Parallele Architekturen für Netzwerkprozessoren*

Dipl.-Ing. Jörg-Christian Niemann, Dr.-Ing. Mario Porrmann, Prof. Dr.-Ing. Ulrich Rückert
Heinz Nixdorf Institut, Universität Paderborn, Deutschland, {niemann, porrmann, rueckert}@hni.upb.de

Kurzfassung

Informationsverarbeitung und Vernetzung von technischen Geräten halten mehr und mehr Einzug in unser tägliches Leben. Um das dabei ständig wachsende Datenaufkommen zu verarbeiten, bedarf es leistungsfähiger Knotenpunkte in Sprach- und Datennetzwerken. In diesem Artikel stellen wir eine Architektur für Netzwerkprozessoren vor, die auf einer homogenen, massiv parallelen Struktur basiert. Sie ist für unterschiedliche Einsatzgebiete skalierbar und durch spezielle Hardwarebeschleuniger für die Zielapplikation optimierbar, um besonders ressourceneffizient zu sein. Wir zeigen Anforderungen für zwei Netzwerk-Anwendungsszenarien auf und stellen die Leistungssteigerung durch die von uns implementierten Hardware-Erweiterungen dar.

1 Einleitung

Mit Begriffen wie *amorphous, nomadic, organic, pervasive* oder *ubiquitous computing* wird derzeit von vielen internationalen Forschungsstätten die Vision einer durchgängig verfügbaren Datenverarbeitung und des damit einhergehenden mühelosen, jederzeitigen Zugriffs auf Informationen und Dienste gekennzeichnet. Ermöglicht wird dieses in erster Linie durch den weiter anhaltenden Fortschritt der Mikro- bzw. Nanotechnologien hin zu immer kleineren, preisgünstigeren und gleichzeitig leistungsfähigeren Geräten. Alltagsgegenstände können so mit Rechen- und Kommunikationsleistung angereichert sowie mit neuen Nutzerschnittstellen ausgestattet werden und fast unscheinbar in unsere berufliche wie private Umwelt zum Nutzen des Menschen integriert werden.
Einige Aspekte dieser technologischen Entwicklung werden im Heinz Nixdorf Institut im Rahmen des Forschungsfeldes Mediatronik behandelt. Unter diesem Kunstwort verstehen wir die situationsbezogene Integration technischer Produkte und Dienste in offene, dynamische Systeme. Ein wichtiges Ziel ist hier die Entwicklung und Bereitstellung von Methoden und Techniken, die es technischen Produkten ermöglichen, effizient zu kommunizieren und zu kooperieren. Wir verfolgen neue Ansätze, die verfügbare Rechenleistung und die Netzwerk-Ressourcen flexibel zu nutzen, um den wachsenden Anforderungen zukünftiger Kommunikationssysteme – z. B. an Übertragungsbandbreite, Dienstgüte und Zuverlässigkeit – optimal gerecht zu werden.
Um das schnell wachsende Datenaufkommen auch in Zukunft bewältigen zu können, bedarf es besonders leistungsfähiger Bausteine, die den Datenverkehr in den Endsystemen und insbesondere in den Knotenpunkten der verschiedenen Netzwerke regeln. In diesem Artikel wollen wir eine flexible, skalierbare Architektur für solche Netzwerkkomponenten vorstellen, die auf dem Konzept massiver Parallelverarbeitung beruht, d. h. eine Vielzahl gleicher Prozessoren verwendet.

2 Netzwerkprozessoren

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 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 Redesigns verbunden. Vor diesem Hintergrund wurden in den letzten Jahren verstärkt so genannte Netzwerkprozessoren entwickelt.

* Die hier beschriebenenen Forschungsergebnisse wurden mit Mitteln des Bundesministeriums für Bildung und Forschung unter dem Förderkennzeichen 01M3062A gefördert. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt bei den Autoren. Diese Arbeit wurde zum Teil unterstützt von Infineon Technologies AG, München, Abteilung CPR ST, Prof. Ramacher.

