Architektur für flexible Paketverarbeitung in Hochgeschwindigkeitskommunikationsnetzen

Von der Fakultät für Informatik, Elektrotechnik und Informationstechnik
der Universität Stuttgart zur Erlangung der Würde
eines Doktor-Ingenieurs (Dr.-Ing.) genehmigte Abhandlung

vorgelegt von

Simon Rüdiger Berthold Hauger

geb. in Filderstadt

Hauptberichter:	Prof. em. Dr.-Ing. Dr. h. c. mult. Paul J. Kühn
1. Mitberichter:	Prof. Dr. sc. techn. Andreas Herkersdorf, TU München
2. Mitberichter:	Prof. Dr.-Ing. Andreas Kirstädter
Tag der Einreichung:	29. Oktober 2010
Tag der mündlichen Prüfung:	19. Juli 2011

Institut für Kommunikationsnetze und Rechnersysteme
der Universität Stuttgart

2011

Kurzfassung

Die wirtschaftliche und gesellschaftliche Bedeutung globaler Datennetze und insbesondere des Internets wächst unablässig. Als Folge davon steigt das zu übertragende Datenvolumen, so dass immer höhere Datenübertragungsraten eingesetzt werden. Folge und zugleich auch Ursache dieser Entwicklung ist, dass Netzbetreiber und Kommunikationsdienstleistungsanbieter eine ständig zunehmende Zahl und Vielfalt von Anwendungen, Diensten und Protokollen über diese Netze bereit stellen. Dies stellt hohe Anforderungen an die Komponenten dieser Netze – insbesondere an die Vermittlungsknoten in und am Rand von Metro- und Kernnetzen.

Der Transport der zu übertragenden Daten in globalen Kommunikationsnetzen erfolgt – wie im Internet – in zunehmendem Maße paketvermittelt. Die Vermittlungsknoten verarbeiten jedes ankommende Paket und leiten es in Richtung seines Ziels weiter. Aufgrund der zunehmenden Datenraten und den ständig neuen oder geänderten Diensten und Protokollen stellen *Leistungsfähigkeit* und *Flexibilität* die wichtigsten Anforderungen für zukünftige Paketverarbeitungssysteme in Vermittlungsknoten dar. Sie sollen einerseits eine hohe Verarbeitungsrate garantieren und andererseits einfach und schnell an neue Verarbeitungsfunktionen und sonstige Anforderungen angepasst werden können.

Diese Dissertation untersucht diese Anforderungen sowie bestehende Lösungsansätze für Paketverarbeitungssysteme in Metro- und Kernnetzen und entwickelt daraus eine neue Architektur, die sowohl die Leistungsfähigkeit als auch die Flexibilität maximiert. Ein Prototyp und die beispielhafte Umsetzung verschiedener Protokollverarbeitungsfunktionen zeigen die Realisierbarkeit und Anwendbarkeit eines solchen Systems mit deterministischem Durchsatz.

Nach einer Einführung in diese Arbeit beschreibt Kapitel 2 zunächst die Grundlagen paketvermittelnder Netze. Hierbei wird sowohl auf die wichtigsten aktuellen und zukünftigen Netztechnologien und Protokolle auf Schicht 2 bis 4 als auch auf die Funktionsweise und Architektur von Hochgeschwindigkeitsvermittlungsknoten eingegangen. Hieraus werden die Anforderungen an zukünftige Paketverarbeitungssysteme bezüglich Funktionalität, Leistungsfähigkeit und Flexibilität abgeleitet.

Mögliche geeignete Bauelemente sind FPGAs (Field Programmable Gate Arrays) und Netzprozessoren. Aufgrund ihrer unterschiedlichen Entwurfsmethodik und Hardware-Infrastruktur weisen beide Bauelemente unterschiedliche Vor- und Nachteile sowohl bezüglich ihrer Flexibilität als auch bezüglich ihrer Leistungsfähigkeit auf. Kapitel 2 schließt mit einer Vorstellung und Diskussion aktueller und zukünftiger Paketverarbeitungssysteme auf Basis von FPGAs und Netzprozessoren aus Industrie und Forschung.

Zur Herleitung und Motivation der in dieser Arbeit vorgestellten neuen Architektur werden zu Beginn von Kapitel 3 zunächst die unterschiedlichen Hardware- und Software-Entwurfsebenen digitaler Systeme analysiert, insbesondere für die Realisierung von Systemen auf Basis von FPGAs und Netzprozessoren. Je nach den spezifischen Anforderungen ist der Hardware- oder der Software-Entwurf zu bevorzugen. Außerdem werden die beiden prinzipiellen Anordnungen von Verarbeitungseinheiten für die Parallelverarbeitung auf funktionaler Ebene untersucht: Die Anordnung als Pool und als Pipeline. Beide werden hinsichtlich Durchsatz, Implementierungsaufwand und Ressourcenbedarf analysiert und verglichen. Als Ergebnis dieser Betrachtungen zeigt sich, dass die Verarbeitung in einer Pipeline-Anordnung vorzuziehen ist, falls ein garantierter hoher Durchsatz bei hoher Modularität und geringem Ressourcenbedarf gefordert wird.

