfrequenz als die der lokalen Busse möglich. Die vorgestellten Kapselungen (*Wrapper*) können erforderlichenfalls die Umsynchronisierung auf den Takt des angeschlossenen Kommunikationsmediums übernehmen. Zur Steigerung der Ressourceneffizienz empfiehlt es sich, Überlegungen dieser Art stets anzustellen, falls dies im Vorfeld der Implementierung aufgrund genügender Informationen bzgl. des späteren Anwendungsszenarios möglich ist.

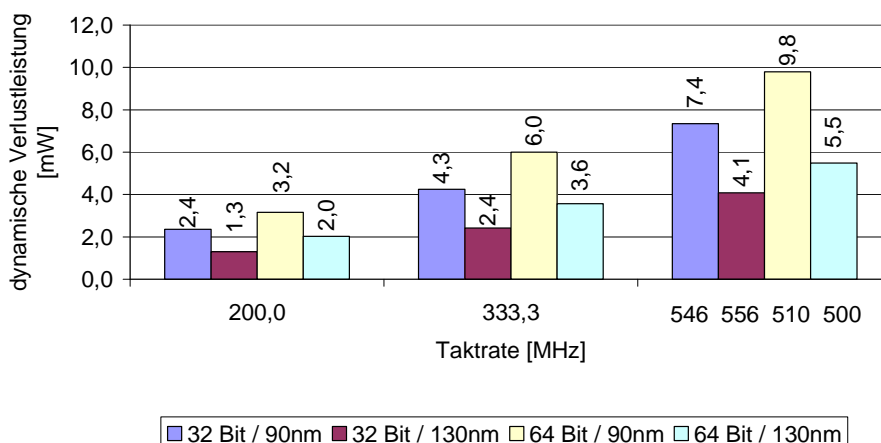
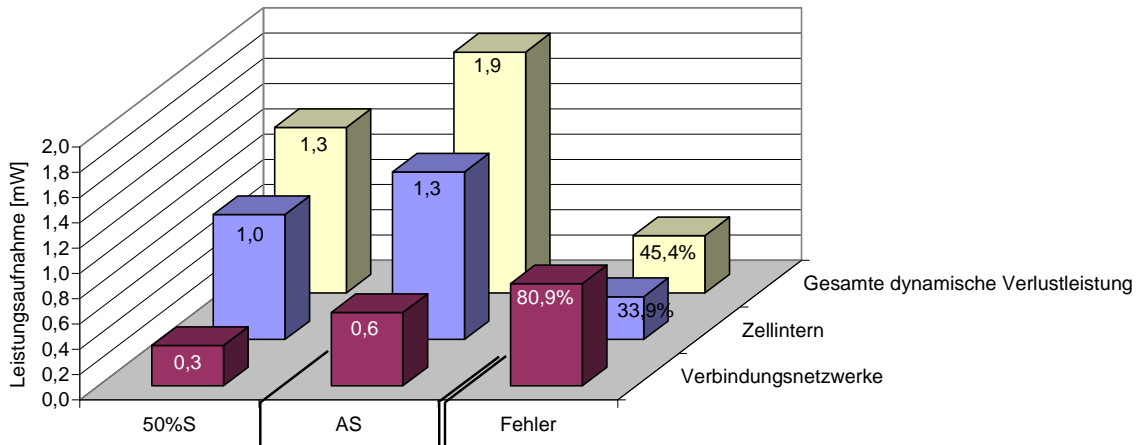


Abbildung 6-21: Vergleich Leistungsaufnahme der verschiedenen IP-Headercheck-Implementierungen in Abhängigkeit von der Betriebsfrequenz und der entsprechenden Realisierung

Ähnlich verhält es sich mit der Leistungsaufnahme und zwar aufgrund der linearen Abhängigkeit von der Frequenz, die in die dynamische Verlustleistung eingeht, und der zusätzlich größeren Fläche der schnelleren Varianten. Die entsprechenden Werte sind in Abbildung 6-21 dargestellt. Allerdings ist zu bemerken, dass diese Angaben auf angenommenen, statistischen Schaltwahrscheinlich-

keiten (**50%S**) des Synthesewerkzeugs beruhen. Hierbei wird eine 50-prozentige Schaltwahrscheinlichkeit der Eingänge angenommen, deren Verhalten sich dann auf die Folgelogik fortpflanzt. Dies spiegelt jedoch nicht die realen Stimuli der Schaltung wieder und kann so nur eine ungefähre Einschätzung der Leistungsaufnahme liefern.



**Abbildung 6-22: Verlustleistungsanalyse (für die 130-nm-Technologie) basierend auf statistischen Schaltwahrscheinlichkeiten (50%) der Gatter und durch Simulation gewonnener Schaltaktivitäten (AS)**

Mit Hilfe der in Abschnitt 5.3 vorgestellten erweiterten PERFMON-Umgebung wurde deshalb zum Vergleich die Leistungsaufnahme durch Annotierung der Schaltaktivitäten (**AS**) während der Laufzeit und der Bearbeitung der *iMix*-Lastverteilung (vgl. Abschnitt 7.2.3) bestimmt. Die Analyse bezieht sich auf die 130-nm-Standardzellentechnologie und die IP-Headercheck-Variante mit 200 MHz Taktfrequenz. Abbildung 6-22 stellt die gewonnenen Ergebnisse für die Realisierung im *Typical-Case (TC)* dar.

Die Fallunterscheidungen *Best Case*, *Typical Case* und *Worst Case* stellen drei unterschiedliche Umgebungsbedingungen der 130- und 90-nm-Standardzellentechnologie dar, die die Eigenschaften der Schaltung beeinflussen. Tabelle 6-4 zeigt die relevanten Parameter für die unterschiedlichen Syntheseparameter.

**Tabelle 6-4: Syntheseparameter für unterschiedliche Umgebungsbedingungen der Standardzellen**

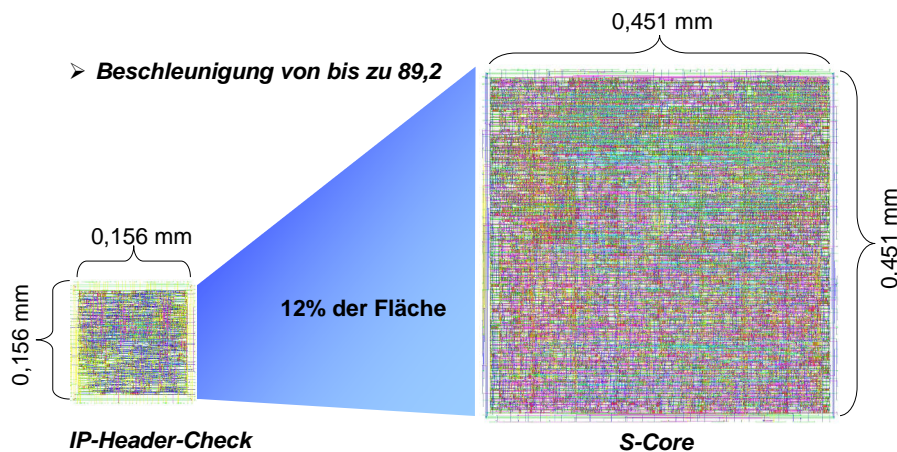
Syntheseparameter	Versorgungsspannung [V]		Umgebungstemperatur [°C]	
	130 nm	90 nm	130 nm	90 nm
Best Case BC	1,32	1,32	0	-40
Typical Case TC	1,2	1,2	25	27
Worst Case WC	1,08	1,08	125	125

Betrachtet man die Leistungsaufnahme für typische Umgebungsbedingungen, so zeigt sich, verglichen mit dem durch **50%S** ermittelten Wert, bei der durch **AS** ermittelten Leistungsaufnahme ein doppelt so hoher Wert bzgl. der Umladevorgänge der Leitungen. Die durch Schaltvorgänge der Standardzellen hervorgerufene Verlustleistung ist um 33,9 % höher, wenn man die Werte mit Hilfe von **AS** abschätzt. Dieses Ergebnis ist dadurch zu erklären, dass durch wahllose Beschaltung der Eingänge bei der **50%S**-Methode die eigentliche Funktion des Hardwarebeschleunigers nicht wiedergespiegelt wird und somit auch die ermittelte Leistungsaufnahme nur eine erste Abschätzung sein kann. Die realistische, dynamische Verlustleistungsaufnahme unter typischen Bedingungen, die mit Hilfe der **AS**-Methode ermittelt wurde, ist um 45,4 % höher, als es die reine Abschätzung des Synthesewerkzeugs zunächst vermuten ließe. Die Verlustleistungsaufnahme des Hardwarebe-

schleunigers von 1,9 mW bei 200 MHz beträgt dennoch nur ein Fünftel der Leistungsaufnahme des N-Core-Prozessorkerns.

Abschließend lässt sich bemerken, dass, wenn es die Zeit und die Umstände (Anwendung muss bereits ausprogrammiert sein) erlauben, eine AS-basierte Ermittlung der dynamischen Verlustleistung des Systems mit Hilfe der in dieser Arbeit vorgestellten Werkzeugkette ratsam ist. Dies ist vor Allem dann der Fall, wenn bereits vor der Chiprealisierung Daten mit einer maximalen Abweichung vom fertigen Chip im einstelligen Prozentbereich (vgl. Abschnitt 5.3) benötigt werden. Für eine grobe Einstufung reicht die 50%S-Methode jedoch aus, liefert sie immerhin noch Werte die zumeist in der gleichen Größenordnung der späteren Schaltung liegen.

Abbildung 6-23 zeigt die Größe der 32-Bit-Variante des IP-Headercheck-Hardwarebeschleunigers im Vergleich zum S-Core nach der Platzierung und Verdrahtung der Standardzellen in 130-nm-Technologie. Mit einer Kantenlänge von nur 0,156 mm beansprucht er lediglich 12 % der Fläche des Prozessorkerns, wobei seine eigentliche Logikfläche durch Füllzellen (*Filler Cells*) um 34 % ansteigt, im Gegensatz zu nur 10 % Flächenzuwachs beim S-Core.



**Abbildung 6-23: IP-Headercheck-Hardwarebeschleuniger im Vergleich zum S-Core in einer 130-nm-Standardzellentechnologie (GDS-II-Plot)**

Die von mir implementierte Spezialhardware benötigt lediglich acht Taktzyklen für eine Überprüfung des *IP-Headers* inklusive der Berechnung der Checksumme und kann mit max. 1,7 GHz im *Typical Case* bzw. 1,1 GHz im *Worst Case* betrieben werden (130-nm-Standardzellentechnologie). Dies ermöglicht 213 Mio. bzw. 138 Mio. *Headerchecks* pro Sekunde im Vergleich zu 921 k bzw. 2,39 Mio. *Headerchecks* pro Sekunde, die vom S-Core-Prozessor bewältigt werden können. Dies bedeutet somit eine rein funktionale Beschleunigung von 89,2 (*Typical Case*) / 57,7 (*Worst Case*) bzw. 13,5, wenn die gleiche Taktfrequenz wie beim Prozessor angesetzt wird [118].

Weiterführende Informationen zu der Leistungsfähigkeit des IP-Headercheck-Hardwarebeschleunigers für dedizierte Anwendungsszenarien werden in Kapitel 7 und in [118][159][109][131] gegeben. Viele der in diesem Abschnitt angestellten Untersuchungen und Methoden, wurden ebenfalls auf die im Folgenden nur kurz vorgestellten, weiteren Hardwarebeschleuniger des GigaNetIC-Systems angewendet und zeigten größtenteils ähnlich positive Ergebnisse bzgl. Flächensparnis, Leistungsaufnahmereduktion und Performanzerhöhung.

## 6.4 Kostenanalyse am Beispiel einer Netzwerkanwendung

Die Vorgehensweise bei der Kostenbetrachtung und der sich ggf. anschließenden Systemoptimierung wurde in Abbildung 3-3 vorgestellt. Im Folgenden sollen die unterschiedlichen Lösungen zur Verarbeitung von *IP*-/Netzwerk-Paketen nach der geschilderten Vorgehensweise exemplarisch einer Kostenbetrachtung nach Definition 9, Kapitel 3, unterzogen werden, und zwar ähnlich der in Abschnitt 7.7 geschilderten Anwendung und u. a. auf den zuvor realisierten Hardwarebeschleunigern aufbauend.

Am Anfang der Kostenanalyse steht die Festlegung der Zielfunktionen  $ZF$  (vgl. Definition 6) der einzelnen Kostenmaße  $\mathbb{K}$  (vgl. Definition 5). Hierzu müssen die relevanten Bewertungsmaße  $BM$  (vgl. Definition 4) im Hinblick auf die Anwendung und die damit verbundenen Randbedingungen  $R$  (vgl. Definition 7) und Schranken  $S$  (vgl. Definition 8) festgelegt werden. Diese bestimmen dann sowohl die Gewichtungen  $c_i$  der ausgewählten Bewertungsmaße  $BM_i$  in  $ZF$  als auch die Gewichtungen  $\alpha_p, \alpha_A, \alpha_T$  und  $\alpha_F$  der Kostenmaße  $\mathbb{K}$  in der Kostenfunktion  $CF$ . Bei der Wahl der Gewichtungen wird gemäß Abschnitt 3.1 folgender Zusammenhang gewählt:  $\sum_i c_i = \sum_i \alpha_i = 1$ , so dass

die relevantesten Bewertungsmaße  $BM_i$  bzw. Kostenmaße  $\mathbb{K}_i$  mit dem größten  $c_i$  bzw.  $\alpha_i$  bedacht werden. Um einen einheitenlosen Kostenwert zu erhalten, sind die Gewichtungen  $c_i$  mit der reziproken Einheit des jeweiligen Bewertungsmaßes  $BM_i$  versehen.

Die jeweiligen Werte der Bewertungsmaße  $BM_i$  der Realisierungsvarianten  $RV_i$  (vgl. Definition 11) müssen ermittelt werden und ergeben nach Einsetzen in die Kostenfunktion  $CF_{RV_i}$  letztendlich die Kosten der jeweiligen Realisierung. Da der hier entwickelte Ansatz im Sinne der *Pareto-Optimierung*  $CF_{pareto}(RV_i) = \mathbf{min!}$  (vgl. Definition 10) eine Minimierung der Kosten vorsieht, ist die Realisierung mit den geringsten Kosten als vielversprechendste Lösung anzusehen.

Zunächst gilt es also die Randbedingungen und Bewertungsmaße festzulegen. Im Falle der Funktion zur Prüfung von Paketdaten (vgl. Abschnitt 6.3.1.1) gibt es folgende Anforderungen, die nach den vier Kostenmaßen  $P, A, T$  und  $F$  (vgl. Definition 5) aufzuschlüsseln sind:

Die Anwendung ist relevant für alle beteiligten Netzwerkteilnehmer, sowohl mobile, als auch Hochleistungssysteme im Kernnetzwerk, d. h. hier sollte, wenn möglich, näher definiert werden, wo das System eingesetzt werden soll, da die Fläche  $A$  und die Leistungsaufnahme  $P$  in Abhängigkeit vom Einsatzort unterschiedlich stark gewichtet werden. Die Prüffunktion ist besonders rechenintensiv und bearbeitet teilweise sehr zeitkritische Daten, so dass die Performanz  $T$ , in gewissen Grenzen abhängig vom Einsatzort, eine wichtige Rolle einnimmt. Die Funktion ist fest definiert und ändert sich für die aktuellen Protokolle nicht mehr, was dem Kostenmaß Zukunftssicherheit bzw. Flexibilität  $F$  eine geringe Gewichtung zuweist. Die Wiederverwendbarkeit wird bei diesem Kostenmaß noch am stärksten gewichtet, da davon ausgegangen wird, dass aufgrund der Standardisierung der gegebenen Anwendung auch zukünftige Systeme auf diese Einheiten aufbauen werden.

Beispielhaft sollen nun die Realisierungsvarianten der Paketprüffunktion  $RV(\text{PaketPrüfung})_i$  für ein mobiles Endgerät für u. a. hochprioräre Datendienste (z. B. *Voice over IP*) mittels der vorgestellten Methode analysiert werden. Die Festlegung der einzelnen Gewichtungen ist in gewissen Grenzen sicherlich subjektiv und bedarf zumeist eingehender Diskussion seitens aller am Entwurfs- und



Vermarktungsprozess beteiligten Stellen, dennoch lassen sich bereits anhand grober Festlegungen erste Abschätzungen für potentielle Lösungsvarianten treffen.

Zunächst erfolgt die Festlegung der Zielfunktionen  $ZF$  der einzelnen Kostenmaße  $\mathbb{K}$ , wobei die jeweiligen Bewertungsmaße  $BM$  in additiver Form (3.4) miteinander verknüpft werden. Als Bewertungsmaß für die Leistungsaufnahme  $\mathbf{P}$  wird  $BM_P = P_{dyn}$  gewählt, also die dynamische Verlustleistung. Die statische Verlustleistung wird in diesem Zusammenhang aufgrund der verwendeten Technologie vernachlässigt. Die dynamische Verlustleistung setzt sich aus den Leistungsanteilen der eigentlichen Verarbeitungseinheit ( $PE$ ), dem evtl. benötigten Controller ( $Ctrl$ ), dem verwendeten Speicher ( $Mem$ ) und der Kommunikationsstruktur ( $Com$ ) zusammen. Es wird folgende Zielfunktion (6.1) für  $\mathbf{P}$  angesetzt:

$$ZF : \mathbf{P} = c_{dyn,PE} \cdot P_{dyn,PE} + c_{dyn,Ctrl} \cdot P_{dyn,Ctrl} + c_{dyn,Mem} \cdot P_{dyn,Mem} + c_{dyn,Com} \cdot P_{dyn,Com} + c_{stat} \cdot P_{stat} \quad (6.1)$$

mit  $c_{dyn,PE} = 0,7$ ,  $c_{dyn,Ctrl} = 0,2$ ,  $c_{dyn,Mem} = c_{dyn,Com} = 0,05$ ,  $c_{stat} = 0$

Die eigentliche Verarbeitungseinheit ( $PE$ ) geht mit der größten Gewichtung in die Zielfunktion ein, ein evtl. zur Ansteuerung eines spezialisierten Hardwarebeschleunigers ( $PE$ ) benötigter Controller ( $Ctrl$ ) wird weniger stark gewichtet. Er kann zusätzliche Aufgaben verrichten und wird deshalb nicht zu 100 Prozent der Paketprüffunktion zugeordnet. In dem hier betrachteten Szenario wird der N-Core als Controller für den Hardwarebeschleuniger eingesetzt.

Für die Fläche  $\mathbf{A}$  wird die Zielfunktion (6.2) aufgestellt. Die Fläche  $\mathbf{A}_{PE}$  für die benötigte Verarbeitungseinheit wird mit der höchsten Gewichtung versehen, wohingegen sich die Flächen für den evtl. benötigten Controller ( $Ctrl$ ), den benötigten Speicher  $\mathbf{A}_{Mem}$  und für die zugrundeliegende Kommunikationsinfrastruktur  $\mathbf{A}_{Com}$  bei diesem Szenario für die einzelnen Implementierungen kaum unterscheiden und deshalb weniger stark gewichtet werden.

$$ZF : \mathbf{A} = c_{PE} \cdot \mathbf{A}_{PE} + c_{Ctrl} \cdot \mathbf{A}_{Ctrl} + c_{Mem} \cdot \mathbf{A}_{Mem} + c_{Com} \cdot \mathbf{A}_{Com} \quad (6.2)$$

mit  $c_{PE} = 0,8$ ,  $c_{Ctrl} = 0,1$ ,  $c_{Mem} = c_{Com} = 0,05$

Für die Performanz  $\mathbf{T}$  wird die Ausführungszeit  $T_{ex,PE}$  (vgl. Definition 18) der Verarbeitungseinheit ( $PE$ ) als Bewertungsmaß eingesetzt. Auch der Jitter  $J$  ist ein relevantes Maß für viele Netzwerkszenarien, kann allerdings bei den hier vorgestellten Realisierungsvarianten systembedingt vernachlässigt werden. Auch der Durchsatz  $D$  (vgl. Definition 19) ist ein weiteres wesentliches Bewertungsmaß der Performanz. Er spiegelt die Leistungsfähigkeit des gesamten Systems wider. Im Gegensatz zu der zuvor genannten Verarbeitungszeit berücksichtigt er zusätzlich die Leistungsfähigkeit der Kommunikationsstruktur und die Speicherlatenz. Dies führt zu der in (6.3) angegebenen Zielfunktion.

$$ZF : \mathbf{T} = c_{Tex,PE} \cdot \mathbf{T}_{Tex,PE} + c_J \cdot \mathbf{T}_J + c_D \cdot \mathbf{T}_D, \text{ mit } c_{Tex,PE} = 1, c_J = c_D = 0 \quad (6.3)$$

Im Zusammenhang mit dem Kostenmaß Zukunftssicherheit bzw. Flexibilität  $\mathbf{F}$  werden folgende Bewertungsmaße  $BM$  berücksichtigt: Die Wiederverwendbarkeit  $WV$  (vgl. Definition 36) ist für eine derart essentielle Funktion zur Prüfung von Netzwerkpaketen von besonderer Bedeutung. Der Einsatz in allen Bereichen des Netzwerks heute und in naher Zukunft erfordert eine gute Portierbarkeit der Realisierungsvariante auch auf neue Technologien. Aufgrund der Beschreibung in einer Hardwarebeschreibungssprache lassen sich die entworfenen Hardwarebeschleuniger relativ einfach auf andere Technologien portieren. Bei der reinen Softwarelösung ( $N\text{-Core SW}$ ) ist die Wiederver-



wendbarkeit noch höher einzustufen, da aufgrund der Realisierung der Funktionalität in der Hochsprache C eine noch leichtere, plattformübergreifende Portierung möglich ist.

Die Programmierbarkeit  $PG$  (vgl. Definition 37) nimmt in diesem Szenario einen weniger wichtigen Stellenwert ein, da sich aufgrund der festen Spezifikation des Algorithmus keine Änderungen der Funktionalität für die etablierten Protokolle ergeben. Lediglich für zukünftige Protokolle, die ggf. zusätzliche Operationen erfordern, wäre ein gewisses Maß an Flexibilität vorteilhaft.

Ein weiterer Aspekt ist die Fehlertoleranz  $FT$  (vgl. Definition 33), die für massiv parallele Systeme in zunehmendem Maße an Bedeutung gewinnt, allerdings für das hier betrachtete Szenario derzeit noch von nur geringer Signifikanz ist. Die Fehlertoleranz, die das betrachtete System ermöglicht, liegt in der softwaregestützten Kontrollfunktion des N-Cores begründet, der evtl. bei eigenem Fehlverhalten oder fehlender Rückmeldung seitens eines Hardwarebeschleunigers eingreifen kann. Diese Möglichkeit ist allerdings nur ein einfaches Mittel, so dass keine hohe Bewertung bzgl. dieses Gesichtspunktes gegeben werden kann. Die sich aus den obigen Betrachtungen ergebende Zielfunktion ist in (6.4) definiert.

$$ZF : \mathbf{F} = c_{WV} \cdot \mathbf{F}_{WV} + c_{PG} \cdot \mathbf{F}_{PG} + c_{FT} \cdot \mathbf{F}_{FT}, \text{ mit } c_{WV} = 0,8, c_{PG} = c_{FT} = 0,1 \quad (6.4)$$

Im Anschluss an die Definition der Zielfunktionen für die vier Kostenmaße kann nun die Kostenfunktion  $CF_{RV(\text{PaketPrüfung})i}$  zum oben geschilderten Anwendungsszenario aufgestellt werden. Die eingesetzten Werte der Bewertungsmaße jeder Realisierungsvariante  $RV(\text{PaketPrüfung})i$  ergeben die spezifischen Kosten der jeweiligen Lösung. Die Variante mit den geringsten Kosten repräsentiert die vielversprechendste Realisierung. In Bezug auf die betrachteten Ansätze stellt sie eine diskrete pareto-optimale Auswahl (nach Definition 13) dar.

Im Falle des oben geschilderten Anwendungsszenarios werden die Kostenfunktionen  $CF_{RV(\text{PaketPrüfung})i}$  (6.5) für die einzelnen Realisierungsvarianten aufgestellt. Die Festlegung der Zielfunktionsgewichtsfaktoren korrespondiert mit den Anforderungen des Einsatzortes und priorisiert Performanz  $T$  und Leistungsaufnahme  $P$ , gefolgt von der Fläche  $A$ . Die Flexibilität  $F$  nimmt mit einer Gewichtung von 0,05 in diesem Szenario nur eine untergeordnete Rolle ein. Zusätzlich wird aufgrund der differierenden Größenordnungen der einzelnen Zielfunktionen eine Normierung aller Zielfunktionen nach (3.10) vorgenommen.

$$CF_{RV(\text{PaketPrüfung})i} : \alpha_P \frac{ZF : \mathbf{P}_{RV(\text{PaketPrüfung})i}}{\max(ZF : \mathbf{P}_{RV(\text{PaketPrüfung})})} + \alpha_A \frac{ZF : \mathbf{A}_{RV(\text{PaketPrüfung})i}}{\max(ZF : \mathbf{A}_{RV(\text{PaketPrüfung})})} \\ + \alpha_T \frac{ZF : \mathbf{T}_{RV(\text{PaketPrüfung})i}}{\max(ZF : \mathbf{T}_{RV(\text{PaketPrüfung})})} + \alpha_F \frac{ZF : \mathbf{F}_{RV(\text{PaketPrüfung})i}}{\max(ZF : \mathbf{F}_{RV(\text{PaketPrüfung})})} \quad (6.5)$$

$$\text{mit } \alpha_P = 0,35, \alpha_A = 0,2, \alpha_T = 0,4 \text{ und } \alpha_F = 0,05$$

Tabelle 6-5 zeigt die Werte für die genannten Bewertungsmaße der einzelnen Realisierungsvarianten. Es wurden als Grundlage für die Analyse die in Abschnitt 6.3.1.5 dargelegten Synthesewerte des Hardwarebeschleunigers und die in Tabelle 8-3, Seite 224 präsentierten Werte aller weiteren Systemkomponenten verwendet. Ausgangspunkt des Hardwaresystems ist ein N-Core, entweder als Controller (*Ctrl*) zur Ansteuerung eines Wishbonebus-gekoppelten Hardwarebeschleunigers (*WB HW Acc*, vgl. Abschnitt 7.7) oder als Verarbeitungseinheit (*PE / N-Core SW*) nebst Speicher und Wishbonebus. Angenommen wird weiterhin, dass, aufgrund des Anwendungsszenarios, nur ein Pro-

zessor und ein Hardwarebeschleuniger am Bus aktiv sind. Es gelten somit die Werte für Kommunikation und Kalkulation aus Abbildung 7-21 a), wobei für die 64-Bit-Varianten aufgrund des 32-Bit-breiten Busses gleiche Taktzahlen für Kommunikation, aber entsprechend geringere Werte zur Kalkulation angesetzt werden (vgl. Abschnitt 6.3.1.5). Zu den Daten der Hardwarebeschleuniger werden zusätzlich die Werte der *Wrapper* (vgl. Tabelle 4-5) hinzugezählt, wobei zu bemerken ist, dass die 64-Bit-Variante ca. 1,5 mal so groß wie die 32-Bit-Variante ausfällt. Zur Betrachtung der Maximalfrequenzen des Hardwarebeschleunigers wurde hypothetisch angenommen, dass der N-Core bei gleichem Flächenbedarf ebenfalls mit den betreffenden Frequenzen betreibbar ist. In der Realität müsste auf eine andere CPU als Controller zurückgegriffen werden, die mit der jeweiligen Frequenz noch zu betreiben ist, vgl. Tabelle 2-3.

**Tabelle 6-5: Auflistung der kostenfunktionsrelevanten Daten der Realisierungsvarianten und der resultierenden Kosten**

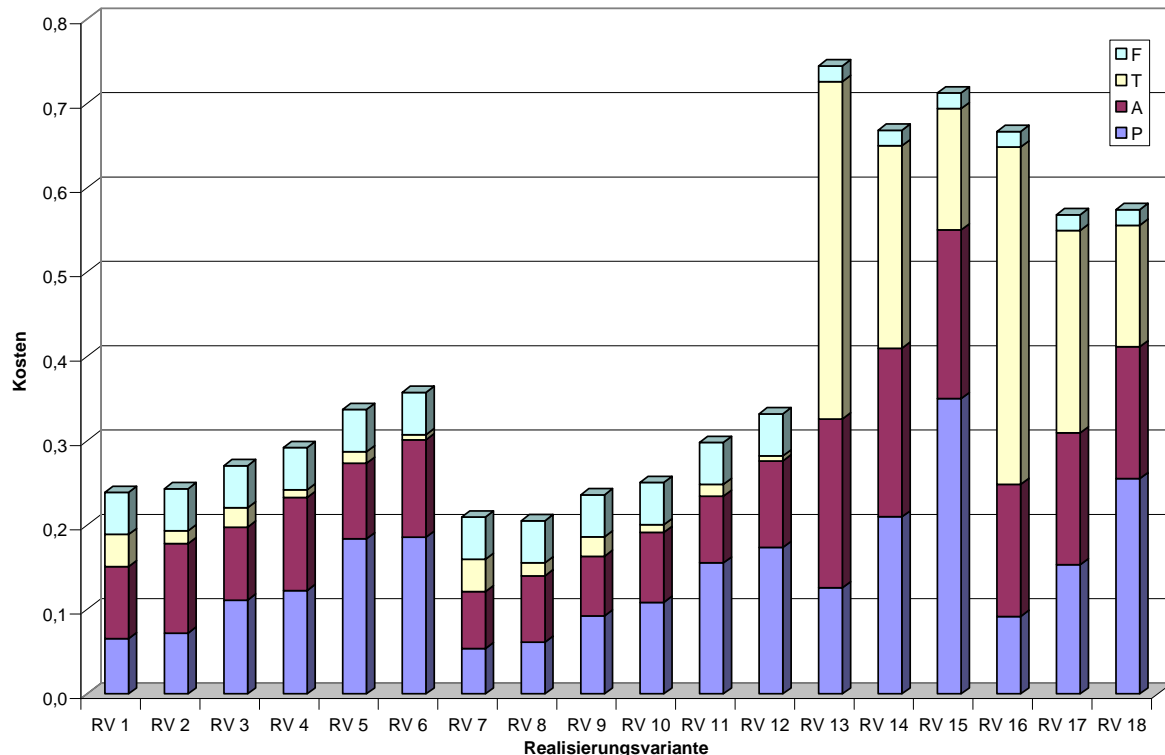
Realisierungsvariante		Bewertungsmaß											Kosten		
		$P_{dyn,PE}$ [mW]	$P_{dyn,Ctrl}$ [mW]	$P_{dyn,Mem}$ [mW]	$P_{dyn,Com}$ [mW]	$A_{PE}$ [mm <sup>2</sup> ]	$A_{Ctrl}$ [mm <sup>2</sup> ]	$A_{Mem}$ [mm <sup>2</sup> ]	$A_{Com}$ [mm <sup>2</sup> ]	$T_{Tex,PE}$ [μs]	$F_{wv}$ [%]	$F_{pg}$ [%]		$F_{ft}$ [%]	
RV 1	130nm	32 Bit / 200 MHz	1,300	10,800	35,600	0,010	0,030	0,160	0,466	0,050	9,295	80	5	5	0,239
RV 2		64 Bit / 200 MHz	2,000	10,800	35,600	0,010	0,050	0,160	0,466	0,050	3,675	80	5	5	0,243
RV 3		32 Bit / 333,3 MHz	2,400	17,998	59,327	0,010	0,031	0,160	0,466	0,050	5,578	80	5	5	0,271
RV 4		64 Bit / 333,3 MHz	3,600	17,998	59,327	0,010	0,054	0,160	0,466	0,050	2,205	80	5	5	0,292
RV 5		32 Bit / 556 MHz	4,100	29,484	97,188	0,010	0,034	0,160	0,466	0,050	3,344	80	5	5	0,337
RV 6		64 Bit / 500 MHz	5,500	27,000	90,780	0,010	0,059	0,160	0,466	0,050	1,470	80	5	5	0,357
RV 7	90nm	32 Bit / 200 MHz	2,400	9,000	10,200	0,006	0,020	0,120	0,466	0,020	9,295	80	5	5	0,210
RV 8		64 Bit / 200 MHz	3,200	9,000	10,200	0,006	0,030	0,120	0,466	0,020	3,675	80	5	5	0,205
RV 9		32 Bit / 333,3 MHz	4,300	14,999	16,998	0,006	0,023	0,120	0,466	0,020	5,578	80	5	5	0,236
RV 10		64 Bit / 333,3 MHz	6,000	14,999	16,998	0,006	0,034	0,120	0,466	0,020	2,205	80	5	5	0,251
RV 11		32 Bit / 546 MHz	7,400	24,570	28,356	0,006	0,031	0,120	0,466	0,020	3,405	80	5	5	0,298
RV 12		64 Bit / 510 MHz	9,800	22,950	28,356	0,006	0,053	0,120	0,466	0,020	1,441	80	5	5	0,332
RV 13	130nm	N-Core (SW) 200 MHz	10,800	0,000	35,600	0,010	0,160	0	0,466	0,050	97,475	95	100	10	0,744
RV 14		N-Core (SW) 333,3 MHz	17,998	0,000	59,327	0,010	0,160	0	0,466	0,050	58,491	95	100	10	0,668
RV 15		N-Core (SW) 556 MHz	30,024	0,000	98,968	0,010	0,160	0	0,466	0,050	35,063	95	100	10	0,712
RV 16	90nm	N-Core (SW) 200 MHz	9,000	0,000	10,200	0,006	0,120	0	0,466	0,020	97,475	95	100	10	0,667
RV 17		N-Core (SW) 333,3 MHz	14,999	0,000	16,998	0,006	0,120	0	0,466	0,020	58,491	95	100	10	0,568
RV 18		N-Core (SW) 556 MHz	25,020	0,000	28,356	0,006	0,120	0	0,466	0,020	35,063	95	100	10	0,574

Werte beziehen sich auf maximal realisierbare Taktfrequenz der Verarbeitungseinheit (PE),  
Werte des N-Cores wurden hypothetisch hochgerechnet

Abbildung 6-24 zeigt die sich nach (6.5) ergebenden Kosten sowie deren Zusammensetzung anhand der Anteile der einzelnen Zielfunktionen für alle Realisierungsvarianten. Für die oben aufgestellten Randbedingungen zeigt sich das System *RV 8* als das kostenoptimale in diesem Vergleich. Aufgrund der relativ hohen Anforderungen an Performanz und der zugleich geringen Leistungsaufnahme liefert die Kostenfunktion die geringsten Kosten für ein System mit dem schnellen 64-Bit-Hardwarebeschleuniger bei gleichzeitig geringster Taktfrequenz von 200 MHz in der 90-nm-Technologie. Die höheren Taktfrequenzen erhöhen die Kosten für die Verlustleistung, 32-Bit-breite Beschleuniger liefern weniger Performanz. Die Software-basierten Lösungen sind sowohl in puncto Verlustleistungsaufnahme als auch bzgl. der Performanz nicht konkurrenzfähig.

Abbildung 6-25 stellt die Kosten für ein Anwendungsszenario mit anderen Randbedingungen dar: Die Performanz wurde hier als die dominierende Charakteristik ausgewählt, die Gewichtungen, die dieser Analyse zugrunde liegen, lauten  $\alpha_P = 0,05$ ,  $\alpha_A = 0,05$ ,  $\alpha_T = 0,85$  und  $\alpha_F = 0,05$ . Die Gewichtungen innerhalb der einzelnen Zielfunktionen wurden identisch belassen. Ein derartiges Anforderungsschema wäre repräsentativ für z. B. Komponenten des Zugangsnetzwerks (vgl. Abschnitt 7.6), wobei ggf. höhere Anforderungen an die Fehlertoleranz bestünden. Unter den neuen Bedin-

gungen liefert ein Kostenvergleich die Realisierungsvariante *RV 10* als „pareto-optimale“ Lösung. Dieses System bietet eine hohe Performanz bei moderater Verlustleistung. Würde ein System mit nahezu maximaler Performanz gesucht und würden die Gewichtungen der Kostenfunktion zu  $\alpha_P = \alpha_A = \alpha_F = 0,01$  und  $\alpha_T = 0,97$  gesetzt, so qualifizierte sich *RV 12* als optimaler Kandidat.



**Abbildung 6-24: Kostenvergleich der unterschiedlichen Realisierungsvarianten zur Paketprüfung, bei einer Wahl der Gewichtungen zu  $\alpha_P = 0,35$ ,  $\alpha_A = 0,2$ ,  $\alpha_T = 0,4$  und  $\alpha_F = 0,05$**

Abbildung 6-26 zeigt die Ergebnisse einer anderen Spezifikation mit den Gewichtungen der Kostenfunktion  $\alpha_P = \alpha_F = 0,49$  und  $\alpha_T = \alpha_A = 0,01$ , wie sie für sehr zuverlässige Systeme mit beschränkten Ressourcen wie z. B. Satelliten oder Weltraumsonden zutreffen könnten. In diesem Fall wären Performanz und Flächenbedarf den beiden anderen Kostenmaßen stark untergeordnet, und die gegebenen Anforderungen würden von *RV 16* am besten erfüllt, einem prozessorbasierten System mit geringer Frequenz in der moderneren Technologie. Zusätzliche Impulse könnte die Einbeziehung der Fertigungskosten für die jeweilige Standardzellentechnologie geben, diese dürfen aber hier aus Gründen der Geheimhaltung nicht genannt werden.

In diesem Abschnitt wurde anhand beispielhafter Analysen die Anwendung des kostenfunktionsbasierten Auswahlverfahrens (vgl. Kapitel 3) zur Bestimmung eines diskreten pareto-optimalen Systems (nach Definition 13) für ein spezifiziertes Anwendungsszenario vorgestellt. Die Bewertung mit Hilfe der Kostenfunktionen unterstützt den Systemarchitekten, aber auch den kaufmännischen Bereich bei der Auswahl eines ressourceneffizienten Entwurfs. In den gezeigten Untersuchungen erscheinen die gefundenen Lösungen nach kurzer Diskussion bereits als die plausibelsten, dies ist aufgrund der gewollten Einfachheit des Beispiels nicht anders zu erwarten. Bei komplexeren Systemen, wie z. B. Chip-Multiprozessoren mit deutlich mehr Komponenten und diffizileren Anforderungen liegen die Lösungen selten so auf der Hand. Gerade hier kann eine formale Bewertung ihre Leistungsfähigkeit beweisen. Eine automatisierte Bewertung mit Hilfe dieser Methode wäre eben-

falls eine sinnvolle Erweiterung des in Abschnitt 7.5 vorgestellten *DSLAM-System-Explorers*. Die Integration der notwendigen Maßnahmen gestaltete sich zudem relativ einfach.

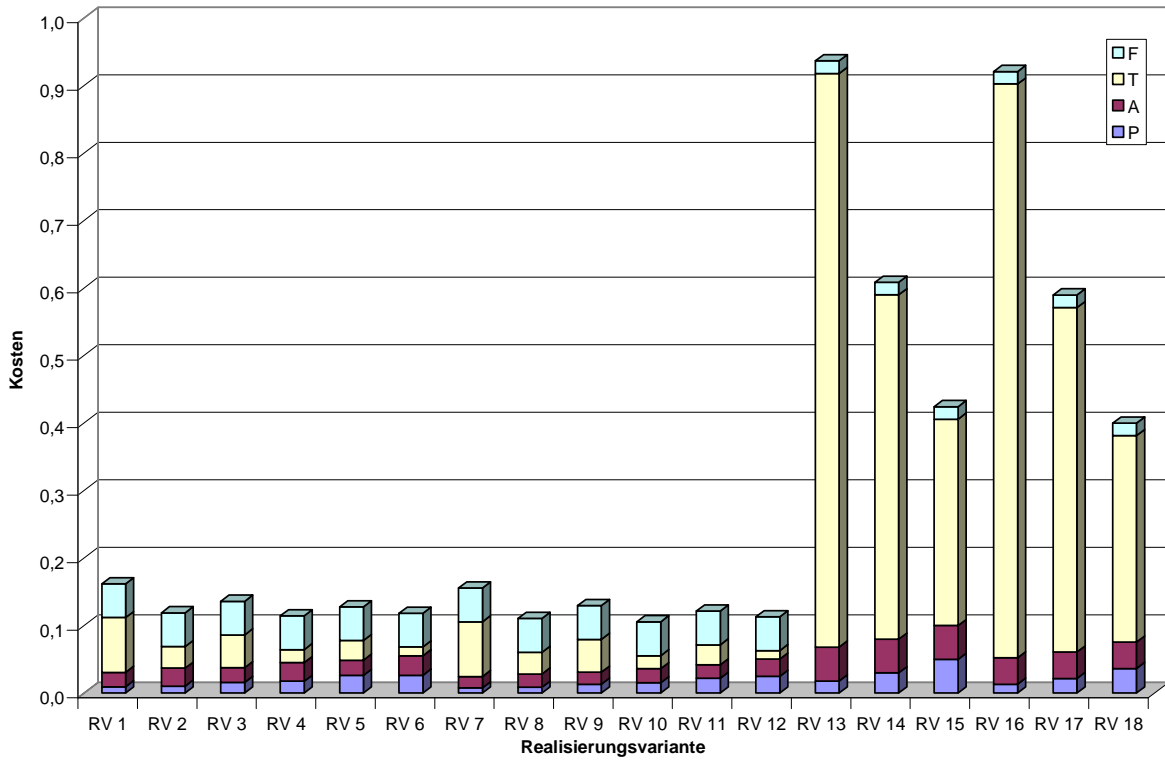


Abbildung 6-25: Kostenvergleich der unterschiedlichen Realisierungsvarianten zur Paketprüfung, bei einer Wahl der Gewichtungen zu  $\alpha_P = 0,05$ ,  $\alpha_A = 0,05$ ,  $\alpha_T = 0,85$  und  $\alpha_F = 0,05$

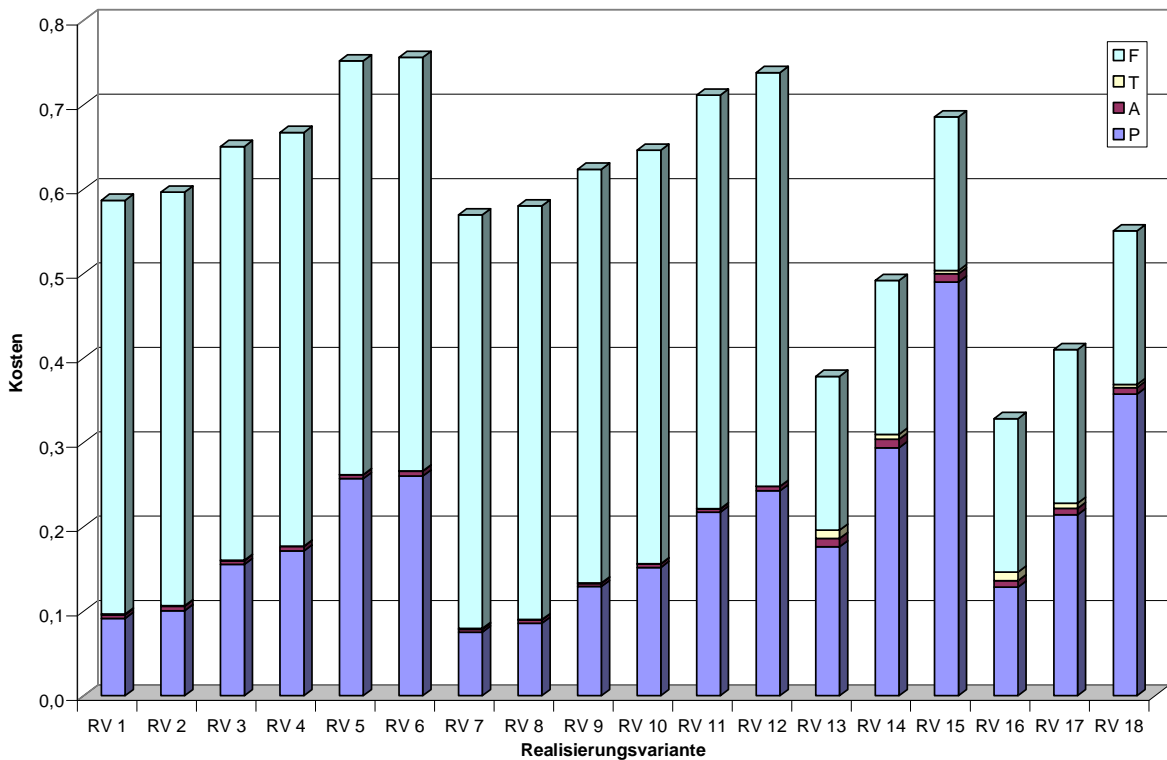


Abbildung 6-26: Kostenvergleich der unterschiedlichen Realisierungsvarianten zur Paketprüfung, bei einer Wahl der Gewichtungen zu  $\alpha_P = \alpha_F = 0,49$  und  $\alpha_T = \alpha_A = 0,01$

## 6.5 Implementierte anwendungsspezifische Hardwarebeschleuniger

Im Folgenden werden wesentliche Erweiterungen der Hardware, die im Rahmen dieser Arbeit für die GigaNetIC-Architektur, zumeist für netzwerkspezifische Anwendungsszenarien (vgl. Kapitel 7) entwickelt wurden, kurz aufgelistet.

Erweiterung	Beschleunigung @200MHz	Fläche @130nm
Net-S-Core-Architurerweiterung	1,0	+16% * +695%
ohne / * mit Speicher		
XORLDW-Befehlssatzerweiterung	1,096	+ 0,0001mm <sup>2</sup>
IXD,IXQ-Befehlssatzerweiterung	1,12	+ 0,001mm <sup>2</sup>
Befehlssatzerweiterungen: ldwxorls8, andshr, orshl8/16/24, ldwaddi, ixwandshr, ldwixw	1,11+ / 1,25*	+ 0,004mm <sup>2</sup>
	+ bei DSLAM / * bei IPsec	
CRC-Hardwarebeschleuniger	8 / 15 / 81 / (437)	+ 0,014mm <sup>2</sup>
IP-Filter (inkl. 13kB SRAM )	6	2,86mm <sup>2</sup> / 0,028mm <sup>2</sup>
IP-Headercheck	13	+ 0,033mm <sup>2</sup>
AES-Verschlüsselung HW-Beschleuniger	160	+ 0,124mm <sup>2</sup>
AES-Entschlüsselung HW-Beschleuniger	175	+ 0,197mm <sup>2</sup>
Content Addressable Memory (CAM)	45 / 1230	+ 0,72mm <sup>2</sup>
Parallelität / Multiprozessor-Cache	abhängig von der Konfiguration und der Parallelisierbarkeit der Anwendung	

Abbildung 6-27: Übersicht der implementierten Erweiterungen zur Performanzsteigerung des GigaNetIC-Systems für die 130-nm-Standardzellentechnologie

Abbildung 6-27 zeigt eine Auswahl der vorgenommenen Erweiterungen, angefangen bei der Erweiterung der Prozessorkerns durch zusätzliche Funktionalitäten (vgl. Abschnitt 4.3.2), die zunächst keine Beschleunigung hervorrufen, aber die Verwendbarkeit des Systems deutlich erhöhen.

Im nächsten Schritt kommen dann die in Abschnitt 6.2 vorgestellten Instruktionssatzerweiterungen hinzu. Diese erhöhen den Flächenbedarf nur marginal, vermögen aber die Leistungsfähigkeit für die betrachteten Anwendungen bis zu 25 % zu beschleunigen.

Ein weiteres Mittel zur Erhöhung der Verarbeitungsgeschwindigkeit stellen die eng-gekoppelten Hardwarebeschleuniger dar. Im Rahmen dieser Arbeit wurden mehrere Hardwarebeschleuniger zur Generierung und Prüfung von unterschiedlichen *CRC* (*Cyclic Redundancy Check*)-Prüfsummen entwickelt. Mit der eng-gekoppelten *CRC*-Hardwareerweiterung wird eine Beschleunigung um den Faktor 15 gegenüber einer reinen Software-Implementierung erzielt [116]. In [118] konnte mit Hilfe einer neuen Beschleuniger-Architektur für CRC8 und CRC32 sogar eine Geschwindigkeitssteigerung gegenüber der Prozessorimplementierung von 81 bzw. sogar 437 bei loser Kopplung und maximaler Betriebsfrequenz erreicht werden.

Hardwarebeschleuniger dieser Stufe mit einer losen Kopplung an das System, mit ggf. sogar deutlich höherer Betriebsfrequenz, erzielen in der Regel weitaus größere Beschleunigungen, benötigen jedoch auch mehr Fläche und Entwicklungsaufwand. Hierzu zählen ein *IP-Filter*-Modul zur Prü-

fung spezifischer Charakteristika in IP-Paketen, der bereits im vorigen Abschnitt vorgestellte IP-Headercheck-Hardwarebeschleuniger, Module zur Ver- und Entschlüsselung nach dem *AES(Advanced Encryption Standard)*-Verfahren, die ebenso für sicherheitsrelevante Zwecke eingesetzt werden wie die bereits beschriebenen Instruktionssatzerweiterungen zur Beschleunigung von *IPSec*-Protokollen (vgl. Abschnitt 6.2.4). Schließlich wurde noch ein inhaltsadressierbarer Speicher, *CAM (Content Addressable Memory)* realisiert, der besonders zur Beschleunigung von Adressraumzugehörigkeitsüberprüfungen (Beschleunigung von 1230 verglichen mit einer reinen Softwarelösung) beiträgt [164].

Viele dieser weniger flexibel, jedoch hoch-performanten Einheiten sind im Zusammenhang mit der Analyse des IP-DSLAM-Referenzbenchmarks [141][119] entstanden und dienen zur weiteren Beschleunigung rechenintensiver Funktionen dieser Netzwerkanwendung. Die Erstellung, Verifikation, Optimierung und anschließende Charakterisierung der Hardwareerweiterungen bzgl. des Anwendungsszenarios erfolgte mit der in den Kapiteln 5 und 6 vorgestellten GigaNetIC-Werkzeugkette.

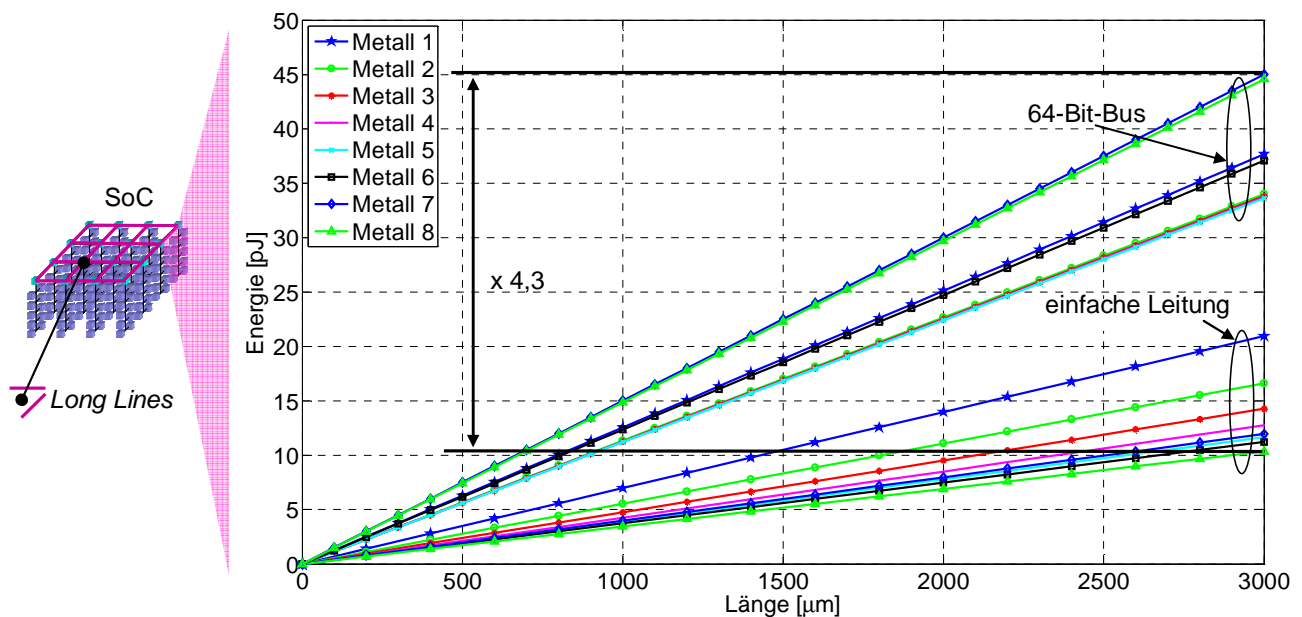
Aufgrund der Vielzahl der hier gewonnenen Ergebnisse können in diesem Rahmen keine weiteren Ausführungen zu den einzelnen Modulen erfolgen. Um einen Eindruck von der Performanzsteigerung durch die einzelnen Hardwarebeschleuniger bzw. durch die Erweiterungen für das Gesamtsystem vermitteln zu können und Auswirkungen von Systemmodifikationen bzw. von Lastveränderungen schnell abschätzen zu können, wurde ein spezielles Analyse- und Visualisierungswerkzeug, der *DSLAM-Explorer* entwickelt. Dieses Werkzeug wird in Abschnitt 7.5 detaillierter vorgestellt.

Die letzte Stufe der Beschleunigung ist der Einsatz der parallelen Struktur der GigaNetIC-Architektur. Dies kann mehrere Prozessoren innerhalb eines Clusters ggf. nebst dem realisierten Multiprozessorcache (vgl. Abschnitt 6.7) bedeuten, oder aber clusterübergreifende Parallelität unter Verwendung eines geeigneten Programmiermodells (vgl. 4.5) und schließt ggf. die Nutzung parallel instanzierter Hardwarebeschleuniger mit ein. Analysen zu diesen Aspekten der Beschleunigung innerhalb der GigaNetIC-Architektur werden in [141][130][118][115][109][131][113], in Abschnitt 6.7 und in Kapitel 8.3 angestellt.

## 6.6 Optimierungspotential der Kommunikationsinfrastruktur

Die GigaNoC-Kommunikationsinfrastruktur der GigaNetIC-Architektur lässt sich ebenfalls auf das jeweils angestrebte Anwendungsszenario hin optimieren. Neben den allgemein üblichen Methoden zur Verlustleistungsminimierung wie z. B. Frequenz- und Spannungsskalierung (vgl. Abbildung 6-3) sind auch systemspezifische Maßnahmen möglich. Stellt das zukünftige Anwendungsszenario besonders hohe Anforderungen an Bandbreite und Latenz der Übertragungswege, so lassen sich z. B. die Flitbreite und oder die Anzahl der Ports der Switch-Boxen erhöhen (vgl. Kapitel 4). Auch die FIFO-Tiefe der Eingangsports könnte deutlich erhöht werden, um etwaige Spitzen im Datenverkehr abpuffern zu können. Auf Clusterebene kann zur Minimierung der Busblockierung bzw. zur Erhöhung des Durchsatzes sowohl bei der Wishbone- als auch bei der AMBA-Realisierung die Datenbreite erhöht werden oder bei häufiger Interprozessorkommunikation ggf. auf die AMBA-Switch-Matrix-Realisierung zurückgegriffen werden. Im Falle der Verlustleistungsreduktion ergibt sich neben den genannten Möglichkeiten noch eine weitere, die die Anforderungen unterschiedlicher Datenströme innerhalb des Chip-Multiprozessors ausnutzt. Die Überlegung basiert auf dem Phäno-

men, dass die Signalübertragung über längere Distanzen (so genannte *Long Lines*) mit Hilfe parallel zueinander geführter Leitungen (in der Darstellung als Bus bezeichnet) in der betrachteten Standardzellentechnologie deutlich mehr Verlustleistung bedeutet bzw. Energie benötigt, als wenn die Daten seriell über eine Leitung übertragen werden. Dieser Zusammenhang ist in [165] detailliert dargelegt, wo ferner Koppelkapazitäten zwischen den einzelnen Bussignalen und benachbarten Metalllagen sowie die eigentliche Leitungskapazität berücksichtigt werden. Besonders auf den höheren Metalllagen machen sich geometriebedingt die Koppelkapazitäten bei der parallelen Übertragung häufig besonders bemerkbar. Diese Lagen (in dem aufgeführten Beispiel betrifft dies vor allem die Metalllagen 7 und 8, da diese für die *Long Lines* Verwendung finden) werden für die Inter-Switch-Box-Verbindungen benötigt, die, wie in Kapitel 4 aufgezeigt, massiv parallel<sup>45</sup> ausgeführt sind. Sollte es in Abhängigkeit vom jeweiligen Anwendungsszenario häufig Datenpakete geben, die eine geringere Übertragungsgeschwindigkeit bzw. eine höhere Latenz tolerieren, so könnte ein bereits vorgesehener serieller Übertragungsmechanismus eine deutliche Reduktion der Verlustleistung bedeuten. Hierzu würde eine oder eine kleinere Anzahl von Leitungen verwendet, wobei bei der Verwendung mehrerer Leitungen diese immer durch einige ungenutzte voneinander getrennt wären, um so die Kopplung möglichst klein zu halten. So würden niederpriorere Datenpakete seriell bzw. partiell seriell übertragen, was gerade im Hinblick auf die Leistungsfähigkeit des GigaNoCs sicherlich für viele Bereiche ausreichen würde.



**Abbildung 6-28: Verlustleistungsoptimierung der Kommunikationsinfrastruktur bei der Übertragung niederpriorer Daten im Hinblick auf eine 130-nm-Standardzellentechnologie**

Abbildung 6-28 verdeutlicht die Verlustleistungsoptimierung der Kommunikationsinfrastruktur bei der Übertragung niederpriorer Daten im Hinblick auf eine 130-nm-Standardzellentechnologie. In dem Diagramm ist der Energiebedarf für die Übertragung von 64 Bit über eine einfache Leitung verglichen mit einer Übertragung über einen 64-Bit-breiten Bus mit minimal zulässigem Abstand der Leitungen in Abhängigkeit von der Länge der Übertragungsstrecke und der Metallebene aufget-

<sup>45</sup> Die derzeitige Implementierung verwendet ca. 200 Signalleitungen zwischen zwei Ports benachbarter Switch-Boxen.

ragen<sup>46</sup>. Bei der Berechnung wurden die anfallenden Koppelkapazitäten für die untersuchte Technologie nach Herstellerangaben berücksichtigt. Der Energiebedarf ist speziell im Bereich einer Leitungslänge von 2 bis 3 mm besonders prägnant und deshalb besonders zu berücksichtigen, da diese Leitungslängen nach dem derzeitigen *Floorplan* (vgl. Abschnitt 8.2.3) in der Größenordnung der Inter-Switch-Box-Verbindungen liegen. Die Übertragung von 64 Datenbits über eine parallele Busstruktur, wie sie derzeit beim GigaNoC vorgesehen ist, benötigt auf der obersten Metalllage (*Metall 8*) 44,58 pJ verglichen mit lediglich 10,35 pJ bei der seriellen Variante auf der gleichen Metalllage. So ließe sich für niederprioritäre Daten bei geringer Auslastung des NoCs das bis zu 4,3-fache an Energie einsparen<sup>47</sup>. Betrachtet man die Energie, die notwendig ist, um ein NAND2-Gatter mit einer Treiberstärke von 2 der gleichen 130-nm-Standardzellentechnologie umzuladen, so liegt diese zwischen 0,009 und 0,013 pJ. Dies liegt fast vier Größenordnungen unter der Energie, die für die Übertragung der Nettodatenmenge eines Flits und die damit verbundenen Umladevorgänge der Koppelkapazitäten der Leitungen aufgebracht werden muss. Der nicht zu vernachlässigende Anteil der Kommunikation am Gesamtenergiebedarf wird hier verdeutlicht.

Mit Hilfe der in Kapitel 5 vorgestellten Entwicklungsumgebungen der GigaNetIC-Architektur für die SoC-Ebene lassen sich die Bandbreitenanforderungen der Anwendungen komfortabel analysieren. Die entsprechenden Pakettypen werden gekennzeichnet und können mit Hilfe dieser langsameren aber energieeffizienteren Übertragungsmethode, z. B. durch Setzen eines bestimmten Steuerbits, mit Hilfe von Instruktionsflits (vgl. Abschnitt 4.2.2.1) transportiert werden. In [166] wird ein ähnlicher Ansatz aufgezeigt, der eine Reduktion des Energiebedarfs um mehr als 30 % ermöglicht, ohne einen nennenswerten Verlust der Übertragungskapazität zu verzeichnen. Eine Bandbreitenreduktion ist hingegen bei dem hier vorgestellten Ansatz bewusst gewählt und erlaubt deshalb die weitaus höhere Reduktion des Energiebedarfs von mehr als 76 %.

Eine weitere Möglichkeit, eine effizientere Kommunikation zu ermöglichen, ist die Realisierung einer global asynchronen, lokal synchronen (*GALS*) Kommunikationsinfrastruktur, die zukünftig in das GigaNoC-On-Chip-Netzwerk einfließen wird. Hierdurch können unterschiedliche Taktomänen sowie lokal unterschiedliche Versorgungsspannungen (*Voltage Islands*) realisiert werden, was ebenfalls zur Verlustleistungsminimierung beiträgt.