Netzwerkprozessoren sind programmierbare Spezialbausteine für den Aufbau von Netzwerkkomponenten, die den geringen Preis und die Flexibilität von RISC-Prozessoren mit der Leistungsfähigkeit und Skalierbarkeit von anwendungsspezifischen Bausteinen kombinieren [1]. 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 [2][3]. Obwohl Implementierungen von Netzwerkprozessoren erst seit wenigen Jahren verfügbar sind, gab es bereits im Jahre 2002 über 30 Hersteller von kommerziellen Netzwerkprozessoren [2], die sich in ihrem Aufbau teilweise deutlich unterscheiden, darunter so bekannte Namen wie Intel (IXP2800), IBM (PowerNP) Motorola (C-5) oder Cisco (PXF).

3 Homogene parallele Architektur für Netzwerkprozessoren

Netzwerkdaten sind aufgrund ihrer inhärenten Parallelität prädestiniert für die Verarbeitung auf parallelen Systemen [4]. Diese sind in der Lage, die vielen, oft unkorrelierten Datenströme gleichzeitig zu verarbeiten, wobei sich die globale Zustandsverwaltung meist als einzige gemeinsame Aufgabe (Task) darstellt.

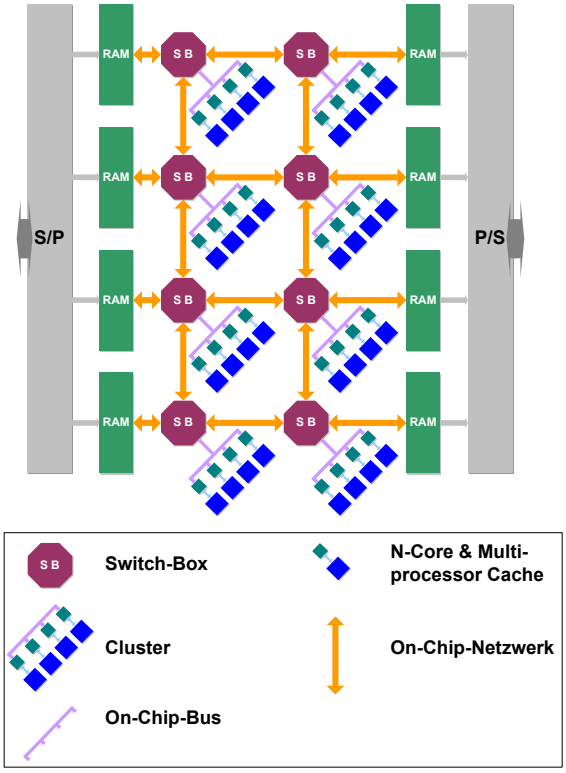


Bild 1: Massiv parallele Systemarchitektur für Netzwerkprozessoren

Die von uns vorgeschlagene Architektur (vgl. Bild 1) basiert auf massiv paralleler Verarbeitung, die durch eine Vielzahl homogener Verarbeitungseinheiten ermöglicht wird. Diese Rechenknoten, die auf einer von uns entwickelten RISC-Architektur basieren, werden in einer hierarchischen System-Topologie über eine leistungsfähige, zweistufige Kommunikationsinfrastruktur miteinander verbunden. Unsere Architektur lässt sich in drei Ebenen unterteilen: *Prozessor-Ebene, Cluster-Ebene* und *SoC(System-on-Chip)-Ebene*. Ein Hauptziel unseres Ansatzes ist, dass der resultierende Netzwerkprozessor in Bezug auf die Anzahl der Cluster, der pro Cluster instanziierten Prozessoren sowie der zur Verfügung gestellten Bandbreite durch die Kommunikationskanäle leicht parametrisierbar sein soll. Auf diese Weise kann eine große Wiederverwendbarkeit dieser Architektur durch Skalierung auf vielfältige Einsatzgebiete gewährleistet werden. Weitere Vorteile einer solchen homogenen Systemarchitektur liegen in dem einheitlichen Programmiermodell [5] und der vereinfachten Testbarkeit und Verifikation. Die drei Ebenen bieten dem Entwickler unterschiedliche Ansatzpunkte zur Optimierung des Systems in Bezug auf den späteren Einsatzzweck.

Auf der *Prozessorebene* stellt ein von uns entwickelter 32-Bit-RISC-Prozessor [6][7] die Kernkomponente des Systems dar. Der N-Core wurde als Soft-Core in der Hardwarebeschreibungssprache VHDL realisiert und kann frei nach den Bedürfnissen des jeweiligen Einsatzgebietes angepasst werden. 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. Weiterhin stellt der N-Core eine Coprozessorschnittstelle für Hardwarebeschleuniger zur Verfügung. Der Prozessorkern wurde in einer aktuellen Standardzellentechnologie implementiert und bei einer Taktrate von 160MHz erfolgreich getestet [6] (Infineon, 130nm, 0,18mm²). Er unterstützt eine byteweise Adressierung des Speicherinhalts, was besonders für Netzwerkapplikationen von Bedeutung ist.

Auf der *Cluster-Ebene* fungieren Switch-Boxen als Hochgeschwindigkeitsroutingknoten [8], die die einzelnen Prozessorcluster miteinander verbinden. Die Datenpakete werden on-chip als Fragmente (Flits) mittels paketorientierter Vermittlung [9] weitergeleitet. Ein lokal angeschlossener Prozessorcluster besteht aus vier Prozessoren, die über einen AMBA-On-Chip-Bus mit der Switch-Box verbunden sind. Aufgrund der generischen Beschreibung dieser On-Chip-Routingknoten in VHDL, die einen frei wählbaren Ausgangsgrad zulässt, ist es uns möglich nahezu beliebige Netztopologien zu implementieren. Für die erste Implementierung des zu fertigenden Chips wurde ein Gitter ausgewählt. Dieses erlaubt sowohl den Parallelbetrieb, als auch ein Pipelining der Prozes-

sorfelder und gewährleistet nahezu gleich lange Verbindungsleitungen und somit identische Signallaufzeiten des Verbindungsnetzwerks zwischen den Switch-Boxen. Die parallele Struktur des Switch-Box-Konzepts erlaubt außerdem eine hohe Fehlertoleranz: Sollte eines der Prozessorfelder ausfallen, können die anderen die anstehenden Aufgaben übernehmen.
Das 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 gesteuert werden.
Das Programmiermodell basiert auf einem parallelisierenden Compiler (auf Clusterebene) und einem überlagerten Werkzeug, das die Abbildung der Anwendung auf die einzelnen Prozessorcluster vornimmt. Als Eingabe für den Compiler dient ANSI-C-Code. Tabelle 1 zeigt eine Flächenabschätzung für die Kernkomponenten des Netzwerkprozessors basierend auf den Syntheseergebnissen für eine 130 nm Standardzellentechnologie. Die Gesamtgröße des hier vorgestellten, 32 Verarbeitungseinheiten umfassenden, Systems liegt bei 29,2 mm², wobei eine Taktfrequenz von 230 MHz (worst case) erreicht wird.

Tabelle 1: Größenangaben des Gesamtsystems für 130nm-UMC-Standardzellentechnologie

SoC-Hauptkomponenten	Fläche [mm²]
32 N-Cores+HW-Beschleuniger [CRC/RAND]	32 x 0,21
8 Switch-Boxen [je 5 Ports]	8 x 0,56
32 MP-Caches, (64K) Mosys 1T-SRAM]	32 x 0,55
8 lokale On-Chip-Busse	8 x 0,05
Gesamt	**29,2**

Wir verfolgen einen hierarchischen, gerichteten Ansatz, um unsere Architektur für den jeweiligen Einsatzzweck zu optimieren. Zeigen die Untersuchungen, dass ein einzelner N-Core-Prozessor für die gegebene Anwendung nicht leistungsfähig genug ist, optimieren wir zunächst die Prozessorarchitektur. Sollte die erzielte Beschleunigung nicht ausreichen, so werden anwendungsbezogene Instruktionssatzerweiterungen und Hardwarebeschleuniger eingesetzt. Ist eine weitere Leistungssteigerung erforderlich, kommen parallele Prozessorfelder zum Einsatz. Im Folgenden wird dieser Ansatz am Beispiel von dedizierten Netzwerkanwendungen dargestellt.