## 6.7 Optimierung im Hinblick auf die Speicherhierarchie

Auf Clusterebene lässt sich die GigaNetIC-Architektur für eine Reihe von Anwendungen durch die Integration des in Abschnitt 4.4.2 beschriebenen Multiprozessorcaches beschleunigen. Inwieweit sich eine Verwendung dieser Hardwareoption für das jeweilige Einsatzgebiet eignet, lässt sich mit

---

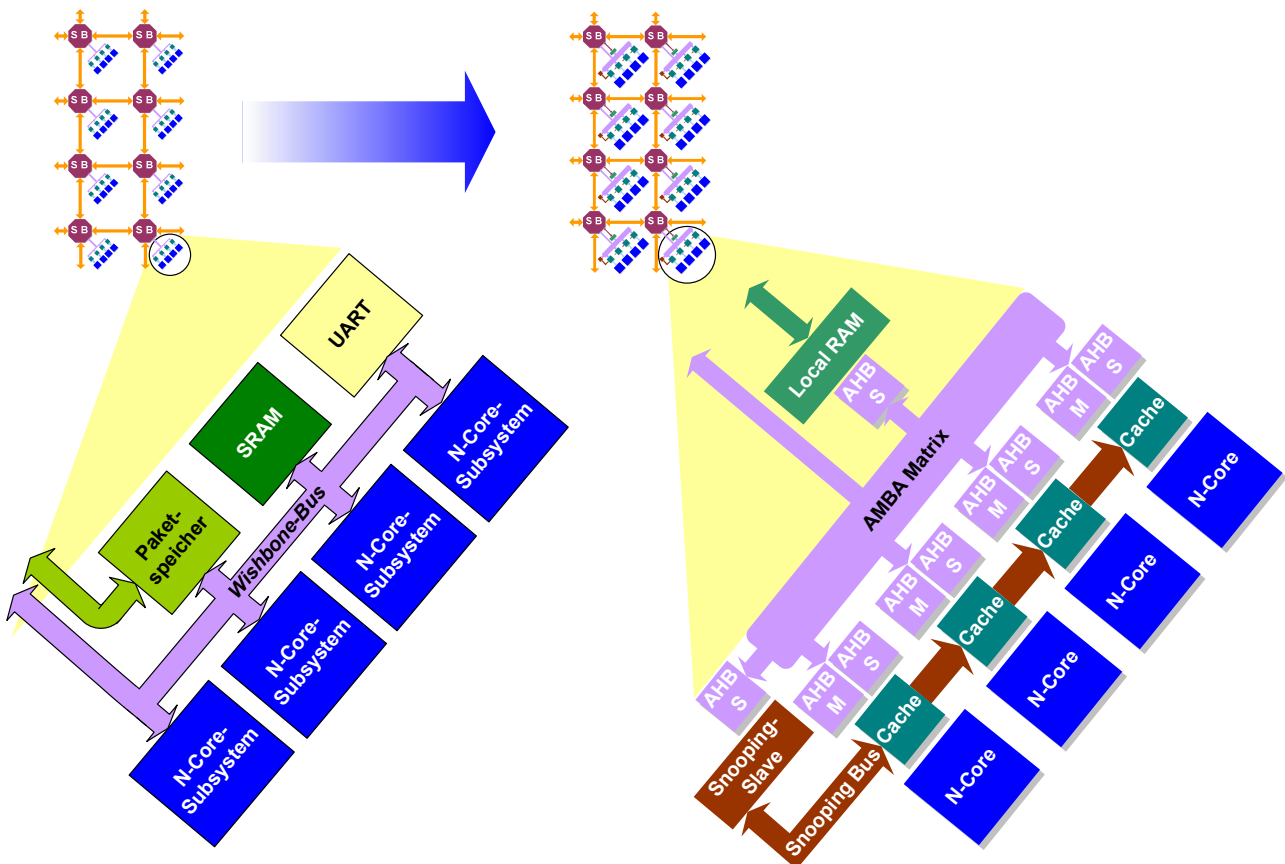
<sup>46</sup> Für die Schaltvorgänge werden die *Worst-Case*-Bedingungen angenommen, d. h. die umzuladenden Kapazitäten werden mit jedem neuen Takt umgeladen. Beim Bus werden für benachbarte Leitungen gegensätzliche Umladevorgänge angesetzt, um die maximal benötigte Energie zum Transport von Nachrichten zu ermitteln.

<sup>47</sup> In diesem Zusammenhang ist zu erwähnen, dass Verlustleistungsanteile der treibenden Ausgangsregister vernachlässigt werden, da diese im Sinne der Schalthäufigkeiten keinen abweichenden Beitrag liefern würden. Lediglich der Mehraufwand durch die zusätzliche, allerdings sehr klein ausfallende Steuerlogik wäre ggf. noch in die Kalkulation einzubeziehen.



den in Kapitel 5 vorgestellten Simulationsumgebungen bereits im Vorfeld komfortabel ermitteln. Sowohl die SystemC-Umgebung SiMPLE, als auch die VHDL-Umgebung PERFMON wurde vollständig für die Analyse des Multiprozessorcaches ausgelegt. Ferner gibt es eine prototypische FPGA-Realisierung für das Rapid-Prototyping System RAPTOR2000 [113].

Abbildung 6-29 zeigt die Eingriffe in das Standard-GigaNetIC-System, die notwendig sind, um es in die Multiprozessorcachevariante zu transformieren. Auf SoC-Ebene sind keine Veränderungen notwendig, auf Clusterebene wird das lokale Bussystem durch die leistungsfähige AHB-Switchmatrix ersetzt, die für die GigaNetIC-Multiprozessorcaches benötigt wird. Das N-Core Subsystem verändert sich insofern, als die in Abschnitt 4.4.2 dargelegten AHB-Schnittstellen integriert werden. Weitere Details zu den Kennwerten dieser Implementierung werden in Abschnitt 8.2.2 gegeben.



**Abbildung 6-29: Transformation vom busbasierten GigaNetIC-System zur Variante mit GigaNetIC-Multiprozessorcache und einer AHB-Switchmatrix auf Clusterebene**

Abbildung 6-30 verdeutlicht die Möglichkeiten zeitgleicher Kommunikation im GigaNetIC-Multiprozessorcachesystem. Diese parallelen Transfers werden u. a. durch den Einsatz der AMBA-Switchmatrix ermöglicht und helfen, die lokale On-Chip-Kommunikation auf Clusterebene im Vergleich zu einem einfachen Bussystem deutlich zu beschleunigen. In Abbildung 6-30 sind folgende, simultan ablaufende Transfers dargestellt: (1) kennzeichnet einen *Snooping*-Zugriff von *Cache Nr. 0*, (2) stellt einen Zugriff von *N-Core 1* auf den Communication-Controller dar, bei (3) handelt es sich um einen Zugriff auf Daten, die sich bereits im *Cache Nr. 1* befinden und durch *Direct-Data-Intervention* (vgl. Abschnitt 4.4.2) in den *Cache Nr. 2* transportiert werden, und (4) beschreibt einen Lesezugriff von *N-Core 3* auf den gemeinsamen lokalen Speicher des Clusters.

In [113] haben wir eine Reihe von Anwendungen<sup>48</sup> im Hinblick auf den Einsatz des Multiprozessorcaches untersucht. Hierbei handelt es sich sowohl um Universalanwendungen wie den *Dhrystone*-Benchmark<sup>4</sup>, um die Sortieralgorithmen *Bubble*- und *Quicksort* und um einen Benchmark aus der Netzwerkdatenverarbeitung. Im Rahmen der Analysen wurde eine Vielzahl von Parametern variiert. Hierzu zählen die Art der Cachearchitektur (*Split* oder *Unified*), die Assoziativität (2, 4 und 8), die Anzahl der *Cachelines* pro Weg (32, 64, 128 oder 256), die Weite einer Cacheline (32, 64, 128 oder 256 Bit) und die Zugriffslatenz auf den Hauptspeicher (0, 5, 10 oder 20 Takte)<sup>49</sup>. Zusätzlich wurde zum Vergleich eine Architekturvariante ohne Cache ausgemessen. Die Simulationen wurden sowohl mit SiMPLE (vgl. Abschnitt 5.2) als auch mit PERFMON (vgl. Abschnitt 5.3) durchgeführt. Obwohl die Abweichungen der Ergebnisse von SiMPLE zur exakten Ausführung mit PERFMON nur im einstelligen Prozentbereich liegen, wurde die sehr zeitintensive HDL-Simulation vorgezogen. Mit Hilfe der in Abschnitt 5.4 vorgestellten MultiSim-Entwurfsraumexploration konnten die Simulationsläufe vollautomatisch ablaufen. Die ermittelten, exakten Laufzeitergebnisse wurden auf vier Arbeitsplatzrechner (P4, 3 GHz HT mit 1 GB Arbeitsspeicher) verteilt und benötigten eine Simulationslaufzeit von ca. je 4 Wochen. Diese Zahlen verdeutlichen den immensen Zeitaufwand, der für die detaillierte HDL-Simulation benötigt wird. Aufgrund der mittlerweile erreichten Genauigkeit der SiMPLE-Umgebung werden solch aufwändige Simulationen immer seltener notwendig sein. Zur Entwurfsraumexploration wird in den meisten Fällen die SiMPLE-Umgebung ausreichend genaue Werte liefern.

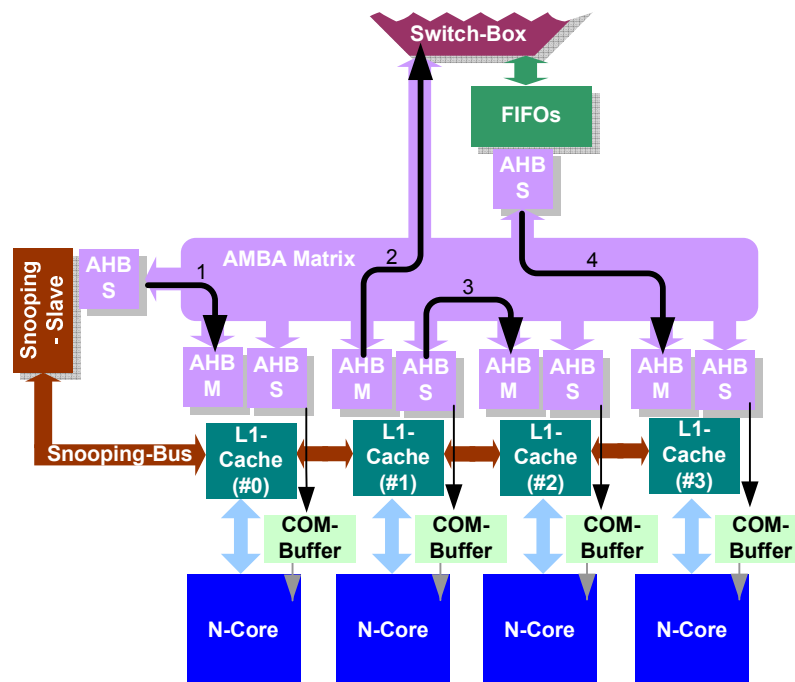
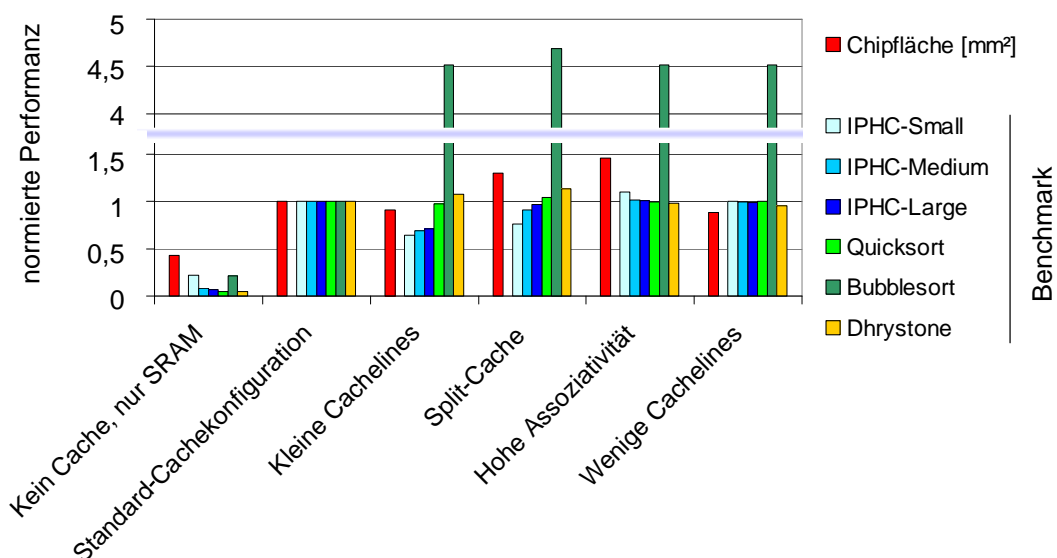


Abbildung 6-30: Parallele Transfers in der AMBA-Matrix auf Cluster-Ebene

<sup>48</sup> Diese Auswahl von Anwendungen wird ebenfalls bei weiteren Analysen der GigaNetIC-Architektur, speziell im Hinblick auf die Ressourceneffizienz, in Abschnitt 8.3 verwendet.

<sup>49</sup> Die Zugriffslatenz auf den Hauptspeicher ist mit maximal 20 Takten eher gering gewählt, allerdings den Gegebenheiten des GigaNetIC-Systems angepasst.

Abbildung 6-31 zeigt die Ergebnisse der detaillierten Benchmarksimulation mit PERFMON. Die dargestellten Werte wurden auf die Standard-Cachekonfiguration (256 Cachelines, 128 Bit breit, Assoziativität von 2, *Unified-Cache* und ein 32-Bit-breites Businterface, 250 MHz) normiert. Zum Vergleich sind ebenfalls Flächenbedarf und Leistungswerte für ein System ohne Cache, aber mit gleicher Latenz zum Hauptspeicher, in dieser Betrachtung sind es zehn Takte, aufgetragen. Es ist zu beachten, dass diese Variante nicht zwangsläufig die in Abschnitt 4.2.4 beschriebene Wishbonebus-Architektur widerspiegelt, da hier normalerweise alle Daten im lokalen SRAM des N-Cores vorgehalten werden und dieser eine Latenz von zwei Takten besitzt. Sollten die Daten jedoch vom Speicher einer benachbarten Switch-Box geholt werden müssen, so würde sich eine vergleichbare Latenz ergeben, vgl. Tabelle 4-6. Es sind jeweils die Trends für vier weitere markante Varianten der insgesamt 2716 untersuchten Cachekonfigurationen aufgezeichnet.



**Abbildung 6-31: Flächenbedarf vs. Performanz von GigaNetIC-Cluster-Konfigurationen für ausgewählte Benchmarkszenarien, normiert auf die Standardcachekonfiguration**

Die angesetzten Parameter der untersuchten Varianten werden in Tabelle 6-6 aufgezeigt. Die konkreten Messwerte der in Abbildung 6-31 gezeigten Untersuchungen sind Tabelle 6-7 zu entnehmen.

**Tabelle 6-6: Ressourcenbedarf der untersuchten Cacherealisierungen in 90-nm-Standardzellentechnologie**

Variante	Cachelinegröße [Bit]	Tiefe	Assoziativität	Split-Cache	Fläche je Cache [mm²]	Taktperiode [ns]	Leistungsaufnahme @250MHz [mW]
Kleine Cachelines	64	256	2	nein	0,61	4,16	136,79
Split-Cache	128	256	2	ja	1,11	4,08	276,13
Hohe Assoziativität	128	256	4	nein	1,31	4,48	376,18
Wenige Cachelines	128	128	2	nein	0,58	3,92	164,87
Standard	128	256	2	nein	0,73	4,1	180,99

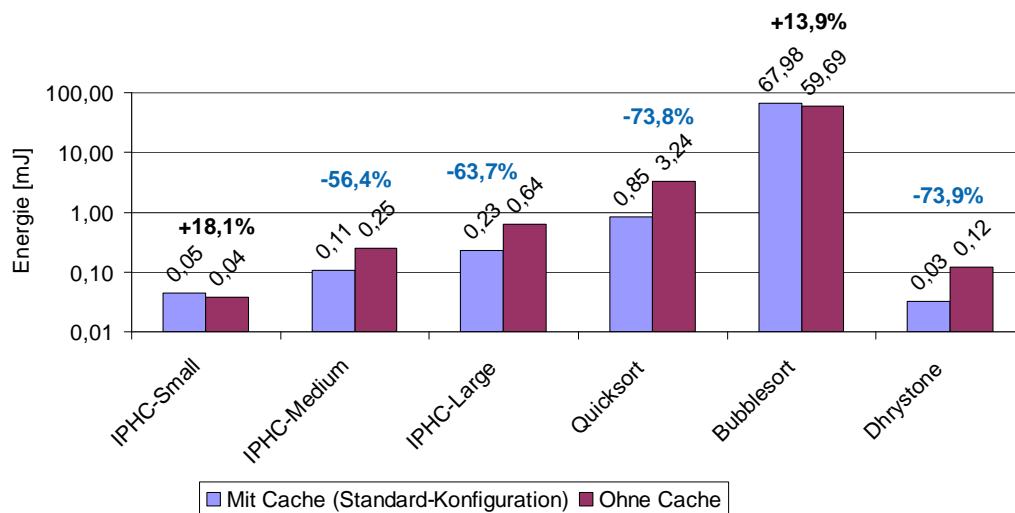
Bei den Analysen zeigt sich, dass durchweg alle Varianten des GigaNetIC-Systems mit Multiprozessorcache deutlich leistungsfähiger sind, als die Realisierung ohne Cache, allerdings zu einem nicht zu vernachlässigenden Preis in Form einer Flächenverdopplung bis hin zu mehr als einer Verdreifachung der Fläche bei der Variante mit hoher Assoziativität. Die Analysen zeigen, dass bereits relativ geringe Variationen der Cacheparameter merkliche Auswirkungen auf die Leistungsfähigkeit und die Ausführungsgeschwindigkeit einzelner Anwendungen haben können. So zeigt z. B. die Verwendung der Variante mit halbiertem Anzahl der Cachelines dennoch einen Geschwindigkeitszuwachs um den Faktor 4,5 bei der Verarbeitung des *Bubblesort*-Algorithmus. Dies wird erreicht,

obwohl die Fläche um 11 % geringer ist als die der Standardvariante. Allerdings zählt der *Bubblesort*-Algorithmus wie auch der *IPHC-Small*-Benchmark zu den beiden untersuchten Anwendungen, die weniger von den hier eingestellten Parametern des Caches profitieren. Bei beiden Anwendungen ist die zeitliche Lokalität relativ gering, d. h. es müssen häufig neue Daten nachgeladen werden. Dieses Phänomen ist bei den anderen Benchmarks weniger stark ausgeprägt, so dass der Geschwindigkeitszuwachs weit höher liegt. So lassen sich durch den Einsatz der *Split-Cache*-Variante der *Quicksort*-Algorithmus und der *Dhrystone*-Benchmark um mehr als 22,8- bzw. 20,8-fach beschleunigt verarbeiten.

**Tabelle 6-7: Kosten-Nutzen-Analyse verschiedener Cachekonfigurationen relativ zur Standardkonfiguration**

Konfiguration	Chipfläche	relative Performanz					
		IPHC-Small	IPHC-Medium	IPHC-Large	Quicksort	Bubblesort	Dhrystone
<b>Standard Cachekonfiguration</b>	<b>100%</b>	<b>100%</b>	<b>100%</b>	<b>100%</b>	<b>100%</b>	<b>100%</b>	<b>100%</b>
Kleine Cachelines	91%	64%	69%	71%	98%	451%	108%
Split-Cache	130%	76%	91%	97%	104%	469%	114%
Hohe Assoziativität	146%	110%	101%	101%	100%	451%	98%
Wenige Cachelines	89%	100%	100%	99%	100%	451%	95%
Kein Cache, nur SRAM	43%	22%	8%	7%	5%	21%	5%

Abbildung 6-32 zeigt die Energieersparnis auf, die sich für die einzelnen Benchmarks unter Verwendung der Standard-Cachevariante erzielen lässt. In der Standardausführung benötigt die Cachevariante teilweise 73 % weniger Energie als das System ohne Cache. Allerdings gibt es auch Anwendungen, wie z. B. *IPHC-Small* (+18,1 %) oder *Bubblesort* (+13,9 %), für die sich die Standardvariante weniger gut eignet. Gleichwohl lassen sich auch hier weitaus höhere Performanzsteigerungen und Energiereduktionen mit anderen Varianten des GigaNetIC-Multiprozessorcaches erzielen, vgl. Tabelle 6-7.



**Abbildung 6-32: Energieersparnis durch Verwendung des GigaNetIC-Multiprozessorcaches für ausgewählte Anwendungen**

Die Analysen zeigen, dass zum einen eine intensive Evaluation des zukünftigen Anwendungsszenarios, sollte dies bereits vor der Chiprealisierung bekannt sein, sehr ratsam ist. Die GigaNetIC-Architektur stellt deshalb eine Vielzahl von Möglichkeiten zur Verfügung, um durch geeignete Parametrisierung eine möglichst optimale Systemkonfiguration für die angestrebten Einsatzzwecke bereits in einer frühen Entwurfsphase zu erkennen, vgl. Kapitel 5. Zum anderen zeigen die angestellten Analysen das Potential des GigaNetIC-Multiprozessorcaches zur Beschleunigung von Anwendungen auf, was allerdings von Fall zu Fall mit den deutlich erhöhten Flächenkosten im Sinne

der Ressourceneffizienz abzuwägen ist. Weitere Analysen und Details zur Steigerung der Ressourceneffizienz durch den hier untersuchten GigaNetIC-Multiprozessorcache sind [113] zu entnehmen.

## 6.8 Optimierung auf SoC-Ebene – Einsatz paralleler Prozessorfelder

Im Rahmen der Leistungssteigerung der GigaNetIC-Architektur sind natürlich auch Aspekte der Softwareoptimierung zu berücksichtigen. Der parallele Aufbau der GigaNetIC-Prozessorcluster bietet eine leistungsfähige Struktur zur Verarbeitung vielfältiger Problemstellungen, die nach GUSTAFSON (vgl. Abschnitt 2.1.2) durch Parallelität beschleunigt verarbeitet werden können. Auf der SoC-Ebene, die clusterübergreifend Aufgaben bearbeitet, ergeben sich mehrere potentielle Optimierungsmöglichkeiten bzgl. der effizienten Nutzung des Chip-Multiprozessors. Dies betrifft zum einen die bereits in Abschnitt 6.2.1 vorgestellte compilerbasierte Werkzeugkette und den daraus resultierenden Compiler, dessen Effizienz ganz entscheidend für die Performanz der hier eingesetzten N-Core-Prozessorkerne ist. Zum anderen spielt das jeweils verwendete Programmiermodell (vgl. Abschnitt 4.5) eine wesentliche Rolle bei der wirksamen Ausnutzung der parallelen Prozessorfelder. Zu diesen beiden Aspekten, die sozusagen die Systemsoftware der Architektur darstellen kommt dann die jeweilige Anwendungssoftware, die auf den jeweiligen Prozessorkernen eingesetzt wird. Diese Optimierungsmaßnahmen seitens der Software helfen so auch nach Fertigstellung der Hardware, die Ressourceneffizienz des Chip-Multiprozessorsystems zu steigern.

### 6.8.1 Optimierung der System- und Anwendungssoftware

Durch zielgerichtete Optimierung der bei der GigaNetIC-Architektur eingesetzten Systemsoftware, die die Verarbeitung durch die einzelnen Prozessorkerne des Chip-Multiprozessors koordiniert und den korrekten Ablauf gewährleistet, lassen sich deutliche Beschleunigungen der Verarbeitung sowie Minimierung bzw. Optimierung der On-Chip-Kommunikation erreichen. Sind die Randbedingungen und Anforderungen der zukünftigen Anwendung bekannt, oder lassen sich diese im Vorfeld abschätzen, so kann dieser Teil des GigaNetIC-Softwaresystems im Hinblick auf Lastverteilung, Kommunikationsmethoden, wie z. B. Art und Anzahl der Synchronisationsbarrieren, und Speicherorganisation angepasst werden.

Die Ergebnisse dieser Arbeit profitieren zum einen von den erzielten Verbesserungen im Bereich der Compileroptimierung, zum anderen von Optimierungen und Erweiterungen der Simulatoren speziell im Bereich der Simulationsgeschwindigkeit<sup>50</sup>.

Beispielhaft sei hier die im Rahmen der in Kapitel 7 näher diskutierten Netzwerkanwendungen implementierte *IP*-Paket-Prüfsummenfunktion als eine potentielle Anwendung der GigaNetIC-Architektur erwähnt. Diese verwirklicht die in Abschnitt 6.3.1 beschriebene Funktionalität des Hardwarebeschleunigers auf dem N-Core-Prozessorkern. Paketköpfe des Internet-Protokolls werden auf Kor-

---

<sup>50</sup> An der Auswertung und der Verbesserung der Aspekte bzgl. der Softwarebestandteile der GigaNetIC-Architektur wird maßgeblich an den Fachgebieten Programmiersprachen und Übersetzer, Prof. Dr. Uwe Kastens sowie Algorithmen und Komplexität, Prof. Dr. math. Friedhelm Meyer auf der Heide gearbeitet. Die Verbesserungen flossen stets in diese Arbeit mit ein, finden in diesem Rahmen jedoch keine weitere Erwähnung, da hier schaltungstechnische Aspekte im Vordergrund stehen. Für nähere Informationen sei deshalb auf die einschlägigen Veröffentlichungen der beiden genannten Fachgebiete der Universität Paderborn verwiesen.

rektheit und Inhalt geprüft. Bei der Evaluierung dieser *IP-Headercheck-Funktion* zeigte sich, dass der Bedarf für eine Beschleunigung der Bearbeitung bestand. Zunächst wurde eine Optimierung der Software vorgenommen. Hierbei wurden vier Varianten des *Headerchecks* implementiert, die mit 16 bzw. 32 Bit breiten Daten arbeiteten. Mit einer optimierten 32-Bit-Version konnte letztendlich eine Reduktion der benötigten Anzahl an Taktzyklen von anfangs 284 auf 108 erzielt werden. Zusätzlich wurde nach Superinstruktionen zur Performanzsteigerung gesucht. Die Analysen zeigten, dass in diesem Fall keine vielversprechenden Instruktionssatzerweiterungen für den gegebenen Algorithmus realisiert werden konnten. Mit den bestehenden Erweiterungen (vgl. Abschnitt 6.2.4) konnte lediglich eine Reduktion um fünf Taktzyklen auf 103 Takte erreicht werden. Dennoch zeigt dieses Beispiel bereits das Potential, dass die Softwareoptimierung bietet, konnte doch eine bereits effiziente Realisierung unter Berücksichtigung spezieller Compilereigenheiten etc. um mehr als das 2,7-fache beschleunigt werden.

Aufgrund der benötigten Bandbreite und der Häufigkeit des Funktionsaufrufs innerhalb des Anwendungsszenarios war die hier erreichte Beschleunigung dennoch nicht ausreichend, so dass, wie in Abbildung 6-5 als Option bereits aufgezeigt, ein spezieller Hardwarebeschleuniger (vgl. Abschnitt 6.3.1) entwickelt werden musste. Dieser erlaubt die Ausführung der inneren Prüffunktion in 8 Takten und dies ggf. bei einer deutlich höheren Betriebsfrequenz als die des N-Cores. Mit Hilfe dieser anwendungsspezifischen Hardware wird so nochmals eine Beschleunigung von mindestens 12,8 verglichen mit der optimierten Software erreicht.

Grundsätzlich wurde bei der Implementierung der in dieser Arbeit entstandenen Software, stets auf eine möglichst effiziente Realisierung geachtet, allerdings lag das Hauptaugenmerk auf einer guten Portierbarkeit auch auf andere Prozessorarchitekturen, die ggf. anstelle des N-Cores in die GigaNet-IC-Cluster integriert werden können. So wurde bewusst weitestgehend auf Besonderheiten des Compilers und auf optimierten Assemblercode verzichtet, um die Interoperabilität zu gewährleisten. Dies bedeutet im Sinne der Ressourceneffizienz einen Kompromiss zwischen Performanz und Zukunftssicherheit. Im Weiteren werden nur finale Softwarerealisierungen betrachtet, wobei die während des Entwicklungsprozesses durchgeführten Optimierungen an der System- und Anwendungssoftware unerwähnt bleiben.

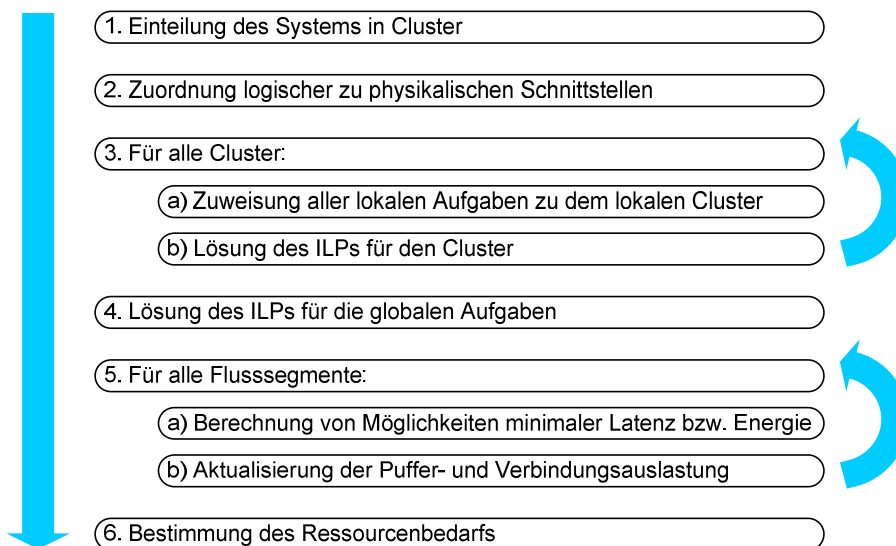
### 6.8.2 Optimierung der Aufgabenverteilung und Interprozesskommunikation

Die Abbildung von protokollverarbeitenden Funktionen bzw. von Programmen allgemein, auf Multiprozessorsysteme stellt zumeist eine zeitaufwändige und zugleich performanzentscheidende Aufgabe dar. Komplexe Hardwarearchitekturen wie der GigaNetIC-Chip-Multiprozessor beinhalten eine Vielzahl von Parametern, wie Betriebsfrequenzen, Datenraten, Speichertypen und -größen etc., die entweder verändert oder zumindest berücksichtigt werden müssen/können. Eine weitgehend automatisierte Lösung dieses Problems würde die Zeit für Softwareentwicklung und -partitionierung deutlich verkürzen und den Softwareentwickler stark entlasten.

Der NoC-basierte Ansatz des GigaNetIC-Systems eröffnet eine gute Skalierbarkeit speziell auch im Hinblick auf zukünftige Schaltungstechnologien, so dass in naher Zukunft weitaus größere Systeme denkbar sind, als derzeit realisierbar. Gerade hier lohnt sich ein automatisiertes Vorgehen.

In [102][103][104] haben wir eine Methodik vorgestellt, die eine automatisierte Abbildung speziell von netzwerkspezifischen Protokollverarbeitungsabläufen auf Chip-Multiprozessoren zur Erhöhung der Ressourceneffizienz durchführt. *NetAMap* (*Network Application Mapper*) ist das resultierende

Werkzeug [104], das, aufbauend auf der PERFMON-Umgebung [116], eine Aufteilung und zeitliche Planung (*Scheduling*) der zu verarbeitenden Prozeduren und die resultierende Interprozesskommunikation durchführt. Hierbei können Zielarchitekturen mit unterschiedlichsten On-Chip-Netzwerken berücksichtigt werden. Der Bestandteil „Network“ im Namen des Softwarewerkzeugs ist hier ambivalent zu sehen. Zum einen bezieht es sich auf die bisher analysierten Anwendungsklassen aus dem Bereich der Netzwerkprozessoren und zum anderen greift es die NoC-basierte Struktur der Zielarchitekturen auf.



**Abbildung 6-33: Prinzipieller Ablauf der automatisierten Anwendungsabbildung für Chip-Multiprozessorsysteme mit Hilfe von NetAMap**

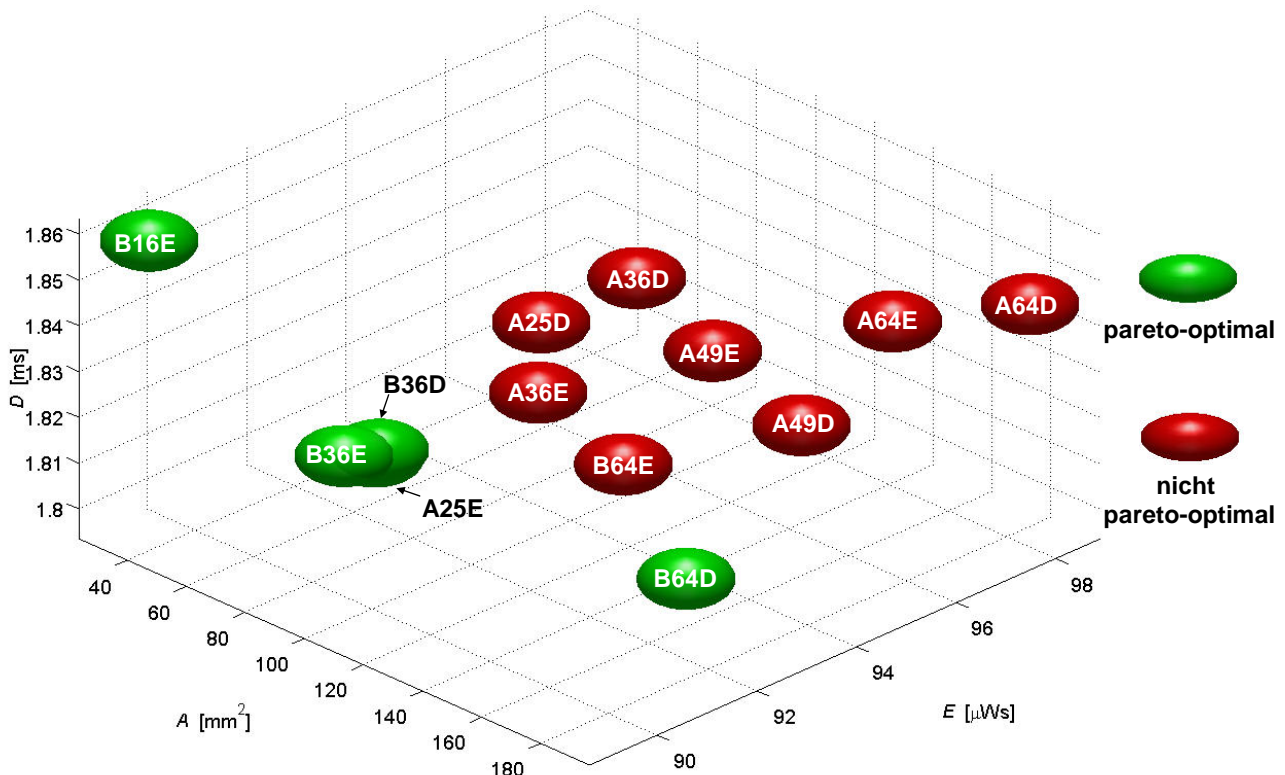
Das durchgeführte *Scheduling* beruht auf dem Ansatz des *Generalized Processor Sharing (GPS)* [167], das u. a. eine Reduzierung des benötigten Pufferspeichers in den einzelnen Knoten bewirkt. Jedem Verarbeitungsschritt (Flusselement bzw. *Flow Segment*) werden separat Kommunikationsbandbreite und Berechnungszeit zugewiesen. Dies erlaubt es dem Algorithmus, individuelle Verarbeitungsprioritäten zu berücksichtigen und zudem eine blockadefreie, deterministische Verarbeitung und Weiterleitung der Daten durchzuführen. Dies setzt derzeit die Kenntnis über die maximale Verarbeitungszeit (*Worst-Case Execution Times - WCET*) von Funktionen schon während der Entwurfszeit voraus, was den Einsatzbereich von NetAMap in einem gewissen Rahmen einschränkt, durch das in [102][103][104] untersuchte, selbst definierte *MANet*-Protokoll aber kompensiert wurde.

Das Problem der Abbildung der Anwendung wird mit Hilfe der ganzzahligen linearen Optimierung (*Integer Linear Programming / ILP*) gelöst. Um die aus komplexitätstheoretischer Sicht NP-schwere Aufgabe in angemessener Zeit lösen zu können, wird hier ein hierarchischer Ansatz gewählt, der die Anzahl der Variablen klein hält [102]. Dies geschieht durch Partitionierung der Problemgröße auf einstellbare Cluster des Zielsystems.

NetAMap unterstützt zwei Optimierungsziele: Entweder wird die Latenz, die die Pakete während der Verarbeitung erfahren, oder aber der Energiebedarf für die Verarbeitung minimiert. Das *ILP* besteht somit aus den Kostenfunktionen und einem Satz technologieabhängiger Bedingungen, die u. a. mit der PERFMON-Umgebung ermittelt werden. Die entworfene Abbildungsmethode ermöglicht eine einfache Softwarepartitionierung für das SoC und eine detaillierte Entwurfsraumexploration und Bewertung potentieller Lösungen. Flaschenhalse bzgl. Performanz und Durchsatz können rela-



tiv leicht lokalisiert und ggf. durch Hard- oder Softwareoptimierung behoben werden. Abbildung 6-33 zeigt den prinzipiellen Ablauf der automatisierten Anwendungsabbildung auf Chip-Multiprozessorsysteme mit Hilfe von NetAMap.



**Abbildung 6-34: Ressourcenbedarf zweier CMP-Architekturen (A = alternative Architektur, B = GigaNetIC) im Hinblick auf pareto-optimale Punkte im Entwurfsraum bzgl. einer MANet-Anwendung [102]**

In [102] haben wir NetAMap eingesetzt, um die Ressourceneffizienz zweier unterschiedlicher Chip-Multiprozessorsysteme in Bezug auf ein selbst entwickeltes Protokoll für mobile Ad-Hoc-Netzwerke (*MANets*) zu bestimmen. Bei beiden CMPs wird der S-Core [108] als Prozessorkern genutzt, die Systeme unterscheiden sich in dem verwendeten On-Chip-Netzwerk. System A verwendet ein Netzwerk, das auf dem *Circuit-Switched*-Prinzip (vgl. Abschnitt 2.3.2) aufbaut und deterministisch bzgl. der Latenz ist. An jeder Switch-Box dieses Systems ist jeweils nur eine Verarbeitungseinheit angeschlossen. Die Switch-Boxen sind flächenmäßig ein Drittel kleiner als die des Systems B. Bei diesem handelt es sich um eine GigaNetIC-Architektur mit vier Prozessorkernen pro Cluster. Abbildung 6-34 zeigt die Resultate der Abbildung der Anwendung mittels NetAMap auf Varianten beider CMP-Architekturen. Die Blasen kennzeichnen die Punkte im Entwurfsraum die durch die jeweiligen Systemvarianten im Hinblick auf Fläche (A), Energiebedarf (E) und Latenz (D) erreicht werden. Die Notation ist dabei wie folgt zu interpretieren: A/B CMP-Architektur (A = Architektur aus [102], B = GigaNetIC-Architektur) – Anzahl der instantiierten Verarbeitungseinheiten – Optimierungsstrategie (D = Optimierung der Latenz, E = Optimierung des Energiebedarfs). Bei der Analyse der Ergebnisse zeigt sich, dass, bis auf die Variante mit 64 Verarbeitungseinheiten und Optimierung auf Energie, alle Systeme der GigaNetIC-Architektur pareto-optimale Punkte im Entwurfsraum darstellen. Pareto-optimal bedeutet in diesem Zusammenhang, dass keine der anderen Systemkonfigurationen besser bzgl. einer der Optimiergrößen und jeweiligen Optimierungsstrategie abschneidet. Den geringsten Energie- und Flächenbedarf zeigt das GigaNetIC-System mit 16 Verarbeitungseinheiten (*B16E*). Diese Variante hat allerdings die größte Latenz. Die GigaNetIC-



Architektur dominiert in diesem Vergleich die alternative Architektur. Hier zeigte sich, dass mehrere Verarbeitungseinheiten pro Routingknoten für die gegebene Anwendung vorteilhafter sind.

Weitere Details zu der hier aufgezeigten Methode der ressourceneffizienten Abbildung von Anwendungen auf Chip-Multiprozessoren sind [102][103][104] und [168] zu entnehmen.

## 6.9 Zusammenfassung

In diesem Kapitel wurde eine Methode vorgestellt, die es dem Entwickler ermöglicht, die zunächst universell einsetzbare und nicht spezialisierte Struktur des GigaNetIC-Chip-Multiprozessors für ein gewünschtes Anwendungsgebiet im Hinblick auf die Ressourceneffizienz der Architektur zu optimieren. Der hierarchisch gerichtete Ansatz bietet den Vorteil, dass, unterstützt durch die entwickelte Werkzeugkette, zunächst mit vergleichsweise geringen Modifikationen die Leistungsfähigkeit bzw. der Ressourcenbedarf der Chip-Multiprozessor-Architektur optimiert werden kann. Durch die leistungsfähigen Profilierungsmöglichkeiten der GigaNetIC-Entwicklungsumgebung lassen sich besonders rechenintensive Funktionen der Anwendungssoftware schnell lokalisieren. Durch anschließende Analyse können sowohl betreffende Stellen der Software als auch, falls notwendig, die Hardware zielgerichtet optimiert werden. Diese Optimierung geschieht im Regelfall hierarchisch gerichtet, angefangen bei Instruktionssatzerweiterungen, über eng-gekoppelte Hardwarebeschleuniger bis hin zu lose gekoppelten Hardwarebeschleunigern. Letztendlich steht dem Softwarearchitekten dann die Nutzung der parallelen Struktur zur parallelen Bearbeitung einer Aufgabe zur Verfügung, deren Leistungsfähigkeit ggf. durch den GigaNetIC-Multiprozessorcache zusätzlich erhöht werden kann.

Die aufzuwendende Zeit für diese Optimierungsmaßnahmen auf Prozessorebene liegt im Bereich von einigen Stunden bis hin zu wenigen Tagen. Sollten Modifikationen auf Prozessorebene nicht genügend Optimierungspotential für die gegebene Anwendung bieten, so kann dies bereits innerhalb der ersten Stunde mit Hilfe der Werkzeugkette festgestellt werden. In diesem Fall können tieferegehende Optimierungen in Form der Realisierung spezialisierter Hardwarebeschleuniger durchgeführt werden. Dieser Prozess benötigt im Allgemeinen deutlich mehr Zeit und Ressourcen.

Die werkzeuggestützte Analyse des jeweiligen Anwendungsszenarios liefert Aussagen sowohl über den Rechenleistungsbedarf aber auch über die benötigten Bandbreiten der On-Chip-Kommunikation. Die GigaNetIC-Architektur eröffnet, aufgrund der generisch gehaltenen Struktur, zahlreiche Möglichkeiten, das System anwendungsgemäß zu optimieren.

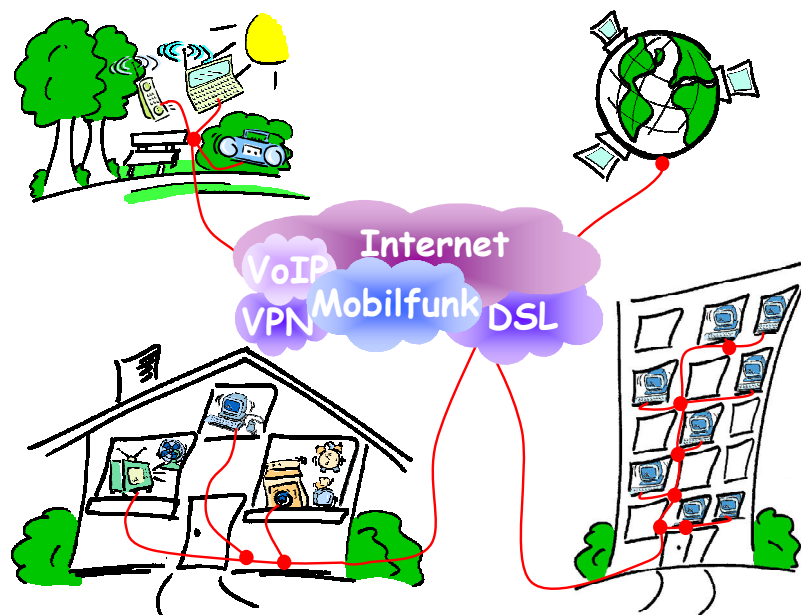
Geeignete Anwendungen lassen sich durch NetAMap automatisiert auf das Chip-Multiprozessor-system abbilden. Dies verkürzt Entwicklungszeiten und führt zu einer besonders effizienten Nutzung der parallelen Architektur. Unterstützt durch die Werkzeugkette lässt sich für die jeweils betrachtete Anwendung ein geeigneter Kompromiss zwischen Leistungszuwachs, Verlustleistungsaufnahme, Flächenbedarf und zusätzlich zu erwartendem Entwicklungsaufwand treffen. Pareto-optimale Punkte des Entwurfsraums können so effizient angenähert werden. Anhand einer exemplarischen Analyse verschiedener Realisierungsvarianten für ein paketverarbeitendes System wurde die in Kapitel 3 vorgestellte Kostenfunktionsmethode verifiziert und deren Leistungsfähigkeit aufgezeigt. Mit Hilfe definierter Parameter für die Zielfunktionen der vier Kostenmaße Leistungsaufnahme, Flächenbedarf, Performanz und Zukunftssicherheit sowie der resultierenden Kostenfunktion wurden in Relation zu den definierten Randbedingungen pareto-optimale Systeme für unterschiedli-

che Einsatzgebiete ermittelt. Systemarchitekten wird hiermit eine nützliche Entscheidungshilfe für den Entwurf ressourceneffizienter Implementierungen an die Hand gegeben.

Im folgenden Kapitel werden am Beispiel von Netzwerkanwendungen die hier vorgestellten Optimierungsmöglichkeiten auf die GigaNetIC-Architektur angewendet und detailliert diskutiert.

## 7 Performanzanalyse skalierbarer GigaNetIC-Netzwerkprozessoren

Eine immer größer werdende Herausforderung der heutigen Informationsverarbeitung stellt das Verarbeiten von Daten in Sprach- und Datennetzwerken dar. Insbesondere das Internet und die damit verbundenen Dienste erfordern neue leistungsfähigere Architektur-Ansätze, die speziell auf die inhärente Parallelität von Netzwerkdaten ausgelegt sein sollten. Für solche Einsatzgebiete sind parallele Architekturen wie die GigaNetIC-Chip-Multiprozessorarchitektur prädestiniert. Eine Vielzahl von Verarbeitungseinheiten kann integriert werden und die Netzwerkverarbeitung voneinander disjunkter, unkorrelierter Netzwerkpakete durchführen, wobei sich die globale Zustandsverwaltung häufig als einzige gemeinsame Aufgabe (*Task*) darstellt, die einem der Kerne aus dem Prozessor-Pool zugewiesen werden kann (vgl. Abschnitt 4.5.3). Diese Klasse von Anwendungsszenarien skaliert sehr gut (vgl. GUSTAFSONS Gesetz, Abschnitt 2.1.2).



**Abbildung 7-1: Zunehmende Vernetzung unserer Umgebung, die durch ressourceneffiziente skalierbare, Netzwerkknoten ermöglicht werden kann**

Die von Netzwerkknoten geforderte Funktionalität geht in immer stärkerem Maße über das reine Weiterleiten von Datenpaketen hinaus. Neben der Auswertung der im Paketkopf (*Header*) abgelegten Adressen, die dazu dient, Daten an den richtigen Empfänger weiterzuleiten, werden zusätzliche Informationen verarbeitet, um erweiterte Dienste wie Verschlüsselung (z. B. für *Virtual Private Networks*), *Network Address Translation (NAT)* oder Priorisierung *Quality of Service (QoS)* anzubieten. Die Leistungsfähigkeit, die in Netzwerkkomponenten für diese Verarbeitungsmechanismen zur Verfügung gestellt werden muss, kann mit herkömmlichen Prozessorarchitekturen nicht erreicht werden. In der Vergangenheit haben sich daher ASIC-basierte Lösungen etabliert, die – teilweise unterstützt durch RISC-Prozessoren – die geforderte Leistungsfähigkeit bieten. Nachteile ASIC-basierter Systeme sind allerdings relativ lange Entwicklungszeiten und hohe Entwicklungs- und Fertigungskosten. Insbesondere Änderungen der Netzwerkprotokolle sind oft mit aufwändigen und teuren Überarbeitungen (*Redesigns*) verbunden. Vor diesem Hintergrund wurden in den letzten Jahren verstärkt so genannte Netzwerkprozessoren entwickelt. Netzwerkprozessoren sind programmierbare Spezialbausteine für den Aufbau von Netzwerkkomponenten, die den geringeren Preis und

die Flexibilität von RISC-Prozessoren mit der Leistungsfähigkeit und Skalierbarkeit von anwendungsspezifischen Bausteinen kombinieren. Eine Optimierung der Architektur für die Paketverarbeitung wird vorrangig durch drei Mechanismen erreicht: *Modifikation des Instruktionssatzes*, *Hinzufügen von Hardwarebeschleunigern* sowie *Entwurf von On-Chip-Architekturen*, die Parallelverarbeitung und Pipelining ausnutzen, vgl. auch Kapitel 6. Obwohl Implementierungen von Netzwerkprozessoren erst seit wenigen Jahren verfügbar sind, gab es bereits im Jahre 2002 über 30 Hersteller von kommerziellen Netzwerkprozessoren [169], die sich in ihrem Aufbau teilweise deutlich unterscheiden, darunter so bekannte Namen wie Intel (IXP2800), IBM (PowerNP), Motorola (C-5) oder Cisco (PXF).

Abbildung 7-1 zeigt wesentliche Einsatzgebiete für Netzwerkprozessoren auf. Augenscheinlich wird hier, dass je nach Funktion der Netzwerkkomponente oft gleichartige Algorithmen verarbeitet werden müssen, allerdings in deutlich unterschiedlichem Umfang, Zeitrahmen und bei mobilen Geräten teilweise mit sehr eingeschränkten Energieressourcen. Die Randbedingungen bzw. Anforderungen unterscheiden sich mitunter beträchtlich. Ein mobiles Endgerät benötigt weniger Rechenleistung als ein Router des Kernnetzwerks, verarbeitet einige wenige Datenströme, hat aber auch ein sehr limitiertes Energiebudget. Dem Router hingegen wird ein höheres Kontingent zugeschrieben, er muss jedoch eine weitaus höhere Bandbreite, verbunden mit deutlich höherer Rechenleistung zur Verfügung stellen. Mittlerweile spielt allerdings auch bei diesen Hochleistungskomponenten des Netzwerks die Leistungsaufnahme eine immer größer werdende Rolle, da die Kosten für Kühlung und Strom bei großen Anlagen einen immer größer werdenden Stellenwert einnehmen [13], vgl. Kapitel 2. Auch die Lärmbelästigung durch etwaig benötigte Lüfter wird, speziell im Bereich der „letzten Meile“, also den Verteileranlagen im Bereich der Hausanschlüsse der Endkunden, immer stärker diskutiert. Könnte man den sich hieraus ergebenden Anforderungen mit einem ressourceneffizienten Architekturkonzept gerecht werden, so würden sich u. a. gravierende wirtschaftliche Vorteile im Sinne von – *Economies of Scale* –, häufig auch als Skaleneffekt bezeichnet, ergeben. Hardwareentwickler wären bei einer modularen, skalierfähigen Bauweise der Netzwerkprozessorarchitektur in der Lage, eine hohe Wiederverwendbarkeit und somit Entwurfszeitersparnisse zu erzielen. Verarbeitungseinheiten mit höheren Rechenleistungsanforderungen könnten aus kleineren leistungsschwächeren Verarbeitungseinheiten zusammengesetzt werden und durch Parallelverarbeitung ihren Anforderungen ressourceneffizient gerecht werden.

Softwareentwickler könnten ein einheitliches Programmiermodell verwenden und die Produktivitätskurve würde aus Gründen der Wiederverwendbarkeit der Software ebenso wie bei der Hardware deutlich gesteigert. Die GigaNetIC-Architektur eröffnet diese Möglichkeiten, weshalb im Folgenden eine tiefere Analyse GigaNetIC-basierter Netzwerkprozessoren für Anwendungen aus dem Zugangsnetzwerkbereich (*Access Networks*) und dem Kernnetzwerkbereich (*Core Network*) vorgestellt werden. Im Bereich mobiler Anwendungen (*MANets*) wurde die Leistungsfähigkeit der GigaNetIC-Architektur bereits skizziert, vgl. Abschnitt 6.8.2.

## 7.1 Einsatzgebiet im Zugangsnetzwerk – DSLAM

Eines der Einsatzgebiete für einen GigaNetIC-basierten Netzwerkprozessor ist die Anbindung der „letzten Meile“, bei der es um die Bündelung bzw. Verteilung von *DSL* (*Digital Subscriber Line*)-Anschlüssen geht. Die hierfür benötigten Knoten, die im Zugangs(*Access*)-Bereich des Netzwerks anzusiedeln sind, nennt man *DSLAMs* (*DSL Access Multiplexer*). Abbildung 7-2 skizziert dieses

Anwendungsszenario, bei dem N-Core-basierte GigaNetIC-Chip-Multiprozessor-Systeme sowohl in dem eigentlichen *DSLAM* aber auch in skaliertem Weise in den Routern des Kernnetzwerks und den Endgeräten (*Customer-Premises Equipment / CPEs*) in den Haushalten der Verbraucher eingesetzt werden, um so eine gute Kosteneffizienz erzielen zu können.

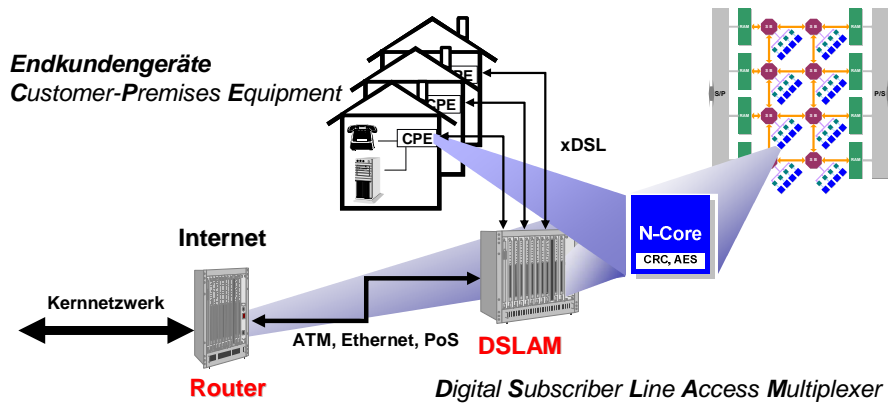


Abbildung 7-2: Einsatz GigaNetIC-basierter Netzwerkprozessoren in einem *DSLAM*-Anwendungsszenario

Hauptkomponenten eines *DSLAMs* sind heute bis zu 64 *Linecards*, die jeweils bis zu 96 *DSL-Anschlüsse (Ports)* zur Verfügung stellen, und eine *Uplinkcard*, die den Verkehr zum Netzbetreiber (*Internet Service Provider / ISP*) regelt. Die *Linecards* sind über eine leistungsfähige Kopplung (*Backplane*) mit einer oder zwei *Uplinkcards* verbunden. Aus Kostengründen werden diese Kommunikationskanäle in Zukunft immer häufiger ethernetbasiert sein [141].

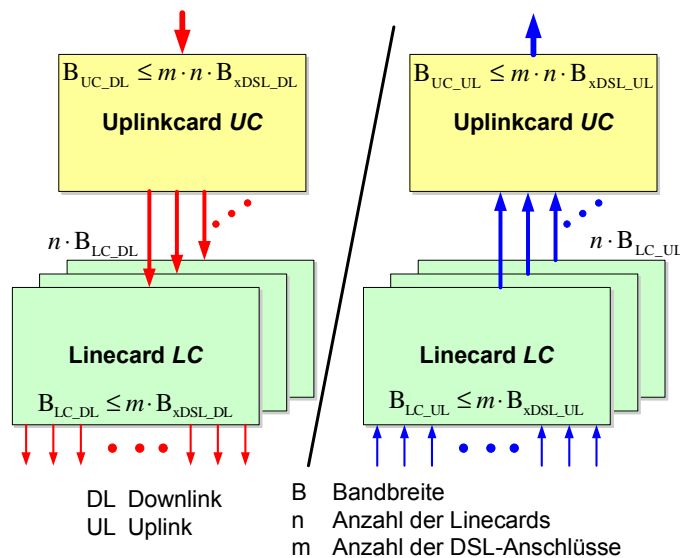


Abbildung 7-3: *DSLAM*-System-Komponenten

Abbildung 7-3 zeigt schematisch die Komponenten und die logischen Datenflussrichtungen innerhalb eines *DSLAMs*. Die sich ergebenden Bandbreiten  $B$  sind für die vier unterschiedlichen Fälle eingetragen. Bei einem *DSLAM* unterscheidet man prinzipiell zwischen zwei Flussrichtungen, den *Uplink*, über den die *CPEs* der Endkunden den Diensteanbieter (*ISP*) erreichen, und den, in den meisten Fällen breitbandigeren *Downlink*, über den die Daten zu den Endkunden übertragen werden. Dieser Systemaufbau erfordert in den einzelnen Komponenten spezielle Funktionen. Diese müssen zudem mit, den auftretenden Bandbreiten angepasst, Ressourcen verarbeitet werden. Die entsprechenden Überlegungen zu den funktionalen, wie auch verkehrsmodellbedingten Anforderungen an einen *IP(Internet Protokoll)*-basierten *DSLAM* (kurz: *IP-DSLAM*) werden im nächsten

Abschnitt 7.2 in einem von uns definierten neuartigen *IP-DSLAM-Benchmark* formuliert [141]. Ein herkömmlicher *DSLAM* ist nach dem OSI-Referenzmodell (vgl. Abschnitt 2.3.2) auf Schicht zwei angesiedelt und so für das *IP*-Protokoll transparent. Der modernere *IP-DSLAM* stellt zusätzliche Dienste zur Verfügung und kann deshalb auch *IP*-Datenverkehr terminieren bzw. verarbeiten.

DSL-basierte Internetzugänge, die zu den so genannten Breitbandanschlüssen zählen, haben, nicht nur in Deutschland, in den vergangenen Jahren einen immensen Zuwachs erfahren. Seit 2002 liegen die jährlichen Zuwachsraten der DSL-Anschlüsse in Deutschland bei ca. 50 %, was Ende 2006 eine Gesamtsumme von 14,9 Mio. Breitbandzugängen ausmachte [170]. Seit 2003 steigt das Breitbandverkehrsvolumen in Deutschland jährlich um ca. 30 % und betrug 2006 nach Schätzungen ca. 876 Mio. GByte [170]. Diese Zahlen zeigen, dass das DSLAM-Anwendungsgebiet sowohl von der Anzahl der benötigten Verarbeitungseinheiten als auch aufgrund der stetig wachsenden Anforderungen an die benötigten Recheneinheiten wirtschaftlich und wissenschaftlich interessant ist. Um den Herausforderungen gerecht werden zu können, müssen skalierbare und zugleich zukunftssichere Netzwerkprozessorarchitekturen im Sinne der Anwendungssoftware und der Programmierbarkeit entwickelt werden. Im Folgenden wird die Verwendbarkeit der GigaNetIC-Architektur für den Einsatz in solchen DSLAM-Anwendungen untersucht und auf spezielle Erfordernisse dieses Anwendungsgebiets hin optimiert.

## 7.2 Definition eines IP-DSLAM-Benchmarks auf Systemebene

Um eine eingängige Bewertung der GigaNetIC-basierten Netzwerkprozessorarchitektur für ein *IP-DSLAM*-Anwendungsszenario durchführen zu können, bedurfte es einer genauen Charakterisierung der Anforderungen die diese Anwendung an die Hard- und Software stellt. Eine Benchmarkdefinition einschließlich eines realistischen Verkehrsmodells (*Traffic Model*) war notwendig. Zu dem Zeitpunkt unserer Analysen waren bereits mehrere Netzwerkprozessorbenchmarks bekannt, von denen wir in [141] acht der bekanntesten in Bezug auf die Anwendbarkeit auf ein *IP-DSLAM*-Szenario diskutieren.