4 Leistungsfähigkeit des Prozessor-Clusters

In einem ersten Schritt wird die Leistungsfähigkeit eines parallelen Systems, basierend auf der N-Core-Verarbeitungseinheit, für zwei unterschiedliche Szenarios aus dem Netzwerkbereich anhand von Benchmarkergebnissen abgeschätzt. Das erste Szenario stellt ein typisches Beispiel aus dem Zugangsbereich des Netzwerks dar, bei der der Rechenleistungsbedarf für die Anbindung und Aggregation von DSL(Digital Subscriber Line)-Anschlüssen ermittelt wird. Bei dem zweiten Anwendungsbeispiel wird die notwendige Performanz eines Systems für den Einsatz im Kernnetzwerk bzw. Edge-Netzwerk bestimmt.

a) DSLAM-Anwendungsszenario
Eines der Einsatzgebiete für unseren Netzwerkprozessor ist die Anbindung der „letzten Meile“, bei der es um die Bündelung bzw. Verteilung von DSL-Anschlüssen geht (vgl. Bild 2). Die hierfür benötigten Knoten, die im *Access*-Bereich des Netzwerks anzusiedeln sind, nennt man DSLAMs (DSL Access Multiplexer).

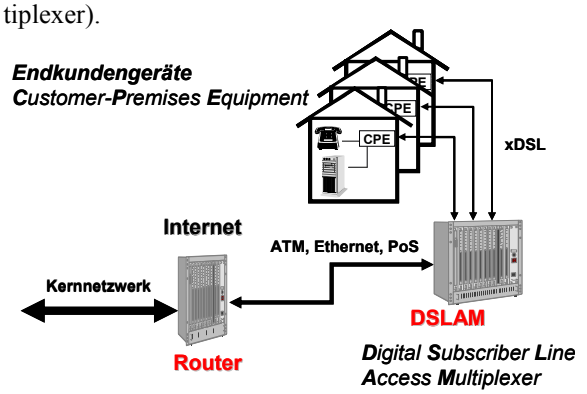


Bild 2: DSLAM-Anwendungsszenario

Hauptkomponenten eines DSLAMs sind heute bis zu 64 Linecards, die jeweils bis zu 96 DSL-Ports zur Verfügung stellen, und eine Uplinkcard, die den Verkehr zum Internet-Service-Provider (ISP) regelt. Grundlegende Funktionen, die in diesem Anwendungsfeld auf den Paketen ausgeführt werden, sind z. B.: Parsing, Headercheck, Classification, Multicast-Duplication, Policing, Conditioning, AAL5, CRC und Ethernet-Framing. Wir haben die zuvor genannten Funktionalitäten in C beschrieben und auf unserem N-Core abgebildet. Die hieraus entstandenen Messergebnisse (vgl. [10]) bei einem zugrunde liegenden iMix-Workload-Modell zeigen, dass auf der Linecard 1,17 Takte/Bit für den Upstream und 1,26 Takte/Bit für den Downstream benötigt werden. Bei einer Taktfrequenz von 230MHz und einer Ausbaustufe von 96 Ports pro Linecard würden bei einem Worst-Case-Szenario folgende Systeme benötigt (vgl. Tabelle 2):

Tabelle 2: Rechenleistung für ein DSLAM-Szenario

DSL-Variante, UL-/DL-Datenrate[Mbps]	N-Cores pro Linecard	
	Upstream	Downstream
ADSL (0,8/8)	1	5
VDSL (3/22)	2	13
SHDSL (2/2)	1	2

Hieraus wird ersichtlich, dass oben vorgestellte SoC basierend auf 32 N-Cores für das Worst-Case-Szenario vollkommen ausreichen würde. Für den Einsatz auf der Uplinkcard wird allerdings deutlich mehr Rechenleistung benötigt, für die zukünftige Realisierungen unseres Chips mit weitaus mehr Verarbeitungseinheiten Verwendung finden könnten. Durch Optimierung der einzelnen Komponenten wird im Anschluss versucht ein möglichst ressourceneffizientes System zu realisieren (vgl. Kapitel 5).

b) Router-Anwendungsszenario

In diesem Abschnitt wird ein Anwendungsbeispiel aus dem Kernnetzwerk bzw. Edge-Netzwerk zur Bewertung des Systems, basierend auf den Leistungsdaten der N-Core-Verarbeitungseinheit, vorgestellt. Hierzu wurden in Anlehnung an die vom EEMBC-Benchmark-Konsortium [11] definierten Netzwerkbenchmarks drei charakteristische Funktionen von Routern (OSPF-Routing, Packet-Flow und Route-Lookup) modelliert und auf die Zielarchitektur abgebildet. Diese stellen wesentliche Funktionen eines aktuellen Routers dar, die durch zusätzliche Funktionalitäten wie zum Beispiel Network Address Translation (NAT) oder Quality of Service (QoS) ergänzt werden können.

Der Router in dem hier betrachteten Szenario (vgl. Bild 3) 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.

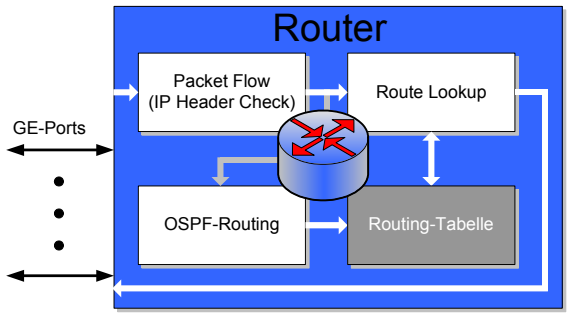


Bild 3: Edge-Router-Szenario

Die eintreffenden IP-Pakete (eingefasst in zufallsverteilte Ethernetpakete minimaler/maximaler Länge) werden durch den Block *Packet Flow* zunächst auf Korrektheit geprüft (*IP-Header-Check*). 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 Routing-Tabelle aktualisiert. Handelt es sich bei dem eingehenden IP-Paket um ein Datenpaket, so übernimmt der *Route-Lookup*-Block anhand eines *Patricia-Trie*-Algorithmusses 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. 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 (bei einer vorinitialisierten Liste von 400 Netzwerkknoten mit je 4 Verbindungen pro Knoten) pro Sekunde erreicht. Heutige Router aktualisieren ihre Routng-Tabelle bis zu 300 mal pro Sekunde [12], so dass hier ein N-Core ausreicht. Tabelle 3 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 8 GE Ports im Vollastbetrieb mindestens 26 dieser Verarbeitungseinheiten zur alleinigen Abwicklung des Protokollstapels benötigt werden.

Tabelle 3: Rechenleistung des N-Cores für ein Edge-Router-Szenario

Funktion	Leistung pro N-Core	Benötigte N-Cores
Packet Flow [Pakete/s]	900.262	14
Route Lookup [Lookups/s]	1.173.000	11
OSPF-Routing [Updates/s]	317	1

Im folgenden Kapitel wird aufgezeigt, wie wir zum einen die Rechenleistung der einzelnen Verarbeitungseinheit für dedizierte Anwendungen optimieren können, und wie zum anderen die Funktion des Gesamtsystems für die beiden Anwendungsbeispiele mit Hilfe eines Simulationsmodells sowohl für die Realisierung der Hardware als auch für die Implementierung der Software überprüft werden kann.