Tabelle 7-1: Charakteristika etablierter Netzwerkbenchmarks

	Granularität	Quellcode	Verfügbarkeit	Verkehrsmodell	Profilierung
<b>CommBench</b>	Mikrobenchmark	C	auf Anfrage	Nein	Nein
<b>EEMBC</b>	Mikrobenchmark	C	Mitgliedschaft	Ja	Ja
<b>MiBench</b>	Mikrobenchmark	C	frei verfügbar	Ja	Nein
<b>NPBench</b>	Mikrobenchmark	C	auf Anfrage	Nein	Nein
<b>NetBench</b>	Mikrobenchmark/ Funktionsebene	C	frei verfügbar	Ja	Nein
<b>NPF-BWG</b>	Funktionsebene	Nein	nur textuell	Ja	Ja
<b>Intel</b>	Funktionsebene	Mikro-Engine-C	nein	k. A.	Ja
<b>LinleyBench</b>	Funktionsebene	Nein	Lizenz	Ja	Ja
<b>GigaNetIC IP-DSLAM-Referenz-BM</b>	Systemebene	C	auf Anfrage	Ja	Ja

Dies waren im Einzelnen: CommBench [171], EEMBC [161], MiBench [172], NetBench [173], NPF Benchmarking Group [174], LinleyBench [175], Intel Corporation Benchmark [176] und NPBench [177]. Die wesentlichen Charakteristika der genannten Benchmarks sind in Tabelle 7-1

aufgeführt. Bei der Granularität der Benchmarks kann zwischen *Mikro-*, *Funktions-* und *Systemebene* unterschieden werden. Die Mehrzahl der hier erwähnten Benchmarks setzt auf *Mikroebene* an, d. h. es kommen elementare Tasks wie CRC-Funktionen und Tabellenauswertung (*Table Lookup*) zum Einsatz. Einige der Benchmarks setzen auf *Funktionsebene* an, wobei die am häufigsten verwendete Funktion im Weiterleiten von IPv4-Paketen (*IPv4-Forwarding*) besteht. Allerdings setzt keiner der Ansätze auf Systemebene an, was für eine realistische Modellierung eines *IP-DSLAM-Benchmarks* zwingend notwendig ist.

Letztendlich kamen wir bei der Analyse dieser Benchmarks, der zugehörigen Verkehrsmodelle sowie der Profilierungsmöglichkeiten zu dem Schluss, dass sie nur unzureichend für ein IP-DSLAM-Szenario auf *System-/Taskgraphebene* anwendbar waren. Deshalb haben wir in [141] einen eigenen IP-DSLAM-Benchmark nebst adaptivem Verkehrsmodell entworfen, der in der Forschungsabteilung von Infineon Technologies zum IP-DSLAM-Referenzbenchmark ausgebaut wurde [178][119].

### 7.2.1 Funktionelle Spezifikation

Im Folgenden werden die für den *IP-DSLAM-Benchmark* relevanten *Tasks* kurz erläutert. Diese formen in Abhängigkeit von der Datenrichtung und der jeweiligen DSLAM-Systemkomponente den eigentlichen *Systemebenen-Benchmark*.

**Task-A – Parser.** Im *Task-A* wird eine Gültigkeitsüberprüfung des Datenpakets durchgeführt. Ein Algorithmus, der die Maßgaben, die ein gültiges IPv4-Paket kennzeichnen, überprüft, wird in diesem Schritt durchlaufen. Schlägt die Überprüfung fehl, wird das Paket verworfen.

**Task-B – IP-Header Verification.** Hier werden mittels eines Algorithmus aus [179] der IP-Paketkopf nach den Vorgaben sowie die *IP-Headerprüfsumme* verifiziert.

**Task-C – Classification.** Der Algorithmus (*IP-Filter*) klassifiziert die Datenpakete unter Berücksichtigung der Quell- und Ziel-IP-Adresse, dem Quell- und dem Zielpport und der Protokollkennzeichnung. Anhand des Klassifizierungsergebnisses dieser fünf Tupel wird ein Zugangs(*Access*)-Status ermittelt, der über das weitere Verfahren mit dem Paket entscheidet. Realisiert wird dies mit Hilfe einer individuell anpassbaren Vergleichstabelle, deren Einträge iterativ mit den jeweiligen fünf Tupeln der Pakete verglichen werden. Kann keine Tupelwert-Übereinstimmung ermittelt werden, wird das Datenpaket verworfen. Im *Downlink* kann der Zugangs-Status für Datenvolumenbegrenzungszwecke (*Conditioning*) eingesetzt werden, im *Uplink* können Bandbreitenzusagen (*Service Level Agreements / SLA*) aufgrund des ermittelten Wertes kontrolliert bzw. eingehalten werden.

**Task-D – MC Address Mapping & Duplication.** *Task-D* beschreibt die Funktionalität der Datenpaketvervielfältigung im Sinne von *Multicast* bzw. *Broadcast* für Netzwerkteilnehmer.

**Task-E – Policing & Conditioning.** Der *Task-E* fungiert als Überwachungsfunktion zum Verkehrsmanagement (*Policing*) von ausgehenden Datenpaketen im *Uplink*. Im *Downlink* werden die Datenraten mit Hilfe von so genannten *Token-Buckets* reguliert. Die Verwendung dieser *Buckets* dient der Datenverkehrsglättung mit dem gewünschten Ziel, eine diskontinuierliche Datenverkehrscharakteristik so zu organisieren, dass annähernd kontinuierliche Datenströme entstehen (*Conditioning / Shaping*).

**Task-F – AAL5 Segmentation.** Um den derzeitigen Standards gerecht zu werden, wurde für die integrierte Nutzung des *ATM(Asynchronous Transfer Mode)*-Übertragungsverfahrens, eine Segmentierung der Datenpakete in *ATM-Zellen* vorgesehen. Der Algorithmus nimmt diese Segmentbildung

unter Maßgabe des *AAL5(ATM Adaptation Layer 5)*-Dienstes vor. Hierzu müssen u. a. die *CRC8* für jede *ATM*-Zelle als auch die *CRC32* für das gesamte Paket berechnet werden.

**Task-G – NPF-ML (Network Processor Forum Message Layer Protocol) – Tag-Generierung.**

Der Einsatz dieses Algorithmus ist nur auf der *Uplinkcard* (im *Downlink*) gegeben und wird zur Parametrisierung der Mehrfachzustellung (*Small Group Multicast*) der Datenpakete von der *Uplinkcard* in Richtung *Linecard* und dort letztendlich hin zum Endkunden verwendet. *Multicast*-Pakete, die als Ziel die gleiche *Linecard* haben, werden nicht dupliziert, sondern mit Hilfe einer Liste auf die entsprechenden Portnummern verteilt. Hierzu wird das *NPF-ML*-Protokoll verwendet.

**Task-H – Ethernet-Encapsulation & Forwarding.** In diesem Schritt wird die Übermittlung des *IP*-Pakets über *Ethernet* initiiert. Nach der Kapselung des Datenpakets in einen so genannten *Ethernet-Frame* erfolgt das Weiterreichen an die Bit-Übertragungsschicht. Mit Hilfe des *CRC32* wird die Prüfsumme unter Berücksichtigung der Ziel- und Quelladressen, des *Type*-Feldes und des *Ethernet*-nutzdatenbereichs gebildet. Der Nutzdatenbereich enthält das zu transferierende Datenpaket.

Für weitere Details zu den genannten Tasks sei an dieser Stelle auf [141] verwiesen.

## 7.2.2 Implementierung

Das erstellte Anwendungsszenario beschreibt eine Testumgebung, die die Funktionalität eines *IP-DSLAMs* unter Verwendung der oben genannten Tasks realisiert und eine Weiterentwicklung des klassischen *DSL-Access-Multiplexers* der *OSI*-Schicht zwei darstellt. Die enthaltenen Erweiterungen berücksichtigen u. a. Aspekte *QoS(Quality of Service)*-basierter Datenverarbeitung der dritten Schicht des *OSI*-Modells [34].

**Tabelle 7-2: Taskzuordnung in Abhängigkeit von DSLAM-Komponente und Datenrichtung**

Tasks	Uplinkcard		Downlinkcard	
	Uplink	Downlink	Uplink	Downlink
Task A: Parser	X	X	X	X
Task B: Headercheck	X	X	X	X
Task C: Classification		X		X
Task D: MC-Duplication		X		X
Task E: Policing & Conditioning		X		X
Task F: AAL5-Segmentation			X	X
Task G: NPF-ML		X		
Task H: Ethernet Encapsulation		X	X	

Die einzelnen Funktionalitäten des *DSLAM*-Anwendungsszenarios wurden in der Programmiersprache *ANSI-C* implementiert und sind in die oben genannten, in sich funktional geschlossenen Aufgaben (*Tasks*) unterteilt. Der Quellcode umfasst 37 Dateien mit insgesamt 4520 Codezeilen. Tabelle 7-2 zeigt die sich in Abhängigkeit von der Datenrichtung des Paketstromes und der hardwaretechnischen Konfiguration ergebende Zuordnung der Tasks (auszuführende Tasks sind mit einem „x“ gekennzeichnet). Zur sinnvollen Verwendung dieses synthetischen Benchmarks, müssen möglichst realitätsnahe Datenströme generiert werden, die sich stark an real auftretenden Gegebenheiten des Anwendungsgebiets orientieren. Die Konzeption dieser Daten wird im folgenden Abschnitt kurz vorgestellt.

## 7.2.3 Verkehrsmodell

Das Verkehrsmodell (*Traffic Model / TM*) definiert die im *DSLAM* auftretende Nutzlast und muss drei wesentliche Kategorien abdecken: Adressbereichsverteilung, *QoS*-Verteilungen und Paketgrößen-Verteilungen.



**Adressbereiche.** Zur Adressierung von Teilnehmergruppen im Anwendungsszenario wird ein IP-Adressraum von 224.0.0.0 bis 239.255.255.255 verwendet. Dieser entspricht einem Klasse-D-Netz, welches zum Adressieren von *Multicastgruppen* genutzt wird. Im hier vorliegenden System sind 32 Gruppen mit je 96 Teilnehmern definiert. Der Anteil der zu bearbeitenden *Multicast*-Pakete beläuft sich auf 1 % bis 9 % aller Pakete. Die übrigen 99 % bis 91 % der Pakete bilden die Menge der *Unicast*-Pakete, die nur an einen Teilnehmer gesendet werden müssen. Zur Adressierung der maximal 3072 Hosts wird ein Adressrahmen von x.y.0.0 bis x.y.12.0 gewählt, um alle Teilnehmer erreichen zu können. Die Quell- und Ziel-IP-Adressen liegen im gleichen Adressbereich. Weiterhin wird festgelegt, dass bis zu 16 Flüsse über den Zielport (*Layer 4*) möglich sind.

**QoS.** Die zu leistende Dienstgüte (*QoS*) ist abhängig von der Adressvereinbarung und den verwendeten Regeln zur Klassifizierung der Pakete. Die Klassifikationsregeln sind so ausgelegt, dass 50 % der Pakete bevorzugt weitergeleitet werden (*Expedited Forwarding*), 36 % den vier Subklassen des *QoS* entsprechend mit jeweils dreifacher Priorisierung bzgl. des Verwerfens (*Assured Forwarding*) zugesichert verarbeitet werden und die restlichen 14 % bei freier Bandbreite weitergeleitet werden (*Best Effort*).

**Paketgröße.** Die Paketgrößen werden nach der etablierten 7:4:1-Verteilung, dem *Internet Mix (iM-ix)* festgelegt. Er spiegelt eine durchschnittliche Paketgrößenverteilung, die häufig in der Realität zu messen ist, wider. Das bedeutet sieben Pakete minimaler Länge à 40 Byte<sup>51</sup>, die z. B. durch *TCP*-Bestätigungen, aber nicht durch Nutzdaten hervorgerufen werden. Weiterhin werden vier Pakete mit 552 bzw. 576 Byte injiziert, die z. B. von *TCP*-Implementierungen die keine *MTU* (*Maximum Transmission Unit*)-Ermittlung durchführen, herrühren. Komplettiert wird das repräsentative Datenaufkommen durch ein Paket maximaler Länge à 1500 Byte, das z. B. von *TCP*-Implementierungen stammt, bei denen die maximale Übertragungsgröße festgelegt wurde. Weitere Details zur Verteilung und Injektion der Pakete sind [141] zu entnehmen. Zu beachten ist, dass diese Paketgrößen sich immer ohne die 18-Byte-großen *Ethernetheader* verstehen.

#### 7.2.4 Bewertungsmethode zum Vergleich unterschiedlicher Architekturen

Vorraussetzung zur Anwendung des *IP-DSLAM*-Benchmarks ist ein zyklenakkurater Simulator mit entsprechender Werkzeugkette, was nahezu unverzichtbar für den Erfolg einer Hardwarearchitektur ist und somit durchaus gefordert werden kann. Aufgrund der Benchmarkimplementierung in der Hochsprache C ist die Portierung auf eine Vielzahl von Zielarchitekturen relativ einfach. Es wird explizit keine Beschleunigung durch optimierte Assemblerrouinen angewendet oder empfohlen, da der Benchmark die Bewertung der gesamten Architektur zum Ziel hat, also auch die Compiler-Werkzeugkette in die Performanzbewertung einbezogen werden soll. Dies liegt darin begründet, dass im untersuchten Anwendungsgebiet schnell auf sich verändernde oder zusätzliche Anforderungen reagiert werden können soll. Außerdem gebietet eine ressourceneffiziente Architektur im Sinne von Definition 36 eine leichte Portierbarkeit auf zukünftige Systeme. Dies spricht für eine Implementierung der Anwendungssoftware in einer Hochsprache. Zusätzlich wird so die Wartbarkeit des Programmcodes deutlich erhöht.

---

<sup>51</sup> In der Literatur wird teilweise auch 46 Byte anstelle der 40 Byte angenommen.

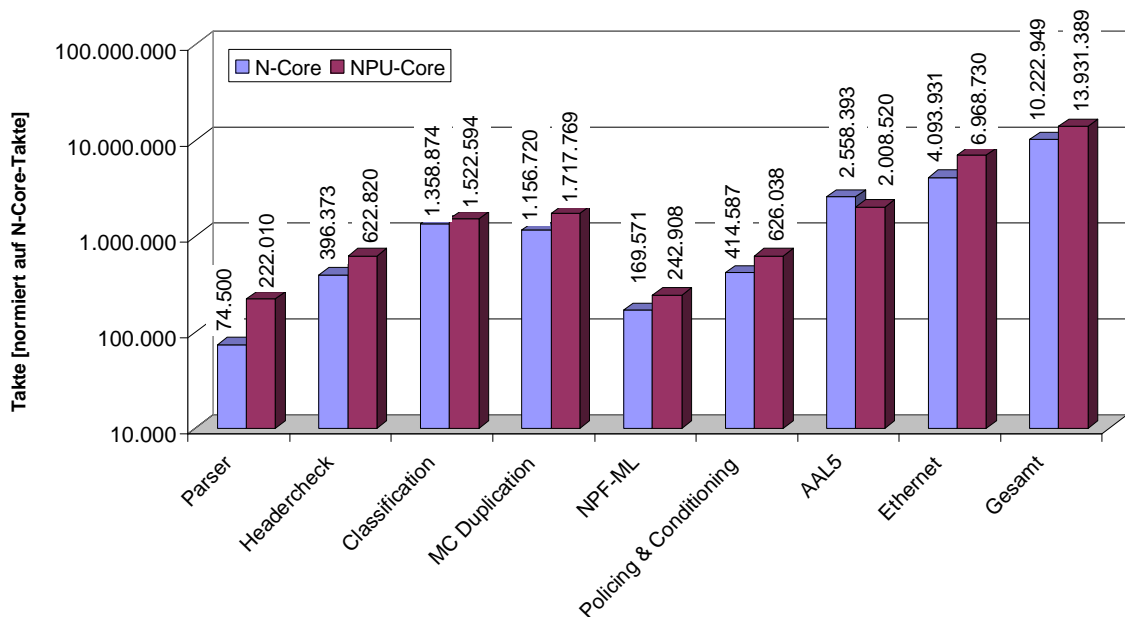
Um aussagekräftige Werte zum Vergleich unterschiedlicher Architekturen bzgl. der Performanz zu erhalten, wird folgende Normalisierungsfunktion angewendet:

$$\frac{\sum_{t \in \text{Tasks}} \text{Takte}_t}{\# \text{Tasks} \cdot \sum_{p \in \text{Pakete}} \text{Größe}_p [\text{Byte}]} = \text{Performanz} \left[ \frac{\text{Takte}}{\text{Bit}} \right] \quad (7.1)$$

$\# \text{Tasks}$  beziffert die Anzahl der eingesetzten *Tasks* und ist im Weiteren mit acht gleichzusetzen.

## 7.2.5 DSLAM-Benchmarkanalysen für skalierbare GigaNetIC-CMPs

In diesem Abschnitt werden erste Benchmarkergebnisse vorgestellt, die mit dem in Abschnitt 7.2 vorgestellten *IP-DSLAM*-Benchmark gewonnen wurden. Zunächst werden zwei eingebettete Prozessoren untersucht, zum einen der N-Core (vgl. Abschnitt 4.3.1) und zum anderen eine speziell auf Paketverarbeitung optimierte Verarbeitungseinheit (*NPU-Core*), ähnlich der aus [180][181]. Beim *NPU-Core* handelt es sich um einen 32-Bit-RISC-Kern, der über eine hardwareunterstützte Behandlung mehrerer *Threads* verfügt. Außerdem erlaubt seine 6-stufige *Pipeline* eine fast doppelt so hohe Betriebsfrequenz wie die des N-Core. Die betrachtete *DSLAM*-Ausbaustufe entspricht der in Abbil-



dung 7-3 vorgestellten Maximalsystemkonfiguration mit  $n = 64$  *Linecards* und  $m = 96$  *xDSL-Ports*.

**Abbildung 7-4: Anforderungen der einzelnen *IP-DSLAM*-Benchmarktasks an die Rechenleistung der eingebetteten Prozessoren**

Um die maximal mögliche Rechenlast zu bestimmen, wird ein *Worst-Case*-Szenario betrachtet, bei dem die maximalen Bandbreiteneanforderungen ausgeschöpft werden, vgl. (7.2).

Das zugrundeliegende Verkehrsmodell entspricht dem aus Abschnitt 7.2.3, wobei 2981 Pakete mit einer Gesamtgröße von 0,955 MByte verarbeitet wurden. Abbildung 7-4 zeigt die benötigten Takte für die Abarbeitung der einzelnen Tasks, sowie die Gesamtzahl der für den Benchmark benötigten

Takte, normiert auf die N-Core-Takte<sup>52</sup>. Augenscheinlich sind die besonders rechenintensiven Aufgaben die Tasks AAL5 und Ethernet, was durch die aufwändigen CRC-Prüfsummen und Kopiervorgänge der Pakete begründet ist. Der N-Core ist bis auf die Funktion AAL5 überall zeiteffizienter, in Bezug auf die Anzahl benötigter Takte entscheidet er jeden *Task* für sich.

$$B_{UC\_UL} = m \cdot n \cdot B_{xDSL\_UL}, \text{ mit } B_{xDSL\_UL} = \begin{cases} 0,8 \text{ Mbps für ADSL} \\ 3 \text{ Mbps für VDSL} \\ 2 \text{ Mbps für SHDSL} \end{cases} \quad (7.2)$$

$$B_{UC\_DL} = m \cdot n \cdot B_{xDSL\_DL}, \text{ mit } B_{xDSL\_DL} = \begin{cases} 8 \text{ Mbps für ADSL} \\ 22 \text{ Mbps für VDSL} \\ 2 \text{ Mbps für SHDSL} \end{cases}$$

Im Durchschnitt benötigt der *NPU-Core* 3,5 Takte/Paket-Bit, wohingegen der N-Core weniger als 1,3 Takte aufwenden muss. Eine grobe Analyse der *Tasks* bezüglich ihrer Abbildung auf die beiden Prozessoren zeigt schnell erste Flaschenhälse der reinen Paketverarbeitungseinheit (*NPU-Core*) auf. Der Instruktionssatz der Paketverarbeitungseinheit ist hoch spezialisiert für besondere Bitoperationen, aber beinhaltet nur eine eingeschränkte Menge an Universalbefehlen.

Tabelle 7-3: Benötigte Takte des N-Cores

N-Core	Linecard		Uplinkcard	
	Downlink	Uplink	Downlink	Uplink
Parser	74.500	74.500	74.500	74.500
Headercheck	396.373	399.353	396.373	417.233
Classification	1.358.874	-	1.358.874	-
MC Duplication	1.156.720	-	1.171.597	-
NPF-ML	-	-	169.571	-
Policing & Conditioning	414.587	-	414.587	-
AAL5	2.558.393	2.543.493	-	-
CRC Beschleuniger	4.093.931	6.024.563	99.099	-
Ethernet Framing	-	309.920	349.886	-
CRC in Software	43.023.639	63.901.032	966.966	-
<b>Gesamt</b>	<b>10.053.378</b>	<b>9.351.829</b>	<b>4.034.487</b>	<b>491.733</b>
Takte/Bit	1,255	1,167	1,986	0,061

Die Analysen zeigen, dass der positive Effekt dieser Spezialoperationen mehr als aufgehoben wird. Der Hauptgrund dafür liegt in der Verteilung der Instruktionshäufigkeiten. Können doch nur 10 % des Codes von den Spezialoperationen profitieren, wohingegen die verbleibenden 90 % des Codes Universalbefehle implizieren. Ein weiterer Punkt liegt in dem Compiler begründet, der weniger gut für die Instantiierung der Spezialbefehle geeignet ist. Abhilfe schaffen könnte eine völlige Restrukturierung des *NPU-Core-Compilers*, oder aber eine aufwändige Programmierung von Hand in Maschinensprache (*Assembler*), was bezüglich der Wartung und Zukunftssicherheit äußerst bedenklich wäre<sup>53</sup> und die Ressourceneffizienz deutlich herabsetzte.

<sup>52</sup> Da der N-Core bzgl. der maximalen Taktfrequenz nur halb so schnell getaktet werden kann wie der *NPU-Core*, wird hier ein N-Core-Takt praktisch mit zwei *NPU-Core*-Takten gleichgesetzt.

<sup>53</sup> Die Ergebnisse dieser Studie führten zu einer kompletten Restrukturierung des Architekturansatzes bei dem bisher verwendeten *NPU-Core*.

Tabelle 7-3 stellt die benötigten Takte des N-Core-Prozessorkerns für ein *IP-DSLAM-Light*-Szenario, ähnlich dem in Tabelle 7-2 vorgestellten Modell, das den asymmetrischen Charakter der Anwendung besonders deutlich macht. In diesem Fall benötigt der N-Core 1,167 Takte/Bit für den *Uplink* und 1,255 Takte/Bit für den *Downlink* auf der *Linecard*, wohingegen im *Uplink* auf der *Uplinkcard* nur 0,061 Takte/Bit anfallen. Besonders rechenintensiv ist der *Downlink* auf der *Uplinkcard* mit 1,986 Takten/Bit. Um einen Eindruck von den Leistungsanforderungen einer *Linecard*<sup>54</sup> mit  $m = 96$  Ports eines *IP-DSLAMs* (vgl. Abbildung 7-3) zu vermitteln, wird in Tabelle 7-4 die Anzahl der jeweils benötigten Verarbeitungseinheiten im Hinblick auf die betrachteten DSL-Varianten, vgl. (7.2), aufgetragen. Eine Leistungseinbuße aufgrund der Parallelverarbeitung wird hierbei mit konstant 10 % angesetzt. Die Taktfrequenz des N-Cores wurde mit 300 MHz und die des *NPU-Cores* mit 600 MHz angenommen.

**Tabelle 7-4: Benötigte Verarbeitungseinheiten für eine *Linecard* des jeweiligen *DSLAM*-Szenarios**

DSL-Version	benötigte N-Cores		benötigte NPU-Cores	
	Uplink	Downlink	Uplink	Downlink
ADSL	1	4	1	5
VDSL	2	10	2	14
SHDSL	1	1	2	2

Mit 14 NPU-Cores bzw. 10 N-Cores ist der *Downlink* bei *VDSL* besonders rechenintensiv und kann nur sehr teuer mit parallelen Prozessoren bewerkstelligt werden. In diesem Szenario muss eine Datenmenge von 2,112 GBit/s pro *Linecard* verarbeitet werden.

Mit Hilfe des hier vorgestellten Benchmarks für das *IP-DSLAM*-Szenario können im Anschluss die in Kapitel 6 vorgestellten Optimierungsmaßnahmen wie Instruktionssatzerweiterungen, Hardwarebeschleuniger, Anpassung der Kommunikationsinfrastruktur sowie Ausnutzung von Parallelität greifen, deren Ergebnisse u. a. in [130][118][109][119][131][113] vorgestellt wurden.

Im folgenden Abschnitt werden u. a. die Performanzgewinne für die Ausführung des hier vorgestellten *IP-DSLAM*-Benchmarks unter Verwendung der in Abschnitt 6.2.4 präsentierten Instruktionssatzerweiterungen aufgezeigt.

### 7.3 Instruktionssatzerweiterungen zur optimierten Protokollverarbeitung

Im Folgenden wird anhand der in Abschnitt 6.2.4 präsentierten Instruktionssatzerweiterungen das Potential dieser Prozessorerweiterungen im Hinblick auf Leistungssteigerung, Reduktion des Energiebedarfs und Codegrößenminimierung aufgezeigt. Die Gegenüberstellung der Analyseergebnisse für den Originalprozessor (*S-Core* [108]) und der erweiterten Varianten (*N-Core* [111]) erfolgt sowohl für den im vorigen Abschnitt diskutierten *IP-DSLAM-Benchmark*<sup>55</sup>, als auch für die in [112] analysierte *IPSec*-Protokollsammlung [158], vgl. auch Abschnitt 6.2.3. So wird zum einen ein Anwendungsgebiet aus dem Zugangsbereich und zum anderen eine bereichsübergreifende

<sup>54</sup> Die Leistungsanforderungen aller Komponenten des *IP-DSLAMs* in Abhängigkeit der Parameter: Prozessorarchitektur, Taktfrequenz, Parallelisierungseinbuße, Hardwarekomponente des *DSLAMs* sowie Flussrichtung etc. zeigt der *DSLAM-System-Explorer* auf (vgl. Abschnitt 7.5).

<sup>55</sup> Im Weiteren wird eine konsekutive Bearbeitung aller Benchmarkszenarien mit reduzierter Paketzahl betrachtet. Dies ermöglicht eine Abschätzung der Auswirkungen der Instruktionssatzerweiterungen für alle *DSLAM*-Komponenten.

Sicherheitsanwendung bezüglich der Steigerung der Ressourceneffizienz durch Befehlssatzerweiterungen untersucht.

**Tabelle 7-5: Benötigte Taktzyklen des S-Cores für die IPSec- und IP-DSLAM-Anwendung und die Beschleunigung durch den Einsatz verschiedener Versionen des N-Cores**

Prozessorvariante	Benötigte Takte		Abnahme der Taktzahl		
	IP-DSLAM	IPSec	IP-DSLAM	IPSec	
<b>S-Core</b>	2.558.959	20.010.034	0	0	
<b>N-Core</b>					
Variante	xorldw	2.390.196	17.077.996	6,59%	14,65%
	andshr	2.407.230	17.126.254	5,93%	14,41%
	ixwandshr	2.407.246	16.100.200	5,93%	19,54%
	orshl81624	2.407.236	17.977.018	5,93%	10,16%
	ldwixw	2.309.388	15.612.431	9,75%	21,98%
	ldwxorls18,xorldw	2.356.116	18.000.664	7,93%	10,04%
	ldwaddi	2.403.618	18.001.336	6,07%	10,04%
<b>N-Core gesamt</b>	<b>2.288.730</b>	<b>15.061.131</b>	<b>10,56%</b>	<b>24,73%</b>	

Tabelle 7-5 zeigt die Ausführungszeit in Takten für die beiden Anwendungsszenarien und die entsprechenden Prozessorvarianten auf. Der N-Core wurde hierbei um jeweils eine der aufgezeigten Instruktionen erweitert, bzw. abschließend wurden alle Instruktionssatzerweiterungen in den N-Core integriert (*N-Core gesamt*). Da die Instruktionssatzerweiterungen ursprünglich für die bereichsübergreifende *IPSec*-Anwendung implementiert wurden, um den N-Core in möglichst vielen Orten des Netzwerks effizienter einsetzen zu können (vgl. Abbildung 7-1), verwundert es nicht, dass die maximal erzielte Performanzsteigerung für ebendiese Verarbeitung erzielt wird. So wird durch Integration der sieben zusätzlichen Befehle eine Reduktion der zur Verarbeitung benötigten Takte von fast 25 % erreicht. Beim *IP-DSLAM* sind es immerhin noch mehr als 10 % Ersparnis. Setzt man die hier gewonnenen Werte zu dem im vorigen Abschnitt 7.2.5 skizzierten *IP-DSLAM*-Szenario in Beziehung, so ließe sich durch die Instruktionssatzerweiterungen z. B. bei *VDSL* im *Downlink* auf der *Linecard* einer von zehn N-Cores einsparen. Diese Laufzeitreduktion geht bei gleichbleibender Taktfrequenz des Prozessorkerns einher mit einer Reduktion des Flächenbedarfs der reinen Prozessoreinheiten von 7,6 % bei der 130-nm-Technologie bzw. einer leichten Flächen-erhöhung um 1 % bei der 90-nm-Technologie, vgl. Abschnitt 6.2.4. Allerdings würde die Reduktion der Prozessoren eine Verringerung des zu instanzierenden Speichers bedeuten, welches somit für beide Technologien eine deutliche Flächenreduktion zur Folge hätte.

Beide Anwendungen profitieren maßgeblich von der *LDWIXW*-Instruktionssatzerweiterung. Hier erreicht man nur unter Verwendung dieses einen zusätzlichen Befehls eine Laufzeitreduktion des gesamten *IP-DSLAM-Benchmarks* von 9,75 % bzw. bei der *IPSec*-Anwendung von fast 22 %. Bei der *IPSec*-Anwendung liegt die Beschleunigung stets im mehrstelligen Prozentbereich, beim *IP-DSLAM* werden immerhin noch Werte von durchgängig über 6 % erreicht. Trotz der hohen Taktzahlreduktion durch die einzelnen Superinstruktionen liegt die erreichte Gesamtreduktion deutlich unter der Summation der Einzelreduktionen. Dies begründet sich darin, dass durch die Verwendung einer Superinstruktion, Einsatzmöglichkeiten anderer Instruktionssatzerweiterungen teilweise eingeschränkt werden. Der Compiler ist somit gefordert, eine möglichst optimale Konstellation der Zusatzbefehle zu finden [112].

Tabelle 7-6 zeigt den Energiebedarf für die Abarbeitung des *IPSec*-Benchmarks bezogen auf die 130-nm-Standardzellentechnologierealisierung. Außerdem wird die sich ergebende Codegröße in

Bytes für die *IPSec*-Anwendung angegeben, da diese ebenfalls einen Beitrag zur Ressourceneffizienz leistet, vgl. Abschnitt 6.2.3.

**Tabelle 7-6: Energiebedarf bezogen auf eine 130-nm-Technologie und Codegröße für die *IPSec*-Verarbeitung der verschiedenen Prozessorvarianten**

Prozessorvariante	Energie [mWs]	Energieabnahme	Codegröße [Bytes]	Codegrößenabnahme	
<b>S-Core</b>	1,266	0%	119.288	0%	
<b>N-Core</b>					
Variante	ldwixw	0,971	30,38%	109.750	8,69%
	ixwandshr	1,018	24,31%	109.626	8,81%
	xorldw	1,194	6,03%	109.470	8,97%
	andshr	1,236	2,43%	109.922	8,52%
	orshl81624	1,250	1,28%	110.010	8,43%
	ldwxorls18	1,277	0,86%	110.478	7,97%
	ldwaddi	1,272	0,47%	110.510	7,94%
	<b>N-Core gesamt</b>	<b>1,045</b>	<b>21,15%</b>	<b>108.158</b>	<b>10,29%</b>

Liegt die Abnahme der Taktzahl, die durch die Instruktionssatzerweiterungen erzielt wird, einheitlich im mehrstelligen Prozentbereich, so verhält sich die Bilanz bzgl. des Energiebedarfs der einzelnen Prozessorvarianten hingegen inhomogen. Die *LDWIXW*-Instruktionssatzerweiterung dominiert die Prozessorvarianten mit einer Energieabnahme von mehr als 30 % gegenüber der Verarbeitung mit dem Original-S-Core. Auch die *IXWANDSHR*-Erweiterung bewirkt eine Reduzierung des Energiebedarfs von fast 25 %. Alle weiteren Instruktionssatzerweiterungen hingegen bewirken nur noch Reduktionen, die größtenteils im unteren einstelligen Prozentbereich liegen. Dies liegt in der Komplexität und Art und Nutzung der prozessor-eigenen Register der jeweiligen Erweiterungen. Die Gesamtenergieabnahme für den N-Core (in der Realisierung von Abschnitt 6.2.4) beträgt immerhin noch über 21 % und liegt für die gesamte Ausführung bei 1,045 mW.

Für künftige Compilergenerationen wären zusätzliche Regelsätze mit Informationen bzgl. der Leistungsaufnahme denkbar, die durch so genannte *Mikrobenchmarks* (vgl. Abschnitt 6.2.3) für den gesamten Befehlssatz ermittelt werden könnten. Im Anschluss könnte eine ressourceneffizientere Codegenerierung im Hinblick auf die sich zur Laufzeit ergebende Leistungsaufnahme geschehen.

Der zweite Teil von Tabelle 7-6 fasst die erzielte Codegröße für den Benchmarkcode der einzelnen Prozessorvarianten zusammen. Bei Verwendung einzelner Superinstruktionen liegt die Abnahme bei durchschnittlich 8,5 %. Bei Einsatz aller Instruktionssatzerweiterungen reduziert sich die benötigte Speichermenge für das Anwendungsprogramm um 10,29 % auf 108.158 Byte.

Zusammenfassend lässt sich für den *IPSec*-Benchmark eine Beschleunigung der Verarbeitung von fast 25 % einhergehend mit einer Abnahme des Energiebedarfs für die Ausführung von 21 % gegenüber dem Original-S-Core konstatieren. Zusätzlich ist noch eine Codegrößenabnahme von mehr als 10 % zu verzeichnen. Diese deutliche Optimierung wird durch eine Erweiterung des Prozessorkerns um sieben leistungsfähige Superinstruktionen erreicht, die für die 130-nm-Implementierung lediglich einen Flächenzuwachs von 2,7 % bezogen auf die ursprüngliche Prozessorfläche ausmachen. Dieses Beispiel zeigt, dass sich die Ressourceneffizienz durch gezielte Instruktionssatzerweiterungen signifikant steigern lässt.

## 7.4 Modulare, effiziente Modellierung von Netzwerkanwendungen

Eine effiziente Erstellung der Anwendungssoftware ist im Sinne der Wirtschaftlichkeit und der Ressourceneffizienz von großer Bedeutung. So sind modular gehaltene Anwendungen mit definierten

Schnittstellen leichter zu erweitern und zu pflegen. In [182][183][118][178][119] wird ein derartiger Ansatz vorgestellt, der u. a. erfolgreich für die GigaNetIC-Architektur angewandt worden ist [118].

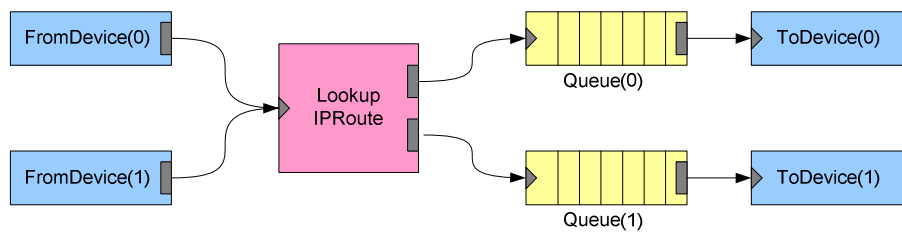


Abbildung 7-5: Beispielhafte Darstellung einer einfachen *Click*-Anwendung

Basis dieses Ansatzes ist *Click*, ein Werkzeug das es dem Benutzer erlaubt, einfach und sehr schnell Netzwerkanwendungen mit Hilfe von Modulen „zusammenzuklicken“ und damit zu beschreiben.

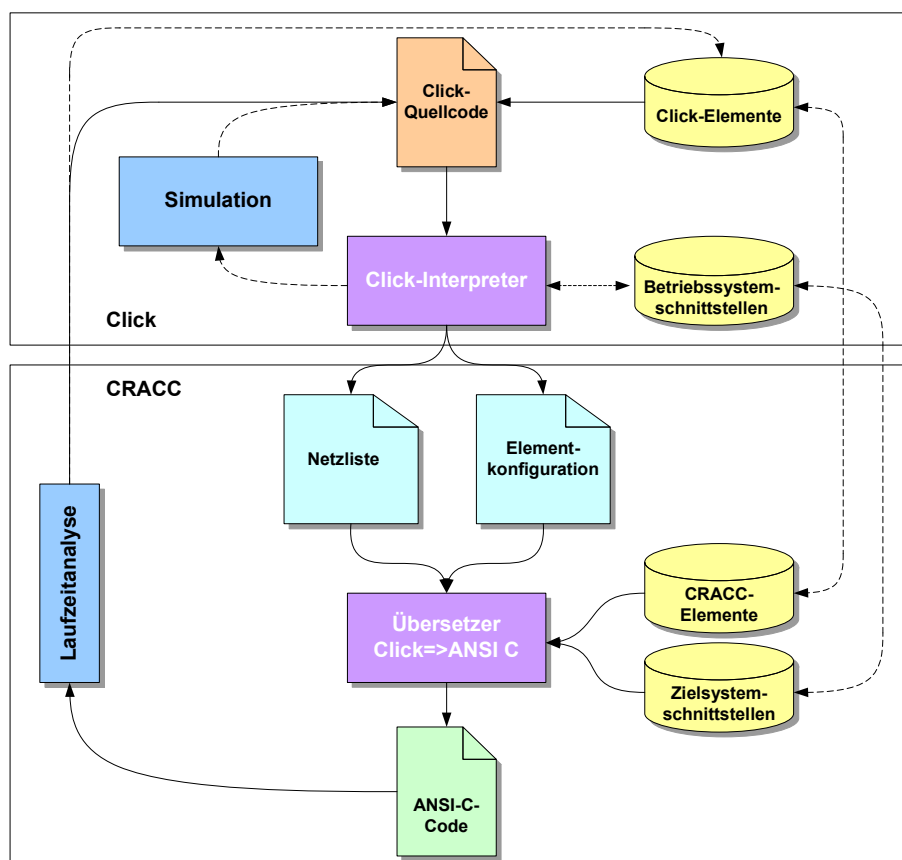


Abbildung 7-6: Zusammenspiel: *Click* – *CRACC*, Ablauf der Generierung von ANSI-C-Code für Netzwerkanwendungen aus einer abstrakten, modulbasierten Beschreibung

Das *Click*-Softwarepaket [184][185][186][187] wurde von der Parallel-and-Distributed-Operation-Systems-Gruppe des Massachusetts Institute of Technology [188], Mazu Networks [189], dem I.C.I.R. Center for Internet Research des International Computer Science Institute [190] und dem Computer Science Department der Universität von Kalifornien [191] entwickelt. Eine *Click*-Verarbeitungseinheit wird aus der Verknüpfung von Modulen, die Elemente genannt werden, gebildet. Diese Elemente bestimmen die Funktion einer Netzwerkverarbeitungseinheit bzw. eines Routers, beginnend bei der Kommunikation mit anderen Netzwerkkomponenten, der Paketmodifikation oder der Handhabung von Regeln zum Verwerfen bestimmter Pakete. Die Elemente sind in der Hochsprache C++ verfasst, was eine komfortable Erweiterung bzw. Anpassung ermöglicht. Parallelverarbeitung ist leicht modellierbar, und es kann auf eine sehr umfangreiche Bibliothek an Netz-

werkfunktionen zurückgegriffen werden. Abbildung 7-5 zeigt eine mit *Click* modulierte einfache Beispielanwendung aus dem Netzwerkanwendungsbereich.

Auf die Beschreibung der Netzwerkverarbeitungseinheit in *Click* setzt der zweite Schritt in unserer Werkzeugkette auf: *CRACC (Click Rapidly Adapted to C Code)* [182][183], das eine nahezu automatische Umsetzung der *Click*-basierten C++-Beschreibung in eine für eingebettete Systeme effizienter umsetzbare Kodierung in C übernimmt. Zusätzlich können in *CRACC* zur Steigerung der Leistungsfähigkeit optimierte *Assembler*-Routinen eingebunden werden. So verbinden sich die Vorteile aus schneller Anwendungserstellung aufgrund der großen Anzahl bereits bestehender Module für die Netzwerkverarbeitung mit der Möglichkeit, ein funktional verifiziertes Modell optimal auf die Zielarchitektur abzubilden. Der Entwurfsablauf mit Hilfe der *Click-CRACC*-Werkzeugkette wird in Abbildung 7-6 dargestellt.

Im Folgenden wird der in Abschnitt 7.2 vorgestellte *IP-DSLAM*-Benchmark mit Hilfe der hier vorgestellten Werkzeugkette nachgebildet und optimiert.

#### 7.4.1 Erweiterung des DSLAM-Benchmarks zum Referenzbenchmark

Aufbauend auf dem aus [141] bekannten *IP-DSLAM*-Benchmark wurde eine sehr realitätsnahe, genaue Modellierung der Abläufe innerhalb eines *IP-DSLAMs* mit Hilfe der *Click-CRACC*-Werkzeugkette realisiert [178]. Dies führte zu dem *IP-DSLAM-Referenzbenchmark*, der hier nur in aller Kürze vorgestellt werden soll. Dieser erweiterte Benchmark ermöglicht nun eine bessere Trennung der Analyse der Anforderungen an die einzelnen DSLAM-Komponenten.

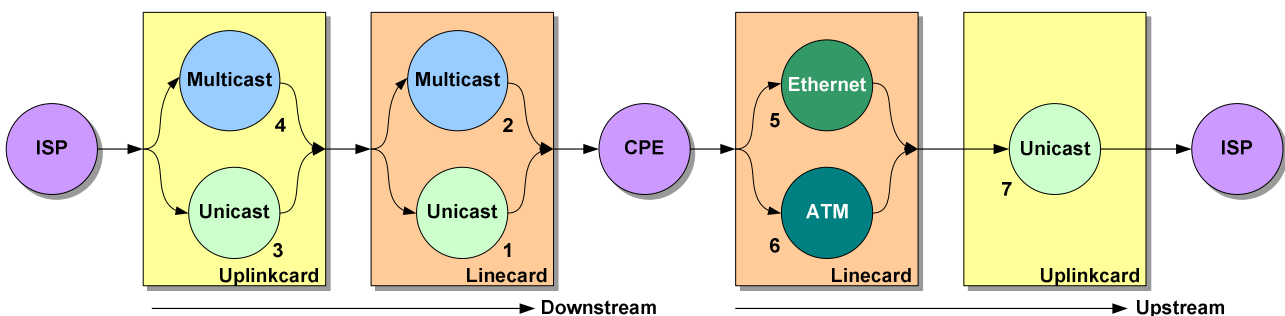


Abbildung 7-7: Die sieben Szenarien des *IP-DSLAM-Referenzbenchmarks*

Abbildung 7-7 zeigt die sich ergebenden sieben Szenarien des *IP-DSLAM-Referenzbenchmarks*<sup>56</sup>. Die Anzahl der *Tasks* wurde auf 11 erhöht, um der Realität der Anwendung noch näherzukommen. Es handelt sich hierbei um: *Ethernet encapsulation*, *AAL5*, *IP header check*, *IP source address / port verification*, *5-tuple classification / destination lookup*, *Traffic policing and QoS*, *Multicast duplication*, *Queuing*, *Set DiffServ codepoint* und *Decrement TTL/HLIM*. Die einzelnen *Tasks* können wiederum mehrere *Click*-Elemente umfassen. Für eine genaue Beschreibung der Funktionalität dieser *Tasks* sei auf [178] verwiesen.

Tabelle 7-7 zeigt die Bandbreiten der verschiedenen DSL-Varianten, die bei der folgenden Analyse des *IP-DSLAM-Referenzbenchmarks* berücksichtigt werden.

<sup>56</sup> Die detaillierte Zuordnung der *Tasks*, die sich teilweise von denen aus Abschnitt 7.2.1 unterscheiden, wird in Abbildung Anhang F-1 dargestellt.



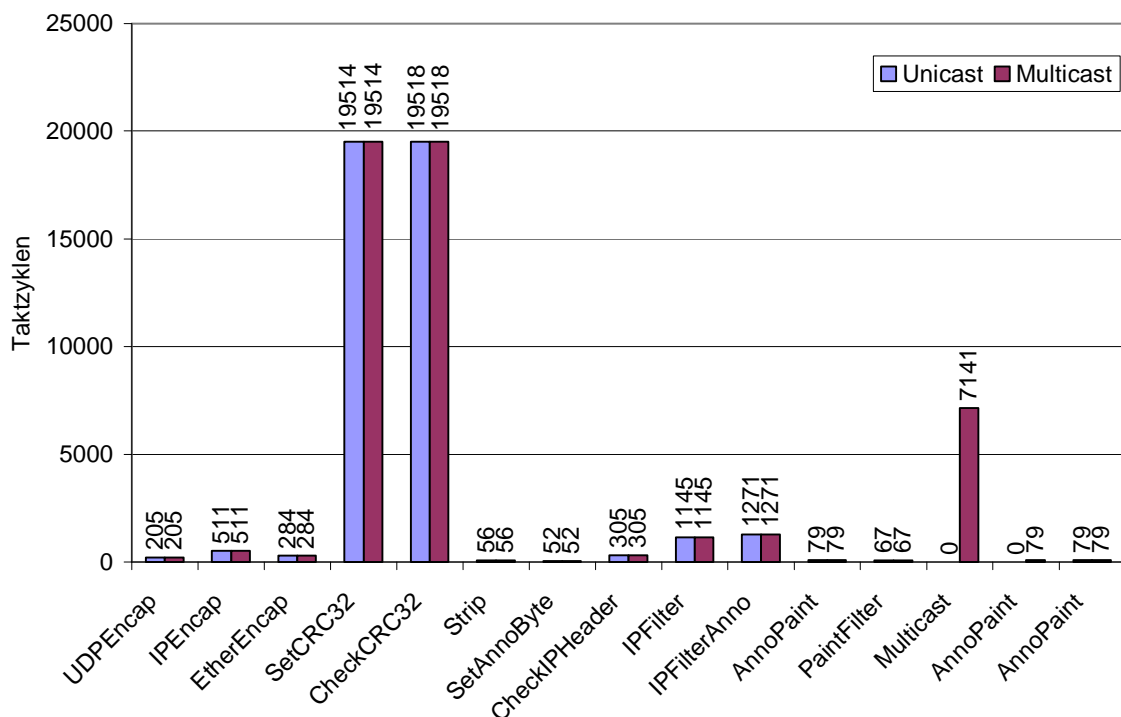
Tabelle 7-7: Bandbreiten der untersuchten DSL-Varianten beim *IP-DSLAM-Referenzbenchmark*

Bandbreite [MBit/s]	Datenrichtung	ADSL	HDSL	SDSL	VDSL	RADSL	ADSL2+
	Downlink	8	20,4	4	51,8	6	24
	Uplink	1	20,4	4	2,3	0,64	3,5

Im Folgenden beziehen sich alle weiteren Analysen, soweit nicht anders erwähnt, auf den *IP-DSLAM-Referenzbenchmark*.

### 7.4.2 IP-DSLAM-Referenzbenchmark – Ergebnisse

In diesem Abschnitt werden die durch zyklenakkurate Simulation gewonnenen Ergebnisse der GigaNetIC-Architektur für den *IP-DSLAM-Referenzbenchmark* präsentiert. Es wird ein ähnliches Verkehrsmodell verwendet wie beim ursprünglichen DSLAM-Benchmark, ebenfalls mit einer *iMix*-basierten Paketgrößenverteilung.

Abbildung 7-8: Benötigte Takte für die einzelnen Tasks auf der *Uplinkcard* im *Downlink*<sup>57</sup>

Exemplarisch werden die Messwerte für die *Uplinkcard* im *Downlink*-Betrieb (vgl. Abbildung 7-8) und die Taktzyklen der maßgeblichen Funktionen der *Linecard* im *Uplink*-Betrieb (vgl. Abbildung 7-9) aufgeführt. Es zeigt sich, dass einige wenige Funktionen, hierzu zählen vor allem die CRC-Generierungs- und Prüffunktionen (insgesamt ca. 90,6 % der Gesamttaktzyklen) sowie die Multicastduplizierung im Multicastszenario (insgesamt ca. 91,8 % der Gesamttaktzyklen) einen Großteil der benötigten Rechenleistung auf sich vereinen. Bei der *Linecard* im *Uplink* kommt neben den beiden CRC-Funktionen noch die Funktion *IPVerifyPort* mit 10,7 % zu den maßgebenden drei mit insgesamt 93,9 % Anteil am Gesamtrechenbedarf hinzu. Diese ist abgeleitet aus dem vorigen *Classification*-Task und führt einen tabellenbasierten Vergleich durch. Im *Uplink* im *ATM*-Szenario sind die *ATM*-spezifischen Funktionen *IP2ATM* und *ATM2IP*, die die Umsetzung von *ATM*-basierten Paketen (zu vergleichen mit *AAL5* aus [141]) maßgeblich mit 90,7 % an der Gesamtzyklenzahl be-

<sup>57</sup> Bezieht sich nach Mittelung der *iMix*-Parameter auf ein theoretisch 788-Byte-großes Paket.

teilt. Die Analyse dieser Zahlen zeigt potentielle Optimierungsmöglichkeiten auf, die durch Softwareoptimierung, Instruktionssatzerweiterungen und, speziell bei den hier erwähnten Funktionen, durch den Einsatz von Hardwarebeschleunigern herbeigeführt werden können.

Abschließend lässt sich zusammenfassen, dass im Durchschnitt 17,3 Takte im *Downlink* bzw. 29,3 Takte im *Uplink* auf der *Linecard* pro Bit eines Pakets zur Verarbeitung benötigt werden. Auf der *Uplinkcard* sind es 28,7 Takte im *Downlink* bzw. 24,3 Takte für den *Uplink*. Diese deutlich höheren Zahlen als beim ersten DSLAM-Benchmark sind u. a. in der aufwändigen CRC-Berechnung begründet.

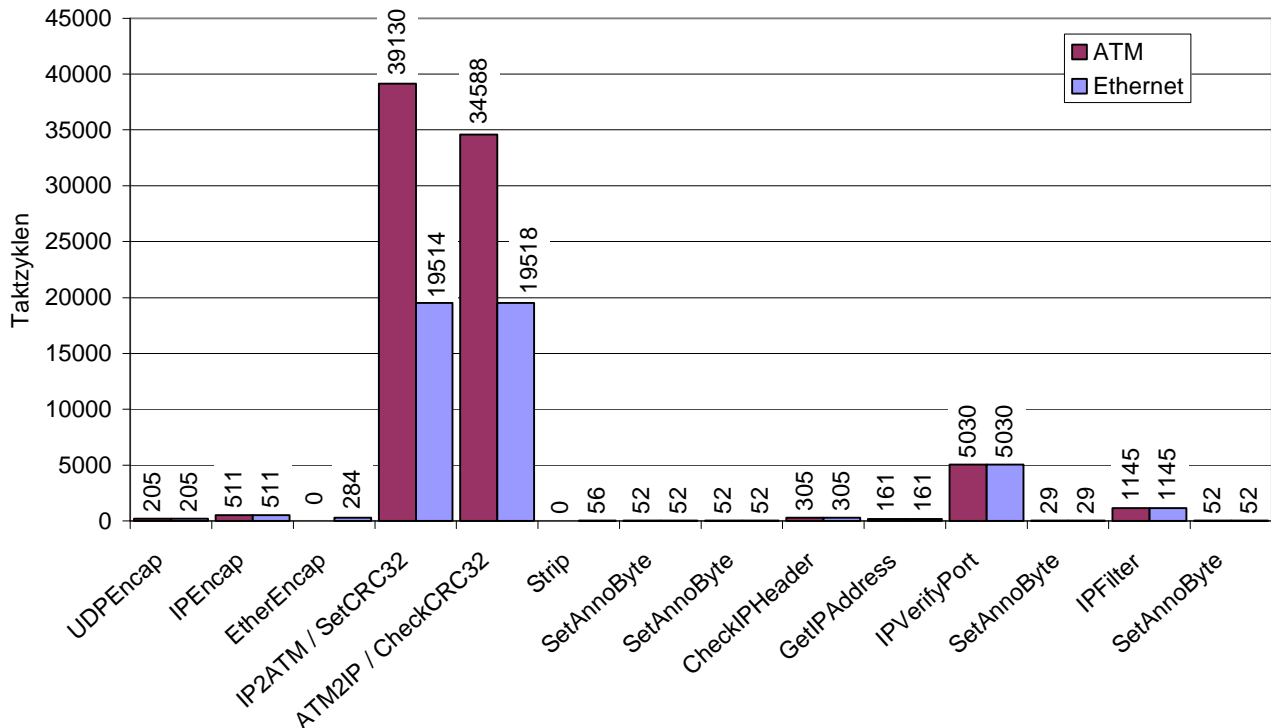


Abbildung 7-9: Benötigte Takte für die *Tasks* auf der *Linecard* im *Uplink* für das *Ethernet-* und *ATM-Szenario*<sup>57</sup>

Für viele der nicht maßgeblichen Funktionen der einzelnen Benchmarkszenarien können ggf. schon durch flächenmäßig geringfügige Eingriffe, wie z. B. durch Instruktionssatzerweiterungen, zusätzliche Performanzgewinne und somit Entlastungen der Verarbeitungseinheiten erzielt werden. Diese können dann im Weiteren zusätzliche Kontrollfunktionen auf den DSLAM-Systemkomponenten übernehmen.

Im Folgenden werden die einzelnen Maßnahmen der Optimierung und ihre Auswirkungen auf die Performanz näher diskutiert. Eine interaktive Entwurfsraumexploration ist mit dem *DSLAM-System-Explorer II* möglich, der die Ergebnisse in Abhängigkeit einstellbarer Systemparameter auswertet bzw. hochrechnet, vgl. Abschnitt 7.5.2.

#### 7.4.2.1 Softwareoptimierung des IP-DSLAM-Referenzbenchmarks

Die Anwendungserstellung durch die abstrakte Modellierung mit *Click* und die automatisierte Abbildung auf die Zielsysteme mittels *CRACC* und der jeweiligen Compiler-Werkzeugkette bietet viele Vorteile (vgl. Abschnitt 7.4), hat jedoch auch einen gewissen Preis bzgl. der erzielbaren Performanz.

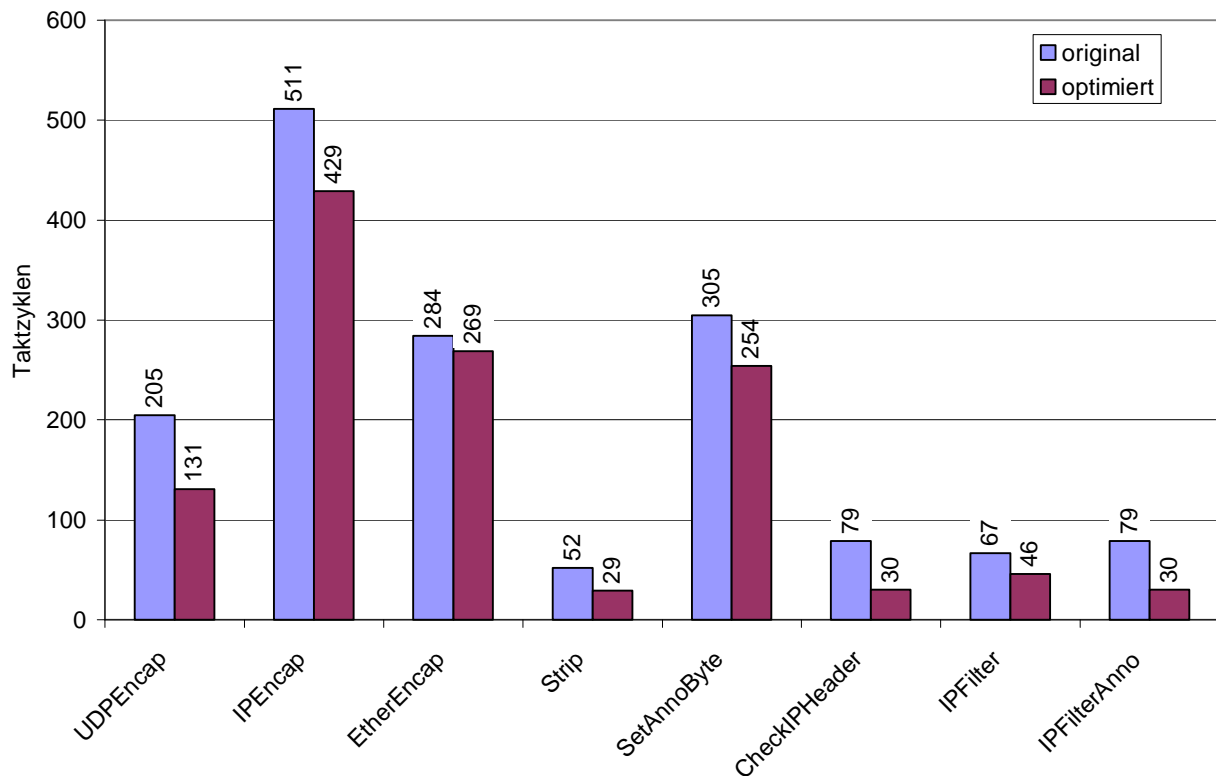


Abbildung 7-10: Vergleich der Originalversionen vs. optimierte Funktionen

Aufgrund der modularen Struktur werden häufig kostspielige Typumwandlungen und geschachtelte Funktionsaufrufe in den automatisch generierten Code eingebracht. Durch manuelle Optimierung des erzeugten C-Quellcodes lässt sich eine beträchtliche Menge an unnötigen Taktzyklen einsparen. Diese Optimierungen wurden für Funktionen, die sowohl auf der *Linecard* als auch auf der *Uplink-card* Verwendung finden, durchgeführt, vgl. Abbildung 7-10. Es lässt sich insgesamt eine deutliche Ersparnis von 21,4 % der benötigten Taktzyklen gegenüber der Originalversion verzeichnen. In Bezug auf die Gesamtleistungssteigerung machen diese Optimierungen jedoch aufgrund der dominierenden Prüfsummenoperationen letztendlich nur einen Geschwindigkeitsvorteil von ca. 1,3 % aus. Deshalb ist eine Suche nach Optimierungs- bzw. Erweiterungsmöglichkeiten der Hardware weiterhin notwendig, um den Einfluss der besonders rechenintensiven Funktionen an der Gesamttaktzahl zu reduzieren. Allerdings zeigt sich, dass für den Praxiseinsatz der automatisch generierte Code im Hinblick auf die oben genannten Schwachstellen untersucht und ggf. optimiert werden sollte. Zur Abschätzung und zum Vergleich der Leistungsfähigkeit unterschiedlicher Zielarchitekturen ist er dennoch ausreichend aussagefähig.

Im Weiteren werden die Untersuchungen bzgl. etwaiger Hardwaremodifikationen gemäß Kapitel 6 durchgeführt.

#### 7.4.2.2 Optimierung durch Instruktionssatzerweiterungen

In Abschnitt 6.2.4 wurden bereits einige Instruktionssatzerweiterungen für den N-Core vorgestellt. Mit Hilfe dieser existierenden und ggf. neuer Erweiterungen, die unter Zuhilfenahme der vorgestellten Werkzeugkette für das *IP-DSLAM*-Szenario implementiert werden, soll die mögliche Leistungssteigerung durch Modifikation des Prozessorkerns aufgezeigt werden. Bei der Analyse der Funktionen stellte sich der Task *IPVerifyPort*, eine der rechenintensiven Aufgaben, als besonders vielversprechend für die Optimierung durch Instruktionssatzerweiterungen heraus. Nach zusätzlich erfolg-

ter, manueller Optimierung der Funktion konnten die in Abbildung 7-11 gezeigten wesentlichen Instruktionpaarhäufigkeiten bestimmt werden.