5 Modellierung des Gesamtsystems und Optimierung

Schon vor der eigentlichen Fertigung des integrierten Schaltkreises muss sichergestellt sein, dass sowohl die Hardware als auch die eingesetzte Software die Spezifikation erfüllen und ordnungsgemäß zusammen arbeiten. Durch Hardware-/Software-Cosimulation – zum Beispiel durch Modellierung des Systems in SystemC – kann bereits im frühen Entwurfsstadium die einzusetzende Software entwickelt und auf Korrektheit geprüft werden. Außerdem lassen sich so schnell komplexe Testumgebungen modellieren, die das System in das umgebende System einbetten. Die Vorgabe, ein möglichst ressourceneffizientes System zu realisieren, das bei möglichst geringer Fläche maximale Leistung und zudem noch Flexibilität für sich verändernde Anforderungen seitens der Anwendungen liefert, erfordert eine Optimierung der einzelnen Sys-

temkomponenten und ggf. des Gesamtsystems für die jeweilige Anwendungsklasse.

a) Modellierung

Um genauere Aussagen über die Leistungsfähigkeit unserer Multiprozessor-Architektur treffen zu können, die über die Ergebnisse zur Leistungsfähigkeit der einzelnen Prozessorkerne und Hardwarebeschleuniger hinausgehen, wurde ein zyklenakkurates Modell unseres Systems in SystemC [12] modelliert (vgl. Bild 4). Dieses entspricht in seinem Aufbau der in Bild 1 gezeigten Struktur. Zunächst wurde ein Prozessor basierend auf einem automatisch generierten Simulationsmodell [5] in eine SystemC-Struktur eingebunden. Darauf aufbauend wurde ein Prozessorcluster mit vier N-Core-CPUs, Cache 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.

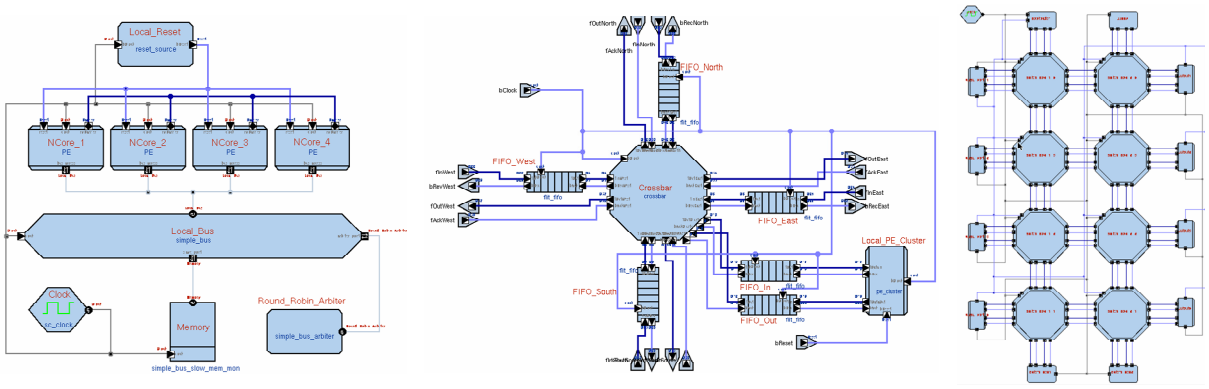
Die Simulation ist immer noch zyklenakkurat und zudem deutlich schneller als eine vergleichbare Simulation auf Basis der VHDL-Beschreibung der Einzelkomponenten. Falls erforderlich, können die einzelnen Hardwarebeschleuniger zur Verifikation des Zusammenspiels mit der Software in einer Cosimulation als VHDL-Beschreibung mit eingebunden werden. Um eine höhere Simulationsgeschwindigkeit zu erreichen, werden diese ansonsten durch funktionale Beschreibungen in C++ bzw. SystemC ersetzt.

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 On-Chip-Netzwerk von einem zentralen *Bootloader* aus an die entsprechenden Speicheradressen gesendet werden. Im Anschluss wird das umgebende System aktiviert, also zum Beispiel Paketgeneratoren, die die eigentlichen Nutzdaten an die Eingangsports des Systems liefern. Die Synopsys CoCentric-Entwicklungsumgebung stellt eine Vielzahl von Bibliothekselementen zur Verfügung, so zum Beispiel auch ADSL-Kanäle, die zur Modellierung der Übertragungsstrecke der Endkunden hin zum DSLAM (vgl. Abschnitt 4 a)) dienen.

Mit Hilfe unserer Systemmodellierung werden wir die oben vorgestellten Netzwerk-Szenarien näher evaluieren und die Hardware sowie die Software für das gesamte System testen und optimieren.

b) Optimierung

Ein Kernpunkt unseres hierarchischen, gerichteten Ansatzes ist die Optimierung der Systemarchitektur im Hinblick auf die betrachtete Anwendungsklasse. Angefangen bei der Architekturoptimierung des Prozessorelements selbst, zum Beispiel durch Hinzufügen von zusätzlichen Pipelinestufen, über Instruktionssatzerweiterungen, bis hin zu eng und lose gekoppelten Hardwarebeschleunigern. Während eng gekoppelte Hardwarebeschleuniger von der CPU angesteuert werden, übernehmen die lose gekoppelten Hardwarebeschleuniger unabhängig vom Prozessor eine Datenvor- oder Nachbearbeitung. Tabelle 4 zeigt eine Auswahl der erreichten Beschleunigungen durch die von uns implementierten Befehlssatzerweiterungen und Hardwareblöcke, welche vorrangig für sicherheitsrelevante Applikationen im Netzwerkverkehr Einsatz finden, die besonders rechenintensiv sind. In unserer Werkzeugkette werden Befehlssatzerweiterungen für den Anwender transparent vom Compiler ausgenutzt. Hierfür werden die zukünftigen Anwendungen zunächst auf oft wiederkehrende Befehlspaare untersucht, die während der Ausführung des Programms aufeinander folgen und auf gleichen Daten operieren. Die Befehlspaare mit dem höchsten Beschleunigungspotential werden anschließend zu so genannten „Superbefehlen“ zusammengefasst und in Hardware umgesetzt [7][14]. Der hier aufgeführte Befehl *XORLDW* fasst die beiden Einzeloperationen



Prozessorfeld Switch-Box Gesamtsystem

Bild 4: SystemC-Modellierungsebenen eingebunden in die Simulationsumgebung CoCentric von Synopsys

LDW und *XOR* des Originalprozessorkerns zusammen. D. h. es wird ein Wort aus dem Speicher geladen und direkt im Anschluss eine XOR-Verknüpfung mit einem weiteren Registerinhalt durchgeführt. Zur Realisierung dieses „Superbefehls“ mussten lediglich zusätzliche Kontrollstrukturen im Instruktionsdekoder des N-Cores eingefügt werden, da die benötigte Logik bereits vorhanden war. Durch diese Instruktion wird eine Beschleunigung von ca. 10% bei Verschlüsselungsverfahren und von ca. 25% bei Prüfsummenbildungen, wie zum Beispiel beim CRC (Cyclic Redundancy Check), erzielt.

Tabelle 4: Erweiterungen zur Performanzsteigerung des Gesamtsystems (130nm-UMC-Technologie)

Erweiterung	Beschleunigung	Fläche [mm²]
N-Core-Architekturoptimierung	1,15	+ 0
N-Core-Strukturerweiterung (2 zus. Pipelinestufen)	1,52	+ 0,021
XORLDW-Befehlssatzerweiterung	1,1 bis 1,25	+ 0,00013
CRC-Hardwarebeschleuniger	15	+ 0,014
AES-Encryption-Hardwarebeschleuniger	160	+ 0,124
AES-Decryption-Hardwarebeschleuniger	175	+ 0,197

Hardwarebeschleuniger, die nächste Stufe der Erweiterung, erzielen deutlich höhere Beschleunigungen, benötigen jedoch weitaus mehr Fläche und Entwicklungsaufwand. Mit der eng gekoppelten CRC-Hardware-Erweiterung [15] erzielen wir eine Beschleunigung um den Faktor 15 gegenüber einer reinen Software-Implementierung. Noch höhere Leistungsgewinne werden durch die lose gekoppelten AES(Advanced Encryption Standard)-Blöcke erzielt.