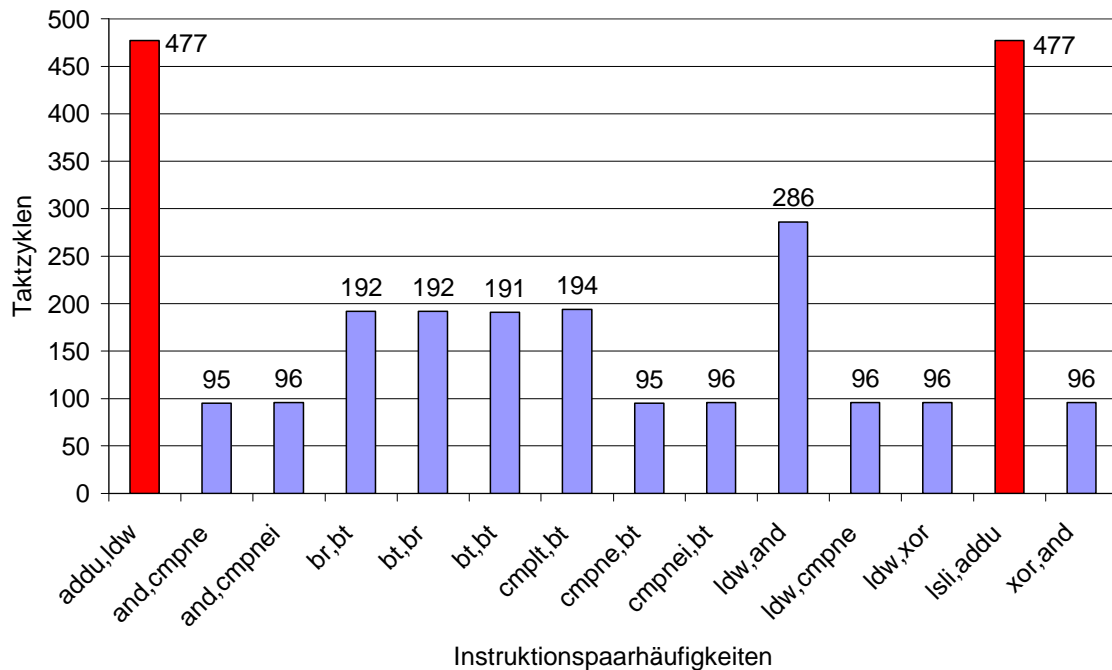


Abbildung 7-11: Wesentliche Instruktionpaarhäufigkeiten bei der Funktion *IPVerifyPort*

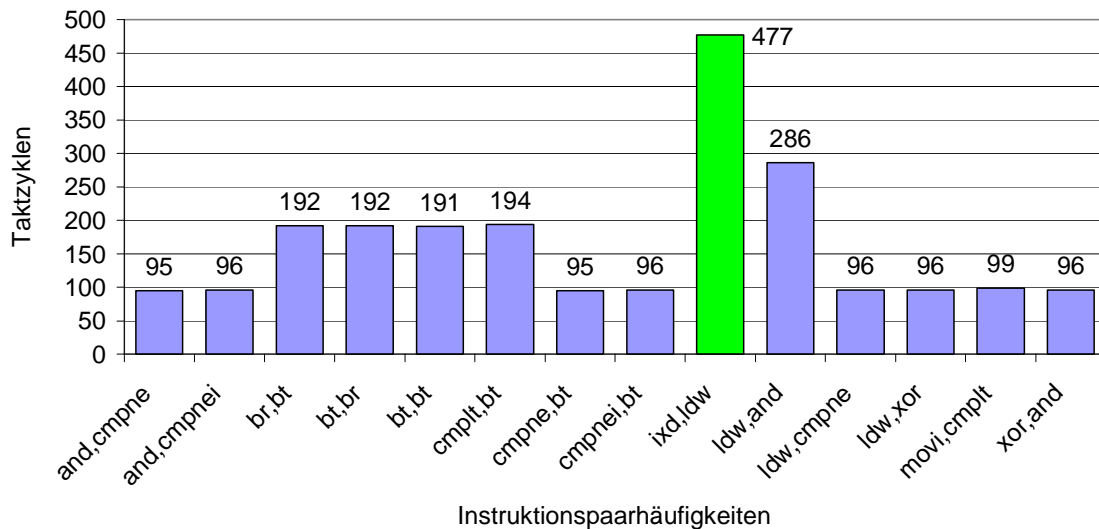


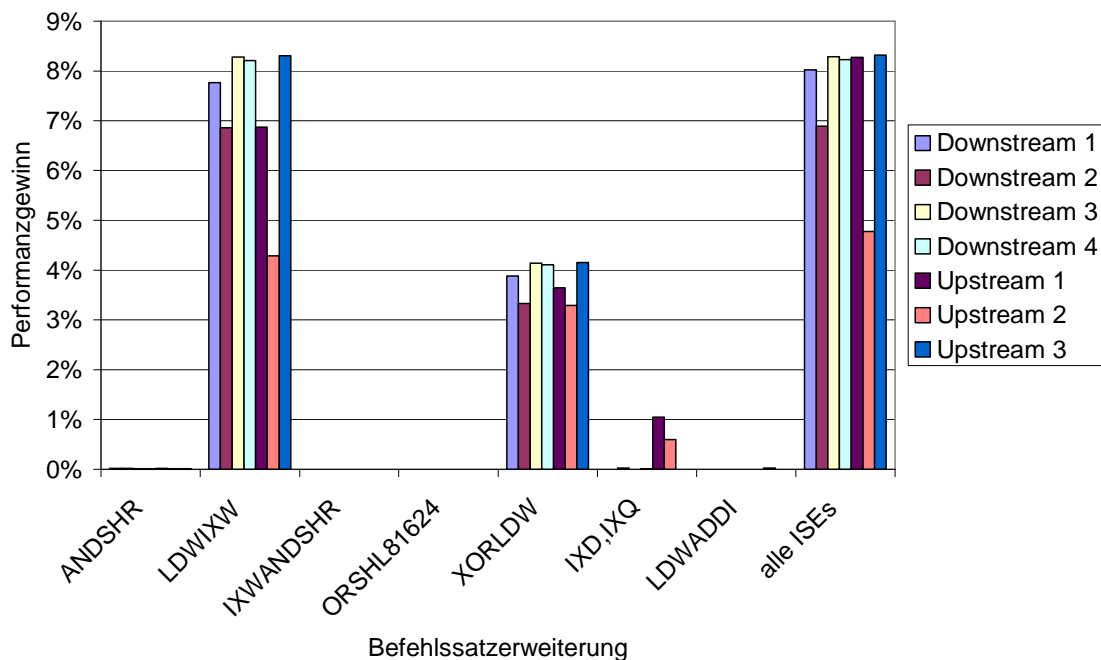
Abbildung 7-12: Wesentliche Instruktionpaarhäufigkeiten bei der Funktion *IPVerifyPort* nach der Instruktionssatzerweiterung durch *IXD*, *IXQ*

Das bei den Analysen sehr häufig aufgetretene Instruktionpaar *lsl, addu*, das eine vielfach benötigte Berechnung zum Vorrücken in der Routingtabellenstruktur ermöglicht, wurde durch die *IXD* (*Index Double*)-Superinstruktion ersetzt. Sie umfasst ein logisches Schieben von *RY* um drei Stellen nach links und das anschließende Addieren eines „Offsets“ auf das Ergebnis, das abschließend in Register *RX* geschrieben wird. Aus ähnlichen Gründen wurde die Instruktion *IXQ* (*Index Quad*) implementiert. Die Funktionsweise der beiden Befehle kann in der Registerzuweisungsnotation wie folgt beschrieben werden:

$$\text{IXD: } RX \leftarrow RX + (RY \ll 3) \quad (7.3)$$

$$\text{IXQ: } RX \leftarrow RX + (RY \ll 4) \quad (7.4)$$

Abbildung 7-12 zeigt die erneute Analyse der *IPVerifyPort*-Funktion nach erfolgter Integration der *IXD*-Superinstruktion. Es wird deutlich, dass durch die Einführung des *IXD*-Befehls eine Verschiebung der resultierenden Instruktionspaare eingetreten ist. Die zuvor aufgetretene Kombination von *addu* und *ldw* wurde eliminiert, dafür könnte eine weitere Superinstruktion *ixd, ldw* auf Basis der gerade neu integrierten *IXD*-Funktion eingebracht werden. Durch die *IXD*-Superinstruktion, die nur einen Takt beansprucht, konnte die Verarbeitung der betrachteten Funktion um 11,7 % beschleunigt werden, wobei der Prozessorkern lediglich um 0,69 % größer geworden ist.



**Abbildung 7-13: Beschleunigung des IP-DSLAM-Referenzbenchmarks durch verfügbare Instruktionssatzerweiterungen des N-Cores für eine *iMix*-Verteilung**

Abbildung 7-13 fasst die Resultate der Performanzgewinne durch die Instruktionssatzerweiterungen für die sieben Benchmarkszenarien und einem *iMix*-Verkehrsmodell zusammen. Es zeigt sich, dass, auf den gesamten Benchmark bezogen, der *IXD*-Befehl nur eine Performanzsteigerung von 1 % ermöglicht, wohingegen die *LDWIXW*-Erweiterung Beschleunigungen von 4 bis häufig 7 % und mehr erreicht. Auch die *XORLDW*-Erweiterung zeigt mit einer beschleunigten Verarbeitung von 3 bis 4 % für alle Benchmarks eine gute Leistung.

Bemerkenswert ist außerdem, dass durch Zuhilfenahme aller Instruktionssatzerweiterungen in fünf von sieben Fällen eine Beschleunigung von mehr als 8 % im Vergleich zum Original-S-Core erreicht wird, obwohl die CPU-Fläche des entsprechenden N-Cores lediglich 2,4 % größer ist. Die absolut gesehen geringe Beschleunigung durch die *IXD*-Instruktion steht im Gegensatz zu der Performanzsteigerung dieser Operation für die Funktion *IPVerifyPort*, die immerhin, wie bereits erwähnt bei 11,7 % liegt. Dies zeigt aber auch, dass Instruktionssatzerweiterungen nicht zwangsläufig alle Funktionen gleichermaßen tangieren und eine entsprechende Analyse im Vorfeld der Chiprealisierung durchaus sinnvoll ist. Im Falle des *IXD*-Befehls wurde die gewünschte Beschleunigung für die ausgewählte Aufgabe erreicht. Für die anderen rechenintensiven Tasks wird im Folgenden versucht, eine Beschleunigung durch spezielle Hardwarebeschleuniger zu erreichen.

### 7.4.2.3 Optimierung durch Hardwarebeschleuniger

Sowohl für die bereits als besonders rechenintensiv identifizierten Tasks *SetCRC32* und *CheckCRC32* im Falle der *Ethernet*-basierten und *IP2ATM* und *ATM2IP* für die *ATM*-basierten Szenarien als auch *IPVerifyPort* für die *Uplink*-Szenarien auf der *Linecard* (vgl. Abbildung 7-8 und Abbildung 7-9) wurden Hardwarebeschleuniger entwickelt. Sie versprechen eine deutlich höhere Beschleunigung, als es durch Instruktionssatzerweiterungen oder, in diesem Fall, durch die Softwareoptimierung möglich ist.

Tabelle 7-8 zeigt die für den *IP-DSLAM-Referenzbenchmark* implementierten Hardwarebeschleuniger und ihre wesentlichen Charakteristika. Die Beschleuniger werden den Tasks bzw. Funktionen des Benchmarks, in denen sie eingesetzt werden, zugeordnet. Der Beschleunigungsfaktor, der durch eine enge Kopplung (vgl. Abschnitt 4.3.3) an den N-Core erreicht werden kann, ist in der folgenden Spalte aufgetragen. Durch den Einsatz der Hardwarebeschleuniger ergibt sich für alle Tasks eine Codegrößenreduktion von durchschnittlich 49,6 %. So wird als positiver Nebeneffekt durch die Verwendung der Beschleunigermodule, noch stärker als durch Instruktionssatzerweiterungen (vgl. Abschnitt 6.2.3), die benötigte Instruktionsspeichermenge reduziert. Aufgrund der engen Kopplung kommt es nur zu einer durchschnittlichen Halbierung der Codegröße, da der N-Core weiterhin alle Speicheroperationen für den jeweiligen *IP(Intellectual Property)*-Block übernimmt. Bei der Ansteuerung von lose-gekoppelten Hardwarebeschleunigern hingegen reduziert sich der Befehlsaufwand auf wenige Instruktionswörter. Im Weiteren sind die Fläche und maximal erreichbare Taktfrequenz der Beschleuniger aufgetragen, gefolgt von der Verlustleistung bezogen auf 1 MHz, zum einen ermittelt durch Schaltwahrscheinlichkeiten (*50%S*) und zum anderen durch laufzeitbedingte Schaltaktivitätenannotation (*AS*) während der Abarbeitung der entsprechenden Funktion, vgl. Abschnitt 6.3.1.5.

**Tabelle 7-8: Charakteristika der implementierten Hardwarebeschleuniger für 130-nm- und 90-Standardzellentechnologien**

HW-Beschleuniger	Funktion	Beschleunigung (relativ zumN-Core, bei gleicher Frequenz), enge Kopplung	Codegrößen- reduktion [%]	Fläche [mm <sup>2</sup> ]		Taktfrequenz max. [MHz]		50%S-Leistung [mW/MHz]		AS-Leistung [mW/MHz]	
				130nm	90nm	130nm	90nm	130nm	90nm	130nm	90nm
<b>CRC</b>	CRC32	7,5	64,7%	0,0150	0,0115	990	1087	0,0061	0,0168	0,0040	0,0109
<b>CAM</b>	IPVerifyPort	45,0	57,1%	0,7186	0,6163	439	380	0,1995	0,1815	0,1808	0,1754
	Lookup	1230,0									
<b>IPFilter</b>	IPFilter	6,0	48,7%	0,3814	0,2470	430	523	0,0827	0,0759	0,0816	0,0746
<b>IPFilter</b>	IPFilterAnno	6,0	46,7%	0,3814	0,2470	430	523	0,0827	0,0759	0,0816	0,0746
<b>IPHeaderCheck</b>	CheckIPHeader	12,0	30,8%	0,0339	0,0329	546	571	0,0060	0,0058	0,0075	0,0098

Der *CRC*-Beschleuniger wurde speziell für dieses Szenario entwickelt und, wie die anderen hier aufgelisteten Hardwaremodule, für eine direkte, enge Kopplung an den N-Core konzipiert. In diesem Betriebsmodus erzielt dieser *CRC-IP*-Block im Vergleich zur Softwarelösung, die auf dem N-Core abgebildet ist, eine Beschleunigung von 7,5. Verfügte er über eine eigene Speicherschnittstelle, könnte sogar eine Performanzsteigerung um Faktor 49,5 erreicht werden. Der *CRC*-Block findet Einsatz in den besonders rechenintensiven Tasks *SetCRC32* und *CheckCRC32*. Das speziell entworfene *CAM*-Modul wird für den Task *IPVerifyPort* verwendet, der insgesamt eine Beschleunigung von 45 erfährt. Die enthaltene Funktion *Lookup*, die fast vollständig durch die Hardwareerweiterung ersetzt wird, kommt so auf eine beschleunigte Ausführung um den Faktor 1230. Der Block *IPFilter* findet in den beiden Tasks *IPFilter* und *IPFilterAnno* Verwendung und erzielt jeweils eine Be-

schleunigung von 6. Letztendlich wird der bereits in Abschnitt 6.3.1 beschriebene *IPHeaderCheck*-Beschleuniger für den *Task CheckIPHeader* als eng an den N-Core gekoppelte Hardwareeinheit eingesetzt. Aufgrund der alleinigen Prüfung der Prüfsumme des *IP*-Paketkopfes und der engen Kopplung ergibt sich in diesem Szenario eine Beschleunigung von 12. Diese Zahlen zeigen, dass mit einem zusätzlichen Flächenaufwand für das Gesamtprozessorsystem von 1,53 mm<sup>2</sup> (130 nm) bzw. 1,15 mm<sup>2</sup> (90 nm), was in etwa dem Zehnfachen der Fläche des N-Cores entspricht, eine akkumulierte Beschleunigung von 76,5 erreicht werden kann. Dies führt zu einer positiven *Flächen-Performanzeffizienz* (nach Definition 38) von über 7 (10/1 zu 77/1 = 7,7).

Betrachtet man die akkumulierte Verlustleistungsaufnahme der Hardwarebeschleuniger, so ergeben sich Werte von 0,36 mW/MHz (130 nm) bzw. von 0,35 mW/MHz (90 nm) was in etwa dem sechsfachen der dynamischen Verlustleistungsaufnahme des reinen N-Core-Prozessorkerns entspricht. Dies führt zu einer *Verlustleistungs-Performanzeffizienz* (nach Definition 38) von 6/1 zu 77/1 = 12,8. Diese Werte sprechen für eine derartige Integration der Hardwarebeschleuniger, ermöglichen sie doch ein deutlich ressourceneffizienteres System, als es allein mit N-Cores möglich wäre. In [118][119][109][131] werden weitere Analysen bzgl. des hier vorgestellten Szenarios gegeben.

Um möglichst schnell Aussagen über die Auswirkungen von verschiedensten Systemparametern unter Auswahl der in den letzten Abschnitten vorgestellten Optimierungsmaßnahmen treffen zu können, fast der *DSLAM-System-Explorer II* die gewonnenen Ergebnisse zusammen und bereitet sie graphisch auf. Dieses Werkzeug wird in Abschnitt 7.5.2 näher vorgestellt und erleichtert die Entwurfsraumexploration für DSLAM-Anwendungen in vielerlei Hinsicht.

## 7.5 Visualisierungswerkzeug zur Entwurfsraumexploration

Im Rahmen der umfangreichen Analysen der Netzwerkszenarien wurde eine Vielzahl von Daten gewonnen, die möglichst umfassend miteinander in Beziehung gesetzt werden sollten. Systemarchitekten zukünftiger GigaNetIC-basierter Netzwerkprozessoren profitieren bei einer Konzeption einer neuen, ressourceneffizienten Systemrealisierung von möglichst umfassenden Informationen durch bereits erstellte Analysen. So entstand der *DSLAM-System-Explorer*, er erlaubt eine schnelle und übersichtliche Visualisierung der gesammelten Ergebnisse der DSLAM-Benchmarks auf einfache Art und Weise, wie es ansonsten nur durch eine Unzahl von Diagrammen möglich wäre. Die wichtigsten Systemparameter des analysierten DSLAM-Szenarios lassen sich interaktiv, mit Hilfe von Auswahlmenüs bzw. mit Schiebereglern übersichtlich gemäß den Anforderungen des Anwendungsgebiets einstellen. Dies erlaubt eine sofortige Analyse des Entwurfsraums, bei der sich u. a. die Leistungsfähigkeit bzw. die Anzahl der benötigten Hardwareeinheiten ablesen lassen. Die Anwendung setzt auf eine einheitliche Datenbasis auf, die komfortabel mit einer Tabellenkalkulationssoftware gepflegt und erweitert werden kann. So lässt sich die Visualisierungssoftware leicht für andere Anwendungsszenarien, Prozessorkerne sowie weitere Systemgrößen (z. B. Fläche oder Leistungsaufnahme) erweitern.

Im nächsten Abschnitt 7.5.1 wird zunächst auf die Resultate der Analysen des ursprünglichen *IP-DSLAM-Benchmarks* eingegangen, die für den N-Core ohne Instruktionssatzerweiterungen (S-Core) und drei weitere eingebettete Prozessoren ermittelt wurden und mit Hilfe des *DSLAM-System-Explorer I* visualisiert werden können. In Abschnitt 7.5.2 wird dann der *DSLAM-System-Explorer II*



vorgestellt, der speziell die implementierten Hardwareerweiterungen für die GigaNetIC-Architektur (vgl. Abschnitt 7.4.2) einbezieht und deren Auswirkungen auf die Performanz bzgl. des *IP-DSLAM-Referenzbenchmarks* aufzeigt.

### 7.5.1 Vergleich eingebetteter Prozessorkerne – DSLAM-System-Explorer I

Der *DSLAM-System-Explorer I* erlaubt den Vergleich der Leistungsfähigkeit von derzeit vier eingebetteten CPUs in Bezug auf den IP-DSLAM-Benchmark aus [141].

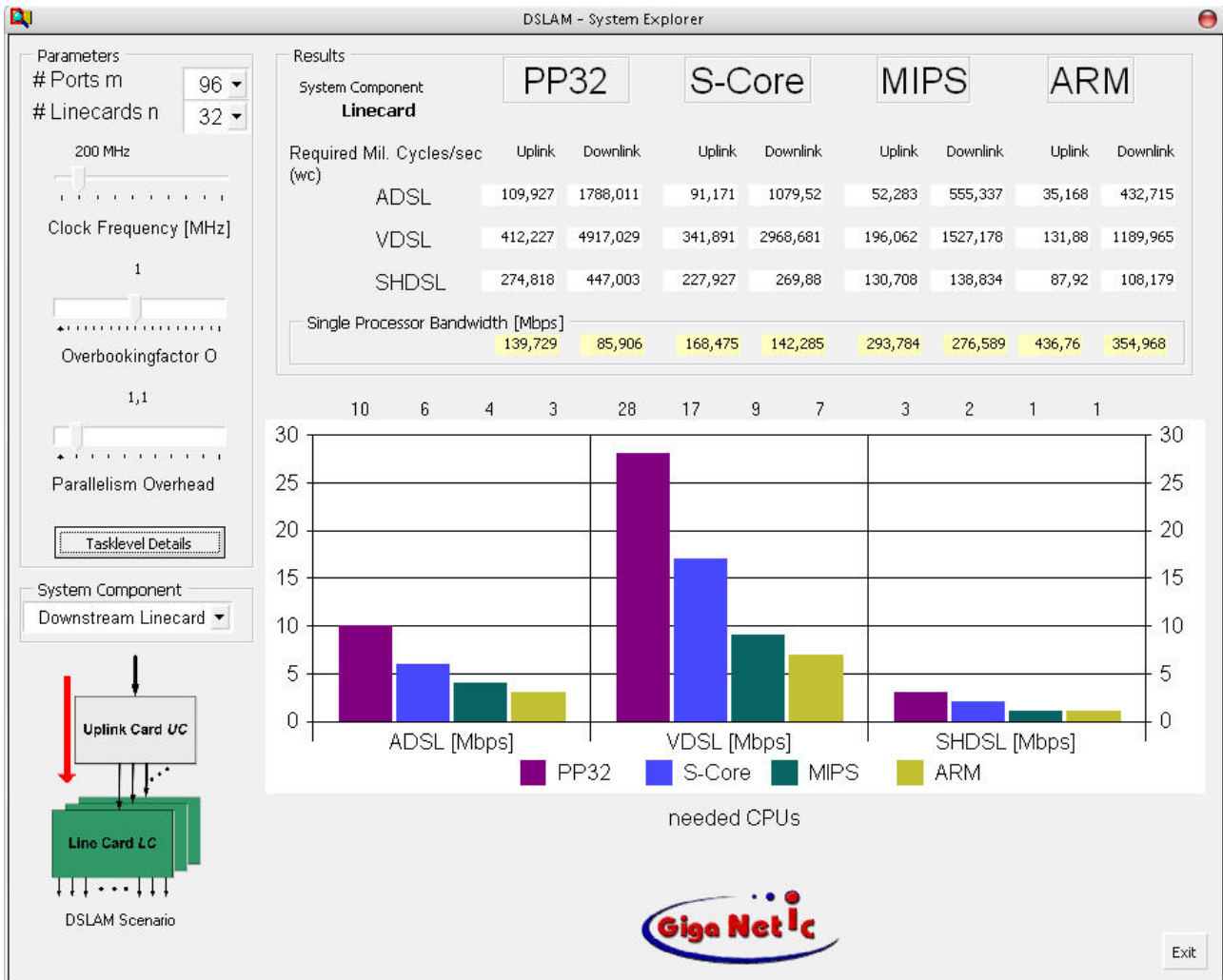


Abbildung 7-14: *DSLAM-System-Explorer I* – Entwurfsraumvisualisierungssoftware für das GigaNetIC-System und weitere eingebettete Prozessoren

Verglichen wird der in Abschnitt 7.2.5 vorgestellte spezialisierte *NPU-Core* (im Folgenden auch *PP32*), der zum Zeitpunkt dieser Analyse mit bis zu 450 MHz in einer 130-nm-Technologie betrieben werden konnte und ca. 0,38 mm<sup>2</sup> Fläche einnimmt. Weiterer Testkandidat ist der N-Core ohne spezielle Instruktionssatzerweiterungen, damit funktional dem S-Core [108] gleichzusetzen, allerdings in 130-nm- und 90-nm-Technologie realisiert ist (vgl. Abschnitt 4.3.1 und 6.2.4). Als nächstes wurde ein MIPS32k-basierter Prozessorkern analysiert, der in 180-nm-Technologie zwischen 200 und 240 MHz erreicht und je nach Konfiguration zwischen 0,8 und 2,5 mm<sup>2</sup> an Fläche beansprucht. In 130-nm-Technologie sind 260 bis 300 MHz bei einer Fläche von 0,4 bis 1,1 mm<sup>2</sup> möglich. Als letztes wurde ein ARM7-basierter Kern untersucht, der in 180-nm-Technologie 80 bis 110 MHz erreicht und eine Fläche zwischen 0,95 und 0,53 mm<sup>2</sup> benötigt. In 130-nm-Technologie steigert sich



die mögliche Taktfrequenz auf 100 bis 133 MHz, wobei sich die Fläche auf 0,42 bis 0,26 mm<sup>2</sup> reduziert.

Abbildung 7-14 zeigt das Hauptanwendungsfenster des *DSLAM-System-Explorers I*. Als Datenbasis dienen Messwerte, die mit zyklenakkuraten Simulatoren der untersuchten Prozessoren ermittelt wurden. Anhand der gemessenen Takte, die zur Verarbeitung der einzelnen Tasks benötigt werden, und aufgrund der Vorgaben durch die hier betrachteten *xDSL*-Szenarien *ADSL* (0,8 Mbps *Uplink*, 8 Mbps *Downlink* pro DSL-Port), *VDSL* (3 Mbps *Uplink*, 22 Mbps *Downlink* pro DSL-Port) und *SHDSL* (2 Mbps *Uplink*, 2 Mbps *Downlink* pro DSL-Port) ergeben sich die Verarbeitungsbandbreiten der Prozessoren.

Die veränderbaren Parameter können im linken Bereich des Hauptfensters eingestellt werden. Als Parameter können der Anwendung die Anzahl der DSL-Anschlüsse (*Ports*) einer *Linecard*, die Anzahl der *Linecards* des gesamten DSLAMs, die mit der *Uplinkcard* verbunden sind, die Taktfrequenz der Verarbeitungseinheit, der Überbuchungsfaktor (*Overbookingfactor*), der bestimmt, zu welchem Anteil die theoretisch notwendige (*WC*) Bandbreite vom *ISP* zur Verfügung gestellt wird und schließlich der zusätzliche Mehraufwand (*Overhead*), der durch den parallelen Betrieb von Verarbeitungseinheiten hervorgerufen wird, übergeben werden. Dieser Mehraufwand wird im Weiteren mit 10 % veranschlagt, was ggf. durch genauere Analysen verifiziert werden muss. Letztendlich kann dann noch eines der vier Szenarien des IP-DSLAM-Benchmarks ausgewählt werden (*Up-/Downlinkcard* / *Up-/Downlink*).

Task	PP32		S-Core		MIPS		ARM	
	Upstream	Downstream	Upstream	Downstream	Upstream	Downstream	Upstream	Downstream
A: Parser	443126	443126	74350	74350	32714	32714	35700	35700
B: Headercheck	1415624	1412650	465010	462101	277893	280621	262739	262739
C: Classify	-	5347252	-	2574465	-	1085057	-	1033050
D: MC_Duplication	-	3309750	-	1094943	-	513899	-	548157
E: Pol / Cond	-	1103830	-	282519	-	209532	-	174750
F: Convertpath	-	-	-	-	-	-	-	-
G: AAL5	3485528	3607462	2486081	2520989	733312	733312	567525	564615
H: CRC Accelerator	5121228	3414152	6166944	4243554	4295916	2933701	2722905	1891605
I: Ethernet Framing	993316	-	311263	-	110176	-	77045	-
J: CRC in Software	53906724	33403968	59550984	40111506	43055240	29013962	26603700	17945680
<b>Aggregate</b>	<b>11458822</b>	<b>18638222</b>	<b>9503648</b>	<b>11252921</b>	<b>5450011</b>	<b>5788836</b>	<b>3665914</b>	<b>4510616</b>
<b>Cycles/bit</b>	<b>1.431</b>	<b>2.328</b>	<b>1.187</b>	<b>1.406</b>	<b>0.681</b>	<b>0.723</b>	<b>0.458</b>	<b>0.563</b>
<b>Code size</b>	<b>4976</b>	<b>27967</b>	<b>4192</b>	<b>11334</b>	<b>4358</b>	<b>11735</b>	<b>4176</b>	<b>9748</b>

# of Packets: 2981  
# of Bytes: 1001324  
System Component: Linecard  
Exit

**Abbildung 7-15: Detaildarstellung der benötigten Taktzyklen für die einzelnen Tasks des IP-DSLAM-Benchmarks sowie für die Gesamtzahl benötigter Takte auf der Linecard**

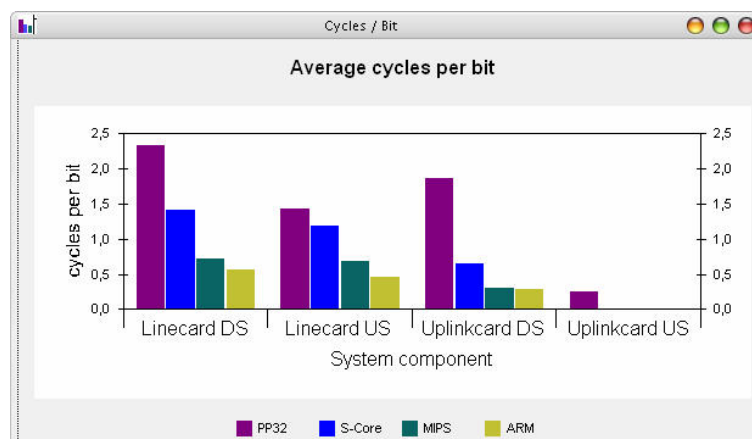
Bei dem dargestellten Szenario aus Abbildung 7-14 wird die *Linecard* im *Downlink* betrachtet. Für *VDSL* werden z. B. 28 NPU-Cores gegenüber 7 ARM-Prozessorkernen zur Bewältigung der Last bei 200 MHz im „schlimmsten Fall“<sup>58</sup> (*Worst Case*) benötigt. Durch Variieren des Schiebereglers kann nun z. B. interaktiv bestimmt werden, bei welcher Frequenz wie viele der jeweiligen Prozessoren

<sup>58</sup> Der schlimmste Fall beschreibt hier den Fall, dass die volle Bandbreite für *Up*- und *Downlink* pro Port der *Linecard* von den Teilnehmern gleichzeitig genutzt wird.

ren benötigt werden, um das entsprechende xDSL-Szenario spezifikationsgemäß bewältigen zu können. Es zeigt sich, dass der N-Core ohne spezielle Instruktionssatzerweiterungen eine deutlich höhere Leistung bietet, als der spezielle Paketverarbeitungsprozessor (vgl. Abschnitt 7.2.5). Auch im Vergleich mit den etablierten eingebetteten MIPS- und ARM-Prozessorkernen ist der N-Core, bezieht man seine geringe Fläche mit ein, durchaus wettbewerbsfähig, benötigt er doch deutlich weniger als die Hälfte der Fläche der kommerziellen Prozessoren. Die Anwendung zeigt ebenfalls die benötigte Anzahl an Taktzyklen für das jeweilige Szenario und die Performanz (gemessen in der zu bewältigenden Bandbreite in Mbps) eines einzelnen Prozessorkerns auf.

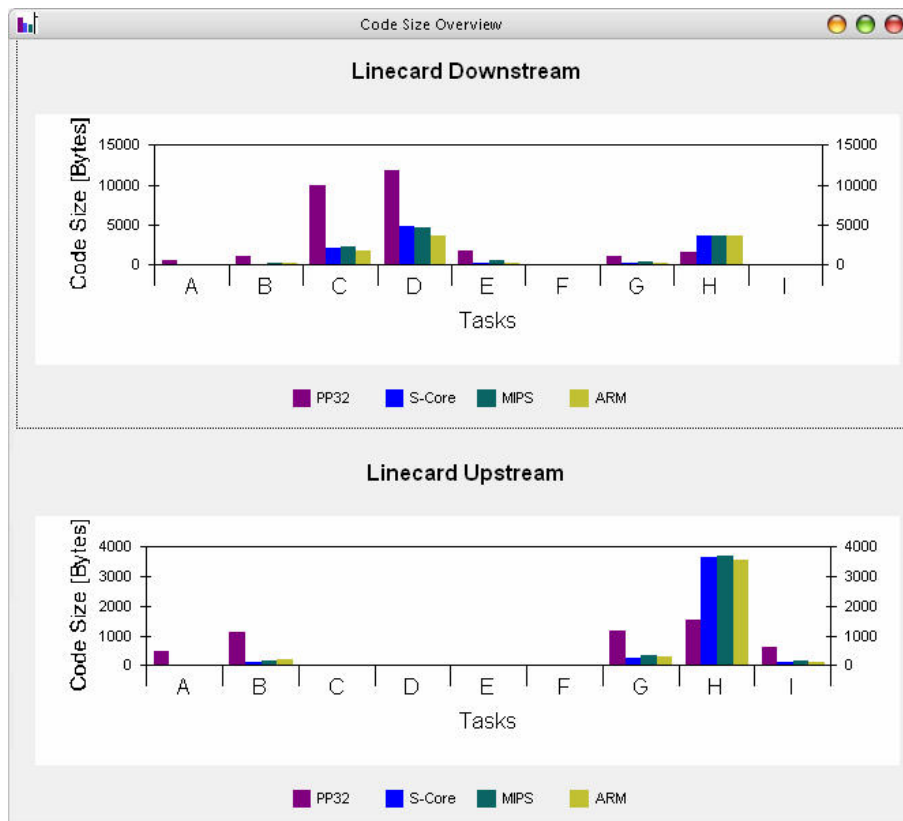
Abbildung 7-15 zeigt die Detailansicht, die genaue Einblicke in die Verteilung der Rechenlast auf die einzelnen Tasks der jeweiligen Szenarien gibt. Es wird ebenfalls deutlich, welche Tasks für das entsprechende Szenario benötigt werden und welche nicht. Auch hier lässt sich die Systemkomponente auswählen. In der gezeigten Darstellung wird Bezug auf die *Linecard* genommen. Außerdem werden Diagramme zu *Cycles/Bit* bzw. *Code size* erzeugt, die Aufschluss über die benötigten Prozessortakte pro Bit eines Pakets bzw. über die Codegröße der einzelnen Tasks geben.

Abbildung 7-16 zeigt die durchschnittliche Taktzahl pro Bit eines Pakets. Diese liegt für den *IP-DSLAM-Benchmark* für alle Szenarien und alle Prozessoren unter 3 Taktten. Deutlich wird auch hier, dass die spezielle Paketverarbeitungseinheit u. a. aufgrund der Schwächen des Compilers (vgl. Abschnitt 7.2.5) weitaus schlechter abschneidet als die anderen drei Prozessorkerne. MIPS und besonders ARM dominieren hier bzgl. der Effizienz benötigter Takte pro Bit eines Pakets. Allerdings ist hier die deutlich komplexere Architektur dieser Kerne, die sich in ihrer größeren Fläche ausdrückt, und die teilweise geringere Taktfrequenz des tatsächlich käuflich erwerbbaeren ARM-Prozessorkerns nicht zu vernachlässigen, so dass der N-Core auch bzgl. dieser Betrachtung durchaus wettbewerbsfähig erscheint. Aus Sicht der Performanz ist hier der MIPS-Kern der leistungsfähigste, ist doch seine maximal realisierbare Taktfrequenz mehr als doppelt so hoch wie die des ARM-Kerns.



**Abbildung 7-16: Darstellung der benötigten Takte pro Paketbit zur Bearbeitung des *IP-DSLAM-Benchmarks* bzgl. des jeweiligen Szenarios**

Abbildung 7-17 stellt die durch den jeweiligen Compiler erzeugten Codegrößen der einzelnen Tasks dar. Auch hier zeigt sich, dass der Paketprozessor (*NPU-Core*) nicht über einen entsprechend leistungsfähigen Compiler verfügt, sondern in der Regel mittels Maschinensprache von Hand programmiert wird, um entsprechend effizient zu sein. Die anderen drei Prozessoren sind in etwa vergleichbar, mit leichten Vorteilen auf Seiten des ARM-Kerns. Hier zeigt sich für den N-Core der Vorteil der 16-Bit-Instruktionsweite, der die Codegröße insgesamt relativ klein hält.



**Abbildung 7-17: Darstellung der Codegrößen der untersuchten Prozessoren für die einzelnen IP-DSLAM-Tasks auf der Linecard**

Abschließend lässt sich bemerken, dass die hier angestellten Analysen zeigen, dass der N-Core verglichen mit etablierten Prozessorkernen bzw. Spezialhardware (NPU-Core) durchaus wettbewerbsfähig im Hinblick auf Leistung und insbesondere auf Flächenbedarf ist. Im folgenden Abschnitt wird, um die Auswirkungen der zusätzlich implementierten Hardwareerweiterungen für die GigaNetIC-Architektur besser auswerten zu können, mit dem *DSLAM-System-Explorer II* ein leistungsfähiges Werkzeug vorgestellt.

### 7.5.2 Einbeziehung von HW-Erweiterungen – DSLAM-System-Explorer II

Im Folgenden wird mit Hilfe des *DSLAM-System-Explorer II* gezeigt, inwiefern, sich die Leistungsfähigkeit und die Ressourceneffizienz des N-Core durch Hardwareerweiterungen in Form von Instruktionssatzerweiterungen und speziell für das Anwendungsszenario entwickelten Hardwarebeschleunigern (vgl. Abschnitt 7.4.2) zusätzlich steigern lässt. Hierbei dient von nun an der *IP-DSLAM-Referenzbenchmark* aus Abschnitt 7.4.1 als Grundlage der Analysen.

Abbildung 7-18 zeigt das Hauptanwendungsfenster des *DSLAM-System-Explorer II*. Er berücksichtigt für sechs bereits vorgestellte xDSL-Varianten die sieben Benchmarkszenarien aus Abschnitt 7.4.1, die im rechten Bereich des Fensters über Auswahlménüs eingestellt werden können. Taktfrequenz des CMPs wie auch die Anzahl der *Ports* und *Linecards* lassen sich ebenfalls konfigurieren. Zusätzlich kann die Art der Hardwareerweiterung bzw. die Softwareoptimierung ausgewählt werden. Die Differenz der benötigten Taktzyklen zum Original wird simultan in der Diagrammansicht des Anwendungsfensters graphisch hervorgehoben. In Abbildung 7-18 werden z. B. die Unterschiede der Originalsoftwareimplementierung gegenüber der optimierten Softwareimplementierung aufgetragen. Der jeweilige durchschnittliche Beschleunigungsfaktor wird ebenfalls zeitgleich be-

rechnet und dargestellt. In diesem Szenario (*Linecard – Uplink – Ethernet*) wird z. B. eine Beschleunigung (*Speedup*) von 10 % erreicht. Für jede der DSL-Varianten wird die benötigte Anzahl an Prozessoren angezeigt. Sollten Hardwarebeschleuniger ausgewählt sein, wird jeweils nur eine dieser Spezialeinheiten berücksichtigt. In zukünftigen Implementierungen wird die Anzahl der Beschleunigereinheiten ebenfalls parametrisierbar sein. So werden z. B. für das in Abbildung 7-18 ausgewählte Szenario der *Linecard* für das *SDSL*-Szenario mindestens 52 N-Core-Prozessoren mit einer Betriebsfrequenz von 200 MHz benötigt. Ohne Softwareoptimierung würden 57 N-Cores und unter Verwendung aller Instruktionssatzerweiterungen nur noch 48 benötigt. Würden alle in Abschnitt 7.4.2 vorgestellten Hardwarebeschleuniger einfach instantiiert und eng-gekoppelt eingesetzt, so reduzierte sich die Anzahl benötigter Prozessorkerne auf lediglich 9. Dies entspricht einer Reduktion von immerhin 84,2 % im Vergleich zur Originalversion. Bei einer Betriebsfrequenz von 1,57 GHz reichte eines dieser Prozessorsysteme, mit allen Hardwarebeschleunigern ausgestattet, aus, um die gesamte Bandbreite einer Linecard im *Uplink* und *Ethernet*-Modus für *SDSL* verarbeiten zu können.

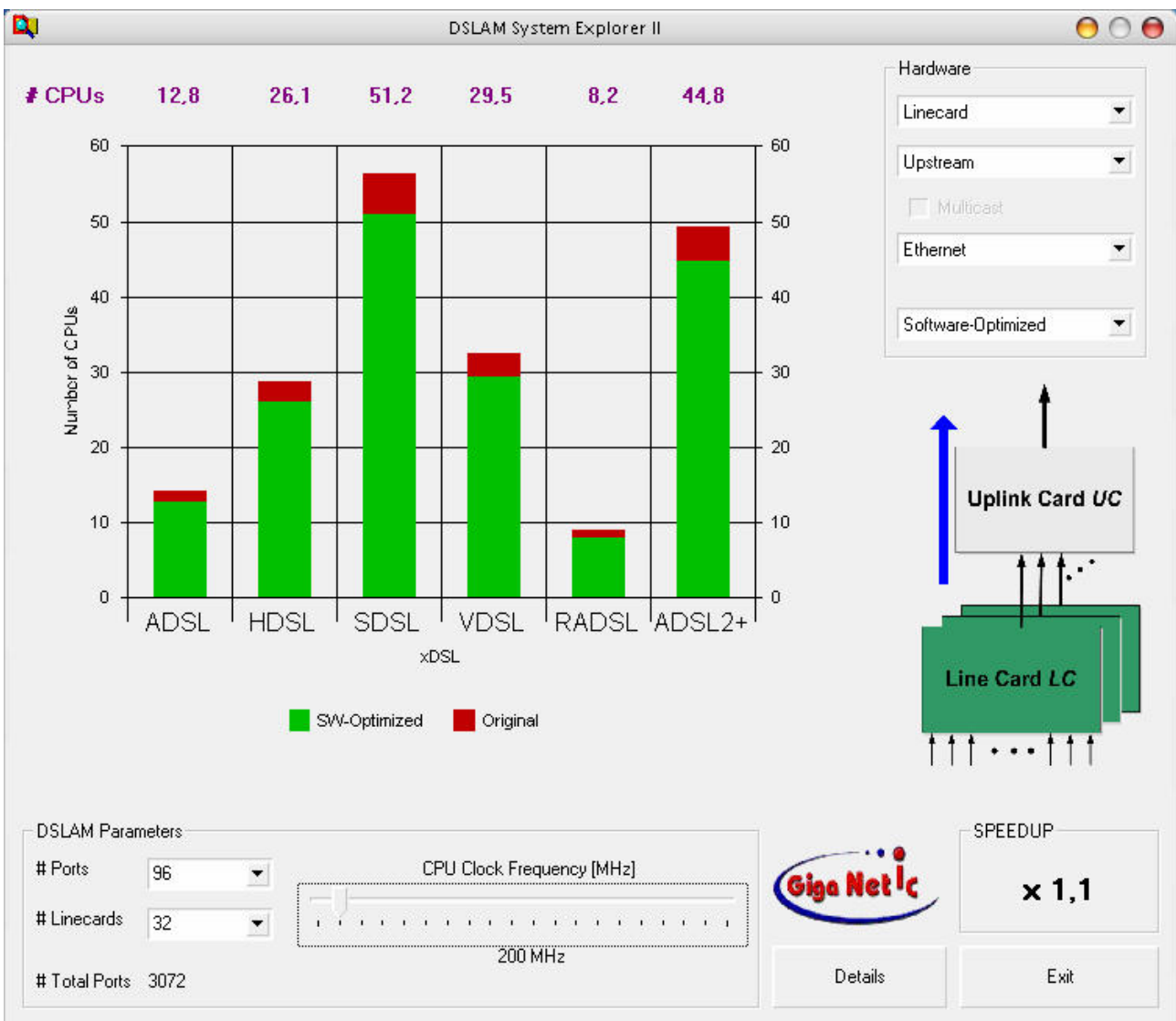
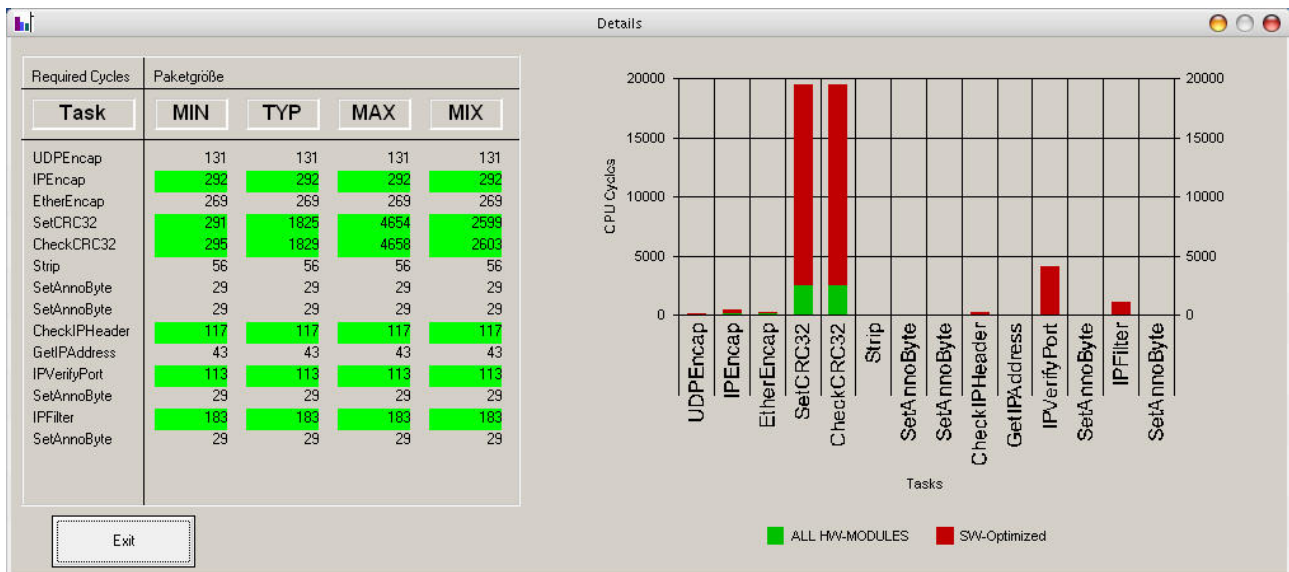


Abbildung 7-18: *DSLAM-System-Explorer II* – Berücksichtigung der Instruktionssatzerweiterungen und Hardwarebeschleuniger für unterschiedliche *IP-DSLAM-Anwendungsszenarien*

In [118] werden detaillierte Analysen zu GigaNetIC-basierten Netzwerkprozessoren vorgestellt, bei denen möglichst ressourceneffiziente Implementierungen unter Verwendung von optimierten Hardwarebeschleunigern angestrebt werden.

Abbildung 7-19 gibt die Detailansicht der Tasks wieder. In der linken Hälfte werden die Funktionen farblich hinterlegt, bei denen die zuvor ausgewählte Optimierung eine Reduktion der benötigten Taktzyklen zur Folge hat. Die benötigten Takte für die drei *iMix*-Paketgrößen sowie das Mittel dieser statistischen Verteilung (788 Byte) werden aufgeführt. In der rechten Hälfte des Fensters, der Diagrammansicht, wird diese ebenfalls durch zwei überlagerte Säulen dargestellt.



**Abbildung 7-19: Darstellung der benötigten Taktzyklen für die einzelnen Tasks des IP-DSLAM-Referenzbenchmarks mit Hervorhebung des Einflusses der ausgewählten Optimierungen**

Bei diesem Beispiel handelt es sich um den Vergleich der optimierten Softwarevariante gegenüber dem Einsatz aller Hardwarebeschleuniger. Es zeigt sich zum einen, dass nicht alle Funktionen von den jeweiligen Optimierungen profitieren, zum anderen wird deutlich, dass viele der Funktionen invariant gegenüber der Paketgröße sind. Dies führt dazu, dass kleine Pakete, wie sie in vielen aktuellen Anwendungen genutzt werden, besonders ineffizient bzgl. der *Nutzdaten-Aufwand-Effizienz* sind. Anhand der Balkendiagramme zeigen sich besonders drastische Reduzierungen der sehr rechenintensiven Funktionen, so wie es in Abschnitt 7.4.2 bereits angedacht war. Wird eine ausgewogene Mischung zwischen Hardwarebeschleunigern und instruktionssatzerweiterten N-Core-Prozessorkernen implementiert, so kann eine deutliche Steigerung der Ressourceneffizienz gegenüber einem nichtoptimierten System, das alleinig mit Universalprozessoren ausgerüstet ist, erreicht werden. Frei werdende Rechenkapazitäten der Prozessorkerne können dann für übergeordnete Kontrollaufgaben und höhere Dienstqualität bzw. zusätzliche, differenzierende Eigenschaften des Gesamtsystems genutzt werden.

Eine Erweiterung der in den letzten beiden Abschnitten vorgestellten *DSLAM-System-Explorer*-Visualisierungswerkzeuge durch die Kostenfunktions-basierte Analyseverfahren (vgl. Kapitel 3) wäre technisch relativ leicht realisierbar und ermöglichte dem Systemarchitekten neben der Visualisierung und Aufbereitung der Synthese- und Messergebnisse zudem eine fundierte Bewertung. Bei einer derart großen Menge an Szenarien und Realisierungsvarianten wäre speziell im Falle des *DSLAM-System-Explorers* eine automatisierte Auswertung im Hinblick auf pareto-optimale und



damit ressourceneffiziente Lösungen äußerst hilfreich. Derzeit muss dies noch vom Anwender selbst anhand von Auswertungen der generierten Diagramme geleistet werden.

## 7.6 Einsatz GigaNetIC-basierter Netzwerkprozessoren als Router

Als weiteres Beispiel der Vielseitigkeit der GigaNetIC-Architektur wird im Folgenden ein Anwendungsbeispiel aus dem Kernnetzwerk bzw. *Edge*-Netzwerk zur Bewertung von GigaNetIC-Systemen, basierend auf den Leistungsdaten der N-Core-Verarbeitungseinheit und spezieller Hardwarebeschleuniger, vorgestellt. Hierzu wurden in Anlehnung an die vom EEMBC-Benchmark-Konsortium [161] definierten Netzwerkbenchmarks drei charakteristische Funktionen von Routern (*OSPF-Routing*, *Packet-Flow* und *Route-Lookup*) modelliert und auf die Zielarchitektur abgebildet [130]. Diese Funktionen stellen wesentliche Aufgaben eines aktuellen Routers dar, die durch zusätzliche Funktionalitäten wie z. B. *Network Address Translation (NAT)* oder *Quality of Service (QoS)* ergänzt werden können.

Der Router in dem hier betrachteten Szenario (vgl. Abbildung 7-20) verfügt über acht *Gigabit-Ethernet (GE)*-Schnittstellen, die im Vollduplex betrieben werden. Über den internen Aufbau des Routers werden keine Angaben gemacht, da hier nur der Rechenaufwand für die einzelnen Algorithmen untersucht werden soll, so dass von einer idealisierten Architektur ausgegangen wird.

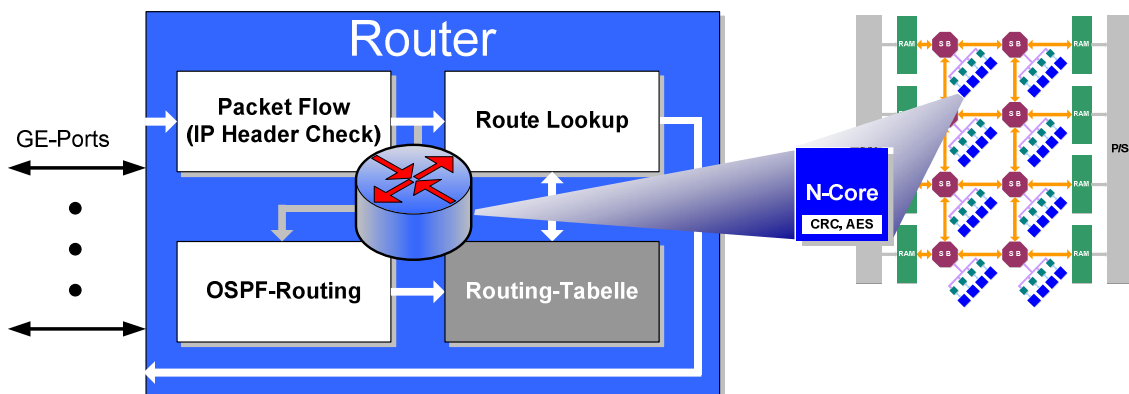


Abbildung 7-20: Leistungsanalyse des N-Cores und der GigaNetIC-Architektur für einen Netzwerkrouter

Die eintreffenden *IP*-Pakete (eingefasst in zufallsverteilte Ethernetpakete minimaler/maximaler Länge) werden durch den Block *Packet Flow* zunächst auf Korrektheit geprüft (*IP-Headercheck*) und das *TTL (Time to Live)*-Feld dekrementiert, da es sich um einen so genannten *Hop* im Netzwerk handelt. Sollte es sich um ein ungültiges Paket handeln, so wird es verworfen. Enthält das Paket Routinginformationen über den Netzwerkstatus, wird dieses dem *OSPF (Open Shortest Path First)*-Block übergeben, der entsprechend dem hier eingesetzten *Dijkstra*-Verfahren ggf. die Routingtabelle aktualisiert. Handelt es sich bei dem eingehenden *IP*-Paket um ein Datenpaket, so übernimmt der *Route-Lookup*-Block anhand eines *Patricia-Trie*-Algorithmus die Zuordnung des Ausgangsports und das Paket wird zum nächsten Netzteilnehmer weitergeleitet. Für eine minimale Paketgröße von 64 Byte und unter Einhaltung der Übertragungsspezifikation ergeben sich 1.488.095 Pakete pro Port. Dies bedeutet eine Gesamtpaketzahl von 11,9 Mio. und stellt zugleich den Fall maximalen Rechenaufwands dar. Für das Aktualisieren der Routingtabelle wird ein eigenständiger N-Core-Prozessorkern eingesetzt, der nur für diese Funktion verwendet wird und bei einer Taktfrequenz von

230 MHz 317 Updates der Routingtabelle pro Sekunde erreicht<sup>59</sup>. Heutige Router aktualisieren ihre Routingtabelle bis zu 300 mal pro Sekunde [192], in diesem Fall reichte hier ein N-Core zur Verarbeitung aus.

Tabelle 7-9 zeigt die mit Hilfe der Benchmarks ermittelten Leistungswerte des N-Cores für den Einsatz in einem *Edge-Router*-Szenario. Es zeigt sich, dass für die hier geforderten acht *GE*-Ports im Volllastbetrieb mindestens 26 dieser Verarbeitungseinheiten zur alleinigen Abwicklung des Protokollstapels benötigt werden. Würde man dieses Standardsystem durch spezielle Hardwarebeschleuniger für die beiden rechenintensiven Aufgaben *Packet Flow* und *Route Lookup* erweitern (optimiertes System), so ließe sich die Anzahl benötigter N-Cores auf drei reduzieren. Zusätzlich wären dann zwei der bereits implementierten Hardwarebeschleuniger (*IP-Headercheck*, eng-gekoppelt und *CAM*, eng-gekoppelt) notwendig (vgl. Abschnitt 7.4.2.3). Diese könnten zusammen mit den drei N-Cores die gleiche Rechenleistung zur Verfügung stellen wie das Standardsystem mit 26 Prozessoren. Tabelle 7-9 zeigt weiterhin die sich durch das optimierte System ergebende Flächensparnis auf. Diese beträgt insgesamt mehr als 70 %. Zusätzlich wird durch die Verwendung dieser optimierten Architektur eine mit 73,6 % nicht unerhebliche Verlustleistungersparnis erzielt. Die Verlustleistung für diese Funktionen wurde mit Hilfe der Annotierung der Schaltaktivitäten (AS), während der Laufzeit bestimmt.

**Tabelle 7-9: Rechenleistung eines N-Core-basierten Standardsystems und eines optimierten GigaNetIC-Systems für ein *Edge-Router*-Szenario**

Funktion	Standardsystem		Optimiertes System			
	Performanz pro N-Core	Benötigte N-Cores	Benötigte N-Cores	Hardwarebeschleuniger	Flächensparnis [%]	Verlustleistungsersparnis [%] (SA)
Packet Flow [Pakete/s]	900.262	14	1	1	91,3%	91,8%
Route Lookup [Lookups/s]	1.173.000	11	1	1	49,6%	57,2%
OSPF-Routing [Updates/s]	317	1	1	-	0%	0%
<b>Gesamt</b>	-	<b>26</b>	<b>3</b>	<b>2</b>	<b>70,1%</b>	<b>73,6%</b>

Auch in diesem Beispiel für Netzwerkanwendungen zeigt sich, dass die Ressourceneffizienz eines Systems aus einem Kompromiss zwischen hoher Leistungsfähigkeit spezialisierter Hardware auf der einen Seite und Flexibilität bzw. Zukunftssicherheit von Universalprozessoren auf der anderen Seite besteht.

## 7.7 Analyse der Anschlussarten von Hardwarebeschleunigern im GigaNoC

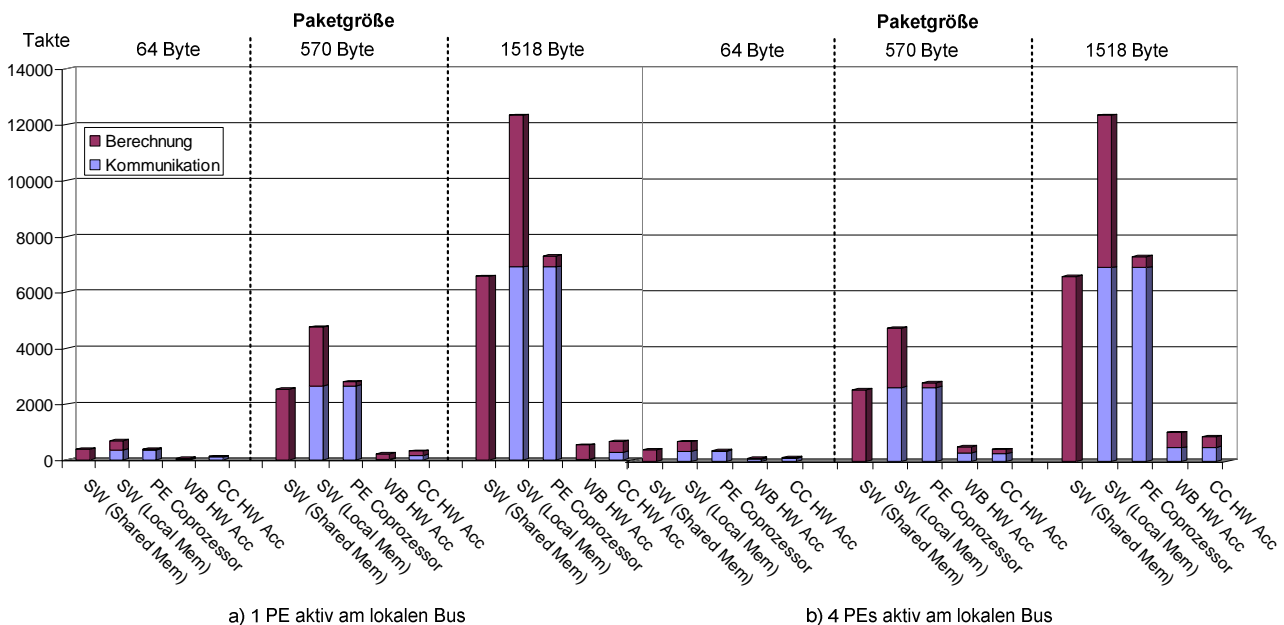
In diesem Abschnitt werden die Ergebnisse von Messreihen zur Software-basierten und Hardwarebeschleuniger-unterstützten Paketverarbeitung vorgestellt. Damit einhergehend wird eine Bewertung der verschiedenen Kopplungsmöglichkeiten (vgl. 4.3.3) der Hardwarebeschleuniger vorgenommen. Der in Abschnitt 6.3.1.1 bereits vorgestellte Hardwarebeschleuniger zur Paketprüfung beschleunigt Funktionen der Netzwerkverarbeitung auf den Netzwerkschichten 3 und 4. Dazu müs-

<sup>59</sup> Hierbei wird eine vorinitialisierte Liste von 400 Netzwerkknoten mit je vier Verbindungen pro Knoten als Vorgabe angenommen.

sen die Pakete auf diesen Schichten terminiert werden, und es bedarf einer protokollkonformen Bildung bzw. Prüfung der Checksummen und der Paketrahmen. Dies kann vollständig in Software geschehen, dann übernimmt der N-Core komplett diesen Task, oder aber: Die rechenintensive Verarbeitung wird auf die speziell hierfür entwickelten Einheiten ausgelagert, die somit den Universalprozessor (*PE*) entlasten und mehr Rechenzeit für zusätzliche Funktionen, wie z. B. *Firewall*- oder weiterreichende Paketprüffunktionen (*Deep Packet Inspection / DPI*), schaffen (vgl. Abschnitt 8.1.2).

Bei der Evaluierung der Leistungsdaten muss zudem die Anzahl der aktiven Verarbeitungseinheiten am lokalen Bus berücksichtigt werden. Der Übersichtlichkeit halber wird sich hier auf zwei Extremfälle beschränkt: Entweder ist eine Verarbeitungseinheit als einzige aktiv, oder aber vier PEs des Clusters sind mit der Paketverarbeitung beschäftigt. Mischfälle von Software-Verarbeitung und hardwarebeschleunigter Verarbeitung werden der Übersichtlichkeit dienend vernachlässigt. Auch für das übergeordnete NoC ist die Anzahl der aktiven Einheiten relevant. Da hier zunächst von einem maximal 32 PE-umfassenden System mit an das Anwendungsszenario angepasster Anzahl von Hardwarebeschleunigern ausgegangen wird, für die das GigaNoC bereits ausreichend dimensioniert ist, sind keine nennenswerten Beeinträchtigungen des Systems zu erwarten [118].