6 Zusammenfassung

Die von uns präsentierte Systemarchitektur stellt eine skalierbare und – durch unsere Werkzeugkette unterstützt – optimierbare, ressourceneffiziente Plattform für Netzwerkanwendungen dar. Die Leistungsfähigkeit, des von uns, speziell für Netzwerkanwendungen, entwickelten Prozessorkerns N-Core wurde für zwei dedizierte Anwendungsbeispiele bestimmt. Durch Optimierung der Architektur, Erweiterungen des Befehlssatzes und Hinzufügen von anwendungsspezifischen Hardwarebeschleunigern haben wir im nächsten Schritt die Performanz des Gesamtsystems erhöht. Mit Hilfe unserer Systemsimulationsumgebung sind wir in der Lage, eine frühe Verifikation von Hardware und Software durchzuführen und die vorgenommenen Optimierungen zu bewerten. In Zukunft werden wir zusätzliche Befehlssatzerweiterungen und Hardwarebeschleuniger integrieren, die die Leistungsfähigkeit der Architektur für Netzwerkanwendungen, über die hier vorgestellten Erweiterungen hinaus, erhöhen.

7 Literatur

[1] D. E. Comer. Network Systems Design using Network Processors. Pearson Education Inc. 2004.

[2] J. Freeman. An Industry Analyst’s Perspective on Network Processors. Network Processor Design, Vol. 1, pp 191-218. Morgan Kaufmann Publishers, 2002.

[3] M. Jain et al. ASIP Design Methodologies: Survey and Issues. In Proc. 14th Int. Conf. on VLSI Design, pp 76-81, 2001.

[4] B. Carlson. Intel® Internet Exchange Architecture & Applications. A Practical Guide to IXP2XXX Network Processors. Intel Press, 2003.

[5] J.-C. Niemann et al. A holistic methodology for network processor design. In Proc. of the Workshop on High-Speed Local Networks, 28th IEEE Conference on Local Computer Networks, pp 583-592, 2003.

[6] 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), 2002

[7] J.-C. Niemann et al. Network Application Driven Instruction Set Extensions for Embedded Processing Clusters. In Proc. PARELEC 2004, International Conference on Parallel Computing in Electrical Engineering, Dresden, Germany, 2004.

[8] A. Brinkmann, J.-C. Niemann, I. Hehemann, D. Langen, M. Porrmann, U. Rückert. On-Chip Interconnects for Next Generation System-on-Chips. In Proc. of the 15th Annual IEEE International ASIC/SOC Conference. Rochester, NY, USA, 2002, pp 212-215.

[9] W. J. Dally, B. Towles. Route packets, not wires: On-chip interconnection networks. In Proc. of DAC 2001, 2001, pp 684–689.

[10] G. Hagen, J.-C. Niemann, M. Porrmann, C. Sauer, A. Slowik, M. Thies. Developing an IP-DSLAM Benchmark for Network Processor Units. In Proc. of ANCHOR 2004, Advanced Networking and Communications Hardware Workshop, Munich, Germany, 2004.

[11] EEMBC, the embedded microprocessor benchmark consortium. http://www.eembc.org.

[12] Agilent Technologies application note. The router performance testing, 2000.

[13] T. Grotker, S. Liao, G. Martin, S. Swan. System Design with SystemC. Kluwer Academic Publishers, 2002.

[14] U. Kastens, D. K. Le, A. Slowik, M Thies. Feedback Driven Instruction-Set Extension. In Proc. of ACM SIGPLAN/SIGBED 2004 Conf. on Languages, Compilers, and Tools for Embedded Systems, 2004.

[15] M. Grünewald, J.-C. Niemann, and U. Rückert. A performance evaluation method for optimizing embedded applications. In Proc. of the 3rd IEEE International Workshop on System-On-Chip for Real-Time Applications, pp 10-15, Calgary, Alberta, Canada, 2003.