Zur Verarbeitung wurden Pakete charakteristischer Größe nach dem bereits erwähnten *Internet-Mix (iMix)* verwendet. Dies bedeutet Ethernet-Pakete von 64, 570 und 1518 Byte. Der hier verwendete 32-Bit-breite Hardwarebeschleuniger erledigt die Aufgabe ca. 14,6 (64 Byte-Paket) bzw. bis zu 16,9 mal (1518 Byte-Paket) schneller, als der Prozessor, bei gleicher Taktfrequenz. Die Tatsache, dass eine deutlich höhere Taktfrequenz des Hardwarebeschleunigers möglich wäre, wird hier außer Acht gelassen. Die Beschleunigung ist bei großen Paketen höher als bei kleineren, da hier der Anteil der fixen Operationen am Gesamtaufwand im Vergleich zu dem datenabhängigen Anteil geringer wird, vgl. Abschnitt 6.3.1.5.



**Abbildung 7-21: Vergleich zwischen Software-basierter Paketverarbeitung und Hardwarebeschleuniger-basierter Verarbeitung unter Berücksichtigung unterschiedlicher Systemanbindungen**

Bei der Analyse wird zwischen Kommunikation (*Communication*) und Berechnung (*Calculation*) unterschieden. Unter Kommunikation wird der Anteil an Zyklen verstanden, der zur Initiierung der



Berechnung bzw. Prüfung benötigt wird, also Ansteuerung des Hardwarebeschleunigers und Übergabe der Adresszeiger und des Steuerwortes. Bei der Software-basierten Variante entfällt dieser Anteil, da hier die CPU keine zusätzliche Kommunikation mit anderen Hardwareblöcken ausführen muss. Unter Kalkulation wird der Anteil der Zyklen verstanden, der für die eigentliche Prüfung der Paketdaten benötigt wird. Abbildung 7-21 gibt einen Überblick über die Verteilung von Kommunikation und Kalkulation bei Software-basierter Paketverarbeitung sowie Hardwarebeschleuniger-basierter Verarbeitung unter Berücksichtigung von unterschiedlichen Systemanbindungen.

Deutlich wird, dass die busbezogene Kommunikation bei einer Anzahl von vier Prozessoren keinen Flaschenhals bzw. Engpass für die Software-basierte Verarbeitung bedeutet. Erst bei einer Anzahl über vier Prozessoren lässt sich eine Herabsetzung der Verarbeitungsgeschwindigkeit feststellen, die aufgrund von konkurrierenden Buszugriffen zu Stande kommt. Dies liegt u. a. in der Zugriffszeit auf den gemeinsamen L2-Speicher von vier bis fünf Takten begründet, vgl. Abschnitt 4.4.

Ein gewisser Flaschenhals zeigt sich hingegen sehr wohl bei dem an den lokalen Bus angeschlossenen Hardwarebeschleuniger (*WB HW Acc*). Hier macht sich die deutlich höhere Verarbeitungsgeschwindigkeit bemerkbar. So nimmt die Anzahl der benötigten Zyklen von 27 (64 Byte-Paket / 1518 Byte-Paket) bei einem aktiven Prozessor durch entstehende Wartezeiten aufgrund der Busarbitrierung auf 72 (64 Byte-Paket) bzw. 477 (1518 Byte-Paket) bei vier Prozessoren zu. Dies entspricht einer Steigerung der Kommunikationskosten um 2,67 bzw. 17,67. Bei vier aktiven Prozessoren am Bus liegen die Kosten der Kommunikation in der gleichen Größenordnung wie die Kosten für die Berechnung. Dies ist ebenfalls der Fall bei der Ansteuerung des an einem Switch-Box-Port angeschlossenen Beschleunigers (*CC HW Acc*). Allerdings wird hier der lokale Bus stärker entlastet, und die Wartezyklen entstehen durch das NoC. Deshalb bietet sich diese Lösung besonders für große Systeme an, bei denen eine Vielzahl von CPUs die rechenintensiven, fixen Tasks auf einige wenige, spezialisierte und über das GigaNoC gut erreichbare Hardwarebeschleuniger, auslagert. Der eng an den Prozessor gekoppelte Hardwarebeschleuniger (*PE Coprozessor*) erfordert aufgrund der Speicherzugriffe über den zwischengeschalteten N-Core (vgl. auch Abschnitt 7.4.2.3) deutlich mehr Taktzyklen für die Kommunikation als die beiden anderen Varianten der Hardwarebeschleunigerkopplung.

Bei der Software-basierten Verarbeitung wird zwischen zwei Varianten unterschieden. Bei der ersten Variante (*SW Shared Mem*) befinden sich die Daten im gemeinsamen Speicher (*Shared Memory*) des Clusters und bedeuten somit häufige verzahnte Buszugriffe der einzelnen Prozessoren. Bei dieser Variante werden die Kosten der Kommunikation zu Null gesetzt, da keine expliziten Datentransfers wie bei den anderen Alternativen notwendig sind. Die Prozessoren nehmen lediglich teurere L2-Speicherzugriffe vor, die jedoch mit zur Kalkulation gezählt werden. Bei der zweiten Variante (*SW Local Mem*) hingegen werden die Daten zunächst vollständig in den lokalen Prozessorspeicher kopiert. Die Kosten für die Kommunikation dieser Variante liegen in der gleichen Größenordnung wie die der eigentlichen Berechnung. Diese Form der Bearbeitung lohnt sich deshalb nur dann, wenn weitere Folgeberechnungen auf den Paketdaten stattfinden und es so im Anschluss zu günstigen L1-Speicherzugriffen käme. Dies würde im weiteren Verlauf den Bus bei einem Mehrprozessorbetrieb entlasten und könnte dann letztendlich zu einer beschleunigten Verarbeitung führen. Beim Einsatz des GigaNetIC-Multiprozessorcaches (vgl. Abschnitt 4.4.2) anstelle des normalen lokalen Prozessorspeichers könnte die Einlagerung neuer Pakete durch *Prefetching* bereits während einer Bearbeitung geschehen, was den Durchsatz des Systems zusätzlich erhöhen würde.

Bei der Software-basierten Berechnung liegt die Bearbeitungszeit zwischen 6,1 Takte/Byte (64 Byte-Paket) und 4,3 Takte/Byte (1518 Byte-Paket), wohingegen der Hardwarebeschleuniger nur 0,4 Takte/Byte (64 Byte-Paket) bzw. 0,3 Takte/Byte (1518 Byte-Paket) benötigt.

Abbildung 7-22 zeigt die maximal zu verarbeitenden Pakete pro MHz Rechenakt des Clusters. Hierzu werden die Varianten mit einer bzw. mit vier aktiven CPUs für die drei verschiedenen Paketgrößen in die Betrachtung einbezogen. Es wird ersichtlich, dass sich der maximale Durchsatz mit den Bus- und NoC-gekoppelten Varianten erzielen lässt. Der höchste Durchsatz, bei dem nur eine CPU aktiv ist, wird bei der busgestützten Hardwarebeschleuniger-Ankopplung (*WB HW Acc*) erzielt. Hier wird bei 1518 Byte großen Paketen ein Durchsatz von 2,82 MByte/MHz erreicht. Die NoC-basierte Lösung (*CC HW Acc*) erreicht bei vier aktiven Prozessoren einen maximalen Durchsatz von 6,9 MByte/MHz bei 1518 Byte großen Paketen. Bezogen auf die in [109] gezeigten Synthesewerte entspräche dies einem maximalen Durchsatz von 1,814 GByte bei einer Betriebsfrequenz von 263 MHz pro Cluster. Ein System dieser Leistungsfähigkeit wäre in der Lage, Pakete auf Layer-3- und Layer-4-Schicht simultan für 900 *ADSL2+*-Anschlüsse (16MBit/s, *Downlink*) unter Vollast zu prüfen. Bei der rein Software-basierten Variante ließen sich noch 120 dieser DSL-Anschlüsse unter Vollast betreiben. Der in Abschnitt 8.1.2 vorgestellte FPGA-basierte Demonstrator könnte bei den derzeitigen 12,5 MHz folglich noch über 86 MByte an Daten pro Sekunde verarbeiten bzw. 42 *ADSL2+*-Anschlüsse bedienen.

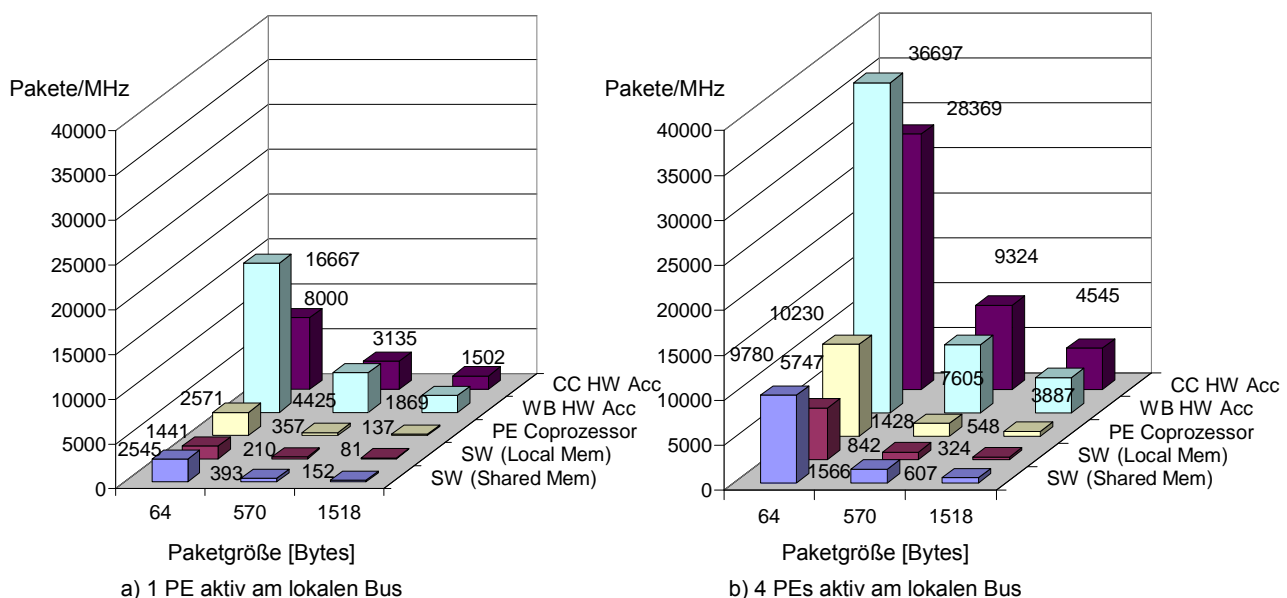


Abbildung 7-22: Bandbreite pro MHz der einzelnen Systemvarianten

Mit Hilfe der leistungsfähigen Analysewerkzeuge der GigaNetIC-Architektur (vgl. Kapitel 5 und 6) kann im Vorfeld einer Chiprealisierung bereits sehr genau bestimmt werden, welche Hardwarebeschleuniger mit welcher Kopplung im GigaNoC eingebunden werden müssen, um möglichst ressourceneffizient eingesetzt werden zu können. In diesem Beispiel konnte gezeigt werden, dass losegekoppelte Beschleuniger (*CC HW Acc* bzw. *WB HW Acc*) besonders bei großen Systemen vorteilhaft eingesetzt werden können. Der enggekoppelte Beschleuniger (*PE-Coprozessor*) eignet sich ggf. in Verbindung mit dem GigaNetIC-Multiprozessorcache. Die Software-basierten Ansätze (*SW Shared Mem* und *SW Local Mem*) brauchen im Gegensatz zu allen anderen Varianten deutlich mehr Taktzyklen. Lediglich die höhere Flexibilität und Zukunftssicherheit der Softwarelösungen erschei-

nen vorteilhaft, was allerdings bei standardisierten Algorithmen wie der hier betrachteten Paketprüfung von geringer Relevanz ist.

## 7.8 Zusammenfassung

In diesem Kapitel wurde die GigaNetIC-Architektur im Hinblick auf den Einsatz für unterschiedliche Netzwerkszenarien untersucht. Gerade im Netzwerkbereich bieten sich parallele Systeme zur Datenverarbeitung an, da hier eine Vielzahl von parallelen, zum Teil nicht korrelierten Datenströmen simultan von den Verarbeitungseinheiten bearbeitet werden kann.

Im Rahmen dieses Kapitels wurde ein *IP-DSLAM-Benchmark* zur Bewertung der GigaNetIC-Architektur für Zugangsnetzwerke vorgestellt. Die Leistungsfähigkeit der Architektur wurde für unterschiedliche Szenarien analysiert und mit anderen Ansätzen verglichen. Im Anschluss wurde die Leistungsfähigkeit des von uns entwickelten Prozessorkerns N-Core für relevante Funktionen durch Optimierung der Architektur, Instruktionssatzerweiterungen sowie Hinzufügen von anwendungsspezifischen Hardwarebeschleunigern deutlich erhöht. So konnten mit Hilfe der implementierten Instruktionssatzerweiterungen Beschleunigungen von über 24 % bei einem marginalen Flächenmehraufwand von 2,7 % für den Prozessorkern erzielt werden. Durch diese Maßnahme können zusätzlich über 20 % an Energie eingespart werden. Verglichen mit den eingebetteten Prozessorkernen ermöglichen die realisierten Hardwarebeschleuniger Beschleunigungen um teilweise mehrere Größenordnungen. Der Flächenbedarf ist höher als der der zusätzlichen Superinstruktionen, der Energiebedarf der Gesamtschaltung wird jedoch deutlich reduziert. Für den *IP-DSLAM-Referenzbenchmark* konnte die benötigte Energie auf weniger als ein Zwölftel reduziert werden.

Es wurde eine modulare Methode zur effizienten Modellierung von Netzwerkanwendungen vorgestellt, mit deren Hilfe der bereits entworfene *IP-DSLAM-Benchmark* auf Systemebene zu einem noch realistischeren Referenzbenchmark erweitert werden konnte. Diese Anwendung wurde ebenfalls auf die GigaNetIC-Architektur portiert und analysiert. Im Anschluss wurde die Hardware für besonders rechenintensive Aufgaben der Zielapplikation optimiert. Mit Hilfe eines eigens entwickelten Visualisierungswerkzeugs, dem *DSLAM-System-Explorer*, können die Leistungsdaten des N-Cores, die erzielten Beschleunigungen der Hardwareerweiterungen und Leistungsvergleiche mit anderen Prozessorfamilien komfortabel veranschaulicht werden. Es lassen sich Hochrechnungen bzgl. des Hardwareaufwands für gewünschte Anforderungen des *IP-DSLAM*-Anwendungsszenarios aufstellen, die eine gezielte Evaluierung des Entwurfsraums ermöglichen.

Eine Analyse der Leistungsfähigkeit der verschiedenen Kopplungsarten von Hardwarebeschleunigern an das GigaNoC der GigaNetIC-Architektur zeigt Vor- und Nachteile der einzelnen Varianten auf. Da die Art der Kopplung und die Anzahl der Hardwarebeschleuniger abhängig von den Anforderungen des jeweiligen Anwendungsszenarios, können vielversprechende Lösungen im Hinblick auf die Ressourceneffizienz mit Hilfe der leistungsfähigen GigaNetIC-Simulationsumgebungen ermittelt werden.

In diesem Kapitel wurde gezeigt, dass die entwickelte, skalierbare Systemarchitektur eine – durch die Werkzeugkette unterstützt – hierarchisch optimierbare, ressourceneffiziente Plattform für Netzwerkanwendungen und Coprozessorsysteme darstellt. Aufgrund der guten Skalierbarkeit der zugrunde liegenden Systemarchitektur können GigaNetIC-basierte Systeme für unterschiedlichste Einsatzbereiche in Netzwerkanwendungen eingesetzt werden. Der Vorteil liegt hier u. a. in dem

gleichbleibenden Architektur- und Programmiermodellansatz und der sich hieraus ergebenden guten Wartbarkeit. Durch Synergieeffekte und sich akkumulierende Lernkurven der Entwickler folgen Zeitersparnis bei der Realisierung neuer Systemvarianten und kürzere *Time-to-Market-Spannen*. Deshalb und aufgrund der hohen Flexibilität der Architektur lassen sich ebenfalls längere *Time-in-Market-Spannen* erzielen.

## 8 Prototypische Implementierung des Systems

Dieses Kapitel beschreibt die vollständige prototypische Umsetzung der bisher konzeptionellen und theoretischen Überlegungen zur Gestaltung der GigaNetIC-Architektur der vorangegangenen Kapitel. Die Realisierung geschieht zum einen in Gestalt eines einsatzfähigen, FPGA-basierten Systems, unter Verwendung der RAPTOR2000-Rapid-Prototyping-Entwicklungsumgebung (vgl. Abschnitt 5.5), und zum anderen als Synthese auf zwei aktuelle CMOS-Standardzellentechnologien.

Für diese beiden disparaten Zieltechnologien wurde ein einheitlicher skriptbasierter Syntheseablauf entwickelt, der weitestgehend automatisiert abläuft. Durch Parametervariation kann er leicht an die spezifizierte Zieltechnologie und nahezu beliebige Systemgrößen und Ausprägungen des Chip-Multiprozessors angepasst werden. Neben der schnellen Systememulation durch den FPGA-basierten Prototypen, die in Abschnitt 5.5 bereits Erwähnung fand und dort u. a. die Entwicklungszyklen für umfangreichere Softwareprojekte erheblich beschleunigen half, ergeben sich zusätzlich wertvolle Impulse zur Verbesserung bzw. zur Fehlerbehebung der Hardwarebeschreibung. Betrachtet man die *NRE*-Kosten für moderne Halbleiterprozesse im Sub-100-nm-Bereich ist dies ein ganz erheblicher Vorteil gegenüber rein durch Simulation verifizierten Hardwareentwürfen. Diese liegen mittlerweile in der Größenordnung von einer Million Euro, so dass eine gründliche Verifikation des Entwurfs mehr als erstrebenswert ist.

Durch Analysen der Codeabdeckung während der einzelnen Simulationsphasen der HDL-Beschreibung als auch durch die Einbindung des FPGA-Prototypen in ein zukünftiges Anwendungsszenario des GigaNetIC-Systems wird die Fehlerwahrscheinlichkeit deutlich minimiert. Zusätzliche Sicherheit bzgl. der Fehlerfreiheit der Hardware könnten Erweiterungen der Hardwarebeschreibung durch die Nutzung von eigenschaftsspezifizierenden Sprachen, wie z. B. *PSL* (*Property Specification Language for Assertion-Based Verification*), die während der Simulation das Verhalten nach vorgegebenen Mustern und Regeln überprüfen, erreicht werden. Letztendlich wäre eine formale Verifikation des Systems im Bezug auf einen fehlerfreien Entwurf wünschenswert. Aufgrund der beschränkten Zeit im Rahmen dieser Arbeit konnten diese beiden Verifikationsmechanismen allerdings noch nicht implementiert werden.

Im folgenden Abschnitt wird auf die FPGA-Realisierung des GigaNetIC-Systems näher eingegangen, bevor in Abschnitt 8.2 die Resultate der ASIC-Realisierung auf Basis zweier CMOS-Standardzellentechnologien vorgestellt werden.

### 8.1 FPGA-Realisierung – GigaNetIC-Prototyping-Plattform

Für die funktionale Verifikation wird die RTL-Beschreibung auf das im Fachgebiet Schaltungstechnik entwickelte *Rapid Prototyping System* RAPTOR2000 abgebildet. Auf Basis feingranular rekonfigurierbarer Bausteine (FPGAs) ermöglicht dieses System eine Emulation des gesamten Chip-Multiprozessors. RAPTOR2000 ist über den PCI-Bus an einen *Host*-Computer angebunden und kann über diesen PC komfortabel konfiguriert werden. Die Prototypen-Umgebung integriert neben rekonfigurierbaren Einheiten, die die Multiprozessorarchitektur emulieren, auch Speicher und externe Schnittstellen. Auf diese Weise kann der Prototyp in eine reale Systemumgebung integriert und verifiziert werden. Die frühe Bereitstellung des Prototyps ermöglicht neben einer Verifika-

tion des zugrunde liegenden RTL-Modells eine effiziente parallele Entwicklung von Software und Hardware. Dabei ist besonders die hohe Geschwindigkeit der Hardware-Emulation von großem Vorteil, die um mehrere Größenordnungen (vgl. Kapitel 5) über der Simulationsgeschwindigkeit der Schaltung auf *RTL*-Ebene liegt. Somit können auch komplexe Testprogramme und Benchmarks auf dem Prototypen in vertretbarer Zeit ausgeführt werden. Der hier zu realisierende FPGA-Prototyp wird zunächst in Bereichen der Netzwerkdatenverarbeitung eingesetzt werden.

Grundsätzlich unterscheidet sich die FPGA-Realisierung nicht von der folgenden ASIC-Implementierung in CMOS-Standardzellentechnologien. Es wird die gleiche Hardwarebeschreibung verwendet, lediglich die verwendete On-Chip-Speichertechnologie muss für die jeweilige Zieltechnologie eingestellt werden. Zur Erzielung der jeweils besten Synthesergebnisse werden für beide Zieltechnologien angepasste Steuerskripte verwendet. In der Größe des realisierbaren Multiprozessorsystems und der erreichbaren Taktfrequenz sind heutige FPGA-Technologien deutlich eingeschränkter als ein moderner standardzellenbasierter ASIC. Bei den FPGA-basierten Prototypen wird das System auf zwei Cluster beschränkt im Gegensatz zu acht bzw. auch 20 in der ASIC-Variante. Auch die erreichbare Taktfrequenz ist ca. 20mal geringer. Dennoch sind für eine schnelle, kostengünstige und zugleich detaillierte Verifikation in der späteren Systemumgebung FPGA-basierte Prototypen von unschätzbarem Wert.

### 8.1.1 Aufbau und Synthesergebnisse

Im Folgenden werden der Aufbau und die Ergebnisse der Synthese auf die verwendete FPGA-Architektur näher erläutert. Abbildung 8-1 zeigt das auf Basis zweier Xilinx Virtex-II 8000 FPGAs zusammen mit einer *Fast-Ethernet*-Schnittstellentochterplatine des RAPTOR2000-Systems realisierte GigaNetIC-Chip-Multiprozessorsystem.

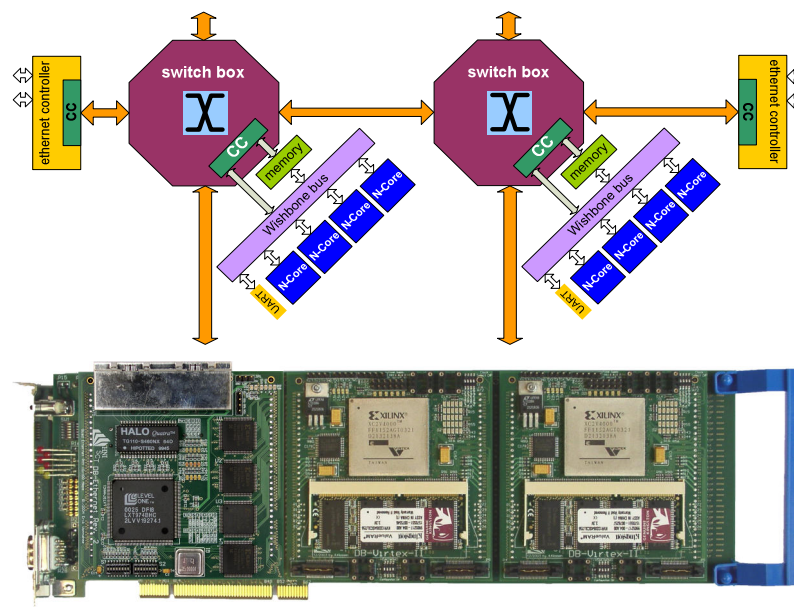


Abbildung 8-1: RAPTOR2000-basierter FPGA-Prototyp der GigaNetIC-Architektur

Das System umfasst zwei Cluster mit jeweils vier N-Cores, die über einen Wishbone-Bus mit dem Communication-Controller und dem gemeinsamen Paketspeicher sowie lokaler Peripherie, in diesem Beispiel einen UART zum Anschluss der zur Interaktion mit dem Benutzer benötigten Touchscreens, verbunden sind. Die beiden Switch-Boxen sind miteinander über die Ports 1 bzw. 3 verbunden, wobei diese Verbindungen über die externen Anschlüsse der beiden FPGAs realisiert

sind. Da die benötigten Anschlüsse nicht in ausreichender Anzahl auf dem RAPTOR2000-Mainboard zur Verfügung stehen wird bei der Übertragung die Serialisierungsfunktionalität der Switch-Box-Ports (vgl. Abschnitt 6.6) ausgenutzt.

Tabelle 8-1 zeigt die Synthesergebnisse für die FPGA-Realisierung eines GigaNetIC-Clusters. Es ist zu beachten, dass der hier angegebene Wert von 14.133 *Slices* für eine Switch-Box mit fünf Ports und einer FIFO-Tiefe von drei gilt. In der Systemkonfiguration aus Abbildung 8-1 werden allerdings nur drei dieser fünf Ports benötigt, so dass sich bei gewünschter Optimierung seitens der Xilinx-Entwicklungsumgebung dieser Wert auf 7.946 *Slices* reduzieren lässt. Die Switch-Box in ihrer vollständigen Ausprägung nimmt fast genauso viele Ressourcen in Anspruch, wie die vier N-Core-Subsysteme mit je 3.662 *Slices*. Dies liegt darin begründet, dass die derzeitige Implementierung Register zur Realisierung der FIFO-Ketten verwendet, die in der FPGA-Implementierung 59 % der benötigten *Slices* in Anspruch nehmen.

**Tabelle 8-1: Synthesergebnisse für die FPGA-Realisierung eines GigaNetIC-Clusters**

Hardware-Blöcke	Slices	RAM16s
4 x N-Core-Subsysteme (inkl. 4 x 32 KB Speicher)	14.648	64
Switch-Box (inkl. Communication-Controller)	14.133	-
Ethernet-Controller (inkl. 2 Ethernet-Ports)	5.544	32
Paketspeicher (32 KB)	53	16
SRAM-Schnittstelle	22	-
Serielle Schnittstelle (UART)	626	-
Wishbone-Arbitrer	13	-
<b>Σ</b>	<b>35.039</b>	<b>112</b>

Die detaillierten Ergebnisse zum N-Core-Subsystem werden in Tabelle 8-2 aufgeschlüsselt. Erwähnenswert ist die Tatsache, dass die Switch-Box zwar die größte Komponente des Clusters darstellt, der kritische Pfad des Systems allerdings vom N-Core zu einer maximalen Taktfrequenz von 16,79 MHz bestimmt wird<sup>60</sup>. Die letztlich verwendete Taktfrequenz des Gesamtsystems beträgt 12,5 MHz.

**Tabelle 8-2: Synthesergebnisse für die FPGA-Realisierung des N-Core-Subsystems**

Hardware-Blöcke	Slices	RAM16s
Address-Dekoder	121	-
N-Core-Kern	3.405	16
ClockGen	6	-
Informationsregister	1	-
LB-Schnittstelle	110	-
PIC	6	-
RR Arbitrer	13	-
<b>Σ</b>	<b>3.662</b>	<b>16</b>

Bei einer Gesamtanzahl der verfügbaren *Slices* des XC2V8000<sup>61</sup> von 46.592 lastet ein GigaNetIC-Cluster dieses FPGA zu 75 % aus. Für die benötigten Speicherkomponenten werden die *BlockRAM*-Ressourcen (*RAM16*) des FPGAs verwendet. Die hierbei kleinste *BlockRAM*-Instanz entspricht einem 18 KBit-großen *Dual-Port*-Speicher mit konfigurierbarer Datenbusbreite. Diese elementaren

<sup>60</sup> Alle Werte beziehen sich auf Xilinx FPGAs des Typs XC2V8000 mit dem langsamsten *Speedgrade* 4.

<sup>61</sup> Der XC2V8000 gehörte 2005 zu den FPGAs mit den meisten verfügbaren Logikelementen (*Slices*) und ist der größte Baustein der Virtex II Familie mit 8 Mio. Systemgattern bzw. 46.592 *Slices*.



Speicherblöcke werden je nach Konfiguration des Speichermoduls nicht immer vollständig ausgenutzt.

### 8.1.2 GigaNetIC-Demonstrator – Einsatz in einem realen Netzwerkszenario

Ähnlich dem im Liberouter-Projekt [193] verfolgten Ansatz setzen wir beim GigaNetIC-Demonstrator auf Internet-Protokoll-basierte Paketverarbeitung mit Hilfe von FPGAs. Allerdings nutzen wir zunächst Cluster von Universalprozessoren wie den N-Core und erweitern dann das System durch zusätzlich benötigte Hardwarebeschleuniger. Vorteilhaft an dem GigaNetIC-System ist die Modularität des RAPTOR2000-Boards. So können bis zu sechs Tochterplatinen mit ggf. sechs interoperablen FPGAs eingesetzt werden, wohingegen der monolithische Aufbau der *Combo-Boards* des Liberouter-Projekts nur ein FPGA integrieren und derzeit keine derartige Erweiterbarkeit vorsehen.

Abbildung 8-2 zeigt den funktionsfähigen RAPTOR2000-basierten GigaNetIC-Demonstrator, der zwei N-Core-Cluster prototypisch auf FPGAs realisiert. Bei diesem Aufbau übernehmen die N-Core-Cluster die komplette Paketverarbeitung eines zwischengeschalteten Routers und übertragen einen *Video-Live-Stream* in Echtzeit zu einem angeschlossenen Netzwerk-Client. Ebenso können die bisher gezeigten Hardwarebeschleuniger integriert und zur beschleunigten Verarbeitung eingesetzt werden. Die Auswirkungen des Hinzuschaltens des IP-Headercheck-Hardwarebeschleunigers z. B. konnten in Form einer deutlich besseren Bildqualität der Videoübertragung des in Abbildung 8-3 beschriebenen Netzwerkszenarios beobachtet werden.

Der Demonstrator wurde auf der CeBIT 2005 sowie auf der Hannover-Messe 2005 auf dem fachgebietseigenen Messestand im Rahmen des Bereichs „Forschungsland NRW“ der Öffentlichkeit vorgestellt.

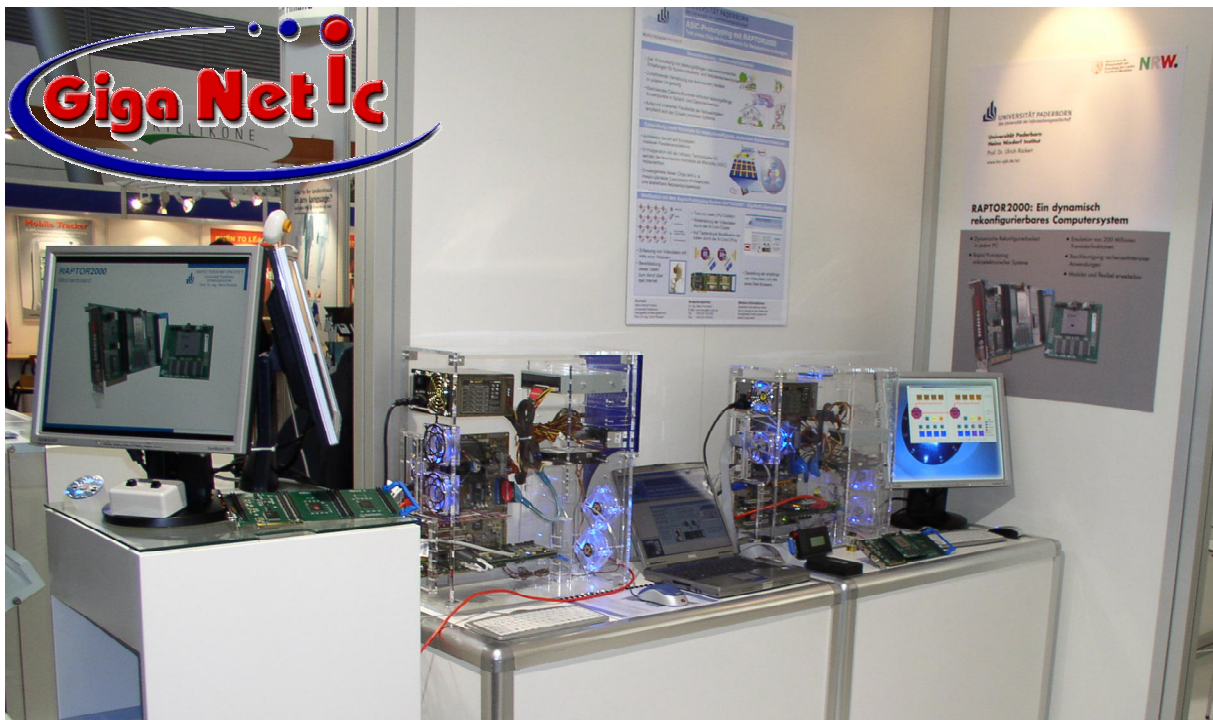
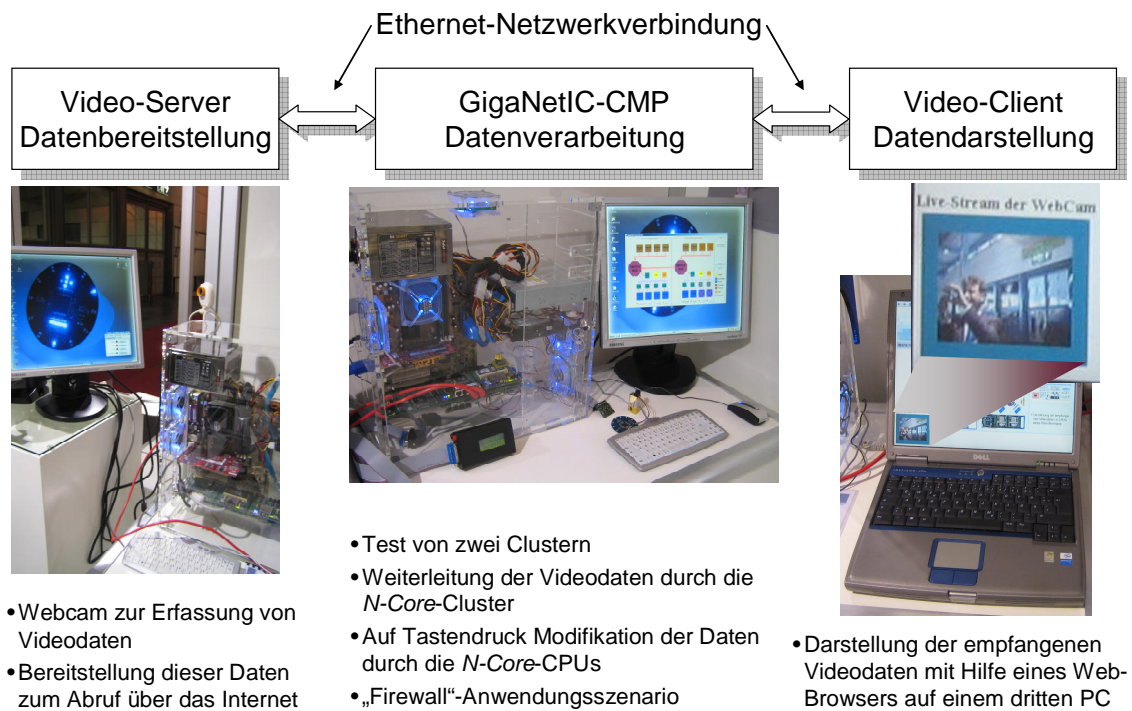


Abbildung 8-2: FPGA-basierter GigaNetIC-Demonstrator, ausgestellt am Stand Forschungsland NRW auf der CeBIT 2005 und Hannover Messe 2005



Das mit dem GigaNetIC-Demonstrator präsentierte Netzwerkanwendungsszenario wird in Abbildung 8-3 detailliert vorgestellt. Ein PC-basierter *Video-Server* stellt die *Livebilder* einer angeschlossenen Webcam zum Abruf über das in diesem Fall kabelgebundene, ethernetbasierte Internet zur Verfügung. Als Zwischenstelle, sozusagen als Router, fungiert hier der GigaNetIC-Chip-Multiprozessor, realisiert als FPGA-Prototyp, basierend auf dem RAPTOR2000-System. Hierbei kommen zwei GigaNetIC-Cluster, wie in Abbildung 8-1 gezeigt, zum Einsatz. Die Datenpakete werden über die Ethernetschnittstelle des GigaNetIC-Clusters in den lokalen Paketspeicher transferiert. Von hier holen die N-Cores die Pakete nach dem *Best-Effort*-Prinzip ab und führen zunächst eine Checksummenprüfung nach [160] durch. Der Benutzer kann mit dem System über eine berührungssensitive Anzeige interagieren. Mit Hilfe von Schaltflächen wird ihm die Möglichkeit gegeben die Farbe des Bildrahmens auszuwählen. Den N-Cores wird dieses Kommando übermittelt, woraufhin sie die Datenpakete nach dem bekannten Muster der Rahmenfarbe durchsucht und diese dann durch die vom Benutzer gewählte ersetzt. Im Anschluss werden die Daten zum Versand wieder zum Ethernetcontroller über das GigaNoC geschickt. Letztendlich werden sie zum anfordernden PC (*Video-Client*) geleitet, der diese innerhalb eines Webbrowsers als *Livestream* anzeigt. Das hier realisierte Netzwerkszenario demonstriert die Funktionsfähigkeit der GigaNetIC-Architektur. Es hat geholfen die entworfene Architektur in einem realen Anwendungsfeld zu testen und in Echtzeit analysieren zu können.



**Abbildung 8-3: GigaNetIC-Demonstrator – reales Netzwerkszenario als Verifikationsbeispiel**

Um mit dem FPGA-Prototypen effektiv arbeiten zu können, wurde eine graphische Benutzeroberfläche (Graphical User Interface / GUI) für die Interaktion mit der Hardware erstellt. Diese verwendet die vom RAPTOR2000-System zur Verfügung gestellte *DLL* (*Dynamic Link Library*), um auf das PCI-basierte Prototypensystem vom PC aus zugreifen zu können.

Über die GigaNetIC-GUI (vgl. Abbildung 8-4) können alle wesentlichen Funktionen des Chip-Multiprozessors gesteuert bzw. abgerufen werden. Sie ermöglicht die einfache **Konfiguration** der verwendeten FPGA-Tochterplatinen mit den jeweiligen Bitstreams, die komfortabel mit Hilfe einer Dateivorschau von den zur Verfügung stehenden Laufwerken ausgewählt werden können. Außer-

dem werden so die jeweiligen Programmdateien den einzelnen N-Cores zugeordnet, die auch während des Betriebs verändert werden können.

Eine weitere wichtige Funktion der Software sind **Kontrollaufgaben** zur Steuerung der Hardware. Hierzu zählen das Starten, Stoppen und Zurücksetzen (*Reset*) aller oder einzelner Prozessoren sowie das Verarbeiten der anliegenden Daten, die über den UART eingehen, so z. B. die Benutzereingaben über den *Touchscreen* als auch dessen Ansteuerung. Mit Hilfe dieser bidirektionalen Schnittstelle kann komfortabel mit dem System interagiert werden und so direkt auf den Programmablauf der einzelnen N-Cores Einfluss genommen werden, wie z. B. beim beschriebenen GigaNetIC-Demonstrator (vgl. Abbildung 8-3). Weiterhin kann die Konfiguration der zur Verfügung stehenden Ethernetports über die GigaNetIC-GUI vorgenommen werden, wie z. B. Übertragungsgeschwindigkeit und Verbindungsstatus.

Mit Hilfe der GigaNetIC-GUI verfügt der Entwickler über leistungsfähige **Debuggingfunktionen**, die es erlauben, graphisch auf relevante Statusinformationen zu den Prozessoren und den anderen Blöcken des Chip-Multiprozessors zuzugreifen. Außerdem lassen sich alle Speicherinhalte anzeigen und bei Bedarf abspeichern (so genannte *Memory Dumps*). Der aktuelle Zustand des Communication-Controllers kann ebenso angezeigt wie gesteuert werden. So können z. B. Pakete zu Testzwecken ins GigaNoC injiziert werden.

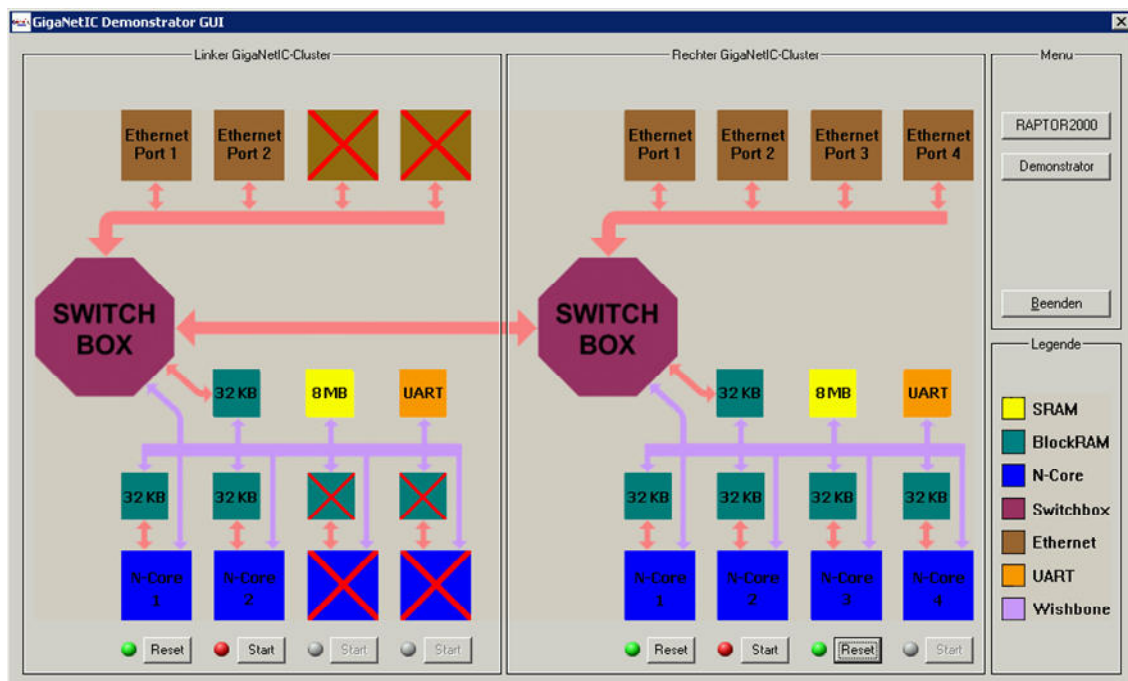


Abbildung 8-4: Graphische Benutzeroberfläche des GigaNetIC-Demonstrators

Die GigaNetIC-GUI ist adaptiv und erkennt die Hardwarekonfiguration der involvierten FPGAs automatisch. Hierzu zählen u. a. die Anzahl verfügbarer N-Cores, Ethernetports und die Größe der verfügbaren Speichermodule. Hardwaremodule, die aufgrund der jeweiligen Konfiguration des FPGAs nicht zur Verfügung stehen, werden automatisch mit einem roten „X“ markiert und stehen nicht zur Verfügung.

Für die spätere ASIC-Realisierung, die zunächst auch auf einer RAPTOR2000-Tochterplatine getestet werden soll, kann auf viele Funktionen dieser Software zurückgegriffen werden. Hierdurch wird ein hoher Wiederverwendungswert erreicht und nur wenige Funktionen müssen ggf. hinzugefügt werden bzw. können im Falle der Auswahl von *Konfigurationsbitstreams* entfallen.

## 8.2 ASIC-Realisierung in CMOS-Standardzellen

Heutige FPGA-Technologien wie die der Xilinx-Virtex-II-Serie, deren Verwendung im vorigen Abschnitt beschrieben wurde, eignen sich noch nicht für die Realisierung sehr komplexer bzw. sehr großer Systeme, wie es z. B. ein GigaNetIC-System mit acht und mehr Switch-Boxen darstellt. Derzeit besteht noch eine deutliche *Flächen-Funktions-Diskrepanz* zwischen der vorgestellten FPGA-Technologie und aktuellen Standardzellen-Technologien. Der Unterschied für die benötigte Fläche für eine gegebene Funktionalität liegt derzeit bei einem Faktor von ca. 40 [194]<sup>62</sup>. Der Unterschied in der maximalen Taktfrequenz zwischen Standardzellen- und FPGA-Entwürfen (schnellstes *Speedgrade*) wird in [194] mit Faktor 2 bis 3 angegeben. Im Vergleich dazu liegen die FPGAs mit dem langsamsten *Speedgrade* ca. Faktor 3 bis 4,5 unter der Leistungsfähigkeit der Standardzellentechnologie. Das Verhältnis bei der dynamischen Verlustleistung bewegt sich lt. [194] zwischen Faktor 9 und 12. Diese Angaben verdeutlichen, dass vor allem noch einiges an der Flächeneffizienz der FPGAs verbessert werden muss, bevor sie als Alternative zur Standardzellentechnologie im Bereich großer Systeme in Frage kommen. Zusätzlich bieten die Standardzellentechnologien bei den angestrebten, hohen Stückzahlen im Netzwerkbereich einen deutlichen Kostenvorteil gegenüber der aktuellen FPGA-Technologie. Diese Umstände empfehlen deshalb, trotz der hohen *NRE*-Kosten, CMOS-basierte Standardzellentechnologien als passende Zieltechnologie für die GigaNetIC-Architektur.

Nach erfolgreichem Test der GigaNetIC-Architektur mit Hilfe des FPGA-basierten Prototyps, sowie durch die Verifikation des Systems durch die zahlreichen Simulationen mit Hilfe der in Kapitel 5 vorgestellten Entwicklungsumgebungen schließt sich die Realisierung in der angestrebten Zieltechnologie, moderne CMOS-basierte Standardzellentechnologien in 130 nm bzw. 90 nm an.

### 8.2.1 GigaNetIC-Architektur mit SRAM-basiertem L1-Speicher

Die folgenden Tabellen zeigen die Syntheseresultate der GigaNetIC-Hauptkomponenten für die eben genannten Technologien auf. Bei der ASIC-Realisierung wird beispielhaft ein System bestehend aus acht Clustern, die in einem 4×2-Gitter angeordnet sind, betrachtet. Jeder Cluster besteht aus vier N-Cores, einer Switch-Box mit fünf Ports mit einer FIFO-Tiefe von drei. Als lokaler Bus wird derzeit für die Variante ohne Cache der Wishbone-Bus favorisiert, aber auch der AMBA-Bus ist denkbar und flächenmäßig nahezu gleich groß. Insgesamt bedeutet dies 32 N-Core-Prozessoren mit 1,25 MByte On-Chip-Speicher.

Tabelle 8-3 stellt zunächst die jeweiligen Flächenanforderungen der Komponenten in beiden Technologien gegenüber. Die Gesamtgröße des hier vorgestellten Systems liegt bei 50,01 mm<sup>2</sup> (130 nm) bzw. 43,7 mm<sup>2</sup> (90 nm). Die sich aus den *S-Parametern* (vgl. Definition 29) ergebende Abschätzung, dass sich bei einer Verkleinerung der minimalen Strukturgröße um den Faktor  $\sqrt{2}$  der resultierende Flächenbedarf in der neuen Technologie halbiert, bewahrheitet sich hier nicht für alle Komponenten. Lediglich für die Switch-Boxen und die lokalen Busse trifft diese „Daumenregel“ zu. Die Fläche des Speichers, der 70 % (130 nm) bzw. 81 % (90 nm) der Gesamtfläche ausmacht,

---

<sup>62</sup> Allen Angaben zu Grunde liegen detaillierte Analysen aktueller FPGA- und Standardzellentechnologien, die zur Abbildung unterschiedlicher Schaltungsentwürfe zur möglichst vollständigen Abdeckung des Entwurfsraums herangezogen wurden.

ist in der 90-nm-Technologie bis auf die dritte Nachkommastelle identisch mit der Speicherfläche der 130-nm-Technologie. Dies liegt in der noch nicht vollzogenen Optimierung der 90-nm-Speicherzellen dieser zum Zeitpunkt der Synthesen neu eingeführten Technologie begründet. Zur Veranschaulichung der Flächenreduktion zeigt Abbildung 8-9 eine Hochrechnung der Clusterfläche bei einer für Speicher eher konservativ angenommenen Skalierung mit  $S^{-2}$ . In diesem Fall reduzierte sich die Fläche von 43,7 mm<sup>2</sup> auf nur noch 25,97 mm<sup>2</sup>, nähme also nur noch 59,4 % der Ursprungsfläche ein.

**Tabelle 8-3: Charakteristika der GigaNetIC-Hauptkomponenten für 130-nm- bzw. 90-nm-Standardzellentechnologie**

SoC-Haupt-Komponenten	Fläche [mm <sup>2</sup> ]		Frequenz [MHz]		Verlustleistung gesamt [mw]		Verlustleistung je IP-Block [mw/MHz]	
	130nm	90nm	130nm	90nm	130nm @205MHz	90nm @285 MHz	130nm	90nm
32 N-Core	32 x 0,160	32 x 0,120	205	285	352,0	410,4	0,054	0,045
8 Switch-Boxen [mit 5 Ports, FIFO-Tiefe 3]	8 x 1,129	8 x 0,530	560	714	660,9	741,0	0,403	0,325
32 lokale Speicher (32 KB)	32 x 0,875	32 x 0,875	400	450	1165,1	465,1	0,178	0,051
8 lokale Paketpuffer (2 x 16 KB)	8 x 2 x 0,466	8 x 2 x 0,466	400	450	518,2	175,6	0,316	0,077
8 lokale On-Chip-Busse (Wishbone / AMBA)	8 x 0,050	8 x 0,020	211	290	15,6	13,7	0,010	0,006
<b>Insgesamt</b>	<b>50,01</b>	<b>43,70</b>	<b>205</b>	<b>285</b>	<b>2711,8</b>	<b>1805,8</b>	<b>13,228</b>	<b>6,336</b>

Die Switch-Boxen zusammen mit den lokalen Bussen nehmen als gesamte Kommunikationsinfrastruktur des Chip-Multiprozessors 18,9 % (130 nm) bzw. 9,3 % (90 nm) der Gesamtfläche des synthetisierten SoCs ein. In DALLY [57] werden ca. 10 % Flächenanteil der Kommunikationsstruktur als durchaus vertretbar angesehen. Dies entspricht ebenfalls den von mir in [14] auf grundlegenden Analysen beruhenden ersten Hochrechnungen zur Ausprägung eines effizienten On-Chip-Netzwerks.

Die maximal erreichbare Taktfrequenz bestimmt beim GigaNetIC-System die langsamste Komponente. So lässt sich derzeit in 130-nm-Technologie eine Taktrate von ca. 205 MHz im Gegensatz zu 285 MHz bei der 90-nm-Technologie erreichen. Derzeit bestimmt der N-Core diese Frequenz durch den längsten kritischen Pfad der Gesamtschaltung. Durch weitere Optimierungen lassen sich allerdings noch höhere Taktfrequenzen erreichen, da hier noch nicht alle Möglichkeiten ausgeschöpft sind (vgl. Abschnitt 6.2).

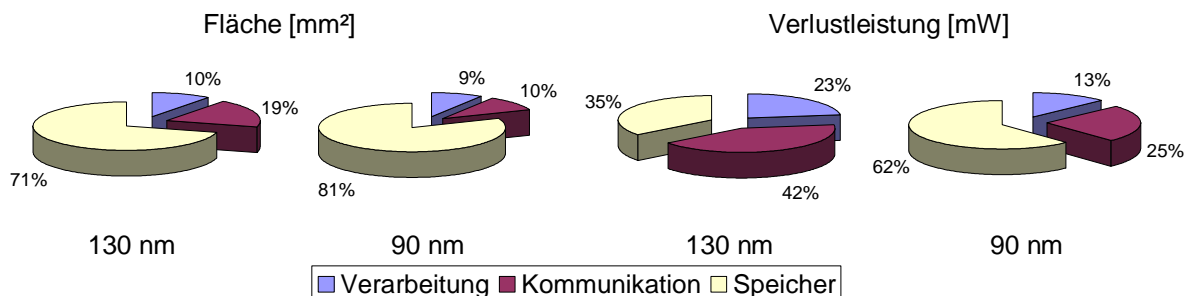
Bei der Verlustleistung, die sich auf die maximal realisierbare Taktfrequenz bezieht, liegt der angenommene Wert bei 2,7 W (130 nm) bzw. 1,8 W (90 nm). Dies ist allerdings eine Abschätzung, die auf Annahmen der Schaltwahrscheinlichkeiten seitens des Synthesewerkzeugs (50%*S*) und nicht auf annotierten Werten (*AS*) beruht. Dies deutet auf eine eher geringere Leistungsaufnahme im normalen Betrieb hin, da hier nicht eine Schaltwahrscheinlichkeit von 50 % in jedem Takt für alle Komponenten anzunehmen ist. Mit Hilfe der in Abschnitt 5.3 vorgestellten Werkzeugkette ist es nun möglich für dedizierte Anwendungen die konkreten Schaltwahrscheinlichkeiten aufzuzeichnen und so sehr genaue Werte zur Leistungsaufnahme zu erzielen (vgl. Abschnitt 6.3.1.5). Bei der relativen Verlustleistung, also aufgenommene mW pro MHz, zeichnet sich eine deutliche Reduktion durch die Verwendung der 90-nm-Technologie ab. Diese fällt auf lediglich 47,9 % des Wertes für die 130-nm-Technologie. Für die 130-nm-Technologie wurde zusätzlich noch die Verlustleistung der Inter-Switch-Box-Links analysiert [115]. Hierzu wurden die relevanten Charakteristika für Leitungen auf den obersten Metalllagen sieben und acht aus dem Datenbuch verwendet. Hieraus kann fol-

gende Berechnungsvorschrift für die Leistungsaufnahme aufgestellt werden:

$$P_{\text{Inter-SB-Links}} = \text{Links} \cdot \text{Ports} \cdot f \cdot l_C \cdot 0,0517 \frac{\text{mW}}{\text{MHz} \cdot \text{m}}, \text{ mit } 93 \text{ Links, } 4 \text{ Inter-Switch-Box-Verbindungen,}$$

einer Betriebsfrequenz von 205 MHz und einer Länge  $l_C$  von 2,3 mm (vgl. Abbildung 8-7) ergibt sich eine zusätzliche Verlustleistung von 9,1 mW für alle globalen Verbindungsleitungen einer Switch-Box bei einer Schaltwahrscheinlichkeit von 50 %. Bezogen auf die Gesamtverlustleistung eines Clusters sind dies ca. 2.7 %. Dieser Anteil ist somit nicht gänzlich zu vernachlässigen, liegt er doch fast in der Größenordnung der Verlustleistungsaufnahme eines Prozessorkerns. Dies spricht ebenfalls für die in Abschnitt 6.6 vorgeschlagene Nutzung der energieeffizienten, serialisierten Datenübertragung für niederpriorre Pakete.

Abbildung 8-5 zeigt die Anteile der drei wesentlichen Hardware-Bestandteile: **Verarbeitung**, **Kommunikation** und **Speicher** der GigaNetIC-Architektur an Fläche und Verlustleistung des Gesamtsystems für die beiden betrachteten Standardzellentechnologien in 130 nm und 90 nm.



**Abbildung 8-5: Anteile der drei wesentlichen Hardware-Bestandteile der GigaNetIC-Architektur an Fläche und Verlustleistung für die 130-nm- und 90-nm-Realisierung**

Es fällt bei beiden Technologien auf, dass der Speicher das System flächenmäßig dominiert. Dies geht einher mit dem in der ITRS [2] beschriebenen Trend, immer höhere Anteile der Chipfläche mit regelmäßigen Strukturen, wie z. B. Speicher zu nutzen. Die Verwendung von derart großen Speichermengen hilft u. a., die sich ansonsten auftuende Entwurfsproduktivitätslücke zu schließen und ist für ein Chip-Multiprozessor-System essentiell. Je größer der zur Verfügung stehende Speicher, desto leichter lassen sich Latenzen klein halten, und die Performanz erhöht sich deutlich im Vergleich zum Einsatz von Off-Chip-Speicher. Bei beiden Standardzellentechnologien nehmen die Verarbeitungseinheiten ca. 10% der Gesamtfläche ein. Dies kann abhängig von Art und Anzahl der verwendeten Hardwarebeschleuniger und je nach Anwendungsszenario variieren. Die restlichen 19 % bzw. 10 % der Fläche werden für die Kommunikationsinfrastruktur benötigt. Allerdings kann hier, je nach Anforderung der Anwendung, noch ein Großteil der Fläche durch Reduktion der FIFO-Tiefe (vgl. Abbildung 4-9) und Optimierung der Switch-Box durch den Einsatz von SRAM-Technologie für die Warteräume eingespart werden.

Bei der Verlustleistungsaufnahme dominiert bei der 90-nm-Technologie der Speicher, wohingegen die Verarbeitungseinheiten nur die Hälfte der Leistungsaufnahme der Kommunikationsinfrastruktur und fast nur ein Fünftel der des Speichers benötigen. Bei der 130-nm-Standardzellentechnologie ist die Aufteilung ausgeglichener, und die Kommunikationsstruktur dominiert hier das System in punkto Verlustleistung. Dies ist zum einen auf den hohen Registeranteil und die damit verbundene relativ hohe Leistungsaufnahme dieser Komponenten in der 130-nm-Technologie zurückzuführen. Zum anderen ist die verwendete Speichertechnologie sehr ausgereift und stromsparend, so dass bei

dieser Komponente sowohl die Fläche als auch die Leistungsaufnahme deutlich besser optimiert sind als bei der 90-nm-Technologie.

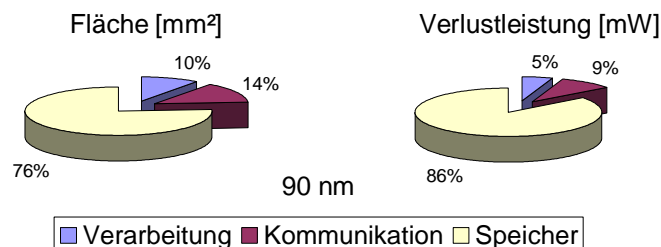
### 8.2.2 GigaNetIC-Architektur mit integrierten Multiprozessorcaches

Tabelle 8-4 gibt Aufschluss über die Flächenanforderungen, möglichen Taktfrequenzen der einzelnen Komponenten sowie die Leistungsaufnahme des bereits in Tabelle 8-3 betrachteten GigaNetIC-Systems, das allerdings im Gegensatz zu dem dort verwendeten Dual-Port-SRAM, die in Abschnitt 4.4.2 beschriebene Multiprozessor-Cache-Architektur als L1-Speicher beinhaltet [113]. Zieltechnologie dieser Variante des GigaNetIC-Chip-Multiprozessors ist die 90-nm-Standardzellentechnologie.

**Tabelle 8-4: Größenangaben des GigaNetIC-Gesamtsystems für 90 nm-Standardzellentechnologie unter Verwendung des GigaNetIC-Multiprozessorcaches**

SoC-Hauptkomponenten [90nm]	Anzahl	Fläche [mm <sup>2</sup> ]	Gesamtfläche [mm <sup>2</sup> ]	Taktfrequenz [MHz]	Leistungsaufnahme je IP-Block@250 MHz [mW]	Gesamtleistungsaufnahme @250 MHz [mW]
Caches	32	0,729	23,34	243,90	180,99	5791,68
N-Cores	32	0,127	4,07	258,00	11,74	375,68
AMBA-Master-Schnittstellen	32	0,008	0,24	354,61	1,11	35,52
AMBA-Slave-Schnittstellen	56	0,001	0,04	465,12	0,15	8,40
AMBA-Matrizen	8	0,097	0,78	265,96	25,16	201,28
Snooping-Slaves	8	0,002	0,02	420,17	0,55	4,40
Paketpuffer	8	0,952	7,62	250,00	35,50	284,00
Switch-Boxen (NoC)	8	0,575	4,60	434,78	53,00	424,00
<b>Gesamt</b>			<b>40,70</b>	<b>243,90</b>		<b>7124,96</b>

Die sich aus Tabelle 8-4 ergebende Gesamtfläche für das betrachtete Referenzsystem, das auf einem 4×2-Gitter basiert, benötigt nur 93,1 % der Fläche des in Tabelle 8-3 aufgezeigten Systems, das Dual-Port-SRAM als L1-Speicher verwendet. Allerdings ist die dem einzelnen N-Core zur Verfügung stehende lokale Speichermenge nur ein Viertel so groß, nämlich 8 KB statt 32 KB. So umfasst die Gesamtspeichermenge dieses Systems nur 0,5 MB anstatt der 1,25 MB Speicher des Vergleichssystems. Die dennoch fast vergleichbare Fläche ist in der hohen Anzahl benötigter Register, die zur Implementierung des Multiprozessorcaches notwendig sind, begründet. Die in dieser Variante derzeit erreichbare maximale Taktfrequenz beträgt 243,9 MHz; sie wird durch den kritischen Pfad des Caches bestimmt.



**Abbildung 8-6: Anteile der drei wesentlichen Hardware-Bestandteile der GigaNetIC-Architektur an Fläche und Verlustleistung bei der Multiprozessorcachevariante in 90-nm-Realisierung**

Abbildung 8-6 zeigt die Anteile der drei wesentlichen Hardware-Bestandteile der GigaNetIC-Architektur an Fläche und Verlustleistung bei der Multiprozessorcachevariante in der 90-nm-Realisierung. Die benötigte Fläche der Kommunikationsinfrastruktur bewegt sich ähnlich wie zuvor bei ca. 14 %. Die Fläche, die ohne zusätzliche Hardwarebeschleuniger von den N-Core-Verarbeitungs-



einheiten beansprucht wird, ist mit einem Zehntel nahezu identisch zu der SRAM-basierten Variante des GigaNetIC Chip-Multiprozessorsystems.

Bei der Leistungsaufnahme dominiert der Speicher mit 86 % der Gesamtverlustleistungsaufnahme noch stärker die anderen Komponenten, als es zuvor bei der SRAM-basierten Variante der Fall war. Bzgl. der Angaben zur Leistungsaufnahme ist zu erwähnen, dass es sich hier um statistische Abschätzungen seitens des Synthesewerkzeugs (50%*S*) handelt, die somit als erste Einstufung zu werten sind, so dass, bei Bedarf, die Leistungsaufnahme für dedizierte Anwendungen durch Analysen mit der GigaNetIC-Werkzeugkette (AS) deutlich genauer abgeschätzt werden kann (vgl. Abschnitt 5.3).

**Schlussbemerkung zur Synthese.** Alle vorgestellten Werte beziehen sich auf den bereits vorgestellten *Typical Case* (vgl. Abschnitt 4.2.4). Selbstverständlich wurden ebenfalls Synthesen für *Best- und Worst-Case*-Bedingungen (vgl. Tabelle 6-4) angestellt, die aus Platzgründen hier nicht näher Erwähnung finden. Anhand der Realisierung des S-Cores in der 130-nm-Infineon-Technologie haben sich die Ergebnisse der *Typical-Case*-Syntheseabschätzung verglichen mit den am Chip gemessenen Werten als realistisch herausgestellt [108], so dass sich für diese Technologie eine gute Übereinstimmung der Synthesewerte mit der Wirklichkeit zeigt. Die Synthesewerte der einzelnen Komponenten aus Tabelle 8-4 unterscheiden sich zum Teil von denen aus Tabelle 8-3. Dies und die leichten Unterschiede der veröffentlichten Zahlen zu den Implementierungen der GigaNetIC-Architektur in den betreffenden Veröffentlichungen [130][117][118][114][115][7][8][109][131][113][110] sind durch mehrere Umstände zu erklären: Zum Teil differieren die Syntheseergebnisse aufgrund der fortwährend eingeflossenen Verbesserungen und Erweiterungen bei jeder neuen Implementierung. Außerdem wurden im Laufe der Zeit Modifikationen an den verwendeten Standardzellenbibliotheken vorgenommen, was zusätzlich für abweichende Werte sorgt. Zusätzlich wurden stets die neuesten Synthesewerkzeuge eingesetzt, die ebenfalls zahlreichen Veränderungen und Optimierungen unterlagen, so dass mit den hier veröffentlichten Werten der derzeit aktuellste Stand der ASIC-Realisierung der GigaNetIC-Architektur wiedergegeben wird, der sich jedoch aufgrund der genannten Faktoren bei zukünftigen Synthesen ebenfalls von den hier genannten Zahlen unterscheiden kann.

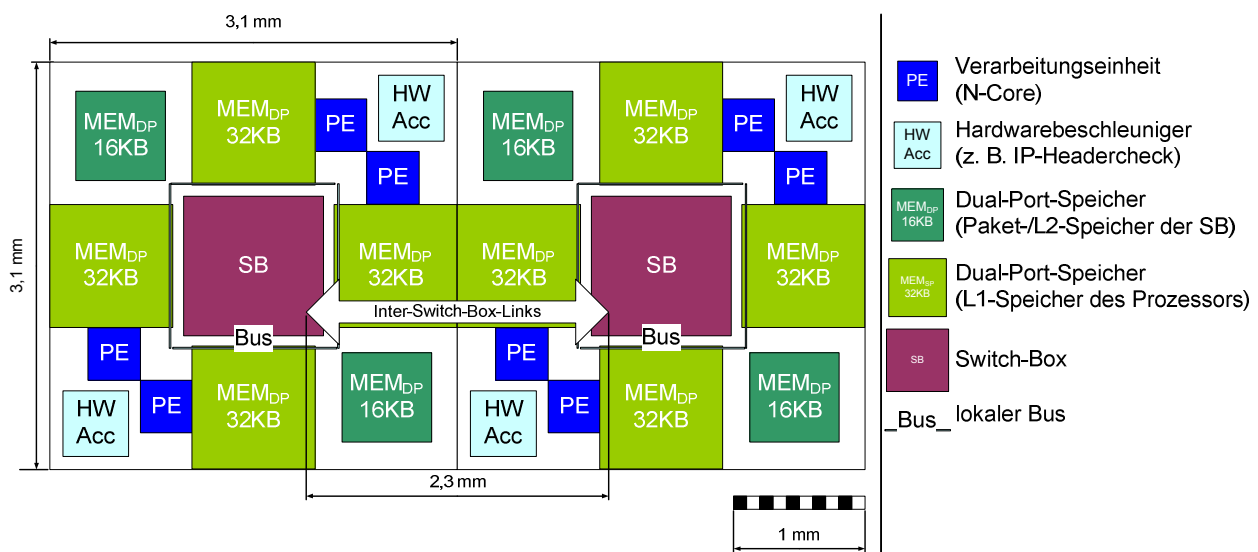
### 8.2.3 „Floorplan“ – ressourceneffiziente, kachelförmige Flächenaufteilung

Basierend auf den präsentierten Synthese Ergebnissen wird als Vorstufe für die weitere Realisierung des Chips und als Planungshilfe ein so genannter *Floorplan* entworfen. Die folgenden Abbildungen zeigen diese maßstabsgetreuen Anordnungen der einzelnen Komponenten für jeweils zwei GigaNetIC-Cluster, die als Kacheln gitterartig aneinander gereiht werden können [115]. Alle Komponenten eines GigaNetIC-Clusters werden hierzu möglichst flächeneffizient in einer quadratischen Kachel angeordnet.

Diese kachelartige Anordnungsoption der GigaNetIC-Cluster, die auch von zahlreichen anderen Implementierungen paralleler Systeme ebenso oder eingeschränkt eingesetzt wird (vgl. Abschnitt 2.8.1), birgt mehrere Vorteile:

- Aufgrund der quadratischen Form sind alle globalen, „langsamen“ Verbindungen (*Inter-Switch-Box-Links*) gleich lang und verursachen somit eine nahezu gleiche Latenz zu allen Nachbarn.

- Die regelmäßige Struktur dieser Kacheln eignet sich besonders für die Realisierung von angepassten Systemen und trägt zu den außergewöhnlich guten Eigenschaften der GigaNetIC-Architektur im Hinblick auf Skalierbarkeit (vgl. Definition 35) bei.
- Der relativ einfache Aufbau und die Regelmäßigkeit der Kacheln erlauben eine gute Wiederverwendbarkeit (vgl. Definition 36), auch im Hinblick auf eine Portierung auf eine modernere Standardzellentechnologie.
- Entwurfsfehler werden durch das Verwenden von stets gleichen, bereits ausgiebig getesteten Schaltungskonzepten reduziert.
- Testbarkeit und Fehlertoleranz können so erhöht werden (vgl. Definition 33).
- Letztendlich trägt dieser kachelartige, makroskopisch<sup>63</sup> gesehene homogene Ansatz zur **Steigerung der Ressourceneffizienz** bei.



**Abbildung 8-7: Maßstabsgetreuer Floorplan für zwei GigaNetIC-Cluster-Kacheln für die 130-nm-Standardzellentechnologie**

Abbildung 8-7 berücksichtigt hierbei die Synthesergebnisse für die 130-nm-Standardzellentechnologie, wohingegen Abbildung 8-8 die Aufteilung für die modernere 90-nm-Technologie skizziert. Eine Kachel besteht in beiden Fällen jeweils aus vier N-Cores und einer Switch-Box (mit fünf Ports und einer internen FIFO-Tiefe von drei). Als L1-Speicher kommen vier lokale SRAM-Blöcke mit je 32 KB zum Einsatz, wobei die ersten Realisierungen in 130-nm-Technologie noch *Single-Port*-Speicher vorsahen, der um 6,1 % kleiner als die *Dual-Port*-Variante in dieser Technologie ausfällt. Weiterhin werden als L2-Speicher in allen Realisierungen zwei lokale *Dual-Port*-SRAM-Paketpuffer mit je 16 KB Speichervermögen verwendet. Weiterer Bestandteil der Kachel ist das

<sup>63</sup> Die Grundstrukturen jeder Kachel sind zunächst gleich, Unterschiede können u. a. in der Art und Anzahl der integrierten Hardwarebeschleuniger, der Variante des Prozessorkerns und der Ausprägung der Kommunikationsinfrastruktur (Switch-Box und lokales Bussystem) bestehen. Selbstverständlich besteht auch die Möglichkeit, Kacheln ohne kompletten GigaNetIC-Cluster in ein bestehendes Gitter zu integrieren. Diese könnten dann z. B. an einen freien Port einer angrenzenden Switch-Box angeschlossen werden und zusätzliche Funktionalitäten (Speicher, schnelle externe Schnittstellen etc.) zur Verfügung stellen.



lokale Bussystem, wobei sowohl die AMBA- oder auch die Wishbone-Implementierung eingesetzt werden können, ohne dass der Flächenbedarf gravierend beeinflusst wird, beanspruchen beide Bussysteme doch weniger als 1 % der Gesamtfläche des Clusters. Zusätzlich sind 3,2 % der Fläche für optionale Hardwarebeschleuniger reserviert. Die freien Flächen können u. a. zur Verdrahtung und lokalen Kommunikation der Module genutzt werden. Die globale Kommunikation über die *Inter-Switch-Box-Links*, die eine ungefähre Distanz von 2,3 mm bei der 130-nm-Variante und 2,2 mm bei der 90-nm-Variante überbrücken muss, findet auf höheren Metalllagen statt (vgl. Abschnitt 6.6). Von der Kantenlänge unterscheiden sich die 130-nm- und die 90-nm-Realisierung nicht gravierend voneinander, ist die Kachel der 130-nm-Variante mit 3,1 mm doch lediglich um 10 % länger als die der 90-nm-Realisierung. Im rechten Cluster von Abbildung 8-8 sind die Ausmaße der Multiprozessorcachevariante des GigaNetIC-Systems innerhalb der der  $MEM_{DP}$ -Blöcke skizziert. Die vier, jeweils 32 KB-großen Speicherblöcke würden in diesem Fall durch die etwas kleiner ausfallenden 8 KB-großen und durch quadratische Begrenzungslinien gekennzeichneten Cache-Blöcke ersetzt. Der lokale Bus würde durch die unwesentlich größere AMBA-Switchmatrix ersetzt.

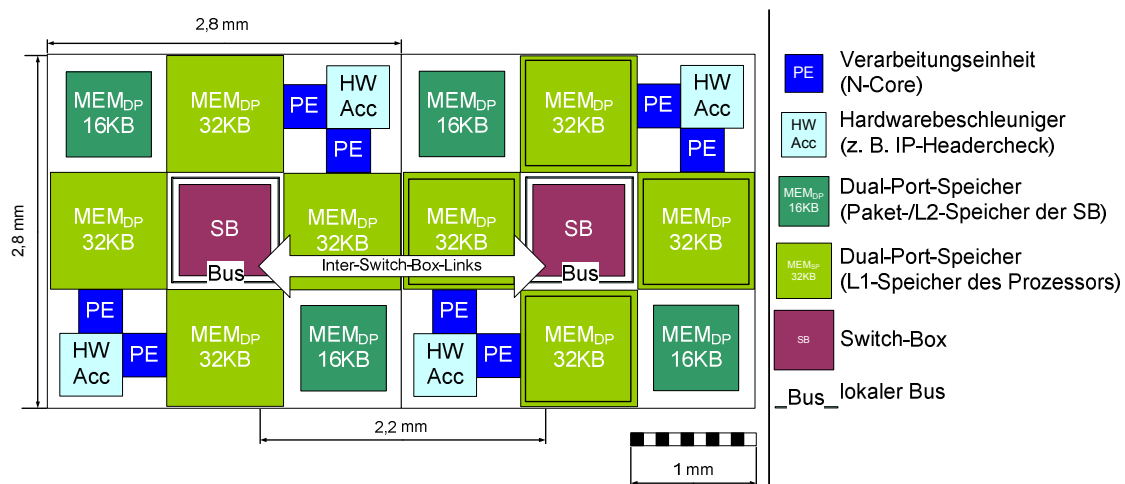


Abbildung 8-8: Maßstabsgetreuer Floorplan für zwei GigaNetIC-Cluster-Kacheln (links L1-Speicher / rechts MP-Cache, angedeutet durch Begrenzungslinien) für die 90-nm-Standardzellentechnologie

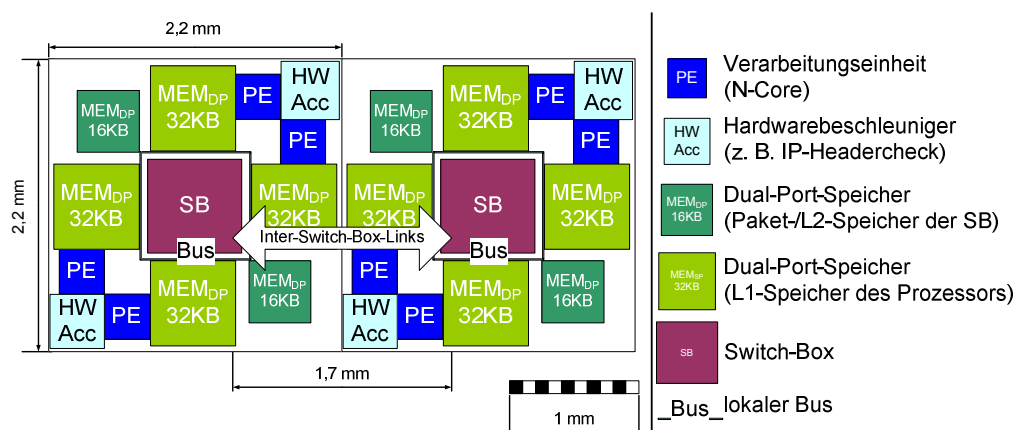


Abbildung 8-9: Maßstabsgetreuer Floorplan für zwei GigaNetIC-Cluster-Kacheln für die 90-nm-Standardzellentechnologie unter Verwendung neuerer Speicherzellen

Abbildung 8-9 zeigt die Realisierung eines GigaNetIC-Clusters in der 90-nm-Technologie unter Verwendung optimierten, um den für Speicher eher konservativen Skalierungsfaktor von  $S^{-2}$  angepassten Speichers. Die Kantenlänge reduzierte sich dann auf nur noch 2,2 mm und die Inter-Switch-Box-Links wären mit ca. 1,7 mm abzuschätzen. Bei Verwendung einer derartigen Kachel könnten

in einem 5×4-Gitter 20 Cluster und somit 80 Verarbeitungseinheiten, ähnlich wie bei der Polaris-Architektur von Intel [92] auf einen Chip integriert werden. Diese Architektur benötigte eine Fläche von weniger als 1 cm<sup>2</sup> in kachelartiger Anordnung. Ein solches System verfügte über 3 MB On-Chip-Speicher und hätte bei lediglich 285 MHz eine aggregierte Intrachip-Nettoübertragungsbandbreite von theoretisch mehr als 445 GB/s<sup>64</sup>. Das System wäre in der Lage mehr als 5,5 G Instruktionen pro Sekunde zu verarbeiten, wobei der Chip nach Tabelle 8-3 theoretisch eine Leistungsaufnahme unter Volllast von nicht einmal 4,5 W hätte. Dies deutet die Leistungsfähigkeit von parallelen und dennoch einfach gehaltenen Strukturen an, die in Abschnitt 8.3 im Vergleich zu Standardprozessoren näher diskutiert wird.

YE und DE MICHELI vertreten in [195] ebenfalls den kachelbasierten *Floorplanning*-Ansatz für Chip-Multiprozessoren aufgrund der inhärenten Homogenität. Mit REGULAY stellen sie ein Werkzeug vor, mit dessen Hilfe höherdimensionale Netzwerke effizienter als bisher auf zweidimensionale Strukturen abgebildet werden können, und zwar mit dem Resultat einer deutlich kürzeren Gesamtverdrahtungslänge der Netzwerkkanten. Eine solche Methode wäre auch für ein GigaNetIC-basiertes, höherdimensionales Netzwerk (vgl. Abschnitt 2.3.1), wie z. B. *Cube-Connected-Cycles*, oder dreidimensionale Gitter oder auch für Switch-Boxen mit höherem Ausgangsgrad anwendbar und könnte ggf. so die Ressourceneffizienz zusätzlich erhöhen. Bei eher geringen Taktfrequenzen von unter oder nur wenigen 100 MHz könnten sich so durchaus auch Chip-Multiprozessoren basierend auf der GigaNetIC-Architektur mit höher-dimensionalen Netzwerktopologien realisieren lassen. Für weniger komplexe Netzwerkstrukturen hingegen ist die GigaNetIC-Architektur aufgrund der Konzeption der quadratischen Kachelstruktur bereits für optimale, gleichmäßige Leitungslängen ausgelegt. Basierend auf diesen geometrischen Anordnungen kann später ein Makroblock erzeugt werden, der, wie eine Kachel an die nächste, auf einfache Weise horizontal wie vertikal aneinanderzureihen ist. Dies ist besonders für die Skalierbarkeit sowie die Wiederverwendbarkeit von Vorteil. Das finale *Place&Route* des gesamten Chips, also das Platzieren und Verdrahten sowie die *Clock-tree*-Synthese und die sich anschließende *Post-Layout-Simulation* stehen noch aus und erfolgen im Rahmen der bereits erwähnten Folgeprojekte des GigaNetIC-Projekts.

### 8.3 Bewertung der Ressourceneffizienz

Ressourceneffizienz ist immer in Bezug auf die Randbedingungen und das jeweilige Anwendungsszenario zu sehen (vgl. Definition 14). In diesem Abschnitt werden exemplarische Szenarien untersucht sowie Varianten der ASIC-basierten GigaNetIC-Architektur, die im vorigen Abschnitt näher vorgestellt worden sind, um diese mit aktuellen Universalprozessoren in Bezug auf Leistungsfähigkeit und Ressourceneffizienz zu vergleichen [131]. Dies geschieht für Anwendungen aus dem Desktopbereich ebenso wie für spezielle Algorithmen aus dem in Kapitel 7 vorgestellten Netzwerkbereich.

---

<sup>64</sup> Dieser Wert resultiert aus 20 Clustern mit je einer Switch-Box, in der Konfiguration mit je fünf Ports und jeweils 64 Datenbit bidirektional, betrieben mit 285 MHz.

### 8.3.1 Einheitliche, werkzeuggestützte Performanzbewertung

Um eine für die Desktop-CPU's möglichst genaue Messung der Leistungsfähigkeit zu erzielen, wurde eine speziell den Anforderungen der Messungen angepasste *Knoppix Live CD*<sup>65</sup> erstellt, die für alle Testkandidaten gleiche Voraussetzungen bzgl. des Betriebssystems und der Compiler-Werkzeuge<sup>66</sup> sowie deren Optionen garantiert. Die Messungen wurden mit *Root*-Rechten der höchsten Priorität durchgeführt, und zur Auswertung der Leistung die im Prozessor integrierten Performanzregister ausgelesen, so dass keine Interferenzen seitens systemeigener Prozesse zu erwarten waren. Zusätzlich wurden die Messungen jeweils 1000fach iteriert, um etwaige Störungen herauszufiltern. Das jeweils beste Ergebnis, im Sinne minimaler Taktzahl, wurde verwendet, was so zusätzlich den Cache der jeweiligen CPU als Systemgröße in die Bewertung einfließen ließ. Die gesamte Kompilierung, das Ausführen der erzeugten Programme, deren Laufzeitauswertung und die Protokollierung geschehen vollkommen automatisch, da skriptbasiert, so dass mögliche Fehlbedienungen bzw. Messwertverfälschungen nahezu ausgeschlossen sind. Die hierfür erstellte Werkzeugkette basiert ebenfalls auf dem in Abschnitt 5.4 vorgestelltem MultiSim. Als Endergebnis werden die Werte in eine Tabellenkalkulation zur weiteren Auswertung exportiert<sup>67</sup>.

Tabelle 8-5 fasst die Hauptcharakteristika der untersuchten Prozessoren zusammen. Zu beachten ist, dass bei dieser Analyse die GigaNetIC-Architekturvarianten über keinen Cache verfügen, sondern in der Wishbone-Bus-basierten Realisierung (vgl. Abbildung 4-20) untersucht wurden. Allerdings kann der lokale Speicher der N-Cores als schneller L1-Cache verstanden werden und wird deshalb in Tabelle 8-5 als solcher gezählt. Alle im Folgenden ermittelten Werte beziehen sich auf eine maximale Taktfrequenz des GigaNetIC-Systems von 250 MHz. Die Werte eines GigaNetIC-Clusters beziehen sich im Folgenden auf eine Realisierung ohne Ethernetports, da diese Schnittstellen als Off-Chip-Schnittstellen angesehen werden. Dies führt zu einer Fläche von ca. 5,3 mm<sup>2</sup> für einen auf vier N-Cores basierenden Cluster, vgl. Abschnitt 8.2. 20 dieser Cluster, in einem Gitter angeordnet (ähnlich Abbildung 4-2), benötigen somit zunächst 106 mm<sup>2</sup>, was der gemittelten Chipfläche der Vergleichsprozessor entspricht und so als flächenäquivalente Alternative zu den betrachteten Desktop-CPU's angesetzt wird. Setzt man allerdings den zuvor dargestellten *Floorplan* voraus, so ließen sich zunächst nur dreizehn anstatt der genannten 20 GigaNetIC-Cluster auf einer Fläche von 101,9 mm<sup>2</sup> integrieren (vgl. Abbildung 8-8). Füllte man allerdings auf Basis neuer skalierten Speicherblöcke mit Hilfe der GigaNetIC-Clusterkacheln bei einer Kantenlänge von nur noch 2,2 mm pro Cluster (vgl. Abbildung 8-9) diese Fläche von 106 mm<sup>2</sup>, so käme man auf über 21 Cluster. Dies lässt die hier getroffene Annahme von 20 gitterartig angeordneten GigaNetIC-Clustern für eine derartige Fläche durchaus realistisch erscheinen.

---

<sup>65</sup> Es wurde als Vorlage die Knoppix 5.0 Boot-CD mit dem 2.6.17 Kernel verwendet.

<sup>66</sup> Als Compiler-Werkzeugkette wurde der GCC 4.0.4 eingesetzt. Die möglichen *Kompilierungs*-Optionen wurden automatisiert eingesetzt und die jeweils besten Resultate als Messwert verwendet. Hierzu wurden sowohl die Optimierungsstufen als auch die speziellen Optimierungen für die unterschiedlichen Architekturen berücksichtigt.

<sup>67</sup> Der gesamte Ablauf, die ausführbaren Kompilate, die *Object*-Dateien und Testergebnisse sowie deren Zusammenfassung und die finale Aufbereitung aller Messungen als *CSV*-Datei wurden auf einen jeweils zusätzlich verwendeten *USB-Stick* als beschreibbares Medium während des automatisierten Ablaufs abgespeichert.

Unter der Annahme, dass die Prozessoren bei der Bearbeitung der Benchmarks nahezu voll ausgelastet sind, wurde als Verlustleistung der *Thermal-Design-Power*(TDP)-Wert angenommen. Dies ist sicherlich eine sehr konservative Abschätzung der Systeme in Bezug auf die Verlustleistungsaufnahme, zumal große Teile der jeweiligen Systeme, wie z. B. Gleitkommaeinheiten oder die Switch-Box-basierte Kommunikationsinfrastruktur nicht für alle Tests relevant sind. 90 % der Chipfläche des Systems mit 20 N-Core-Clustern wird von Speicherzellen belegt, die die überproportional hohe Anzahl an Transistoräquivalenten, im Vergleich zu den anderen CPUs, erklärt. Die Verlustleistungsangabe für das 20 N-Core-Cluster umfassende GigaNetIC-System von 14,88 W bei 250 MHz ist unter Berücksichtigung der in Tabelle 8-3 ermittelten Werte ebenfalls als eher konservativ einzustufen, käme man doch unter Verwendung der Synthesergebnisse auf eine Verlustleistung von lediglich 3,96 W bei 250 MHz.

**Tabelle 8-5: Wesentliche Merkmale der analysierten Prozessoren**

Prozessor	Kerne	Name	FSB [MHz]	L1 Cache [kB]	L2 Cache [kB]	Taktfrequenz [GHz]	Technologie [nm]	Spannung [V]	TDP [W]	Die-Größe [mm <sup>2</sup> ]	Transistoren [Mil.]
Pentium 4 3000	1	Northwood	200	8	512	3,00	130	1,550	110	146	125
Intel Pentium M 1600	1	Banias	100	16	1024	1,60	130	1,485	24	100	77
Intel Pentium M 1700	1	Banias	100	16	1024	1,70	130	1,485	25	100	77
Intel Pentium M 2100	1	Dothan	100	64	2048	2,10	90	1,340	21	84	140
Intel Duo Core T2400	2	Yonah	166	64	2048	1,83	65	1,325	31	91	151
Intel Duo Core T2500	2	Yonah	166	64	2048	2,00	65	1,325	31	91	151
AMD Athlon 64 3000+	1	Venice	200	128	512	1,80	90	1,400	89	83	69
AMD Athlon 64 3200+	1	NewCastle	200	128	512	2,19	130	1,500	89	144	69
AMD Athlon 64 3700+	1	San Diego	200	128	1024	2,20	90	1,400	89	115	105
N-Core	1	GigaNetIC	250	32	0	0,25	90	1,200	0,05	0,96	2
N-Core-Cluster (4PEs)	4	GigaNetIC	250	128	0	0,25	90	1,200	0,74	5,30	10
20 N-Core-Cluster	80	GigaNetIC	250	2560	0	0,25	90	1,200	14,88	106,05	200

### 8.3.2 Universalbenchmarks zur Bewertung der GigaNetIC-Architektur

Die Ergebnisse der Analysen der Universalbenchmarks werden in Abbildung 8-10 dargestellt. Ziel des Vergleichs der einzelnen Systeme war nicht, den verwendeten Benchmark-Code jeweils auf Assemblerebene zu optimieren, sondern vielmehr die Leistungsfähigkeit der gesamten Architektur zu bewerten, wozu auch die Softwareumgebung nebst Übersetzer zu zählen ist (vgl. Definition 3). Deshalb wurde auf manuelle Optimierung bewusst verzichtet, zumal dies im realen Einsatz nur bedingt und mit relativ hohem Zeitaufwand durchgeführt werden könnte. Als Universalanwendungen kamen der zwar schon etwas in die Jahre gekommene aber trotzdem noch heute oft für eingebettete Systeme verwendete Dhrystone-Benchmark zur Beurteilung der Integer-Performanz und zwei Sortieralgorithmen<sup>68</sup> der *Stanford-Benchmark-Sammlung* zum Einsatz.

Als Essenz der gewonnenen Ergebnisse lässt sich für diese Anwendungen, die nicht aus dem Netzwerkbereich stammen, sagen, dass die Leistungsfähigkeit eng an die Taktfrequenz der Systeme gekoppelt ist. Die durchschnittliche Taktzahl, die von den Universalprozessoren für einen Dhrystone-durchgang benötigt wird, liegt bei 703 Takten, wohingegen die deutlich einfacher gehaltene N-Core-Architektur 1119 Takte benötigt, also 59 % mehr Taktzyklen. In Anbetracht der zusätzlich sieben- bis zehnfach höheren Taktfrequenzen der Universalprozessoren sind deren Performanzvorsprünge von 10,3 bis 16,3 sofort nachvollziehbar. Betrachtet man hingegen die Sortieralgorithmen,

<sup>68</sup> Hierbei wurden zum einen der *Quicksort*- und zum anderen der *Bubblesort*-Algorithmus verwendet, wobei 200 Integerwerte sortiert werden mussten. Die Anzahl der zu sortierenden Elemente wurde bewusst relativ gering gehalten, da die GigaNetIC-Analysen mit Hilfe der sehr langsamen HDL-Simulation durchgeführt wurden.

so kann die N-Core-Architektur den relativen Vergleich für sich entscheiden. Der N-Core benötigt ca. 509 Tausend Takte bei der Verwendung des *Bubblesort*-Algorithmus für die Sortieraufgabe und 29 Tausend Takte beim *Quicksort*-Verfahren, wohingegen der Durchschnitt der Universalprozessoren 662 Tausend bzw. 41 tausend Takte aufwenden muss. In diesem Fall benötigt die einfachere RISC-Architektur zwischen 30 % und 40 % weniger Taktzyklen. Hieraus resultiert der geringere Performanzvorteil der Universalprozessoren von 4,5 bis 11,6.

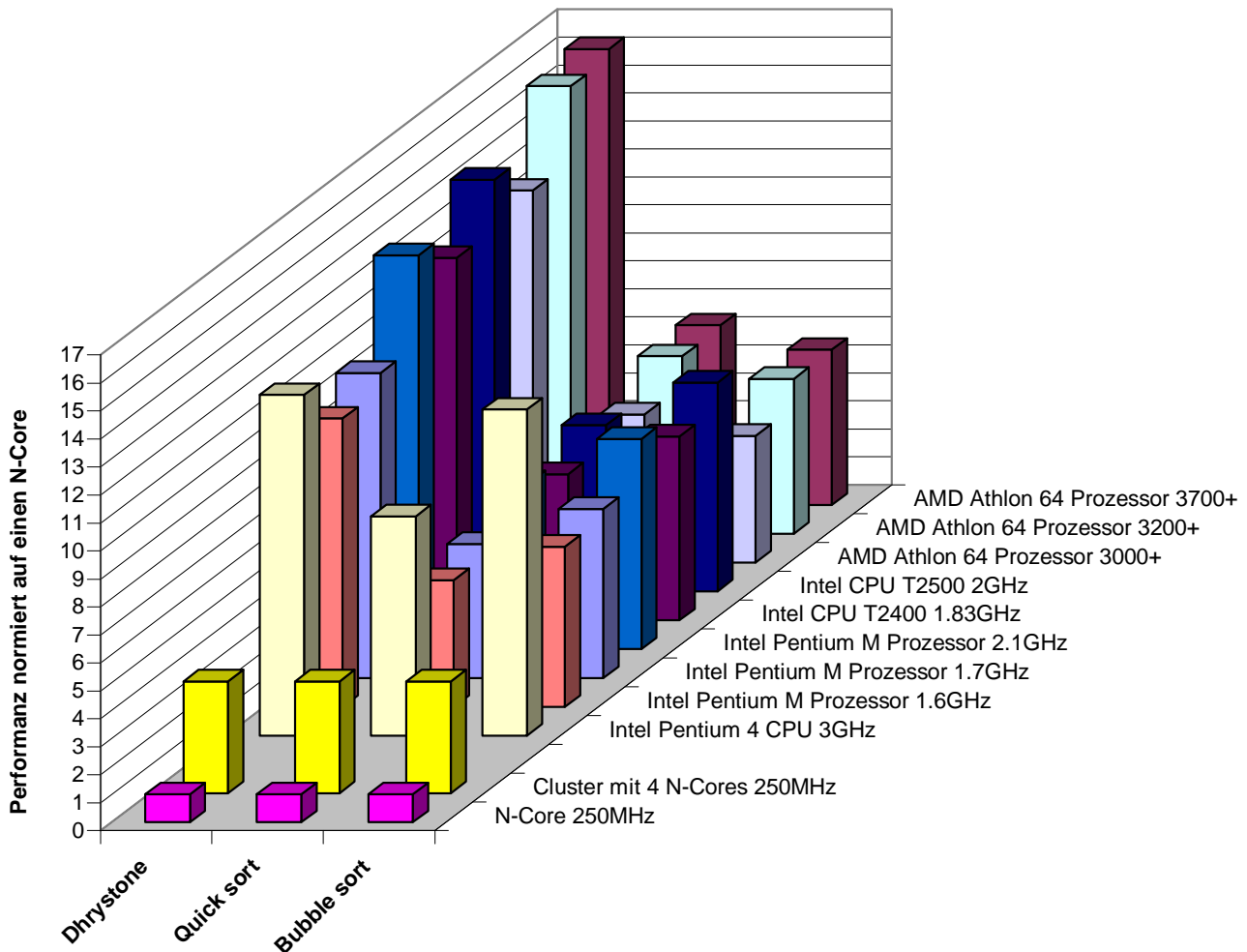


Abbildung 8-10: Leistungsvergleich zwischen Universal-CPU's und der GigaNetIC-Architektur für einfache Anwendungsszenarien

### 8.3.3 Netzwerkbenchmark zur Bewertung der GigaNetIC-Architektur

Im Folgenden wird der Bereich der Netzwerkverarbeitung, der bereits in Kapitel 7 ausführlich betrachtet worden ist, nochmals in Bezug auf die Ressourceneffizienz untersucht. Hierbei kommt die inhärente Parallelität der Netzwerkdaten der GigaNetIC-Architektur mit den zuschaltbaren Hardwarebeschleunigern zu Gute.

In Abbildung 8-11 wird die Leistungsfähigkeit des in dieser Arbeit entwickelten Systems in Bezug auf dieses Anwendungsszenario dokumentiert. Der zugrunde liegende Benchmark bestand aus der Verarbeitung von einer Million IP-Pakete nach *iMix*-Verteilung (vgl. Abschnitt 7.2.3). Das Diagramm zeigt als resultierende Hochrechnung die Anzahl der verarbeitbaren Pakete pro Sekunde. Die Universalprozessoren sind um den Faktor 1,7 bis 8,1 mal schneller als ein einzelner N-Core. Ein Cluster mit vier N-Cores ermöglicht einen Geschwindigkeitsvorteil gegenüber dem einzelnen N-

Core von 3,83. Eine Beschleunigung von vier ist aufgrund von Arbitrierungslatenzen und möglicher Busblockaden nicht realistisch. Unter Hinzunahme des in Abschnitt 6.3.1 vorgestellten Hardwarebeschleunigers übertrifft bereits ein GigaNetIC-Cluster fast alle Universalprozessoren. Es wird sowohl die Kopplung des Beschleunigers am lokalen Bus (*WB HW Acc*) als auch am Port einer benachbarten Switch-Box<sup>69</sup> (*CC HW Acc*) berücksichtigt.

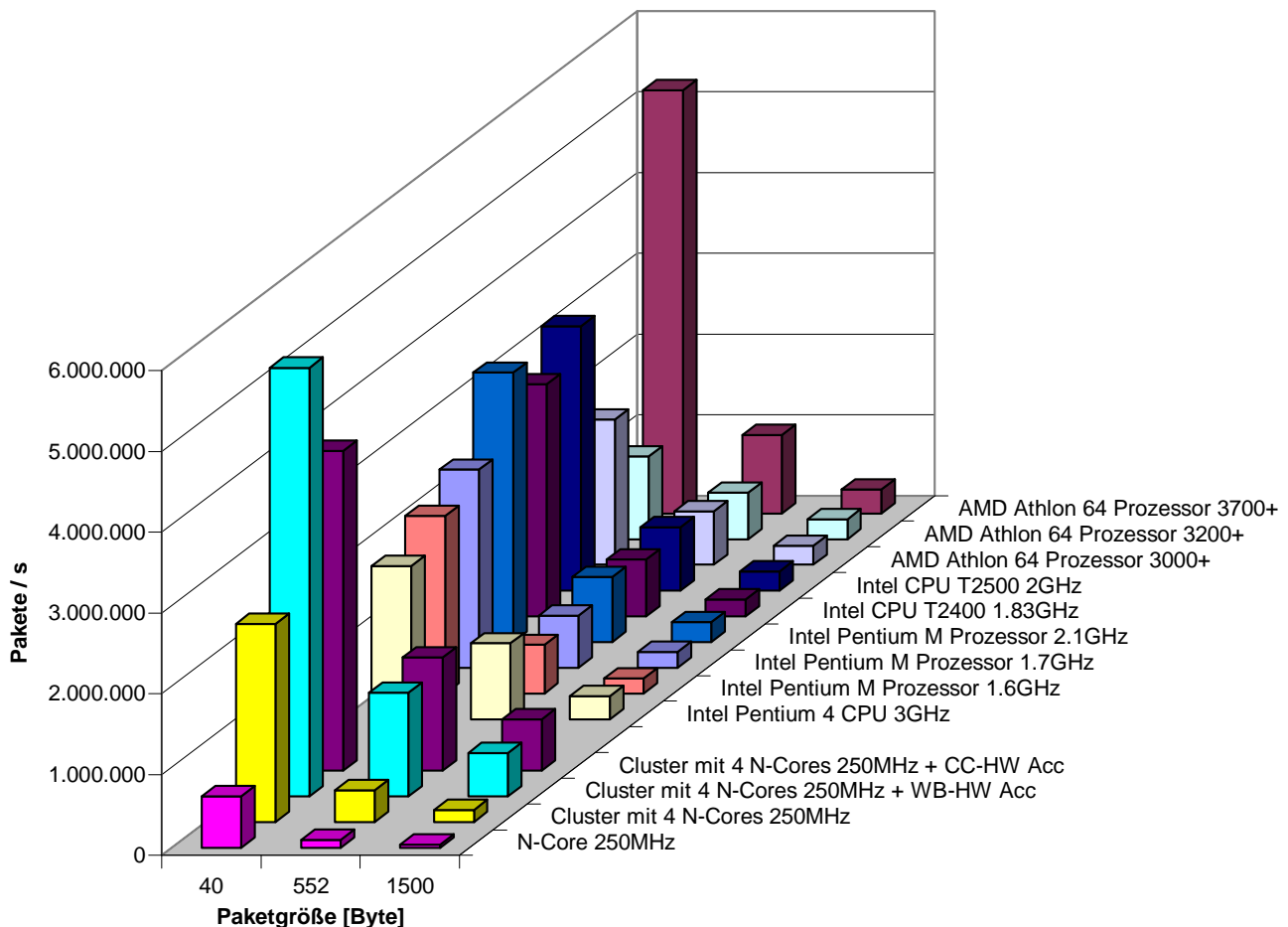


Abbildung 8-11: Leistungsvergleich zwischen Universal-CPU und der GigaNetIC-Architektur für eine Paketverarbeitungsanwendung

Abbildung 8-12 führt zwei neue Dimensionen des Entwurfsraums in die Betrachtung ein: den **Energiebedarf** (im Hinblick auf den entsprechenden Benchmark) und die **Chipgröße** der Systeme, die durch den Flächeninhalt der Blasen symbolisiert wird. Das flächenmäßig kleinste System ist der einzelne N-Core (J), das zudem die geringste Energieaufnahme für die Abarbeitung des Benchmarks aufweist, allerdings ist die benötigte Zeit auch deutlich höher als die der Universalprozessoren. Die Universalprozessoren (A-I) bilden eine Menge in der rechten Hälfte des Diagramms bzw. Entwurfsraums, was einer Bearbeitungszeit von ca. 1 s (0,7 bis 1,48 s) bei einer Energieaufnahme von 19 bis 132 Ws entspricht. Nahezu die gleiche Zeit (1,6 s) benötigt der normale N-Core-Cluster (K), wobei sein Flächenbedarf nur ein 20stel der durchschnittlichen Universalprozessorfläche aus-

<sup>69</sup> Den Messungen liegt eine Beschleunigerkopplung an eine Switch-Box zugrunde, die innerhalb einer Distanz von einem Hop liegt, so dass der Beschleuniger nicht unmittelbar an die Switch-Box des Clusters angeschlossen werden muss. Die Taktzahlen für Kommunikation und Berechnungen wurden in Abschnitt 7.7 näher diskutiert.

macht. Sein Energiebedarf liegt bei 1,3 Ws, wohingegen der durch den am lokalen Bus angeschlossenen Hardwarebeschleuniger erweiterte Cluster weniger als 0,5 s zur Bearbeitung der gestellten Aufgabe benötigt und zudem noch einen deutlich geringeren Energiebedarf aufweist (0,4 Ws).

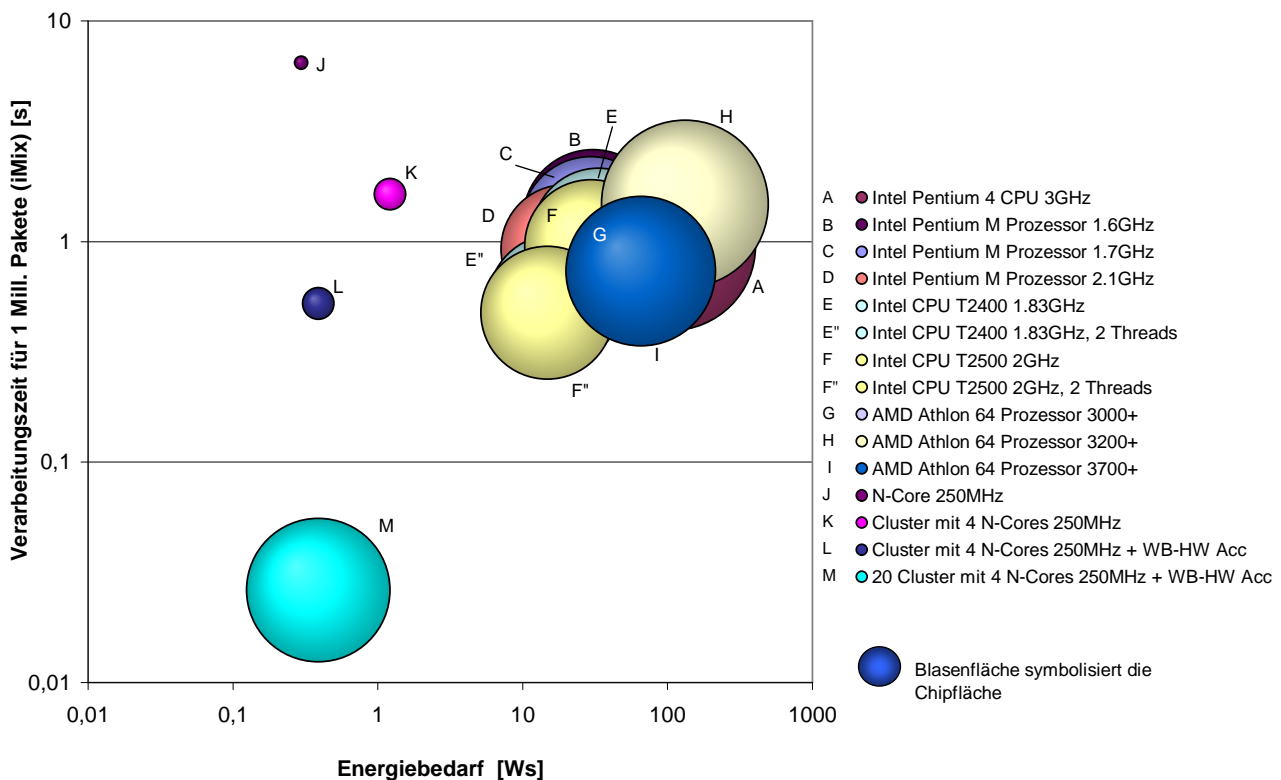


Abbildung 8-12: Ressourceneffizienz und Entwurfsraumvisualisierung von Universalprozessoren verglichen mit GigaNetIC-Systemen in Bezug auf das Paketverarbeitungsszenario (*iMix*)

Um die Mehrkernprozessoren der Intel-Core-Architektur entsprechend ausnutzen zu können, wurde zusätzlich eine *Thread*-basierte Variante des Paketverarbeitungsbenchmarks realisiert (E'', F''). Hier konnte nahezu eine Verdopplung der Verarbeitungsgeschwindigkeit gemessen werden (0,54 s und 0,48 s) bei einem Energiebedarf von 16,6 Ws bzw. 14,8 Ws. Bei einer Erhöhung der *Threadanzahl* über zwei konnte darüber hinaus eine Reduktion der Leistungsfähigkeit dieser beiden Prozessoren bei dem betrachteten Anwendungsszenario festgestellt werden. Dies liegt in der rechenintensiven Aufgabe begründet, die Speicherlatenzen nahezu ausblendet und daher mehr als zwei Threads als ineffektiv herausstellt. Das gleiche Verhalten konnte bei der *Hyperthreading*-Architektur von Intel (A) beobachtet werden, bei der viele *Threads* um die beschränkte Anzahl der exzessiv genutzten Funktionseinheiten konkurrieren müssen und so Geschwindigkeit eingebüßt wird.

Das GigaNetIC-Multiprozessorsystem, bestehend aus 20 N-Core-Clustern demonstriert, beeindruckend die Ressourceneffizienz dieser Architektur insbesondere am Beispiel dieses Paketverarbeitungsszenarios, bei der die massiv parallele Architektur besonders von der inhärenten Parallelität der Aufgabe profitiert. Mit einer durchschnittlichen Chipfläche einer Desktop-CPU ist es möglich, die Aufgabe innerhalb von 0,026 s, also 28mal schneller als die schnellste Desktop-CPU, zu erledigen und dies bei einem Energiebedarf von nur 0,4 W.

Sicherlich handelt es sich bei dem vorgestellten Benchmark um ein sehr spezielles Szenario, von dem nicht auf allgemeine Anwendungen geschlossen werden darf, die z. B. Fließpunktberechnungen beinhalten oder größere Speicheranforderungen stellen. Sehr wohl aber zeigen die gewonnenen Zahlen das Potential der GigaNetIC-Architektur in Bezug auf den immer wichtiger werdenden As-

pekt der Ressourceneffizienz auf. Die einfache Erweiterbarkeit aufgrund definierter Schnittstellen und Protokolle sowie die Optimierungsmöglichkeiten der GigaNetIC-Systemarchitektur (vgl. Kapitel 6) erlauben die Konzeption eines angepassten parallelen Systems. Für ausgewählte Anwendungsgebiete zeigt sich dieser Ansatz im Hinblick auf eine möglichst hohe Ressourceneffizienz vielversprechender als *Single-Core*-Architekturen, bei denen im Wesentlichen durch Erhöhung der Taktfrequenz steigenden Anforderungen Rechnung getragen wird.

Abschließend lässt sich feststellen, dass sich sowohl mit Hilfe des GigaNoC-On-Chip-Netzwerks und der damit verbundenen einfachen Kopplung anwendungsspezifischer Hardwarebeschleuniger, als auch durch massiv parallele Verwendung relativ einfach gehaltener RISC-Prozessoren wie dem N-Core sowie durch die Kombination dieser Maßnahmen beachtliche Resultate für gegebene Anwendungen erreichen lassen. Sowohl im Hinblick auf die Leistungsfähigkeit als auch auf den Energiebedarf kann durch Parallelität und durch die vielen Möglichkeiten der Systemerweiterung eine hohe Ressourceneffizienz erreicht werden. Die gute Skalierbarkeit der Architektur ist ein weiterer Pluspunkt der GigaNetIC-Architektur, ermöglicht sie doch einen effizienten Einsatz in unterschiedlichsten Bereichen des Entwurfsraums.

## 8.4 Zukünftige Architekturen

Bereits derzeitige Prozessorarchitekturen heutiger Arbeitsplatzrechner sowie die jüngsten Veröffentlichungen zu Aktivitäten wie der Polaris-Architektur und dem *Tera-Scale*-Projekt von Intel (vgl. Abschnitt 2.8.1) zeigen, dass die nahe Zukunft den Parallelprozessoren gehört. Im Laufe der nächsten zehn Jahre wird die Zahl der integrierten Prozessorkerne stetig zunehmen und Architekturen wie die GigaNetIC-Architektur werden zum Standard gehören, vgl. [2]. Die Art und Anzahl der Verarbeitungseinheiten pro Chip wird dann hauptsächlich vom jeweiligen Einsatzgebiet bestimmt werden. Deshalb werden sich skalierbare Architekturen mit einheitlichem Programmiermodell nicht nur aufgrund der fertigungstechnischen Vorteile (vgl. Kapitel 4 und Abschnitt 8.2) und der damit verbundenen Kostenvorteile besonders stark hervortun. Der Aspekt der Ressourceneffizienz wird zunehmend an Bedeutung gewinnen. Die *NRE*-Kosten für komplexe ASICs im Strukturgrößenbereich von wenigen Nanometern steigen kontinuierlich. Komplexe Systeme erfordern nicht zuletzt deshalb eine möglichst vollständige Verifikation, formal und prototypisch, um die Erfolgchancen einer fehlerfreien Realisierung zu maximieren. Die steigende Mobilität der Anwendungen erhöht stetig die Anforderungen an geringstem Energiebedarf. Gründe wie Laufzeitmaximierung mobiler Geräte auf der einen Seite sowie die Wärmeentwicklung und die damit verbundenen Probleme komplexer Systeme auf der anderen Seite erfordern ebenfalls die Forcierung der Entwicklung besonders ressourceneffizienter Architekturen speziell im Hinblick auf die Leistungsaufnahme.

Wie bereits vorgestellt, wird die Entwurfsproduktivitätslücke nur von annähernd regelmäßigen Strukturen und großen Speicherblöcken zu schließen sein. Immer kleiner werdende Strukturgrößen stellen immer mehr Funktionalität pro Fläche zur Verfügung, dies allerdings, begründet durch die Laufzeiten und die schwierige Synchronisierung, auf einer vergleichsweise stark eingeschränkten Fläche. Kleinere Kacheln können so lokal immer höhere Taktfrequenzen erreichen. Komplexere, flächenintensivere Hardwareblöcke hingegen werden aufgrund der relativ gesehen größer werdenden Signallaufzeiten im Vergleich zu den Schaltzeiten der Transistoren deutlich an Leistungsfähigkeit einbüßen. Aufgrund der größer werdenden Diskrepanz zwischen stetig steigender Schaltgeschwindigkeit der Logiktransistoren auf der einen Seite und der im Vergleich dazu geringen Ge-



schwindigkeitszuwächse für globale Verbindungsleitungen auf der anderen Seite werden in Zukunft neue Ansätze zur Realisierung von hochperformanten On-Chip-Verbindungen notwendig. In diesem Zusammenhang könnten sich optische Signalführungen für Intrachipleitungen als sehr leistungsfähige Alternative zu den bisher verwendeten Metallleiterbahnen erweisen.

Vielleicht werden neuartige Schaltungstechniken in der Lage sein, diese Probleme, getreu dem MOORESchen Gesetz, aufzulösen oder weiter in Richtung Zukunft zu verschieben. Sicherlich eröffnen diese „Nano-Technologien“ auch neue Perspektiven und Einsatzgebiete für leistungsfähige FPGA-Strukturen. Rekonfigurierbarkeit könnte von der hohen Packungsdichte speziell bei den Speicherzellen profitieren. Die Parallelität könnte so zusätzlich durch hohe Flexibilität ergänzt werden und die Entwicklung von komplexen, dynamisch rekonfigurierbaren Bausteinen als Alternative zu Universalprozessoren fördern.

## 8.5 Zusammenfassung

In diesem Kapitel wurde die prototypische Realisierung der in Kapitel 4 beschriebenen GigaNetIC-Architektur vorgestellt. Nach Durchlaufen aller in Kapitel 5 vorgestellten Entwurfsschritte zur Verifikation und Optimierung wurde die GigaNetIC-Architektur als FPGA-Prototyp und in zwei aktuellen Standardzellentechnologien implementiert.

Basierend auf den Ergebnissen der FPGA-Realisierung kann im Folgenden sehr schnell ein vorlagenbasiertes RAPTOR2000-Tochtermodul für den noch zu realisierenden GigaNetIC-ASIC erstellt werden. Das hier erworbene Wissen um die Systemintegration kann in der Folge genutzt werden, um den ASIC in einer bereits erstellten und erprobten Umgebung schnell und komfortabel testen zu können. Dieses Vorgehen kann einem potentiellen Industriepartner die Einführung eines neuen Produktes erleichtern und hilft so die *Time-To-Market*-Spanne deutlich zu verkürzen.

Der auf dem FPGA-Prototypen basierende GigaNetIC-Demonstrator konnte erfolgreich einem breitem Publikum auf der Cebit 2005 und der Hannover Messe 2005 auf einem fachgebietseigenen Messestand im Rahmen des Bereichs „Forschungsland NRW“ präsentiert werden. Darüber hinaus hilft er, das System in einer realen Umgebung detailliert und zugleich um Größenordnungen schneller, als es andere Formen der Simulation ermöglichen, zu verifizieren.

Im Anschluss an diese positiv verlaufene Verifikation wurde die Architektur auf zwei aktuelle CMOS-basierte Standardzellentechnologien abgebildet und die Ergebnisse detailliert analysiert. Basierend auf den gewonnenen Syntheseergebnissen konnten fundierte Aussagen über die zu erwartenden Flächen- und Leistungsanforderungen und die realisierbaren Taktfrequenzen der Einzelkomponenten und des Gesamtsystems getroffen werden. Ein exemplarisches System mit acht Clustern zu je vier N-Core-Prozessoren und insgesamt 1,25 MB SRAM benötigt weniger als 44 mm<sup>2</sup> in der verwendeten 90-nm-Technologie. Die maximale Betriebsfrequenz liegt bei 285 MHz. Im Rahmen dieser Implementierung wurde ein Konzept der ressourceneffizienten Anordnung der einzelnen Komponenten eines GigaNetIC-Clusters in Form quadratischer Kacheln vorgestellt. Auf Basis dieser Kacheln ist es möglich, GigaNetIC-Systeme effizient zu skalieren und identische Signallaufzeiten zu gewähren, was für eine Maximierung der Performanz essentiell ist.

Es konnte das große Potential der skalierbaren GigaNetIC-Architektur in Bezug auf Leistungsfähigkeit und insbesondere bezogen auf ihre Ressourceneffizienz aufgezeigt werden. Beim Einsatz eines massiv parallelen GigaNetIC-Systems, bestehend aus 20 Clustern mit je vier N-Cores und

zusätzlichen Hardwarebeschleunigern und mit der Fläche einer durchschnittlichen Desktop-CPU, konnte ein Leistungsvorsprung um zwei Größenordnungen mit einhergehendem, zusätzlich um zwei Größenordnungen geringerem Energiebedarf im Vergleich mit den derzeitigen Universalprozessoren festgestellt werden. Sowohl im Hinblick auf Universalanwendungen und im Besonderen bei Netzwerkanwendungen konnte das hier entwickelte Chip-Multiprozessorsystem seine Leistungsfähigkeit unter Beweis stellen. Anhand der hier angestellten Untersuchungen lassen sich gute Zukunftschancen für Architekturen wie die der GigaNetIC-Architektur prognostizieren. Derartige Architekturen sind zum einen in der Lage, die Vorteile, die durch die stetigen Strukturverkleinerungen der Halbleiterprozesse entstehen, zu nutzen. Zum anderen zeigen solche Architekturen Möglichkeiten auf, die sich ergebenden Nachteile zu kompensieren.

## 9 Zusammenfassung und Ausblick

Aktuelle Forschungsarbeiten von Intel zeigen, dass die Rechenleistung, die vor zehn Jahren noch dem schnellsten Supercomputer, der ein Einfamilienhaus hätte füllen können, vorbehalten war, mittlerweile von einem einzigen Halbleiterbaustein bereitgestellt werden kann. Dies geht einher mit einer nahezu um vier Größenordnungen kleineren Verlustleistungsaufnahme des Chip-Multiprozessors (*CMP*). Dieser Technologiesprung wird zum einen durch die stetig verbesserten Herstellungsverfahren der Halbleiterindustrie und zum anderen durch die Ausnutzung von massiv paralleler Verarbeitung in integrierten Schaltkreisen ermöglicht.

Zur Einordnung der hardwarebezogenen Themenbereiche dieser Arbeit wurden grundlegende Abschätzungen zur Leistungssteigerung durch paralleles Rechnen und die damit verbundenen Anforderungen an die Systeme aufgezeigt. Es wurden elementare Grundlagen zu den Kernkomponenten eingebetteter Parallelrechner vorgestellt: *On-Chip-Netzwerke*, *eingebettete Verarbeitungseinheiten*, *Speicherhierarchien* sowie deren *Anwendungen*. Unterschiede und Gemeinsamkeiten existierender Ansätze im Hinblick auf die GigaNetIC-Architektur wurden herausgearbeitet und charakterisieren so die Besonderheiten der von mir entworfenen Systemarchitektur.

**Analytische Modellierung.** Im Anschluss an die Definition von Ressourceneffizienz und wesentlicher Begriffe zur kostenfunktionsbasierten Analyse von Chip-Multiprozessoren wurden Formalismen zur Bewertung solcher Systemimplementierungen eingebetteter Parallelrechner und ihrer Komponenten eingeführt.

**Effiziente CMP-Architektur.** Im Rahmen dieser Arbeit wurde eine neuartige skalierbare Chip-Multiprozessor-Architektur entworfen, die aufgrund einer sehr flexibel gestalteten, parametrisierbaren Hardwarestruktur an verschiedenste Anforderungen angepasst werden kann, um so für unterschiedlichste Anwendungsszenarien eine möglichst ressourceneffiziente Lösung zu bieten. Rückgrat dieses Chip-Multiprozessorsystems bildet das eigens für diese Architektur entworfene neuartige hierarchische GigaNoC-On-Chip-Netzwerk. In Verbindung mit einem umfassenden Konzept zur Kopplung unterschiedlichster Verarbeitungseinheiten an die verschiedenen SoC(System-on-Chip)-Ebenen erlaubt es einen hohen Grad an Flexibilität und Leistungsfähigkeit. Durch die spezielle Konstruktion der On-Chip-Routingknoten der Switch-Boxen ist nicht nur eine gute Skalierbarkeit auf Chip-Ebene während des Entwurfs gegeben, sondern auch die Möglichkeit einer späteren Kombination von GigaNetIC-basierten CMPs auf Leiterplattenebene. Je nach Anwendungsgebiet und dessen Anforderungen kann zwischen normalem SRAM oder einem eigens entwickelten Multiprozessorcache als *On-Chip-Speicher* der Verarbeitungseinheiten gewählt werden.

Das GigaNetIC-CMP-System dient und diene als Basis für weitere Forschungsvorhaben der Universität Paderborn, wie z. B. für die DFG-Sonderforschungsbereiche SFB 376 „Massive Parallelität: Algorithmen, Entwurfsmethoden, Anwendungen“ und SFB 614 "Selbstoptimierende Systeme des Maschinenbaus". Es wird in den erfolgreich beantragten Folgeprojekten PlaNetS, MxMobile, EasyC oder auch DFG weiterhin genutzt und erweitert.

Im Rahmen zukünftiger Arbeiten wäre eine Flächenreduktion der Switch-Box durch Einsatz von SRAM anstelle von Registerzellen für die Warteschlangen sinnvoll. Die Implementierung einer vollwertigen *Broad-* und *Multicast*-Funktionalität würde die Möglichkeiten des On-Chip-Netzwerks für einige Anwendungsszenarios zusätzlich erhöhen. Für sehr große Systeme mit einer Vielzahl von

Clustern sollten zusätzliche Elemente zur Realisierung eines GALS(global asynchronen, lokal synchronen)-Konzepts hinzugefügt werden. Für den finalen Baustein ist der *Bootloader*, der für die Initialisierungsphase der Prozessoren verantwortlich zeichnet, noch von der SystemC-Beschreibung in eine synthetisierbare Form zu transferieren. Eine zusätzliche Erhöhung der Performanz speziell im Bereich der Netzwerkanwendungen könnte die Integration eines Hardwareblocks zur Lastverteilung (*Loadbalancer*) bedeuten, der dynamisch sowohl auf sich ändernde Lastverteilungen als auch auf die Auslastung der einzelnen Cluster und Prozessoren reagieren könnte und die Aufgaben adaptiv verteilt. Besonderes Augenmerk sollte in Zukunft auf Aspekte der Fehlertoleranz gelegt werden, die für derart komplexe Systeme wie dem GigaNetIC-Chip-Multiprozessor mit teilweise mehreren Hundertmillionen Transistoren immer wichtiger werden, erhöhen sie doch die Ausbeute (*Yield*) bei der Produktion und die Ausfallsicherheit während des Betriebs, was speziell in hochverfügbaren Netzwerkkomponenten von besonderer Bedeutung ist.

**Entwicklungsumgebung – in sich geschlossene Werkzeugkette.** Parallel zur Realisierung der GigaNetIC-Hardwarebeschreibung wurde in Kooperation mit den Projektpartnern der Universität Paderborn eine geschlossene und ineinander verzahnte Werkzeugkette entworfen: angefangen beim Prozessorentwurf über die automatische Generierung des Compilers und eines C-basierten zyklennakuraten Instruktionssatzsimulators bis hin zu rückannotierten *RTL(Register-Transfer-Level)*-Beschreibungen. Letztere liefern detaillierte Informationen über Leistungsaufnahme, Flächenbedarf und Leistungsfähigkeit der integrierten Schaltung und geben Impulse für Instruktionssatzerweiterungen und Hardwarebeschleuniger sowie für Systemoptimierungen allgemeiner Natur und helfen so die Ressourceneffizienz des Systems zu steigern. Die GigaNetIC-Architektur stellt Simulations- sowie Emulationsumgebungen unterschiedlicher Abstraktionsstufe und unterschiedlicher Simulationsgeschwindigkeiten zur Verfügung. Der C-basierte Cluster-Simulator dient vornehmlich der schnellen Simulation und Optimierung der N-Core-Prozessorkerne und liegt bei einer Simulationsgeschwindigkeit von ca. 10 MHz. Die SystemC-Simulationsumgebung SiMPLE erlaubt hingegen die zyklennakurate Simulation des gesamten Chip-Multiprozessors mit ca. 100 kHz. Sie dient als Plattform für frühe Softwaretests in der Entwurfsphase, aber auch als Evaluationsplattform für neue Hardwarekonzepte. Zukünftig wäre eine Erweiterung des Simulationsmodells durch Annotierung der jeweiligen Verlustleistung der einzelnen Komponenten sinnvoll. So könnten bereits im frühen Stadium einer Entwicklung ausreichend genaue Abschätzungen durch Schaltaktivitäten in deutlich kürzerer Zeit ermittelt werden, als es derzeit die HDL-Simulation erlaubt. Deutlich detaillierter, allerdings auch weitaus langsamer mit ca. 100 Hz Simulationsgeschwindigkeit ist die HDL-basierte Simulation mit der erweiterten GigaNetIC-PERFORM-Umgebung, mit der ebenfalls die Simulation des gesamten Chip-Multiprozessorsystems möglich ist. Das für die GigaNetIC-Architektur genutzte FPGA-basierte Rapid-Prototyping-System RAPTOR2000 dient zum einen als Vorstufe zur ASIC-Realisierung und damit als finaler Test der Hardwarebeschreibung, mit 20 MHz Simulationsgeschwindigkeit zum anderen aber auch als besonders schnelle Plattform zur Analyse sehr zeitintensiver Softwaretests auf CMP-Ebene. Zusätzlich erlaubt diese Plattform es, GigaNetIC-Systeme mit externen Schnittstellen, wie es z. B. für die Netzwerkprozessor-Realisierung notwendig ist, zu testen.

Die einheitliche GigaNetIC-Übersetzer-Werkzeugkette ermöglicht einen reibungslosen Übergang zwischen den einzelnen Plattformen und garantiert ein funktional gleiches Verhalten des GigaNetIC-Systems in allen Simulatoren. Etwaige irrelevante Unterschiede der einzelnen Plattformen bleiben für den Systementwickler transparent.

Ohne eine derart geschlossene Werkzeugkette wäre eine effektive Nutzung eines Chip-Multiprozessorsystems nur sehr eingeschränkt möglich, denn erst das Zusammenspiel von gut aufeinander abgestimmter Hardware und Software ermöglicht eine ressourceneffiziente Lösung.

Ein weiterer Entwicklungsschritt wäre eine Automatisierung der Werkzeugkette im Hinblick auf eine automatische Generierung der Hardwarebeschreibung des Prozessorkerns anhand der UPSLA-Spezifikation. Im nächsten Schritt könnten automatisierte Modifikationsläufe mit anschließender Auswertung der Resultate der Kostenfunktionen neue, effizientere Systeme generieren.

**Optimierung.** Der bei der GigaNetIC-Architektur konsequent verfolgte ganzheitliche Ansatz sieht neben der reinen Simulation bzw. Emulation des CMP-Systems auch eine Optimierung der Architektur im Hinblick auf Anforderungen spezieller Einsatzgebiete vor. Hierbei wird ein auf die GigaNetIC-Architektur angepasster hierarchisch gerichteter Optimierungsansatz verfolgt, der es Systemarchitekten und Softwareentwicklern ermöglicht, eine werkzeuggestützte anwendungsspezifische Anpassung und Optimierung einzelner bzw. aller Komponenten vorzunehmen. Dies hilft, die Ressourceneffizienz des Chip-Multiprozessors im Bezug auf die jeweiligen Anforderungen, Randbedingungen und Schranken im Vergleich zur universellen Variante zu steigern. Der hierarchisch gerichtete Ansatz bietet den Vorteil, dass, unterstützt durch die entwickelte Werkzeugkette, zunächst mit vergleichsweise geringen Modifikationen die Leistungsfähigkeit bzw. der Ressourcenbedarf der Chip-Multiprozessor-Architektur teilweise deutlich optimiert werden kann. Durch die leistungsfähigen Profilierungsmöglichkeiten der GigaNetIC-Entwicklungsumgebung lassen sich besonders rechenintensive Funktionen der Anwendungssoftware schnell lokalisieren. Dies geschieht in der Regel hierarchisch gerichtet, angefangen bei Instruktionssatzerweiterungen, über eng-gekoppelte Hardwarebeschleuniger bis hin zu lose gekoppelten Hardwarebeschleunigern. Letztendlich steht dem Softwarearchitekten dann die Nutzung der parallelen Struktur zur parallelen Bearbeitung einer Aufgabe zur Verfügung, deren Leistungsfähigkeit ggf. durch den GigaNetIC-Multiprozessorcache zusätzlich erhöht werden kann. Unterschiedliche Programmiermodelle des GigaNetIC-CMPs erlauben eine angepasste, möglichst effiziente Nutzung der parallelen Verarbeitungseinheiten für das jeweilige Anwendungsszenario.

Die werkzeuggestützte Analyse des jeweiligen Anwendungsszenarios liefert Aussagen sowohl über den Rechenleistungs- als auch den Energiebedarf, aber auch über die benötigten Bandbreiten der On-Chip-Kommunikation. Die GigaNetIC-Architektur eröffnet, aufgrund der generisch gehaltenen Struktur, zahlreiche Möglichkeiten, das System anwendungsgemäß zu optimieren.

Die anhand vorausgegangener werkzeuggestützter Analysen eingebrachten Optimierungen erlauben eine besonders effiziente Nutzung der parallelen Architektur. Unterstützt durch die Werkzeugkette lässt sich für die jeweils betrachtete Anwendung ein geeigneter Kompromiss zwischen Leistungszuwachs, Verlustleistungsaufnahme, Flächenbedarf und zusätzlich zu erwartendem Entwicklungsaufwand treffen. Pareto-optimale Punkte des Entwurfsraums können so effizient angenähert werden. Anhand anwendungsspezifischer Instruktionssatzerweiterungen des N-Core-Prozessorkerns konnten mit Hilfe einzelner Superinstruktionen Performanzzuwächse von bis zu 25 % für Netzwerkanwendungen erzielt werden – und dies bei einem Flächenzuwachs von teilweise unter einem Prozent, verbunden mit einer Reduktion des Energiebedarfs um 20 %. Die implementierten Hardwarebeschleuniger im Bereich von Netzwerkanwendungen ermöglichen teilweise eine Reduktion der Verarbeitungszeit um drei Größenordnungen bei lediglich moderater Flächenzunahme. Zusätzlich wurde der Energiebedarf der angepassten Systeme deutlich reduziert. Für den *IP-DSLAM*-

*Referenzbenchmark* konnte hier eine Reduktion der benötigten Energie um mehr als Faktor 12 erreicht werden, so dass insgesamt eine merkliche Steigerung der Ressourceneffizienz erzielt wurde.

Anhand einer exemplarischen Analyse verschiedener Realisierungsvarianten für ein paketverarbeitendes System wurde die Kostenfunktionsmethode verifiziert und deren Leistungsfähigkeit aufgezeigt. Mit Hilfe definierter Parameter für die Zielfunktionen der vier Kostenmaße Leistungsaufnahme, Flächenbedarf, Performanz und Zukunftssicherheit sowie der resultierenden Kostenfunktion wurden in Relation zu den definierten Randbedingungen pareto-optimale Systeme für unterschiedliche Einsatzgebiete ermittelt. Dem Systemarchitekten wird hiermit eine hilfreiche Entscheidungshilfe für den Entwurf ressourceneffizienter Implementierungen an die Hand gegeben.

**Netzwerkanwendungsszenarien.** Im Rahmen dieser Arbeit wurde die GigaNetIC-Architektur vornehmlich im Hinblick auf den Einsatz in Netzwerkszenarien untersucht. Gerade im Netzwerkbereich bieten sich parallele Systeme zur Datenverarbeitung an, da hier eine Vielzahl von parallelen, zum Teil nicht korrelierten Datenströmen simultan von den Verarbeitungseinheiten bearbeitet werden kann.

Zur Bewertung der GigaNetIC-Architektur für Zugangnetzwerke wurde ein neuartiger *IP-DSLAM-Benchmark* vorgestellt, ferner die Leistungsfähigkeit der GigaNetIC-Architektur für unterschiedliche Szenarien analysiert und mit anderen Ansätzen verglichen. Im Anschluss wurde die Leistungsfähigkeit des von uns entwickelten Prozessorkerns N-Core für relevante Funktionen durch Optimierung der Architektur, Instruktionssatzerweiterungen sowie Hinzufügen von anwendungsspezifischen Hardwarebeschleunigern deutlich erhöht.

Zudem wurde eine modulare Methode zur effizienten Modellierung von Netzwerkanwendungen vorgestellt, mit deren Hilfe der bereits entworfene *IP-DSLAM-Benchmark* auf Systemebene zu einem noch realistischeren Referenzbenchmark erweitert werden konnte. Mit Hilfe eines eigens entwickelten Visualisierungswerkzeugs, dem *DSLAM-System-Explorer*, konnten die Leistungsdaten des N-Cores, die erzielten Beschleunigungen der Hardwareerweiterungen und Leistungsvergleiche mit anderen Prozessorfamilien komfortabel veranschaulicht werden. Hochrechnungen bzgl. des Hardwareaufwands für gewünschte Anforderungen des *IP-DSLAM*-Anwendungsszenarios lassen sich aufstellen, die eine gezielte Evaluierung des Entwurfsraums ermöglichen. Die Integration der kostenfunktionsbasierten Bewertungsmethode zur Ressourceneffizienz wäre eine wesentliche Erweiterung dieses Werkzeugs, die Vielzahl der Messwerte könnte neben der eigentlichen Visualisierung zusätzlich zur automatisierten Bewertung der untersuchten Realisierungsvarianten herangezogen werden. In zukünftigen Arbeiten könnten außerdem tiefergehende Analysen der GigaNetIC-Architektur mit weiteren etablierten Netzwerkprozessoren von Interesse sein, bei denen die ursprünglich universelle Struktur des GigaNetIC-Systems mit dem hoch spezialisierten, speziell auf Netzwerkanwendungen optimierten Aufbau dieser ASIPs verglichen wird. Ggf. könnten anhand dieser Untersuchungen weitere Optimierungspotentiale der GigaNetIC-Architektur bestimmt werden.

Eine Analyse der Leistungsfähigkeit der verschiedenen Kopplungsarten von Hardwarebeschleunigern an das GigaNoC der GigaNetIC-Architektur zeigt Vor- und Nachteile der einzelnen Varianten am Beispiel einer Netzwerkanwendung auf. Da die Art der Kopplung und die Anzahl der Hardwarebeschleuniger abhängig von den Anforderungen des jeweiligen Anwendungsszenarios ist, können vielversprechende Lösungen im Hinblick auf die Ressourceneffizienz mit Hilfe der GigaNetIC-Simulationsumgebungen ermittelt werden.

**Prototypen.** Im Rahmen dieser Arbeit wurden Varianten der GigaNetIC-Architektur als FPGA-Prototyp und in zwei aktuellen Standardzellentechnologien implementiert. Der auf dem FPGA-Prototypen basierende GigaNetIC-Demonstrator hilft, das Chip-Multiprozessorsystem in einer realen Umgebung detailliert und zugleich um Größenordnungen schneller, als es andere Formen der Simulation ermöglichen, zu verifizieren.

Für zwei aktuelle CMOS-basierte Standardzellentechnologien mit 130 nm und 90 nm Strukturgröße wurde u. a. prototypisch eine 4×2-Gitter-Architektur abgebildet und die Ergebnisse detailliert analysiert. Basierend auf den gewonnenen Syntheseergebnissen konnten fundierte Aussagen über die zu erwartenden Flächen- und Leistungsanforderungen und die realisierbaren Taktfrequenzen der Einzelkomponenten und des Gesamtsystems getroffen werden. Das exemplarische 4×2-System mit acht Clustern zu je vier N-Core-Prozessoren und insgesamt 1,25 MB SRAM benötigt weniger als 44 mm<sup>2</sup> in der verwendeten 90-nm-Technologie und umfasst ca. 80 Millionen Transistoren. Die maximale Betriebsfrequenz liegt bei 285 MHz. Im Rahmen dieser Implementierung wurde ein Konzept der ressourceneffizienten Anordnung der einzelnen Komponenten eines GigaNetIC-Clusters in Form quadratischer Kacheln vorgestellt. Auf Basis dieser Kacheln ist es möglich, GigaNetIC-Systeme effizient zu skalieren und identische Signallaufzeiten zu gewähren, was für eine Maximierung der Performanz essentiell ist.

Der Vergleich eines massiv parallelen GigaNetIC-Systems mit derzeitigen Universalprozessoren zeigte für ein spezielles Netzwerkanwendungsszenario einen Leistungsvorsprung des optimierten GigaNetIC-CMPs um zwei Größenordnungen einhergehend mit einem um zwei Größenordnungen geringeren Energiebedarf bei einem Zehntel der durchschnittlichen Taktfrequenz. Das GigaNetIC-System umfasste 20 Cluster mit je vier N-Cores nebst zusätzlichen Hardwarebeschleunigern mit einer Gesamtfläche von 106 mm<sup>2</sup>, der durchschnittlichen Fläche der untersuchten Desktop-CPU's.

Sowohl im Hinblick auf Universalanwendungen als auch im Besonderen bei Netzwerkanwendungen konnte das hier entwickelte Chip-Multiprozessorsystem seine Leistungsfähigkeit unter Beweis stellen. Anhand der angestellten Untersuchungen lassen sich gute Zukunftschancen für Architekturen wie die der GigaNetIC-Architektur prognostizieren, vorausgesetzt, dass die notwendigen Softwarekomponenten in diese Richtung optimiert werden. Derartige Architekturen sind zum einen in der Lage, die Vorteile, die durch die stetigen Strukturverkleinerungen der Halbleiterprozesse entstehen, zu nutzen. Zum anderen zeigen solche Architekturen Möglichkeiten auf, die sich ergebenden Nachteile zu kompensieren.





# Verzeichnis verwendeter Formelzeichen und Abkürzungen

## Abkürzungen

50%S	Verlustleistungsbestimmung des Synthesewerkzeugs auf Basis statistischer Schaltwahrscheinlichkeiten
AAL5	ATM Adaptation Layer 5
ADSL	Asymmetric Digital Subscriber Line
AES	Advanced Encryption Standard
ALU	Arithmetic Logic Unit
AmI	Ambient Intelligence
AS	Annotierung der Schaltaktivitäten durch Simulation
ASIC	Application-Specific Integrated Circuit
ASIP	Application-Specific Instruction Set Processors
ATM	Asynchronous Transfer Mode
BC	Best Case
BE	Best Effort
BIST	Build-in Self Test
BSP	Bulk Synchronous Parallel
CAM	Content Addressable Memory
CC	Communication-Controller
CISC	Complex Instruction Set Computer
CMP	Chip-Multiprozessor
CPE	Customer-Premises Equipment
CPU	Central Processing Unit
CRACC	Click Rapidly Adapted to C-Code
CRC	Cyclic Redundancy Check
CS	Circuit Switching
DLL	Dynamic Link Library
DMA	Direct Memory Access
DMIPS	Dhrystone MIPS
DSL	Digital Subscriber Line
DSLAM	Digital Subscriber Line Access Multiplexer
DSM	Distributed Shared Memory Multiprocessor
DSP	Digital Signal Processor bzw. Digitale Signalverarbeitungsprozessoren
EEMBC	Embedded Microprocessor Benchmark Consortium
EIB	Element Interconnect Bus
FIER	Fast Interrupt Enable Registers
FIFO	First In First Out
FINT	Fast Interrupt
Flit	Flow Control Digits
FLOPS	Floating Point Operations Per Second
FPGA	Field Programmable Gate Array
FSM	Finite Statemachine
GALS	global asynchron, lokal synchron
GDS II	Graphic Data System II
GE	Gigabit Ethernet
GPS	Generalized Processor Sharing
GT	Guaranteed-Throughput-Traffic

---

HAL	Hardware Abstraction Layer
HDSL	High Data Rate Digital Subscriber Line
HOL	Head-of-Line-Blocking
ILP	Instruction-Level Parallelism
ILP	Integer Linear Programming
iMix	Internet Mix
IP	Intellectual Property
IP	Internet Protocol
IPSec	Internet Protocol Security
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
ISE	Instruction Set Extension
ISP	Internet Service Provider
ITRS	International Technology Roadmap for Semiconductors
MANet	Mobiles Ad-Hoc-Netzwerk
MIC	Memory Interface Controller
MIPS	Millionen Instruktionen pro Sekunde
MOPS	Millionen Operationen pro Sekunde
MPU	Memory / Processor Module
MSB	Most Significant Bit
MTU	Maximum Transmission Unit
NAT	Network Address Translation
NIC	Network Interface Card
NINT	Normal Interrupt
NoC	Network on Chip
NPU	Network Processing Unit
NRE	Non-recurring Engineering
NUMA	Nonuniform Memory Access
OCP	Open Core Protocol
OTP	One-Time Programmable
PC	Personal Computer
PC	Program Counter
PDA	Personal Digital Assistant
PE	Processing Element
Phit	Physical Unit
PIC	Programmierbarer Interruptcontroller
PS	Packet Switching
QoS	Quality of Service
RADSL	Rate Adaptive Digital Subscriber Line
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
ROM	Read-Only Memory
RTL	Register Transfer Level
RTOS	Real-Time Operating System
SAF	Store and Forward
SB	Switch-Box
SDRAM	Synchronous Dynamic Random Access Memory
SDSL	Symmetric Digital Subscriber Line
SIMD	Single Instruction Multiple Data
SiP	System in Package

SLA	Service Level Agreements
SMP	Symmetric Multi-Processing
SoC	System-on-Chip
SPE	Synergistic Processor Elements
STBus	Split Transaction Bus
TC	Typical Case
Tcl	Tool command language
TDM	Time Division Multiplex
TDP	Thermal Design Power
TLP	Task / Thread Level Parallelism
TM	Traffic Model
TTL	Time to Live
UART	Universal Asynchronous Receiver Transmitter
ULSI	Ultra-Large Scale Integration
UMA	Uniform Memory Access
UPSLA	Unified Processor Specification Language
VC	Virtual Channel
VCT	Virtual Cut Through
VDSL	Very High Data Rate Digital Subscriber Line
VHDL	Very High Scale IC Hardware Description Language
VLSI	Very Large Scale Integration
VoIP	Voice over IP
VOQ	Virtual Output Queueing
WC	Worst Case
WCET	Worst-Case Execution Times

### Formelzeichen<sup>70</sup>

$\alpha$	sequentieller Anteil eines Programms
$\alpha_A$	Gewichtung bzw. Gewichtungsfaktor der <i>A</i> -Zielfunktion zur Kostenfunktion
$\alpha_F$	Gewichtung bzw. Gewichtungsfaktor der <i>F</i> -Zielfunktion zur Kostenfunktion
$\alpha_i$	Gewichtungen bzw. Gewichtungsfaktoren der einzelnen Zielfunktionen einer Kostenfunktion
$\alpha_P$	Gewichtung bzw. Gewichtungsfaktor der <i>P</i> -Zielfunktion zur Kostenfunktion
$\alpha_T$	Gewichtung bzw. Gewichtungsfaktor der <i>T</i> -Zielfunktion zur Kostenfunktion
$\delta$	Grad eines Netzwerks
$\delta_x$	Grad eines Knotens <i>x</i>
$\varepsilon_n(N)$	nominale Effizienz paralleler Verarbeitung ( <i>nominal parallel efficiency</i> ) unter Verwendung von <i>N</i> Prozessoren
$\varepsilon_{Tr}$	Übertragungseffizienz
$\tau$	Schaltzeit
$\mathbb{A}$	Architektur
$A_{93}^{CC}$	Fläche des <i>CCs</i> bei einer Flitbreite von 93 Bit
$A_{93}^{Crossbar}$	Fläche des <i>Kreuzschienenverteilers</i> bei einer Flitbreite von 93 Bit
$A_{Fiforeg}$	Registerfläche der FIFOs der Switch-Box
$A_{93}^{Fiforeg}$	Registerfläche der FIFOs der Switch-Box bei einer Flitbreite von 93 Bit

<sup>70</sup> Aufgrund der unterschiedlichen Themengebiete kann es zu Mehrfachverwendungen eines Formelzeichens kommen. Die jeweilige Bedeutung ist dementsprechend kontextbezogen zu sehen.

$A_{93}^{AB}$	Fläche des Advanced Buffers bei einer Flitbreite von 93 Bit
$A$	Fläche
$A_{AB}$	Fläche des Advanced Buffers
$A_{CC}$	Fläche des Communication-Controllers
$A_{Com}$	Fläche der Kommunikationsstruktur (Com)
$A_{Crossbar}$	Fläche des Kreuzschienenverteilers
$A_{Ctrl}$	Fläche eines Controllers (Ctrl) zur Ansteuerung eines Hardwarebeschleunigers
$A_{Inputports}$	Fläche der Eingangsports
$A_{Mem}$	Fläche eines Speichers (Mem)
$A_{PE}$	Fläche einer Verarbeitungseinheit (PE)
$A_{SB}$	Fläche der Switch-Box
$B_B$	Bisektionsbandbreite
$b_c$	Bandbreite eines Kanals
$B_C$	Kanalbisektion
$B_{LC\_DL}$	akkumulierte Downlink-Bandbreite der Linecard
$B_{LC\_UL}$	akkumulierte Uplink-Bandbreite der Linecard
$BM$	Bewertungsmaß
$BM_P$	Bewertungsmaße zur Leistungsaufnahme
$B_{UC\_DL}$	akkumulierte Downlink-Bandbreite der Uplinkcard
$B_{UC\_UL}$	akkumulierte Uplink-Bandbreite der Uplinkcard
$C$	Kanäle in einem Netzwerk
$C(N_1, N_2)$	Schnitt bzw. Teilung eines Netzwerks
$C(x,y)$	Kanal $C$ zwischen Quellknoten $x$ und Zielknoten $y$
$C_{eff}$	effektive Kapazität
$CF$	Kostenfunktion
$CF_{pareto}$	pareto-optimale Lösung = pareto-optimaler Kostenfunktionswert eines Problems für gegebene Gewichtungen $c_i$ und $\alpha_i$
$c_i$	Gewichtungen bzw. Gewichtungsfaktoren der einzelnen Bewertungsmaße einer Zielfunktion
$C_{Ix}$	Menge der Eingangskanäle
$C_{last}$	Lastkapazität
$C_{Ox}$	Menge der Ausgangskanäle
$c_{ox}$	flächenspezifische Oxidkapazität
$C_{ox}$	Gate-Kapazität
$CPI$	<i>Cycles Per Instruction</i> , durchschnittliche Anzahl der benötigten Takte bzw. Zyklen zur Verarbeitung einer Instruktion
$c_{Wachstum}$	erwartete jährliche durchschnittliche Wachstumsrate der Entwurfskomplexität
$D$	Durchmesser
$D(P)$	physikalische Distanz eines Pfades
$D(P)$	Verzögerung eines Pfades
$d_c$	Zielknoten
$D_M$	Durchsatz eines Speichers $M$
$D_{SB}$	Durchsatz einer Switch-Box
$\mathbb{E}$	Entwurfsraum
$\mathbb{E}_{HW}$	hardwarebezogener Entwurfsraum
$E$	Feldstärke
$EM$	Effizienzmaß
$F$	Zukunftssicherheit/Flexibilität
$f$	Taktfrequenz
$f_c$	Betriebsfrequenz eines Kanals

$F_{FT}$	Bewertungsmaß Fehlertoleranz zugehörig zum Kostenmaß Flexibilität
$FI$	Fehlerimmunität
$F_{PG}$	Bewertungsmaß Programmierbarkeit zugehörig zum Kostenmaß Flexibilität
$FT$	Fehlertoleranz
$F_{WV}$	Bewertungsmaß Wiederverwendbarkeit zugehörig zum Kostenmaß Flexibilität
$g$	Gap (Lücke zwischen zwei aufeinander folgenden Übertragungen)
$H_{\varnothing}$	durchschnittliche Hopanzahl
$h$	Anzahl Hops
$H(x, y)$	Anzahl an Hops eines minimalen Pfades zwischen den Knoten $x$ und $y$
$H_{max}$	maximale Hopanzahl
$H_{min}$	durchschnittliche minimale Anzahl an Hops eines Netzwerks zwischen allen Quell- und Zielknoten
$I$	Strom
$IC$	dynamische Instruktionsanzahl ( <i>Dynamic Instruction Count</i> )
$I_D$	Drainstrom
$I_{leck}$	Leckströme
$I_{quer}$	Querströme
$I_{schalt}$	Kurzschlussströme
$J$	Jitter
$\mathbb{K}$	Kostenmaß
$k$	Kante eines Graphen
$L$	Latenz
$L$	Länge des Transistor-Gates
$Latenz_{Paket}$	Latenz des Datentransfers eines Pakets über einen bestimmten Pfad des GigaNoC
$L_c$	Latenz eines Kanals
$l_c$	Länge eines Kanals
$L_{M(R)}$	Lese-Latenz eines Speichers
$L_{M(W)}$	Schreib-Latenz eines Speichers
$m$	Gesamtanzahl der ggf. zu segmentierenden Paketdatenbyte
$M$	Paket bestehend aus $m$ Byte
$m_f$	Anzahl der Datenbyte eines Flits
$m_h$	Anzahl der Headerbyte eines Flits
$N$	Endknoten eines Netzwerks
$n$	Hierarchieebenen eines GigaNetIC-Systems
$N$	Länge eines Datenpakets in Flit
$N^*$	Knoten eines Netzwerks
$N^+$	Routingknoten eines Netzwerks
$n_{FLITS}$	Gesamtzahl der benötigten Flits für die Übertragung eines segmentierten Pakets
$o$	<i>Overhead</i> (Mehraufwand)
$ P $	Anzahl der Hops
$P$	Anzahl paralleler Prozessoren
$P$	Pfad
$P$	Leistungsaufnahme
$P$	Preis eines Systembestandteils in €
$P_{dyn}$	dynamische Verlustleistung
$P_{dyn,Com}$	dynamische Verlustleistung der Kommunikationsstruktur (Com)
$P_{dyn,Ctrl}$	dynamische Verlustleistung eines Controllers (Ctrl) zur Ansteuerung eines Hardwarebeschleunigers
$P_{dyn,Mem}$	dynamische Verlustleistung eines Speichers (Mem)
$P_{dyn,PE}$	dynamische Verlustleistung einer Verarbeitungseinheit (PE)
$P_{Entwicklung}$	erwartete jährliche durchschnittliche Produktivitätswachstumsrate

$PG$	Programmierbarkeit
$P_{Inter-SB-Links}$	Leistungsaufnahme für Inter-Switch-Box-Verbindungen
$P_{last}$ bzw. $P_{load}$	Lastumladeverlustleistung
$P_{pareto}$	pareto-optimaler Punkt im Entwurfsraum
$P_{SB}$	Anzahl der Ports einer Switch-Box
$P_{schalt}$	Schaltverlustleistung
$P_{stat}$	statische Verlustleistung
$R$	Randbedingung
$R_E$	Ressourceneffizienz im Sinne des schaltungstechnischen Entwurfs
$reuse(n)$	Anzahl wiederverwendeter Blöcke im Jahre $n$ vom Referenzjahr
$RV$	Realisierungsvariante eines spezifizierten Systems
$R_{xy}$	Gesamtheit aller minimalen Pfade
$\mathbb{S}$	System
$\mathbb{S}_e$	Systementität
$\mathbb{S}_{e(HW)}$	hardwarebezogene Systementitäten
$\mathbb{S}_{e(SW)}$	softwarebezogene Systementitäten
$S$	Schranke
$S$	Skalierungsfaktor zweier CMOS-Technologien
$S$	Anzahl konkurrierender FIFO-Ketten am Eingang einer Switch-Box
$S(P)$	<i>Speedup</i> (Beschleunigung) in Abhängigkeit von der Anzahl der Prozessoren
$SB_{FIFO-Tiefe}$	Tiefe der Switch-Box FIFO-Register-Warteschlange
$s_c$	Quellknoten
$S_n$	Anzahl konkurrierender FIFO-Ketten am Eingang der $n$ -ten Switch-Box
$S_o$	obere Schranke
$S_u$	untere Schranke
$\mathbf{T}$	Leistung bzw. Performanz
$T$	Taktperiode
$T_{ex}$	Ausführungszeit
$T_{ex,PE}$	Ausführungszeit einer Verarbeitungseinheit (PE)
$t_{ox}$	Siliziumoxiddicke
$T_{superstep}$	Übertragungszeit für eine Nachricht einfacher Länge unter kontinuierlichem Netzwerkverkehr beim BSP-Modell
$\Delta U$	Signalhub
$U$	Spannung bzw. Spannungshub
$U_B$	Versorgungsspannung
$U_{DD}$	Versorgungsspannung
$U_T$	Schwellspannung
$v$	charakteristische Ausbreitungsgeschwindigkeit eines Kanals
$v_{sim}$	Simulationsgeschwindigkeit, gemessen in benötigter Zeit pro simuliertem Takt des Zielsystems
$W$	Weite des Transistor-Gates
$w_c$	Weite eines Kanals $C$
$W_{Flit}$	ganzzahlige Anzahl der 32-Bit-Datenworte, die in einem Flit enthalten sind
$w_{max}$	Gesamtkosten eines Supersteps
$WV$	Wiederverwendbarkeit
$ZF$	Zielfunktion
$ZF_{RV(\text{System})i}(\mathbf{P}, \mathbf{A}, \mathbf{T}, \mathbf{F})_{normiert}$	normierte Schar von Zielfunktionen für implementierte Realisierungsvarianten eines spezifizierten Systems

## Literaturverzeichnis

- [1] G. E. MOORE. *Cramming more components onto integrated circuits*. Electronics, vol. 38, pages 114–117, 1965.
- [2] SEMICONDUCTOR INDUSTRY ASSOCIATION. *International Technology Roadmap for Semiconductors, 2005 Edition*. 2005.
- [3] K. KREWELL. *Intel cancels 4 GHz P4*. Microprocessor Report, pages 12–14, November 2004.
- [4] K. KREWELL. *Multicore showdown. Multicore moving from embedded to servers to clients*. Microprocessor Report, pages 41–45, May 2005.
- [5] K. KREWELL. *Sun`s Niagra pours on the cores*. Microprocessor Report, pages 11–13, October 2004.
- [6] O. BONORDEN, N. BRÜLS, D. K. LE, U. KASTENS, F. MEYER AUF DER HEIDE, J.-C. NIEMANN, M. PORRMANN, U. RÜCKERT, A. SLOWIK AND M. THIES. *A holistic methodology for network processor design*. In Proceedings of the Workshop on High-Speed Local Networks held in conjunction with the 28th Annual IEEE Conference on Local Computer Networks, pages 583–592, Königswinter, Germany, October 2003.
- [7] J.-C. NIEMANN. *GigaNetIC-Final report*. Working groups of System and Circuit Technology (Prof. Rückert), Programming Languages and Compilers (Prof. Kastens), Algorithms and Complexity (Prof. Meyer auf der Heide) University of Paderborn, 2005.
- [8] J.-C. NIEMANN, M. PORRMANN, M. THIES, D. KHOI LE, A. SLOWIK, O. BONORDEN AND G. SCHOMAKER. *GigaNetIC-BMBF-Schlussbericht*. May 2005.
- [9] H. KALTE, M. PORRMANN AND U. RÜCKERT. *A Prototyping Platform for Dynamically Reconfigurable System on Chip Designs*. In Proceedings of the IEEE Workshop Heterogeneous reconfigurable Systems on Chip (SoC), Hamburg, Germany, 2002.
- [10] M. PORRMANN, H. KALTE, U. WITKOWSKI, J.-C. NIEMANN AND U. RÜCKERT. *A Dynamically Reconfigurable Hardware Accelerator for Self-Organizing Feature Maps*. In Proceedings of the 5th World Multi-Conference on Systemics, Cybernetics and Informatics, SCI 2001, pages 242–247, Orlando, Florida, USA, July 2001.
- [11] J. KAHLE ET AL. *Introduction to the Cell multiprocessor*. IBM Journal of Research and Development, vol. 49, pages 589–604, September 2005.
- [12] M. BARON. *The Cell, At One*. Microprocessor Report, pages 9–20, March 2006.
- [13] G. LAWTON. *Powering Down the Computing Infrastructure*. Computer, vol. 40, pages 16–19, 2007.
- [14] A. BRINKMANN, J.-C. NIEMANN, I. HEHEMANN, D. LANGEN, M. PORRMANN AND U. RÜCKERT. *On-Chip Interconnects for Next Generation System-on-Chips*. In Proc. of the 15th Annual IEEE International ASIC/SOC Conference, pages 211–215, Rochester, NY, USA, 2002.
- [15] G.M. AMDAHL. *Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities*. In Proc. of the AFIPS Spring Joint Computer Conference, Atlantic City, New Jersey, USA, AFIPS Press, Reston, Virginia, USA, pages 483–485, 1967.
- [16] J. L. GUSTAFSON. *Reevaluating Amdahl's law*. Communications of the ACM, vol. 31, pages 532–533, 1988.

- [17] D. CULLER, R. KARP, D. PATTERSON, A. SAHAY, K.E. SCHAUSER, E. SANTOS, R. SUBRAMONIAN AND T. VON EICKEN. *LogP: towards a realistic model of parallel computation*. Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming, pages 1–12, 1993.
- [18] L. G. VALIANT. *A bridging model for parallel computation*. Communications of the ACM, vol. 33, pages 103–111, 1990.
- [19] D. CULLER, J. P. SINGH AND A. GUPTA. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann Publishers, 1998.
- [20] J.D.C. LITTLE. *A proof of the queueing formula*. Operations Research, vol. 9, pages 383–387, 1961.
- [21] T. S. SITES. [www.top500.org](http://www.top500.org), May 2007.
- [22] W.J. DALLY AND B. TOWLES. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.
- [23] D. LANGEN, A. BRINKMANN AND U. RÜCKERT. *High Level Estimation of the Area and Power Consumption of On-Chip Interconnects*. In Proceedings of the 13th Annual IEEE International ASIC/SOC Conference, pages 297–301, September 2000.
- [24] Y. ZHANG, W. YE, R. OWENS AND M. IRWIN. *The Power Analysis of Interconnect Structures*. In Proceedings of the ASIC'97 Conference, September 1997.
- [25] Y. ZHANG, W. YE AND M. IRWIN. *An Alternative Architecture for On-Chip Global Interconnects: Segmented Bus Power Modeling*. In Proceedings of the 36th Asilomar Conference on Signals, Systems, and Computers, November 1998.
- [26] J. DUATO, S. YALAMANCHILI AND L. NI. *Interconnection Networks: An Engineering Approach*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1997.
- [27] D.M. CHAPIRO. *Globally-asynchronous Locally-synchronous Systems*. Stanford University, 1984.
- [28] J. MUTTERSBACH, T. VILLIGER, H. KAESLIN, N. FELBER AND W. FICHTNER. *Globally-asynchronous locally-synchronous architectures to simplify the design of on-chip systems*. ASIC/SOC Conference, 1999. Proceedings. Twelfth Annual IEEE International, pages 317–321, 1999.
- [29] R. HO, K. W. MAI AND M. A. HOROWITZ. *The Future of Wires*. Proceedings Of The IEEE, vol. 89, pages 490–504, 2001.
- [30] P. KAPUR, J. MCVITTIE AND K. SARASWAT. *Technology and reliability constrained future copper interconnects. I. Resistance modeling*. IEEE Transactions on Electron Devices, vol. 49, pages 590–597, 2002.
- [31] P. KAPUR, G. CHANDRA, J. MCVITTIE AND K. SARASWAT. *Technology and reliability constrained future copper interconnects. II. Performance implications*. IEEE Transactions on Electron Devices, vol. 49, pages 598–604, 2002.
- [32] D. SYLVESTER AND K. KEUTZER. *Impact of small process geometries on microarchitectures in systemson a chip*. Proceedings of the IEEE, vol. 89, pages 467–489, 2001.
- [33] M. HOROWITZ AND W. DALLY. *How scaling will change processor architecture*. Solid-State Circuits Conference, 2004. Digest of Technical Papers. ISSCC. 2004 IEEE International, pages 132–133, 2004.
- [34] ISO. *Information Technology-Open Systems Interconnection-Basic Reference Model: The Basic Model*. 7498-1 Edition, 1994.
- [35] G. D. MICHELI AND L. BENINI. *Networks on Chips. Technology and Tools*. Morgan Kaufmann, September 2006.



- [36] W.J. DALLY AND C.L. SEITZ. *The torus routing chip*. Distributed Computing, vol. 1, pages 187–196, 1986.
- [37] W. DALLY AND C. SEITZ. *Deadlock-free message routing in multiprocessor interconnection networks*. IEEE Transactions on Computers, vol. 36, pages 547–553, 1987.
- [38] M. DEHYADGARI, M. NICKRAY, A. AFZALI-KUSHA AND Z. NAVABI. *Evaluation of Pseudo Adaptive XY Routing Using an Object Oriented Model for NOC*. The 17th International Conference on Microelectronics ICM 2005, pages 204–208, 2005.
- [39] U. FEIGE AND P. RAGHAVAN. *Exact analysis of hot-potato routing*. Proceedings of the 33rd Annual Symposium on Foundations of Computer Science, pages 553–562, 1992.
- [40] M. MAJER, C. BOBDA, A. AHMADINIA AND J. TEICH. *Packet Routing in Dynamically Changing Networks on Chip*. Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, April 2005.
- [41] A. JANTSCH AND H. TENHUNEN. *Networks on Chip*. Kluwer Academic Publishers, 2003.
- [42] A.S. TANENBAUM. *Computer Networks*. Prentice Hall PTR, 2002.
- [43] P. GUERRIER AND A. GREINER. *A generic architecture for on-chip packet-switched interconnections*. In Proceedings of the conference on Design, Automation and Test in Europe, pages 250–256, 2000.
- [44] C. CIORDAS, T. BASTEN, A. RADULESCU, K. GOOSSENS AND J. VAN MEERBERGEN. *An event-based monitoring service for networks on chip*. ACM Transactions on Design Automation of Electronic Systems (TODAES), vol. 10, pages 702–723, 2005.
- [45] D. SHOEMAKER. *An Optimized Hardware Architecture and Communication Protocol for Scheduled Communication*. Massachusetts Institute of Technology, 1997.
- [46] M. MILLBERG, E. NILSSON, R. THID AND A. JANTSCH. *Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network on chip*. Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings, vol. 2, 2004.
- [47] K. GOOSSENS, J. DIELISSSEN AND A. RADULESCU. *AEtheral network on chip: concepts, architectures, and implementations*. Design Test of Computers, IEEE, vol. 22, pages 414–421, 2005.
- [48] T. BJERREGAARD AND J. SPARSO. *A Router Architecture for Connection-Oriented Service Guarantees in the MANGO Clockless Network-on-Chip*. In DATE '05: Proceedings of the conference on Design, Automation and Test in Europe, IEEE Computer Society, pages 1226–1231, Washington, DC, USA, 2005.
- [49] T. BJERREGAARD, J. SPARSO AND D. TEKLATECH. *Implementation of guaranteed services in the MANGO clockless network-on-chip*. IEE Proceedings: Computers and Digital Techniques, vol. 153, pages 217–229, 2006.
- [50] A. ADRIAHANTENAINA, H. CHARLERY, A. GREINER, L. MORTIEZ AND C. A. ZEFERINO. *SPIN: A Scalable, Packet Switched, On-Chip Micro-Network*. In Proceedings of the conference on Design, Automation and Test in Europe, IEEE Computer Society, pages 70–73, Washington, DC, USA, 2003.
- [51] W. WEBER, J. CHOU, I. SWARBRICK AND D. WINGARD. *A Quality-of-Service Mechanism for Interconnection Networks in System-on-Chips*. In DATE '05: Proceedings of the conference on Design, Automation and Test in Europe, IEEE Computer Society, pages 1232–1237, Washington, DC, USA, 2005.
- [52] ARTERIS-THE NETWORK-ON-CHIP COMPANY. [www.arteris.com](http://www.arteris.com). 2007.

- [53] A. MELLO, L. TEDESCO, N. CALAZANS AND F. MORAES. *Virtual channels in networks on chip: implementation and evaluation on hermes NoC*. In SBCCI '05: Proceedings of the 18th annual symposium on Integrated circuits and system design, ACM Press, pages 178–183, New York, NY, USA, 2005.
- [54] K. H. YUM, E. J. KIM, C. R. DAS, M. YOUSIF AND J. DUATO. *Integrated Admission and Congestion Control for QoS Support in Clusters*. In CLUSTER '02: Proceedings of the IEEE International Conference on Cluster Computing, IEEE Computer Society, pages 325–332, Washington, DC, USA, 2002.
- [55] N. KAVALDJIEV, G. SMIT AND P. JANSEN. *A virtual channel router for on-chip networks*. Proceedings of the IEEE International SOC Conference, pages 289–293, 2004.
- [56] E. BOLOTIN, I. CIDON, R. GINOSAR AND A. KOLODNY. *QNoC: QoS architecture and design process for network on chip*. Journal of Systems Architecture, vol. 50, pages 105–128, 2004.
- [57] W. J. DALLY AND B. TOWLES. *Route Packets, Not Wires: On-Chip Interconnection Networks*. In Proceedings of the Design Automation Conference, pages 684–689, Las Vegas, Nevada, USA, June 2001.
- [58] T. MARESCAUX, A. BARTIC, D. VERKEST, S. VERNALDE AND R. LAUWEREINS. *Interconnection Networks Enable Fine-Grain Dynamic Multi-tasking on FPGAs*. In FPL '02: Proceedings of the Reconfigurable Computing Is Going Mainstream, 12th International Conference on Field-Programmable Logic and Applications, Springer-Verlag, pages 795–805, London, UK, 2002.
- [59] D. BERTOZZI AND L. BENINI. *Xpipes: a network-on-chip architecture for gigascale systems-on-chip*. Circuits and Systems Magazine, IEEE, vol. 4, pages 18–31, 2004.
- [60] K. GOOSSENS, J. DIELISSSEN, J. V. MEERBERGEN, P. POPLAVKO, A. RADULESCU, E. RIJPKEMA, E. WATERLANDER AND P. WIELAGE. *Guaranteeing the quality of services in networks on chip*. Kluwer Academic Publishers, pages 61–82, Networks on Chip. Edition, 2003.
- [61] J. POSTEL. *RFC 791, Internet Protocol: DARPA Internet Program Protocol Specification*. Information Sciences Institute, <http://www.ietf.org/rfc/rfc079.txt>, vol. 791, 1981.
- [62] A. IVANOV AND G. D. MICHELI. *Guest Editors' Introduction: The Network-on-Chip Paradigm in Practice and Research*. Design Test of Computers, IEEE, vol. 22, pages 399–403, 2005.
- [63] T. T. YE. *On-Chip Multiprocessor Communication Network Design and Analysis*. Stanford University, 2003.
- [64] P. P. PANDE, C. GRECU, M. JONES, A. IVANOV AND R. SALEH. *Performance Evaluation and Design Trade-Offs for Network-on-Chip Interconnect Architectures*. IEEE Transactions on Computers, vol. 54, pages 1025–1040, 2005.
- [65] D. COMER AND L. PETERSON. *Network Systems Design Using Network Processors*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 2003.
- [66] XILINX. [www.xilinx.com](http://www.xilinx.com). 2007.
- [67] SEMICONDUCTOR INDUSTRY ASSOCIATION. *International Technology Roadmap for Semiconductors, 2006 Update*. 2006.
- [68] R. P. WEICKER. *Dhrystone: a synthetic systems programming benchmark*. Communications of the ACM, vol. 27, pages 1013–1030, 1984.
- [69] J. A. BOOTH. *ARM dominated MCU market is ready for change*. [www.electronicweekly.com](http://www.electronicweekly.com), January 2007.
- [70] ARC. [www.arc.com](http://www.arc.com), October 2007.

- [71] L. A. BARROSO. *The price of performance*. Queue, vol. 3, pages 48–53, 2005.
- [72] T. R. HALFHILL. *Massively Parallel Digital Video*. Microprocessor Report, pages 17–22, January 2006.
- [73] J. SILC, B. ROBIC AND T. UNGERER. *Processor Architecture*. Springer New York, 1999.
- [74] L. CODRESCU, D. S. WILLS AND J. D. MEINDL. *Architecture of the Atlas Chip-Multiprocessor: Dynamically Parallelizing Irregular Applications*. IEEE Transactions on Computers, vol. 50, pages 67–82, 2001.
- [75] K. OLUKOTUN, B. A. NAYFEH, L. HAMMOND, K. WILSON AND K. CHANG. *The Case for a Single-Chip Multiprocessor*. In Proc. of the Seventh International Symposium on Architectural Support for Parallel Languages and Operating Systems, 1996.
- [76] S. KUMAR, A. JANTSCH, J.-P. SOINIEN, M. FORSELL, M. MILLBERG, J. TIENSYRJÄ AND A. HEMANI. *A network on chip architecture and design methodology*. In Proc. of the IEEE Computer Society Annual Symposium on VLSI, pages 117–124, 2002.
- [77] C. GRECU, P. P. PANDE, A. IVANOV AND R. SALEH. *Structured interconnect architecture: a solution for the non-scalability of bus-based SoCs*. In GLSVLSI '04: Proceedings of the 14th ACM Great Lakes symposium on VLSI, ACM Press, pages 192–195, New York, NY, USA, 2004.
- [78] E. WAINGOLD, M. TAYLOR, D. SRIKRISHNA, V. SARKAR, W. LEE, V. LEE, J. KIM, M. FRANK, P. FINCH, R. BARUA AND OTHERS. *Baring it all to software: Raw machines*. Computer, vol. 30, pages 86–93, 1997.
- [79] M. B. TAYLOR, J. KIM, J. MILLER, D. WENTZLAFF, F. GHODRAT, B. GREENWALD, H. HOFFMAN, P. JOHNSON, J. LEE, W. LEE, A. MA, A. SARAF, M. SENESKI, N. SHNIDMAN AND V. S. *The Raw Microprocessor: A Computational Fabric for Software Circuits and General-Purpose Programs*. IEEE Micro, vol. 22, pages 25–35, 2002.
- [80] L. HAMMOND, B. A. HUBBERT, M. SIU, M. K. PRABHU, M. CHEN AND K. OLUKOTUN. *The Stanford Hydra CMP*. IEEE Micro, vol. 20, pages 71–84, 2000.
- [81] B. ACKLAND, A. ANESKO, D. BRINTHAUPT, S. DAUBERT, A. KALAVADE, J. KNOBLOCH, E. MICCA, M. MOTURI, C. NICOL AND J. O'NEILL. *A single-chip, 1.6-billion, 16-b MAC/s multiprocessor DSP*. IEEE Journal of Solid-State Circuits, vol. 35, pages 412–424, 2000.
- [82] S. GOLDSTEIN, H. SCHMIT, M. BUDIU, S. CADAMBI, M. MOE AND R. TAYLOR. *PipeRench: a reconfigurable architecture and compiler*. Computer, vol. 33, pages 70–77, 2000.
- [83] N. SUZUKI, S. KUROTAKE, M. SUZUKI, N. KANEKO, Y. YAMADA, K. DEGUCHI, Y. HASEGAWA, H. AMANO, K. ANJO, M. MOTOMURA AND OTHERS. *Implementing and Evaluating Stream Applications on the Dynamically Reconfigurable Processor*. Proceedings of the Field-Programmable Custom Computing Machines, 12th Annual IEEE Symposium on (FCCM'04), pages 328–329, 2004.
- [84] PACT. *XPP III Processor Overview White Paper*. 2006.
- [85] PACT. [www.pactxpp.com](http://www.pactxpp.com), Mai 2007.
- [86] L. A. BARROSO, K. GHARACHORLOO, R. MCNAMARA, A. NOWATZYK, S. QADEER, B. SANO, S. SMITH, R. STETS AND B. VERGHESE. *Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing*. In Proc. of the 27th Annual International Symposium on Computer Architecture, 2000.
- [87] RAPPORT. [www.rapportincorporated.com](http://www.rapportincorporated.com), Mai 2007.
- [88] NVIDIA. [www.nvidia.com](http://www.nvidia.com), Mai 2007.
- [89] ATI. [www.ati.com](http://www.ati.com), Mai 2007.

- [90] AGEIA. *Ageia PhysX*. www.ageia.com, Mai 2007.
- [91] J. HELD, J. BAUTISTA AND S. KOEHL. *From a Few Cores to Many: A Tera-scale Computing Research Overview*. Intel, 2006.
- [92] INTEL. <http://www.intel.com>. October 2007.
- [93] AMD. www.amd.com, Mai 2007.
- [94] D. PHAM, H. ANDERSON, E. BEHNEN, M. BOLLIGER, S. GUPTA, P. HOFSTEE, P. HARVEY, C. JOHNS, J. KAHLE, A. KAMEYAMA, J. KEATY, B. LE, S. LEE, T. NGUYEN, J. PETROVICK AND MYD. *Key features of the design methodology enabling a multi-core SoC implementation of a first-generation CELL processor*. In ASP-DAC '06: Proceedings of the 2006 conference on Asia South Pacific design automation, ACM Press, pages 871–878, New York, NY, USA, 2006.
- [95] J.L. HENNESSY AND D.A. PATTERSON. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 2003.
- [96] J. LI AND J. F. MARTÍNEZ. *Power-performance considerations of parallel computing on chip multiprocessors*. ACM Trans. Archit. Code Optim., vol. 2, pages 397–422, 2005.
- [97] H. KLAR AND W. HEIMSCH. *Integrierte digitale Schaltungen MOS, BICMOS*. Springer, 1996.
- [98] W. WOLF AND A. JERRAYA. *Multiprocessor Systems-on-Chips*. Morgan Kaufmann, 2004.
- [99] T. KURODA. *Optimization and Control of Vdd and Vth for Low-Power, High-speed CMOS Design*. In International Conference on Computer aided Design, pages 28–34, 2002.
- [100] D. DUARTE, Y. TSAI, N. VIJAYKRISHNAN AND M. J. IRWIN. *Evaluating Run-Time Techniques for Leakage Power Reduction*. In ASP-DAC, IEEE, pages 31–38, 2002.
- [101] P. E. LANDMAN. *Low-Power Architectural Design Methodologies*. Berkeley Wireless Research Center, 1994.
- [102] M. GRÜNEWALD, J.-C. NIEMANN, M. PORRMANN AND U. RÜCKERT. *A framework for design space exploration of resource efficient network processing on multiprocessor SoCs*. In Proc. of the 3rd Workshop on Network Processors & Applications, pages 87–101, Madrid, Spain, 2004.
- [103] M. GRÜNEWALD, J.-C. NIEMANN, M. PORRMANN AND U. RÜCKERT. *A mapping strategy for resource-efficient network processing on multiprocessor SoCs*. In Proc. of DATE: Design, Automation and Test in Europe, pages 758–763, CNIT La Défense, Paris, France, 2004.
- [104] M. GRÜNEWALD, J.-C. NIEMANN, M. PORRMANN AND U. RÜCKERT. *A framework for design space exploration of resource efficient network processing on multiprocessor SoCs*. In Network Processor Design: Issues and Practices, vol.3, chapter 12, Editors: Crowley, P. and Franklin, M. A. and Hadimioglu, H. and Onufryk, P. Z., Morgan Kaufmann Publishers, pages 245–277, 2005.
- [105] G. BACCARANI, M. WORDEMAN AND R. DENNARD. *Generalized scaling theory and its application to 1/4  $\mu\text{m}$  MOSFET design*. IEEE Transactions on Electron Devices, pages 452–462, 1984.
- [106] T. JUHNKE. *Die Soft-Error-Rate von Submikrometer-CMOS-Logikschaltungen*. 2003.
- [107] T. BASTEN, L. BENINI, A. CHANDRAKASAN, M. LINDWER, J. LIU, R. MIN AND F. ZHAO. *Scaling into Ambient Intelligence*. In Proceedings of Design Automation and Test in Europe (DATE'03), IEEE Computer Society Press, Los Alamitos, CA, USA, 2003, Munchen, Germany, 2003.

- [108] D. LANGEN, J.-C. NIEMANN, M. PORRMANN, H. KALTE AND U. RÜCKERT. *Implementation of a RISC Processor Core for SoC Designs – FPGA Prototype vs. ASIC Implementation*. In Proc. of the IEEE-Workshop: Heterogeneous reconfigurable Systems on Chip (SoC), Hamburg, Germany, 2002.
- [109] J.-C. NIEMANN, C. PUTTMANN, M. PORRMANN AND U. RÜCKERT. *GigaNetIC – A Scalable Embedded On-Chip Multiprocessor Architecture for Network Applications*. In ARCS'06 Architecture of Computing Systems, pages 268–282, March 2006.
- [110] C. PUTTMANN, J.-C. NIEMANN, M. PORRMANN AND U. RÜCKERT. *GigaNoC – A Hierarchical Network-on-Chip for Scalable Chip-Multiprocessors*. In 33rd EUROMICRO Conference on Digital System Design DSD, pages 495–502, Lübeck, Germany, August 2007.
- [111] M. GRÜNEWALD, U. KASTENS, D. K. LE, J.-C. NIEMANN, M. PORRMANN, U. RÜCKERT, M. THIES AND A. SLOWIK. *Network Application Driven Instruction Set Extensions for Embedded Processing Clusters*. In PARELEC 2004, International Conference on Parallel Computing in Electrical Engineering, pages 209–214, Dresden, Germany, September 2004.
- [112] U. KASTENS, D. K. LE, A. SLOWIK AND M. THIES. *Feedback Driven Instruction-Set Extension*. In Proceedings of ACM SIGPLAN/SIGBED 2004 Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'04), Washington, D.C., USA, June 2004.
- [113] J.-C. NIEMANN, C. LIB, M. PORRMANN AND U. RÜCKERT. *A Multiprocessor Cache for Massively Parallel SoC Architectures*. In ARCS'07 Architecture of Computing Systems, pages 83–97, Zurich, Switzerland, March 2007.
- [114] R. EICKHOFF, J.-C. NIEMANN, M. PORRMANN AND U. RÜCKERT. *Adaptable Switch boxes as on-chip routing nodes for networks-on-chip*. In From Specification to Embedded Systems Application, International Embedded Systems Symposium (IESS), A. Rettberg, M. C. Zanella and F. J. Rammig Ed., pages 201–210, Manaus, Brazil, August 2005.
- [115] B. JÄGER, J.-C. NIEMANN AND U. RÜCKERT. *Analytical approach to massively parallel architectures for nanotechnologies*. In Proceedings of the 17th International Conference on Application-Specific Systems, Architecture and Processors (ASAP 2005), IEEE Computer Society Press, pages 268–275, 2005.
- [116] M. GRÜNEWALD, J.-C. NIEMANN AND U. RÜCKERT. *A performance evaluation method for optimizing embedded applications*. In Proceedings of the 3rd IEEE International Workshop on System-On-Chip for Real-Time Applications, pages 10–15, Calgary, Alberta, Canada, June 2003.
- [117] J.-C. NIEMANN, M. PORRMANN AND U. RÜCKERT. *A Scalable Parallel SoC Architecture for Network Processors*. In IEEE Computer Society Annual Symposium on VLSI 2005 (ISVLSI 2005), IEEE Computer Society Press, pages 311–313, Tampa, FL, USA, May 2005.
- [118] J.-C. NIEMANN, M. PORRMANN, C. SAUER AND U. RÜCKERT. *An Evaluation of the Scalable GigaNetIC Architecture for Access Networks*. In Advanced Networking and Communications Hardware Workshop (ANCHOR), held in conjunction with the ISCA 2005, Madison, Wi., USA, July 2005.
- [119] C. SAUER, M. GRIES, J.-C. NIEMANN, M. PORRMANN AND M. THIES. *Application-driven Development of Concurrent Packet Processing Platforms*. In 5th International Symposium on Parallel Computing in Electrical Engineering, Bialystok, Poland, 2006.
- [120] K. YOSHIGOE AND K. CHRISTENSEN. *A parallel-pollled virtual output queued switch with a buffered crossbar*. High Performance Switching and Routing, 2001 IEEE Workshop on, pages 271–275, 2001.

- [121] M. KAROL, M. HLUCHYJ AND S. MORGAN. *Input Versus Output Queueing on a Space-Division Packet Switch*. IEEE Transactions on Communications, vol. 35, pages 1347–1356, 1987.
- [122] A. GUPTA, F. G. GUSTAVSON, M. JOSHI AND S. TOLEDO. *Design and implementation of a fast crossbar scheduler*. ACM Transactions on Mathematical Software, vol. 24, pages 74–101, 1998.
- [123] U. TELLERMANN. *Implementierung von aktiven Kommunikationsknoten für On-Chip-Netzwerke*. Schaltungstechnik, Heinz Nixdorf Institut, Universität Paderborn, September 2004.
- [124] SILICORE CORP. *WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores, Rev. B.3*. [www.silicore.net/pdfiles/wishbone/specs/wbspec\\_b3.pdf](http://www.silicore.net/pdfiles/wishbone/specs/wbspec_b3.pdf), September 2002.
- [125] ARM LTD. *AMBA Specification (Rev. 2.0)*. [www.arm.com/products/solutions/AMBA\\_Spec.html](http://www.arm.com/products/solutions/AMBA_Spec.html), 1999.
- [126] C. SAUER, M. GRIES, S. DIRK, J.-C. NIEMANN, M. PORRMANN AND U. RÜCKERT. *A Lightweight NoC for the NOVA Packet Processing Platform*. In Design, Automation and Test in Europe DATE, Future Interconnect and Network-on-Chip (NoC) Workshop, Munich, Germany, March 2006.
- [127] D. LANGEN. *Abschätzung des Ressourcenbedarfs von hochintegrierten mikroelektronischen Systemen*. Universität Paderborn, Heinz Nixdorf Institut, Schaltungstechnik, 2005.
- [128] MOTOROLA. *M-Core Reference Manual*. 1998.
- [129] GCC. *GCC, the GNU Compiler Collection*. 2007.
- [130] J.-C. NIEMANN, M. PORRMANN AND U. RÜCKERT. *Parallele Architekturen für Netzwerkprozessoren*. In Ambient Intelligence, VDE Kongress, VDE Verlag, pages 105–110, Berlin, Deutschland, Oktober 2004.
- [131] J.-C. NIEMANN, C. PUTTMANN, M. PORRMANN AND U. RÜCKERT. *Resource Efficiency of the GigaNetIC Chip Multiprocessor Architecture*. In Journal of Systems Architecture - the Euromicro Journal, Elsevier, pages 285–299, 2006.
- [132] AMD. *AMD64 Technology AMD64 Architecture Programmer's Manual Volume 2: System Programming*. September 2006.
- [133] J. A. FISHER, P. FARABOSCHI AND C. YOUNG. *Embedded computing: a VLIW approach to architecture, compilers and tools*. Morgan Kaufmann Publishers, 2005.
- [134] E. STÜMPPEL, M. THIES AND U. KASTENS. *VLIW Compilation Techniques for Superscalar Architectures*. In Proc. of 7th International Conference on Compiler Construction CC'98, K. Koskimies Ed., 1998.
- [135] M. THIES. *UPSLA (Unified Processor Specification Language) Language Description and Reference*. Universität Paderborn, 2001-2005.
- [136] J. T. D. FISCHER AND R. WEPER. *Efficient architecture/compiler co-exploration for ASIPs*. In ACM SIG Proc. International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES 2002), pages 27–34, Grenoble, France, Oct. 2002.
- [137] O. BONORDEN, B. JUURLINK, I. VON OTTE AND I. RIEPING. *The Paderborn University BSP (PUB) library*. Parallel Computing, vol. 29, pages 187–207, 2003.
- [138] J.-C. NIEMANN, U. WITKOWSKI, M. PORRMANN AND U. RÜCKERT. *Extension Module for Application-Specific Hardware on the Minirobot Khepera*. In Autonomous Minirobots for Research and Edutainment (AMiRE 2001), pages 279–288, Paderborn, Germany, October 2001.
- [139] J.-C. NIEMANN. *RAPTOR2000 DB-VS FPGA-Modul, Datenblatt*. Heinz Nixdorf Institut, 2000.

- [140] M. PORRMANN AND J.-C. NIEMANN. *Teaching Reconfigurable Computing-Theory and Practice*. In International Workshop on Reconfigurable Computing Education, March 2006.
- [141] G. HAGEN, J.-C. NIEMANN, M. PORRMANN, C. SAUER, A. SLOWIK AND M. THIES. *Developing an IP-DSLAM Benchmark for Network Processor Units*. In ANCHOR 2004, Advanced Networking and Communications Hardware Workshop, held in conjunction with the 31st Annual International Symposium on Computer Architecture (ISCA 2004), Munich, Germany, 2004.
- [142] A. BONA, V. ZACCARIA AND R. ZAFALON. *System Level Power Modeling and Simulation of High-End Industrial Network-on-Chip*. In DATE, IEEE Computer Society, pages 318–323, 2004.
- [143] A. SINHA AND A. P. CHANDRAKASAN. *JouleTrack-A Web Based Tool for Software Energy Profiling*. In Proceedings of the Design Automation Conference, June 2001.
- [144] M. T.-C. LEE, V. TIWARI, S. MALIK AND M. FUJITA. *Power Analysis and Minimization Techniques for Embedded DSP Software*. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 5, March 1997.
- [145] S. STEINKE, M. KNAUER, L. WEHMEYER AND P. MARWEDEL. *An Accurate and Fine Grain Instruction-Level Energy Model Supporting Software Optimizations*. In Proceedings of the International Workshop: Power and Timing Modeling, Optimization and Simulation (Patmos), September 2001.
- [146] T.K. TAN, A. RAGHUMATHAN, G. LAKSHMINARAYANA AND N. K. JHA. *High-level Software Energy Macro-modeling*. In Proceedings of the 38th Design Automation Conference, 2001.
- [147] J. FLINN AND M. SATYANARAYANAN. *PowerScope: A Tool for Profiling the Energy Usage of Mobile Application*. In Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications, pages 2–10, 1999.
- [148] G. QU, N. KAWABE, K. USAMI AND M. POTKONJAK. *Functional-Level Power Estimation Methodology for Microprocessors*. In Proceedings of the 37th Design Automation Conference, June 2000.
- [149] R. MURESAN AND C. H. GEBOTYS. *Current Consumption Dynamics at Instruction and Program Level for a VLIW DSP Processor*. In Proceedings of the 14th International Symposium on System Synthesis, pages 130–135, October 2001.
- [150] R. Y. CHEN AND M. J. IRWIN. *Architecture-Level Power Estimation and Design Experiments*. ACM Transactions on Design Automation of Electronics Systems, vol. 6, pages 50–66, 2001.
- [151] D. BROOKS, V. TIWARI AND M. MARTONOSI. *Wattch: A Framework for Architectural-Level Power Analysis and Optimizations*. In Proceedings of the 27th International Symposium on Computer Architectures, pages 83–94, 2000.
- [152] W. QIFA, T. YUJING AND X. WEI. *Oki Techno Centre Design Team Achieves Lowest Power Consumption Using Power Compiler*. Compiler, Synopsys, December 2002.
- [153] SYNOPSYS. *Power Compiler User Guide*. [www.synopsys.com](http://www.synopsys.com), 2003.
- [154] H. KALTE. *Einbettung dynamisch rekonfigurierbarer Hardwarearchitekturen in eine Universalprozessorumgebung*. Universität Paderborn, Heinz Nixdorf Institut, Schaltungstechnik, 2004.
- [155] COWARE. <http://www.coware.com>, October 2007.
- [156] T. COMPILER. <http://www.retargot.com>, October 2007.
- [157] TENSILICA. <http://tensilica.com>, October 2007.

- [158] S. KENT AND R. ATKINSON. *RFC2401: Security Architecture for the Internet Protocol*. Internet RFCs, 1998.
- [159] J.-C. NIEMANN, C. SAUER, M. PORRMANN AND U. RÜCKERT. *Flexibler Hardware-Beschleuniger zur Verarbeitung von Paketdaten und Prüfsummen zum Einsatz in kommunizierenden eingebetteten On-Chip-Systemen*. Patentanmeldung, Erfindungsmeldung. München, Deutschland, Dezember 2005.
- [160] R. BRADEN, D. BORMAN AND C. PARTRIDGE. *RFC1071: Computing the Internet checksum*. Internet RFCs, 1988.
- [161] EMBEDDED MICROPROCESSOR BENCHMARK CONSORTIUM (EEMBC). <http://www.eembc.org>, July 2007.
- [162] S. DEERING AND R. HINDEN. *RFC 2460: Internet Protocol, Version 6 (IPv6) specification*. December 1998.
- [163] T. HENRIKSSON, N. PERSSON AND D. LIU. *VLSI Implementation of Internet Checksum Calculation for 10 Giga-bit Ethernet*. Proceedings of Design and Diagnostics of Electronics, Circuits and Systems, pages 114–121, 2002.
- [164] T. JUNGEBLUT. *Implementierung ressourceneffizienter Fehlerkorrekturverfahren für die Datenübertragung in drahtlosen Netzwerken*. Schaltungstechnik, Heinz Nixdorf Institut, Universität Paderborn, March 2004.
- [165] J.H. CHERN, J. HUANG, L. ARLEDGE, P.C. LI, P. YANG, T.I. INC AND T. DALLAS. *Multilevel metal capacitance models for CAD design synthesis systems*. Electron Device Letters, IEEE, vol. 13, pages 32–34, 1992.
- [166] N. HATTA, N.D. BARLI, C. IWAMA, L.D. HUNG, D. TASHIRO, S. SAKAI AND H. TANAKA. *Bus Serialization for Reducing Power Consumption*. IPSJ Digital Courier, vol. 2, pages 165–173, 2006.
- [167] A. K. PAREKH AND R. G. GALLAGER. *A generalized processor sharing approach to flow control in integrated service networks: The single node case*. IEEE / ACM Transactions on Networking, vol. 1-3, pages 344–357, 1993.
- [168] M. GRÜNEWALD. *Protokollverarbeitung mit integrierten Multiprozessoren in drahtlosen Ad-hoc-Netzwerken*. Universität Paderborn, Heinz Nixdorf Institut, Schaltungstechnik, 2007.
- [169] J. FREEMAN. *An Industry Analyst's Perspective on Network Processors*. In *Network Processor Design: Issues and Practices*, vol.1, chapter 9, Editors: Crowley, P. and Franklin, M.A. and Hadimioglu, H. and Onufryk, P.Z., Morgan Kaufmann Publishers, pages 191–218, 2002.
- [170] VERBAND DER ANBIETER VON TELEKOMMUNIKATIONS- UND MEHRWERTDIENSTEN (VATM). <http://www.vatm.de>, July 2007.
- [171] T. WOLF AND M. FRANKLIN. *CommBench—a telecommunications benchmark for network processors*. IEEE International Symposium on Performance Analysis of Systems and Software ISPASS, pages 154–162, 2000.
- [172] M.R. GUTHAUS, J.S. RINGENBERG, D. ERNST, T.M. AUSTIN, T. MUDGE AND R.B. BROWN. *MiBench: A free, commercially representative embedded benchmark suite*. IEEE 4th Annual Workshop on Workload Characterization, pages 83–94, 2001.
- [173] G. MEMIK, W.H. MANGIONE-SMITH AND W. HU. *NetBench: a benchmarking suite for network processors*. Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design, pages 39–42, 2001.
- [174] S. AUDENAERT AND P. CHANDRA. *Network processors benchmark framework*. NPF Benchmarking Workgroup.
- [175] THE LINLEY GROUP. *LinleyBench*. <http://www.linleygroup.com/benchmark>, July 2007.
- [176] P. CHANDRA, F. HADY, R. YAVATKAR, T. BOCK, M. CABOT AND P. MATHEW. *Benchmarking Network Processors*. *Network Processor Design: Issues and Practices*, vol. 1, pages 11–25, 2002.



- [177] B. K. LEE AND L. JOHN. *NpBench: a benchmark suite for control plane and data plane applications for network processors*. Proceedings of the 21st International Conference on Computer Design, pages 226–233, 2003.
- [178] C. SAUER, M. GRIES, S. SONNTAG AND I. TECHNOLOGIES. *Modular Reference Implementation of an IP-DSLAM*. Computers and Communications, 2005. ISCC 2005. Proceedings. 10th IEEE Symposium on, pages 191–198, 2005.
- [179] F. BAKER. *RFC 1812, Requirements for IP Version 4 Routers*. Vol. IETF Network Working Group, June 1995.
- [180] X. NIE, L. GAZSI, F. ENGEL AND G. FETTWEIS. *A new network processor architecture for high-speed communications*. IEEE Workshop on Signal Processing Systems, SiPS 99, pages 548–557, 1999.
- [181] J. WAGNER AND R. LEUPERS. *C compiler design for a network processor*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 20, pages 1302–1308, 2001.
- [182] S. SONNTAG, C. SAUER AND M. GRIES. *Click/CRACK-A Programming Model for NP Platforms?*. In 7th NPU Workshop, Infineon Technologies, Corporate Research, Munich, November 2004.
- [183] C. SAUER, M. GRIES AND S. SONNTAG. *Modular domain-specific implementation and exploration framework for embedded software platforms*. Proceedings of the 42nd annual conference on Design automation, pages 254–259, 2005.
- [184] E. KOHLER, R. MORRIS, B. CHEN, J. JANNOTTI AND M. F. KAASHOEK. *The Click modular router*. ACM Transactions on Computer Systems, vol. 18, pages 263–297, August 2000.
- [185] R. MORRIS, E. KOHLER, J. JANNOTTI AND M. F. KAASHOEK. *The Click modular router*. In Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP '99), pages 217–231, Kiawah Island, South Carolina, December 1999.
- [186] E. KOHLER. *The Click Modular Router*. Massachusetts Institute of Technology, February 2001.
- [187] THE CLICK MODULAR ROUTER PROJECT. <http://pdos.csail.mit.edu/click>, August 2005.
- [188] MASSACHUSETTS INSTITUTE OF TECHNOLOGY. <http://pdos.csail.mit.edu>, August 2005.
- [189] MAZU NETWORKS. <http://www.mazunetworks.com>, August 2005.
- [190] INTERNATIONAL COMPUTER SCIENCE INSTITUTE. *The ICSI Networking Group*. <http://www.icsi.berkeley.edu>, August 2005.
- [191] UCLA COMPUTER SCIENCE DEPARTMENT. <http://www.cs.ucla.edu>, August 2005.
- [192] AGILENT TECHNOLOGIES. *True Router Performance Testing*. Application Notes, May 2000.
- [193] LIBEROUTER. *CESNET Liberouter, Programmable Hardware*. <http://www.liberouter.org>, July 2007.
- [194] I. KUON AND J. ROSE. *Measuring the gap between FPGAs and ASICs*. Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field Programmable Gate Arrays, pages 21–30, 2006.
- [195] T. T. YE AND G. DE MICHELI. *Physical Planning for Multiprocessor Networks and Switch Fabrics*. In International Conference on Application-Specific Systems, Architectures and Processors, ASAP 2003, pages 97–107, June 2003.



# Eigene Veröffentlichungen

## Ausgewählte, chronologisch:

### Buchbeiträge

- [104] M. GRÜNEWALD, J.-C. NIEMANN, M. PORRMANN AND U. RÜCKERT. *A framework for design space exploration of resource efficient network processing on multiprocessor SoCs*. Morgan Kaufmann Publishers, vol.3, chapter 12, pages 245–277, 2005.

### Journalbeiträge

- [131] J.-C. NIEMANN, C. PUTTMANN, M. PORRMANN AND U. RÜCKERT. *Resource Efficiency of the GigaNetIC Chip Multiprocessor Architecture*. In *Journal of Systems Architecture - the Euromicro Journal*, Elsevier, 2006.

### Patente

- [159] J.-C. NIEMANN, C. SAUER, M. PORRMANN AND U. RÜCKERT. *Flexibler Hardware-Beschleuniger zur Verarbeitung von Paketdaten und Prüfsummen zum Einsatz in kommunizierenden eingebetteten On-Chip-Systemen*. Patentanmeldung, Erfindungsmeldung, München, Deutschland, Dezember 2005.

### Begutachtete Konferenzbeiträge, international:

- [10] M. PORRMANN, H. KALTE, U. WITKOWSKI, J.-C. NIEMANN AND U. RÜCKERT. *A Dynamically Reconfigurable Hardware Accelerator for Self-Organizing Feature Maps*. In *Proceedings of The 5th World Multi-Conference on Systemics, Cybernetics and Informatics, SCI 2001*, pages 242–247, Orlando, Florida, USA, July 2001. **Best Paper Award**
- [138] J.-C. NIEMANN, U. WITKOWSKI, M. PORRMANN AND U. RÜCKERT. *Extension Module for Application-Specific Hardware on the Minirobot Khepera*. In *Autonomous Minirobots for Research and Edutainment (AMiRE 2001)*, pages 279–288, Paderborn, Germany, 22-24 October 2001.
- [14] A. BRINKMANN, J.-C. NIEMANN, I. HEHEMANN, D. LANGEN, M. PORRMANN AND U. RÜCKERT. *On-Chip Interconnects for Next Generation System-on-Chips*. In *Proc. of the 15th Annual IEEE International ASIC/SOC Conference*, pages 211–215, Rochester, NY, USA, September 2002.
- [108] D. LANGEN, J.-C. NIEMANN, M. PORRMANN, H. KALTE AND U. RÜCKERT. *Implementation of a RISC Processor Core for SoC Designs – FPGA Prototype vs. ASIC Implementation*. In *Proc. of the IEEE-Workshop: Heterogeneous reconfigurable Systems on Chip (SoC)*, Hamburg, Germany, 2002.
- [116] M. GRÜNEWALD, J.-C. NIEMANN AND U. RÜCKERT. *A performance evaluation method for optimizing embedded applications*. In *Proceedings of the 3rd IEEE International Workshop on System-On-Chip for Real-Time Applications*, pages 10–15, Calgary, Alberta, Canada, June 2003.

- [6] O. BONORDEN, N. BRÜLS, D. K. LE, U. KASTENS, F. MEYER AUF DER HEIDE, J.-C. NIEMANN, M. PORRMANN, U. RÜCKERT, A. SLOWIK AND M. THIES. *A holistic methodology for network processor design*. In Proceedings of the Workshop on High-Speed Local Networks held in conjunction with the 28th Annual IEEE Conference on Local Computer Networks, pages 583–592, Königswinter, Germany, 20–24 October 2003.
- [102] M. GRÜNEWALD, J.-C. NIEMANN, M. PORRMANN AND U. RÜCKERT. *A framework for design space exploration of resource efficient network processing on multiprocessor SoCs*. In Proc. of the 3rd Workshop on Network Processors & Applications, pages 87–101, Madrid, Spain, 2004.
- [103] M. GRÜNEWALD, J.-C. NIEMANN, M. PORRMANN AND U. RÜCKERT. *A mapping strategy for resource-efficient network processing on multiprocessor SoCs*. In Proc. of DATE: Design, Automation and Test in Europe, pages 758–763, CNIT La Défense, Paris, France, 2004.
- [111] M. GRÜNEWALD, U. KASTENS, D. K. LE, J.-C. NIEMANN, M. PORRMANN, U. RÜCKERT, M. THIES AND A. SLOWIK. *Network Application Driven Instruction Set Extensions for Embedded Processing Clusters*. In PARELEC 2004, International Conference on Parallel Computing in Electrical Engineering, pages 209–214, Dresden, Germany, 2004.
- [141] G. HAGEN, J.-C. NIEMANN, M. PORRMANN, C. SAUER, A. SLOWIK AND M. THIES. *Developing an IP-DSLAM Benchmark for Network Processor Units*. In ANCHOR 2004, Advanced Networking and Communications Hardware Workshop, held in conjunction with the 31st Annual International Symposium on Computer Architecture (ISCA 2004), Munich, Germany, 2004.
- [117] J.-C. NIEMANN, M. PORRMANN AND U. RÜCKERT. *A Scalable Parallel SoC Architecture for Network Processors*. In IEEE Computer Society Annual Symposium on VLSI 2005 (ISVLSI 2005), IEEE Computer Society Press, pages 311–313, Tampa, Fl., USA, 11–12 May 2005.
- [118] J.-C. NIEMANN, M. PORRMANN, C. SAUER AND U. RÜCKERT. *An Evaluation of the Scalable GigaNetIC Architecture for Access Networks*. In Advanced Networking and Communications Hardware Workshop (ANCHOR), held in conjunction with the ISCA 2005, Madison, Wi., USA, 4–8 July 2005.
- [114] R. EICKHOFF, J.-C. NIEMANN, M. PORRMANN AND U. RÜCKERT. *Adaptable Switch boxes as on-chip routing nodes for networks-on-chip*. From Specification to Embedded Systems Application, International Embedded Systems Symposium (IESS), A. Rettberg, M. C. Zanella and F. J. Rammig Ed., pages 201–210, Manaus, Brazil, 15–17 August 2005.
- [115] B. JÄGER, J.-C. NIEMANN AND U. RÜCKERT. *Analytical approach to massively parallel architectures for nanotechnologies*. In Proceedings of the 17th International Conference on Application-Specific Systems, Architecture and Processors (ASAP 2005), IEEE Computer Society Press, pages 268–275, 2005.
- [126] C. SAUER, M. GRIES, S. DIRK, J.-C. NIEMANN, M. PORRMANN AND U. RÜCKERT. *A Lightweight NoC for the NOVA Packet Processing Platform*. In Design, Automation and Test in Europe DATE, Future Interconnect and Network-on-Chip (NoC) Workshop, Munich, Germany, 6–10 March 2006.
- [109] J.-C. NIEMANN, C. PUTTMANN, M. PORRMANN AND U. RÜCKERT. *GigaNetIC – A Scalable Embedded On-Chip Multiprocessor Architecture for Network Applications*. In ARCS'06 Architecture of Computing Systems, pages 268–282, 13–16 March 2006. **Best Paper Award**
- [140] M. PORRMANN AND J.-C. NIEMANN. *Teaching Reconfigurable Computing-Theory and Practice*. In International Workshop on Reconfigurable Computing Education, March 2006.

- [119] C. SAUER, M. GRIES, J.-C. NIEMANN, M. PORRMANN AND M. THIES. *Application-driven Development of Concurrent Packet Processing Platforms*. In 5th International Symposium on Parallel Computing in Electrical Engineering, Bialystok, Poland, 2006.
- [113] J.-C. NIEMANN, C. LIB, M. PORRMANN AND U. RÜCKERT. *A Multiprocessor Cache for Massively Parallel SoC Architectures*. In ARCS'07 Architecture of Computing Systems, pages 83–97, Zurich, Switzerland, 12-15 March 2007.
- [110] C. PUTTMANN, J.-C. NIEMANN, M. PORRMANN AND U. RÜCKERT. *GigaNoC – A Hierarchical Network-on-Chip for Scalable Chip-Multiprocessors*. In 33rd EUROMICRO Conference on Digital System Design DSD, pages 495–502, Lübeck, Germany, 27-31 August 2007.

**Begutachtete Konferenzbeiträge, national:**

- [130] J.-C. NIEMANN, M. PORRMANN AND U. RÜCKERT. *Parallele Architekturen für Netzwerkprozessoren*. In Ambient Intelligence, VDE Kongress, VDE Verlag, pages 105–110, Berlin, Germany, 18-20 Oktober 2004.



## Anhang A (GigaNetIC-C-Bibliotheksfunktionen)

Die folgende Abbildung eines C-Codebeispiels zeigt, wie die GigaNoC-Funktionen nach Einbindung der *giganetic.h* C-Bibliothek in benutzerspezifischen Anwendungen genutzt werden können:

```
#include <stdio.h>
#include <giganetic.h>
int main()
{
    unsigned int *packet_addr;
    unsigned int *ack;
    unsigned int data;
    unsigned short packet_length;
    unsigned int start_time, stop_time;

    // initialisiere Zeitgeber für Cluster 6 zum genauen Profiling
    gn_timer_init(6,-1);
    // warte auf Paket
    packet_addr = (unsigned int*)gn_wait_for_packet();
    // erhalte Zeiger auf das zugewiesene Paket
    while(!packet_addr)
        packet_addr = (unsigned int*)gn_wait_for_packet();
    // identifiziere Paketlänge
    packet_length = *((volatile unsigned short*)packet_addr)+5);
    // Anweisung für den Communication-Controller, das Paket an einen HW-Beschleuniger zu
    // versenden. Die Adressierung enthält bereits Steuerbefehle für den
    // "Memory Mapped-I/O"-Hardwarebeschleuniger
    *packet_addr = 0x80190039;
    gn_print("Sending packet to CC_HW_ACC\n");
    // Profiling-Kommando zur taktgenauen Zeitmessung
    start_time = gn_timer_get_counter();
    // Versendung des Pakets Nr. 1 an einen benachbarten Cluster
    gn_send_data(0,-1,6,1,packet_length+14,packet_addr);
    // warte auf erfolgreiche Versendung des Pakets Nr. 1
    ack = gn_get_ack(1);
    while(!ack)
        ack = gn_get_ack(1);
    // Profiling Kommando zur taktgenauen Zeitmessung
    stop_time = gn_timer_get_counter();
    // gibt den Paketspeicher des CC des Pakets frei
    gn_free_packet(packet_addr);
    return 1;
}
```

**Abbildung Anhang A-1: Codebeispiel zur Verwendung elementarer GigaNoC-Bibliotheksfunktionen**

Die GigaNoC-Bibliotheksfunktionen sind an dem vorangestellten *gn\_* zu erkennen und in dem Beispiel fett gedruckt. Alle zur Verfügung stehenden Funktionen der Bibliothek können hier aus Platzmangel nicht erläutert werden, sind aber auf der GigaNetIC-Linux-Live-CD (vgl. Abschnitt 4.8) im Softwareverzeichnis enthalten und detailliert kommentiert.

Abbildung Anhang A-2 zeigt den Aufbau der Befehle, die seitens der Prozessorkerne an den Communication-Controller der Switch-Box gesendet werden können. Die Befehle sind in der Adresse kodiert. Adressen die mit „E“ oder „F“ im obersten Oktett beginnen sind für diese Kommandos reserviert. Dies reduziert die theoretisch adressierbare Speichergröße pro Cluster auf 3,5 GByte, was für den Einsatz der Architektur in Chip-Multiprozessoren derzeit mehr als ausreichend ist. Bei den Befehlen handelt es sich sowohl um Abfragen (*ldw*-Befehl) seitens der Prozessoren, als auch um Schreibzugriffe (*stw*-Befehl). Die Daten, die bei Lesezugriffen zurückgeliefert werden beinhalten die angeforderte Information, die vom Communication-Controller der anfragenden CPU übermittelt wird (wie z. B. Registerauslastung oder die Paketadresse bei einer Empfangsabfrage). Bei Schreib-

zugriffen wird die entsprechende Paketadresse als Datum übertragen, um so z. B. ein bestimmtes Paket zum Zielknoten im NoC zu versenden oder den durch ein Paket belegten Speicher wieder frei zu geben. Beim Versand eines Pakets entspricht die Kodierung der Zielkoordinaten dem GigaNoC-Protokoll (vgl. Abschnitt 4.2.2). Die Aufträge zum Versenden einzelner Pakete werden im CC in FIFO-Strukturen abgelegt und sequentiell abgearbeitet. Die drei dunkel hinterlegten Abfragen stellen blockierende Anfragen an den Communication-Controller dar. Diese werden bei der Wishbone-Bus-basierten Variante aufgrund der Spezifikation dieses Bussystems nicht unterstützt. Bei anderen Bussystemen, wie z. B. dem ebenfalls implementierten AMBA-Bus sind diese zulässig und können genutzt werden. Bei einer *Spezialabfrage* handelt es sich um eine Kombination aus Versand- und Empfangsabfrage. Um die Performanz des Systems zu steigern wird mit dieser Abfrage zum einen überprüft, ob ein in Auftrag gegebenes Paket versendet worden ist (*Versandabfrage*) und zum anderen ggf. eine neue Paketadresse eines zu bearbeitenden Pakets zurückgegeben (*Empfangsabfrage*). So können zwei Aufgaben mit einer Transaktion erledigt werden. Die Füllstandsabfrage ermöglicht den Prozessoren eine Bestandsaufnahme der derzeitigen Systemlast und kann für *Load-balancing*-Aufgaben herangezogen werden. Diese Befehle werden, für den Softwareentwickler transparent, durch die oben gezeigten GigaNetIC-Bibliotheksfunktionen aufgerufen. Die Struktur der CC-Kommandos lässt noch weitere Befehle für den Communication-Controller zu, so dass auf zukünftige Anforderungen flexibel reagiert werden kann. Bei negativer Quittierung einer Abfrage wird NULL, also der Null-Vektor (0x0) zurückgegeben, der eindeutig von einer gültigen Adresse unterschieden werden kann, da die Basis des Paketspeichers bei Adresse 0x40000000 liegt (vgl. Abbildung 4-18).

Operation	Adresse								Daten	
	31							0	31	0
Füllstandsabfrage	F	0	0	0	0	8	0	0	FIFO-Registerauslastung	
Spezialabfrage	F	0	0	0	0	6	0	CPU_ID	CPU_ID / Paketadresse / NULL	
Versandabfrage	F	0	0	0	0	5	0	CPU_ID	CPU_ID / NULL	
Empfangsabfrage	F	0	0	0	0	4	0	0	Paketadresse / NULL	
Spezialabfrage	F	0	0	0	0	3	0	CPU_ID	CPU_ID / Paketadresse	
Versandabfrage	F	0	0	0	0	2	0	CPU_ID	CPU_ID	
Empfangsabfrage	F	0	0	0	0	1	0	0	Paketadresse	
Speicherfreigabe	F	0	0	0	0	0	0	0	Paketadresse	
Paket versenden	E	Port	X	Y	Paketlänge			CPU_ID	Paketadresse	

← 4 Bit → \* 3 Bit \* ← 4 Bit → \* 4 Bit → ← 11 Bit → \* 6 Bit →

Abbildung Anhang A-2: Befehlsübersicht des Communication-Controllers



## Anhang B (Parametrisierbarkeit der GigaNetIC-Architektur)

Die folgenden Parameter sind die wichtigsten generisch gehaltenen Konstanten des GigaNetIC-Systems, die in den *VHDL-Design-Packages* der jeweiligen Design-Bibliothek enthalten sind. Die relativ große Anzahl gibt bereits einen kleinen Eindruck von der Komplexität, die der Entwurf eines skalierbaren flexiblen Chip-Multiprozessors mit sich bringt.

### Globales GigaNetIC-System

- *CLUSTER\_PER\_ROW*: Anzahl der Cluster bzw. Switch-Boxen in x-Dimension (1 bis beliebig)
- *CLUSTER\_PER\_COL*: Anzahl der Cluster bzw. Switch-Boxen in y-Dimension (1 bis beliebig)
- *NUM\_OF\_PE*: Legt die Anzahl der Prozessorelemente pro Cluster fest (derzeit realisierbar: 1 bis 8).
- *NUM\_OF\_ETH*: Legt die Anzahl der Ethernet-Ports pro Cluster fest (derzeit realisierbar: bis 4).

### Switch-Box

- *NUMBER\_OF\_PORTS*: Anzahl der I/O-Ports der Switch-Box.
- *LOG\_NUMBER\_OF\_PORTS*: Ganzzahliger Zweierlogarithmus des Wertes *NUMBER\_OF\_PORTS*. Dieser Wert dient der generischen Dimensionierung der Steuerleitungen der Multiplexer.
- *DATA\_WIDTH*: Dieser Parameter legt die physikalische Breite eines Flits fest. Er ist von vielen der bereits erwähnten Parameter, die das Format des Flitkopfes sowie des Flitrumpfes betreffen, abhängig (vgl. Abschnitt 4.2.2). Deshalb sind die entsprechenden Regeln bei der Änderung dieses Wertes zu befolgen.
- *FIFO\_DEPTH*: Gibt die Tiefe der einzelnen FIFO-Ketten in den Ports der Switch-Box an und bestimmt somit die Größe des Warteraums.
- *COST\_WIDTH*: Breite der Kostentabelle für ein entsprechendes Routing-Verfahren.
- *COMMAND\_WIDTH*: Breite des Kommandoteils im Flitkopf (vgl. Abschnitt 4.2.2).
- *X\_WIDTH*: Breite der X-Koordinate im Flitkopf. Sie besteht aus einem Vorzeichenbit und einer binär kodierten Zahl. Die Zahl entspricht den Netzwerkknoten, die das Flit in Richtung der X-Achse durchlaufen muss. Bei einem Netzwerk von 8×8-Knoten ist eine Breite von 4 notwendig.
- *Y\_WIDTH*: Die Breite der Y-Koordinate, verhält sich analog zu *X\_WIDTH*. Sie sind unabhängig voneinander einstellbar, um auch unsymmetrische Netzwerkdimensionen zuzulassen.
- *FLIT\_ID\_WIDTH*: Kennzeichnet im Flitkopf die Identifikationsnummer des Flits innerhalb eines Pakets. Mit dieser Größe sind Pakete mit bis zu 16 KB Nutzdaten möglich. Die Bearbeitung der größten auftretenden DSL-Pakete von 9000 Byte führte zu dem derzeitigen Wert 11.
- *FLOW\_ID\_WIDTH*: Die *Flow-ID* wird benötigt, um ein Paket eindeutig zu kennzeichnen. Dabei genügt es, eine *ID* für jeden Absender zu vergeben. Es ist also eine Nummer pro Switch-Box und Eingang des Netzwerks erforderlich. Bei einem 4×4-Gitter sind das maximal 32 Absender, die in fünf Bit kodiert werden können.
- *CPU\_ID\_WIDTH*: Die ID einer CPU setzt sich aus der Nummer des N-Cores und der Paketnummer zusammen. Damit ist ein Paket unverwechselbar in dem Sende-FIFO des *Communication-Controllers* markiert. Die CPU kann unter Angabe der *CPU ID* erfragen, ob das Paket mit dieser Nummer vollständig versendet wurde. Der derzeitige Wert 6 entspricht der Minimalbreite, um eine eindeutige Identifizierung zu ermöglichen.

- *ADDR\_WIDTH*: Legt für den CC die Größe des Speichers für die Eingangspakete fest. Derzeit ist die Breite 15. Es können also  $2^{15}$  Flit-Nutzdaten à 8 Byte im Speicher abgelegt werden. Das entspricht einer Datenmenge von 256 KByte bei einer maximalen Paketgröße von 16 KByte.
- *ADDR\_FIFO\_DEPTH*: Die eingehenden Pakete werden vom *Communication-Controller* im Speicher abgelegt. Damit eine Verarbeitungseinheit mit einem Paket arbeiten kann, muss sie eine Anfrage an den CC stellen, ob ein Paket vorliegt. Ist dies der Fall, so erhält sie die Startadresse des Pakets, um es aus dem Speicher zu lesen. Diese Anfragen der Prozessoren sind nicht deterministisch einplanbar. Es ist also zu erwarten, dass sich zeitweise mehrere Pakete aufstauen können. Die Startadressen dieser Pakete legt der CC in einer FIFO-Registerkette ab, deren Tiefe mit *ADDR\_FIFO\_DEPTH* eingestellt wird. Um mehrere Pakete pro Prozessor gleichzeitig verwalten zu können, ist die Tiefe derzeit mit 16 eingestellt und erlaubt bei vier Prozessoren im Cluster je vier Speicherplätze in dem Adress-FIFO.
- *SEND\_FIFO\_DEPTH*: Der *Communication-Controller* kann Pakete nur sequentiell verschicken, so dass sich Versandaufträge aufstauen können. Da die Prozessoren möglichst unbelastet vom Versendeprozess bleiben sollen, muss ein gewisser Warteraum für diese Aufträge eingerichtet werden. Damit wird die Gefahr reduziert, dass CPUs aufgrund einer ausgelasteten Sendeeinheit Anfragen mehrmals stellen müssen. Die Tiefe dieses Sende-FIFOs ist derzeit auf 16 gesetzt.
- *MEMORY\_WIDTH*: Der *Communication-Controller* ist hauptsächlich mit dem Empfang und Versand von Paketen beschäftigt. Er muss also ständig auf den Speicher zugreifen. Damit er nicht permanent den Bus belastet, sondern ein passiver Busteilnehmer bleibt, wird ein Dual-Ported-Speicher verwendet. Dieser hat zwei unabhängige Ports, die gleichzeitig benutzt werden können. So kann er als Slave an den Bus angeschlossen werden und ist damit den Prozessoren zugänglich. Der zweite Anschluss ist direkt mit dem CC verbunden. Die Breiten der Daten- und Adressleitungen müssen nicht zwangsläufig an beiden Ports übereinstimmen. Während die Breite der Adresse für den Port am Bus mit *BUS\_ADDRESS\_WIDTH* definiert ist, ist für den Anschluss des CCs ein eigener Parameter *MEMORY\_WIDTH* erforderlich.
- *BUS\_ADDRESS\_WIDTH*: Die Breite der Adressen, die ein Bussystem verwalten kann, ist nicht zwangsläufig bei allen Bussen gleich. Aus diesem Grund wird die Größe der Adressweite generisch gehalten, so dass das GigaNetIC-System weitestgehend unabhängig von der Struktur des lokalen Busses ist. Die derzeitige Implementierung sieht 32-Bit-breite Busadressen, passend für AMBA und Wishbone (vgl. Abschnitt 4.2.4), vor.
- *BUS\_DATA\_WIDTH*: Stellt die Datenbreite des Busses ein und ist derzeit zu 32 Bit gewählt, da der N-Core-Prozessor (vgl. Abschnitt 4.3.1) ebenfalls mit 32-Bit-breiten Daten arbeitet. Durch diesen Parameter ist das GigaNetIC-System leicht an andere Prozessorkerne anpassbar.

### Wishbone-Bus-Deklarationen

- *WB\_ADDR\_WIDTH*: Breite des Wishbone-Adressbusses (derzeit 32).
- *WB\_DATA\_WIDTH*: Breite des Wishbone-Datenbusses (derzeit 32).
- *WB\_SEL\_WIDTH*: Breite des Select-Busses zur Auswahl der Busteilnehmer (derzeit 4).

### AMBA-Bus-Deklarationen

- *AMBA\_BUS\_WIDTH*: Breite der AMBA-Daten-Busse (von 32 bis 1024).
- *NO\_OF\_MASTERS*: Bestimmt die Anzahl der Master im System (von 1 bis 16, je 1 Master pro Layer).
- *NO\_OF\_SLAVES*: Legt die Anzahl der Slaves mit einem oder mehr Ports im System fest (von 1 bis 16).

- *NO\_OF\_PORTS*: Ist die Gesamtzahl der Ports aller Slaves (von 1 bis 16).
- *WRITE\_BUS\_WIDTH*: Festlegung der Breite der Write-Busse (von 8 bis 1024).
- *READ\_BUS\_WIDTH*: Legt die Breite der Read-Busse fest (von 8 bis 1024).
- *ADDR\_WIDTH*: Definiert die Breite der Adressen im System (von 8 bis 1024).

### GigaNetIC-Multiprozessor-Cache

- *UNIPROCESSOR\_SYSTEM*: Legt fest, ob ein Einprozessor-Cache oder ein Multiprozessor-Cache instantiiert wird.
- *SPLIT\_CACHE*: Legt fest, ob ein Split-Cache generiert wird oder eine Unified-Cache-Architektur zum Einsatz kommt. Beim Unified-Cache wird die Datencachestruktur verwendet.
- *D\_LINE\_WIDTH*: Gibt die Anzahl der Bits pro Daten-Cache-Line an (von 32 bis 1024).
- *D\_ASSOC*: Beziffert die Assoziativität des Daten-Caches (von 2 bis 32).
- *D\_NO\_OF\_LINES*: Anzahl der Daten-Cache-Lines ( $\geq 8$ , je mit Vielfachem von 4).  
 $\Rightarrow$  Daten-Cache-Größe [Bit] =  $D\_LINE\_WIDTH * D\_ASSOC * D\_NO\_OF\_LINES$
- *I\_LINE\_WIDTH*: Gibt die Anzahl der Bits pro Instruktions-Cache-Line an (von 32 bis 1024).
- *I\_ASSOC*: Beziffert die Assoziativität des Instruktions-Caches (von 2 bis 32).
- *I\_NO\_OF\_LINES*: Anzahl der Instruktions-Cache-Lines ( $\geq 8$ , mit je Vielfachem von 4).  
 $\Rightarrow$  Instruktions-Cache-Größe [Bit] =  $I\_LINE\_WIDTH * I\_ASSOC * I\_NO\_OF\_LINES$

### N-Core

- *CPU\_ID*: 32-Bit-breite, eindeutige Identifikationsnummer, die dem Prozessorkern zugewiesen wird. Sie wird direkt an den GSR-Eingangsport des Prozessors angelegt und ist Bestandteil des N-Core-Registersatzes. Dies ermöglicht einen schnellen Zugriff im Programmcode zur Feststellung der jeweiligen Prozessorinstanz.
- *INT\_WIDTH*: Bestimmt die Anzahl der externen Interrupt-Signale. Die Breite des Interrupt-Eingangsvektors und das PIC-Modul werden hierdurch angepasst. Derzeit wird lediglich der Interrupt des Timer-Moduls genutzt.
- *RAM\_WIDTH*: Entspricht der benötigten Breite des Speicher-Adressbusses und leitet sich von der verwendeten Speichergröße ab. Ist das Prozessorsystem z. B. mit 32-KB-Speicher ausgestattet, ist  $RAM\_WIDTH = 13$  ( $2^{13} \times 32$  Bit = 32 KB).
- *MEM\_FILE*: Legt den Dateinamen des Speicherabbilds fest und wird nur bei der Simulation ausgewertet. Zu Beginn der Simulation wird diese Speicherabbilddatei zur Initialisierung des N-Core-Programmspeichers geladen.

### Hardwarebeschleuniger

- *NUM\_OF\_HW\_ACC*: Anzahl der an die Switch-Box angeschlossenen Hardwarebeschleuniger.
- *NUM\_OF\_WBM\_HW\_ACC*: Anzahl der an den lokalen Bus angeschlossenen Hardwarebeschleuniger.

### Speicher

- *PE\_MEM*: Speicher pro CPU, beinhaltet generische Anpassung der Adressleitungen etc. ( $2^5 = 32$  KByte).
- *SB\_MEM*: Speicher der Switch-Box, generische Anpassung der Adressleitungen etc. ( $2^5 = 32$  KByte).
- *ETH\_MEM*: Speicher pro CPU, generische Anpassung der Adressleitungen etc. ( $2^4 = 16$  KByte).

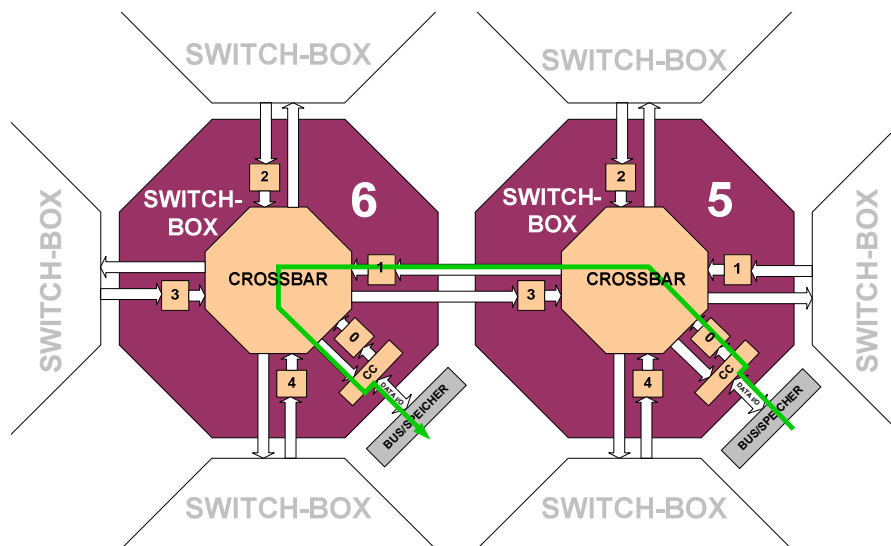
Wie bereits erwähnt handelt es sich hier nur um eine Auswahl der wichtigsten, variierbaren Parameter des GigaNetIC-Systems. Für einen vollständigen Eindruck ist der im Versionsverwaltungssystem des Heinz Nixdorf Instituts abgelegte Quellcode einzusehen.

## Anhang C (Ablauf der Kommunikation auf Switch-Box-Ebene)

Tabelle Anhang C-1 schildert den zeitlichen Ablauf einer Paketinjektion seitens eines Prozessors und die Terminierung am Zielknoten in einer benachbarten Switch-Box. Abbildung Anhang C-1 zeigt die hierfür angenommene Hardwarestruktur zur Erläuterung.

**Tabelle Anhang C-1: Beispielhafter Ablauf einer Paketinjektion seitens eines Prozessors und dessen Terminierung in einer anderen Switch-Box**

Zeit	Vorgang
1. Takt	Die Anfrage eines N-Cores wird im Sende-FIFO gespeichert.
2. Takt	Der Sendevorgang beginnt. Die Adresse des ersten Flits liegt am Speicher an.
3. Takt	Die Daten des ersten Flits liegen am Speicherausgang und werden an den OUTPUT_BUFFER ausgegeben.
4. Takt	Das Flit steht im Eingangsregister der Eingangsports der Switch-Box 5.
5. Takt	Das Flit wird in die FIFO-Schlange für Ausgang 3 übernommen.
6. Takt	Das Flit steht im Puffer hinter der FIFO-Kette und wird durch die Crossbar an die Switch-Box 6 geleitet.
7. Takt	Das Flit steht im Eingangsregister des Eingangsports in Switch-Box 6.
8. Takt	Das Flit wurde in die FIFO-Schlange für Ausgang 0 übernommen.
9. Takt	Das Flit steht im Puffer hinter der FIFO-Kette und wird durch die Crossbar an den Communication-Controller geleitet.
10. Takt	Das Flit steht im Eingangsregister der CC.
11. Takt	Das Flit wird in das Adressregister des CCs übernommen, und die Daten werden im Speicher angelegt.



**Abbildung Anhang C-1: Die dem in Tabelle Anhang C-1 beschriebenen Ablauf der Paketinjektion zugrunde liegende Hardwarestruktur**



## Anhang D (Instruktionssatz des N-Cores)

### Instruktionssatz

Der S-Core und folglich der N-Core verfügen über eine dreistufige Pipeline. Instruktionen werden geholt, anschließend dekodiert und die benötigten Register adressiert. In der letzten Stufe werden die Operationen ausgeführt und die Ergebnisse zurückgeschrieben.

Der Originalinstruktionssatz des S-Cores verfügt über arithmetische und logische Instruktionen, Befehle für Bitoperationen, Byte-Extraktion, Schiebe-, Lade-, Speicher- und Sprungbefehle.

Er lässt sich in drei verschiedene Instruktionstypen einteilen:

#### 1. Register-zu-Register-Instruktionen

- a) Bei der Ein-Registeradressierung spezifizieren die untersten vier Bit des Befehlswortes (*rx*-Feld) das Quell-/Zielregister. Folgende Instruktionen bedienen sich dieses Instruktionsformats: *abs*, *asrc*, *brev*, *clrf*, *clrt*, *decf*, *decgt*, *declt*, *decne*, *dect*, *ffl*, *incf*, *inct*, *lslc*, *lsrc*, *mvc*, *mvcv*, *not*, *sextb*, *sextl*, *tstnbz*, *xsr*, *zextb* und *zextl*.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode												rx			

- b) Dyadische Registeradressierung wird bei den Befehlen *addc*, *andda*, *and*, *andn*, *asr*, *bgenr*, *cmp[hs,lt,ne]*, *ixh*, *ixw*, *lsl*, *lsr*, *mov*, *movf*, *movt*, *mult*, *or*, *rsub*, *subc*, *subu*, *tst* und *xor* benutzt. Die *rx*- und *ry*-Felder bezeichnen dabei ein Quell- und ein weiteres Quell- oder Zielregister.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode								ry				rx			

- c) Einregisterbefehle mit einem fünf Bit langen *immediate*-Feld: Das vier Bit große Registerfeld *rx* stellt das zu verwendende Quell-/Zielregister. Das fünf Bit lange *immediate*-Feld spezifiziert bei den Befehlen *andi*, *asri*, *bclri*, *bgeni*, *bmaski*, *bseti*, *btsti*, *cpmnei*, *lsli*, *lsri*, *rotli* und *rsubi* den zweiten Operanden als einen vorzeichenlosen *immediate*-Wert. Bei den Instruktionen *addi*, *subi* und *cmplti* wird dieser *immediate*-Wert so dekodiert, dass ein Wertebereich zwischen 1 und 32 abgedeckt wird.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode								Imm5				rx			

- d) Registerbefehle mit einem sieben Bit langen *immediate*-Feld: Bei diesem Adressierungsformat für die *movi*-Instruktion ist das vier Bit breite *rx*-Feld das Zielregister und das sieben Bit große *immediate*-Feld ist der vorzeichenlose Wert.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode					Imm7							rx			

- e) Kontrollregisteradressierung wird bei den Befehlen *mfcrr* und *mtrr* eingesetzt. Dabei ist das vier Bit breite *rx*-Feld das normale Quell-/Zielregister und das fünf Bit breite *CReg* Feld das Kontrollregister.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Opcode</i>								<i>CReg</i>				<i>rx</i>			

## 2. Speicherzugriffsinstruktionen mit

- a) 4-Bit-*immediate*-Adressierung: Diese wird bei den *ld*- und *st*-Instruktionen verwendet. Der Wert aus dem im *rx*-Feld definierten Register plus den vorzeichenlosen Wert aus *imm4* ergibt die Zugriffsadresse. Das *rz*-Feld ist das Quellregister für den Speicherbefehl bzw. das Zielregister für den Ladebefehl.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Opcode</i>				<i>Rz</i>				<i>Imm4</i>				<i>rx</i>			

- b) Laden und Speichern eines zusammenhängenden Registerbereichs. Das *rx*-Feld definiert den Startpunkt im Speicher, an dem die Register *r4* bis *r7* gespeichert oder geladen werden. Nur die *ldq*- und *stq*-Anweisungen verwenden diesen Modus.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Opcode</i>												<i>rx</i>			

- c) Laden und Speichern mehrerer Register. Die durch das *rf*-Feld spezifizierten Register werden entweder ab der in Register *r0* definierten Speicheradresse gespeichert oder von der in Register *r0* definierten Adresse geladen. Diesen Modus verwenden die Instruktionen *ldm* und *sdm*.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Opcode</i>												<i>rf</i>			

- d) Laden eines relativen Wortes ist mit der Instruktion *lrw* möglich. Dazu wird der *disp8* Wert um 2 Stellen nach links geschoben und zum nächsten *PC*-Wert hinzuaddiert. Das Ergebnis ist die Speicheradresse, von der das Wort geladen und dann in das *rz*-Register abgelegt wird. (Register *r0* und *r15* dürfen nicht verwendet werden.)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Opcode</i>				<i>Rz</i>				<i>disp8</i>							

## 3. Kontrollfluss-Instruktionen

- a) 11-Bit-Verschiebung. Die Sprungadresse bei den Instruktionen *br*, *bf*, *bt* und *bsr* wird wie folgt berechnet: *disp11* wird durch ein Linksschieben mit zwei multipliziert und dem Wert des nächsten Programmzählers (*PC*+2) hinzuaddiert.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>Opcode</i>					<i>disp11</i>										



- b) Registeradressierung. Bei den Instruktionen *jmp* und *jsr* ist durch das 4-Bit-*rx*-Feld die Sprungadresse vorgegeben.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode												rx			

- c) Indirekte Adressierung. Die *jmp*- und *jsr*-Anweisungen verwenden dieses Format, um ein 32-Bit-Wort relativ zum Programmzähler (PC) zu adressieren.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode								dipl8							

- d) Negative 4-Bit-Verschiebung. Die *loopt*-Anweisung verwendet diese Adressierungsart zur Berechnung der effektiven Adresse. Dabei wird die Zieladresse mittels Addition des nächsten Werts des Programmzählers und des vier Bit langen *disp4*-Wertes, der zuvor mit 1 zum negativen Wert erweitert und um ein Bit nach links geschoben wurde, berechnet.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode								disp4				rx			

Abbildung Anhang D-1 zeigt den verbliebenen freien Opcode-Bereich des N-Core-Prozessorkerns. Zusätzlich implementierte Superinstruktionen sind namentlich aufgeführt und reduzieren die Anzahl verfügbarer Befehlswoorte, so dass nunmehr maximal noch 725 einfache Instruktionen, die in zehn separaten Binärbereichen liegen, hinzugefügt werden können.

Opcode				min	max	#Stellen	1r    imm4	2r    1r+imm4	1r+imm<4	1r+imm>4
0000	0000	0000	0111	1	1	0	0	0	0	0
0000	0000	0000	11xx	1	4	2	0	0	0	0
0000	0000	0010	xxxx	1	16	4	1	0	0	0
0000	1000	mem[RY]	RX	xorldw						
0000	1001	mem[RY]	RX	xorldwls8						
0010	011i	iiii	rrrr	1	512	9	32	2	4	1
0010	1100	001x	rrrr	1	32	5	2	0	1	0
0010	1100	01xx	rrrr	1	64	6	4	0	1	0
0011	0010	0000	rrrr	1	16	4	1	0	0	0
0011	0010	001x	rrrr	1	32	5	2	0	1	0
0011	0010	010x	rrrr	1	32	5	2	0	1	0
0011	0010	0110	rrrr	1	16	4	1	0	0	0
010RY	RY0	C	RX	andshr						
010RY	RY1	C	RX	ixwandshr						
0110	1000	RY	RX	orshl8						
0110	1001	RY	RX	orshl16						
0110	1010	RY	RX	orshl24						
0110	1011	RY	RX	ldwixw						
0110	11C	RY	RX	ldwaddi						
Total				10	725		45	2	8	1

Abbildung Anhang D-1: Freier Opcode-Bereich und verwendeter Bereich für die zusätzlich implementierten Instruktionssatzerweiterungen des N-Cores

Im Folgenden werden die Funktionsweisen der zusätzlichen N-Core-Befehle kurz vorgestellt:

In (9.1) wird die Funktionsweise des Befehls *ANDSHR* dargestellt, der Inhalt des *RX*-Registers wird zunächst um eine Konstante *C* nach rechts geschoben und im Anschluss bitweise mit dem Inhalt von Register *RY* und-verknüpft. Das Ergebnis der Operation wird nach *RX* zurückgeschrieben. Die Operation benötigt anstatt ursprünglich zwei nur noch einen Takt.

$$\text{ANDSHR: } RX \leftarrow (RX \gg C) \text{ AND } RY \quad (9.1)$$

(9.2) erläutert die Instruktion *IXWANDSHR*. Der Inhalt von Register *RX* wird um *C* nach rechts geschoben mit dem Inhalt von *RY* und-verknüpft, um 2 nach links geschoben und abschließend mit dem Inhalt von *RI* aufsummiert. Das Ergebnis der Operation wird nach *RX* zurückgeschrieben. Die Operation benötigt einen Takt. Die ursprüngliche Funktionsabfolge benötigte drei Takte.

$$\text{IXWANDSHR: } RX \leftarrow RI + (((RX \gg C) \text{ AND } RY) \ll 2) \quad (9.2)$$

Bei *LDWADDI* (9.3) wird zu dem durch *RY* adressierten Speicherinhalt eine Konstante *C* hinzuaddiert und das Ergebnis innerhalb von zwei an Stelle von drei Takten in Register *RX* abgelegt.

$$\text{LDWADDI: } RX \leftarrow \text{mem} [RY] + C \quad (9.3)$$

Die *LDWIXW*-Instruktion (9.4) lädt einen Wert der durch die Summe des *RX*-Registerinhalts und den um zwei nach links geschobenen Inhalt des *RY*-Registers adressiert wird aus dem Speicher ins *RX*-Register. Die benötigte Taktzahl wird von ursprünglich drei auf zwei Takte reduziert.

$$\text{LDWIXW: } RX \leftarrow \text{mem} [RX + (RY \ll 2)] \quad (9.4)$$

Innerhalb eines Taktes ermöglicht der *ORSHL8,16,24*-Befehl (9.5) eine bitweise Verschiebung des *RX*-Registerinhaltes mit dem um wahlweise um 8, 16 oder 24 Bit geschobenen Inhalt des *RY*-Registers. Das Ergebnis wird einen Takt schneller als im Original S-Core ins *RX*-Register geschrieben.

$$\text{ORSHL8,16,24: } RX \leftarrow RX \text{ OR } (RY \ll (8,16,24)) \quad (9.5)$$

Die in Abschnitt 6.2.3 ausführlicher diskutierte *XORLDW*-Instruktionserweiterung lädt den unter *RY* adressierten Speicherinhalt und führt mit dem *RX*-Registerinhalt eine *XOR*-Operation durch und speichert das Resultat im *RX*-Register innerhalb von zwei Takten. Dies entspricht einer Reduktion um einen Takt, im Vergleich mit dem unmodifizierten Prozessor.

$$\text{XORLDW: } RX \leftarrow \text{mem} [RY] \text{ XOR } RX \quad (9.6)$$

Die *LDWXORLSL8*-Instruktion schiebt den Inhalt des *RX*-Registers um acht Stellen nach links und führt dann eine *XOR*-Verknüpfung mit dem unter *RY* adressierten Speicherinhalt durch. Das Resultat der insgesamt zwei anstatt vier Takte umfassenden Operation wird in Register *RX* geschrieben.

$$\text{LDWXORLSL8: } RX \leftarrow RX \ll 8 \text{ XOR } \text{mem} [RY] \quad (9.7)$$

## Anhang E (Details zum IP-Headercheck-Hardwarebeschleuniger)

Im Folgenden werden weitere Details zu der entworfenen Prüfsummenberechnungseinheit vorgestellt. Abbildung Anhang E-1 zeigt den internen Aufbau der 32-Bit-Implementierung des IP-Headercheck-Hardwarebeschleunigers.

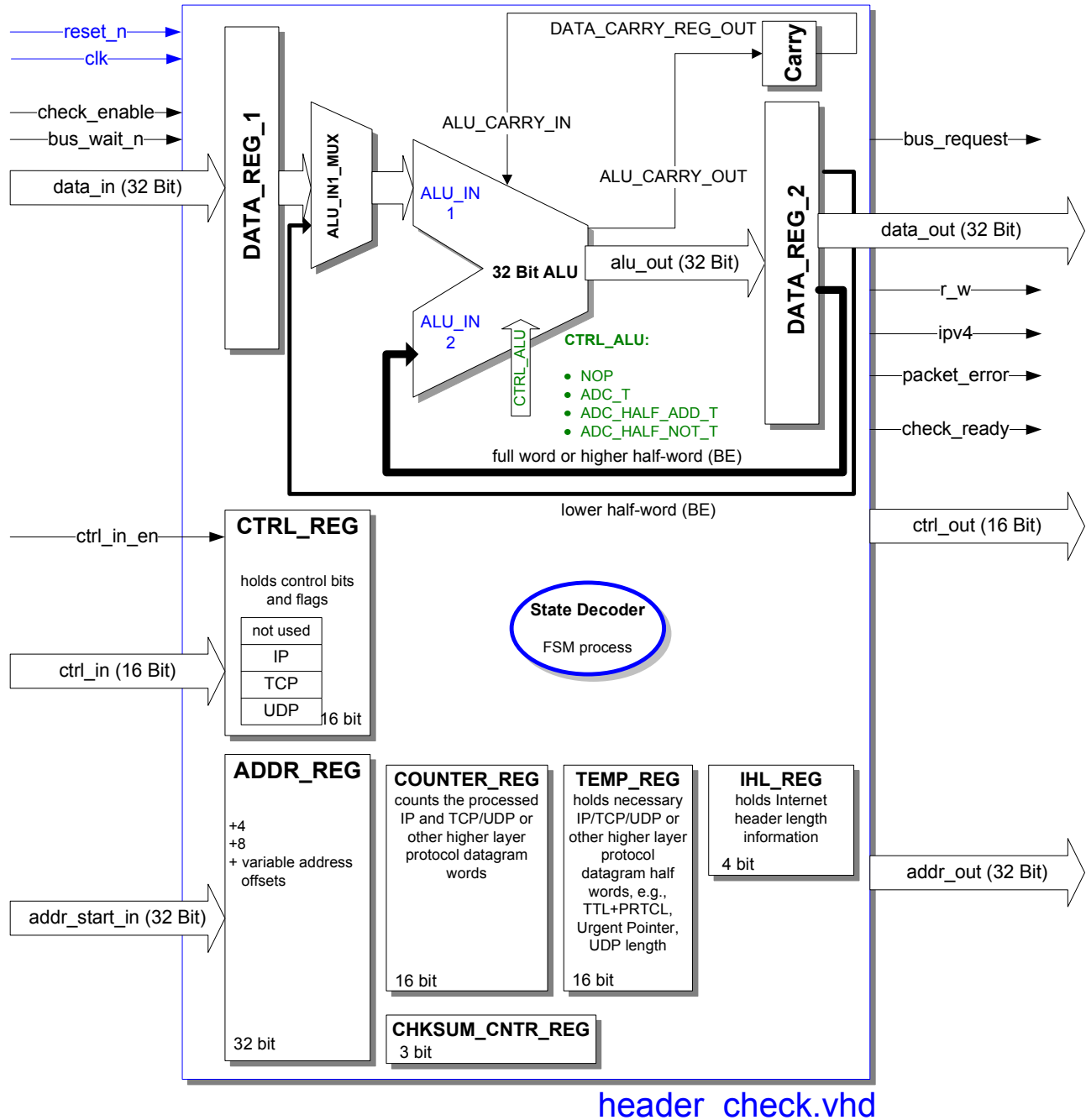


Abbildung Anhang E-1: Blockschaltbild der 32-Bit-Implementierung des Header-/Packetcheck-Hardwarebeschleunigers

Abbildung Anhang E-2 stellt die prinzipielle Funktionsweise des Hardwarebeschleunigers anhand eines Ablaufplans dar.



## Anhang F (IP-DSLAM-Referenzbenchmark)

Abbildung Anhang F-1 zeigt die Anordnung der Tasks des IP-DSLAM-Referenzbenchmarks für die unterschiedlichen möglichen Szenarien (Ethernet / ATM - Linecard / Uplinkcard - Downlink / Uplink).

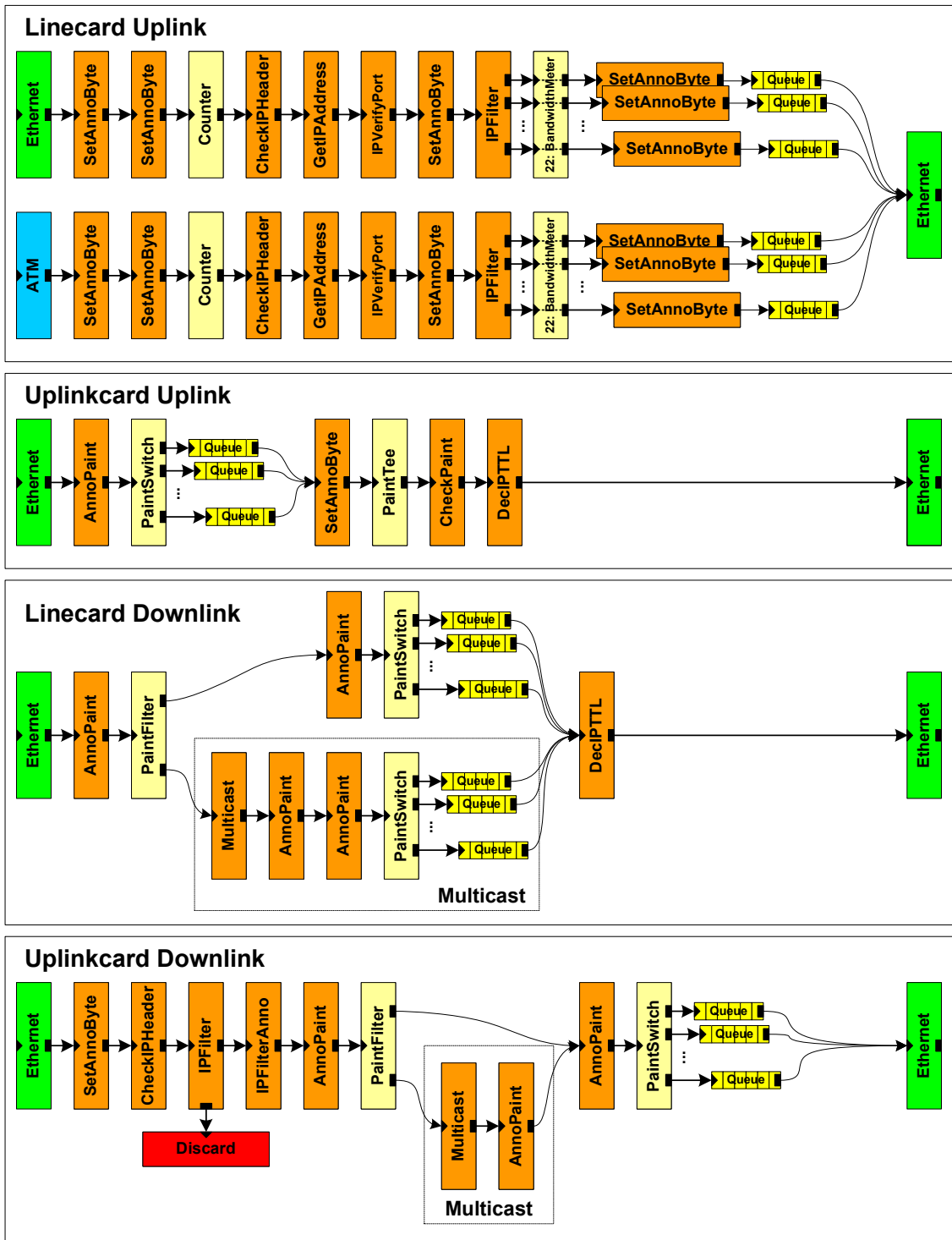


Abbildung Anhang F-1: Tasks der sieben unterschiedlichen IP-DSLAM-Referenzbenchmarkszenarien