Die Zielsetzung beim Entwurf der neuen Architektur war, die Leistungsfähigkeit und die Flexibilität zu maximieren. Die Architektur basiert auf einer Pipeline-Struktur, die einen deterministischen Durchsatz von mindestens einem Paket minimaler Größe pro Taktperiode garantiert. Die Pipeline besteht aus Verarbeitungsmodulen, die diese Durchsatzanforderungen einhalten, intern jedoch beliebig aufgebaut sein können. Somit kann man sowohl Module mit vielen parallelen Prozessorkernen als auch Module basierend auf programmierbarer Logik einsetzen. Damit hat man die Möglichkeit, je nach den konkreten Anforderungen einer Verarbeitungsfunktion, diese auf Software-Ebene oder auch auf Register-Transfer-Ebene zu realisieren. Zusätzlich ist es möglich, Module mit vorgefertigten Funktionen von Universitäten oder Firmen als Open-Source- oder Intellectual-Property-Module zu integrieren.

Die Architekturen verschiedener Verarbeitungsmodule werden diskutiert und beispielhafte Implementierungen eines Prozessor-Pool-Moduls und eines Datenfluss-Pipeline-Moduls vorgestellt. Das Prozessor-Pool-Modul enthält einen Pool von kleinen RISC-Prozessorkernen mit einem für die Paketverarbeitung spezialisierten Befehlssatz. Das Datenfluss-Pipeline-Modul basiert auf einer Pipeline von Verarbeitungseinheiten, die Software-gesteuert jeweils einen Befehl auf den durchwandernden Datenworten ausführen. Außerdem werden Anwendungsszenarien gezeigt, wie man ein Paketverarbeitungssystem mit der neuen Architektur flexibel und entsprechend der jeweiligen Anforderungen anpassen kann und wie geeignete Software-Werkzeuge diesen Prozess unterstützen können.

Die Realisierbarkeit und Anwendbarkeit der neuen Architektur beweisen eine prototypische Implementierung eines entsprechenden Paketverarbeitungssystems sowie die Umsetzung verschiedener Protokollverarbeitungsfunktionen in Kapitel 4. Der Prototyp wurde in VHDL beschrieben, für verschiedene FPGA-Familien synthetisiert und auf der Universellen Hardware-Plattform im Institut des Autors testweise betrieben. Funktionale Simulationen und Tests validieren das korrekte Verhalten des Prototyps. Die Place & Route-Ergebnisse zeigen, dass ein solches System mit aktueller Technologie realisierbar und auf 100 Gbit/s-Line-Cards mit deterministischem Durchsatz einsetzbar ist. Um außerdem die Anwendbarkeit der verschiedenen Verarbeitungsmodule zu demonstrieren, wird abschließend die Umsetzung der Fast-Path-Funktionen eines IPv4-Routers sowie von Funktionen möglicher zukünftiger Protokollmechanismen für Router-unterstützte Überlastregelung gezeigt. Hierbei wurden die Funktionen in Software und in programmierbarer Logik unter Einsatz dreier verschiedener Modularten realisiert.

Zusammengefasst stellt diese Dissertation eine modulare Architektur für die Paketverarbeitung in Vermittlungsknoten vor, die einen hohen Durchsatz garantiert und den flexiblen Entwurf von Verarbeitungsfunktionen auf einer jeweils passenden Abstraktionsebene erlaubt.

Abstract

Architecture for Flexible Packet Processing in High-Speed Communication Networks

The economic and societal significance of global data networks—especially of the Internet—grows and grows. As a result, the data volume that has to be transmitted increases, so that ever higher data transmission rates are employed. Consequence and also cause of this development is that network operators and communication service providers offer more and more innovative applications, services and protocols over these networks. This places high demands on the components of these networks—particularly on the switching nodes of metro and core networks.

Data transport in global communication networks is increasingly performed *packet-switched*—like in the Internet. The switching nodes, called routers or switches, process each incoming packet and forward it towards its destination. Due to the ever increasing data rates and the continuous introduction and modification of services and protocols the most important requirements for future packet processing systems in switching nodes are *throughput* and *flexibility*. These systems have to guarantee a high processing rate and they must be adaptable to new processing functions and other requirements fast and easily.

This thesis analyses these requirements as well as existing approaches for packet processing systems in metro and core networks and develops a new architecture that maximizes both throughput and flexibility. A prototype and the exemplary implementation of various protocol processing functions show the feasibility and applicability of such a system with deterministic throughput.

After an introduction, Chapter 2 first describes the basics of packet-switching networks. It addresses the most important current and future networking technologies and protocols on layer two to four as well as the tasks and the architecture of high-speed routers and switches. This leads to the derivation of the requirements for future packet processing systems in terms of functionality, performance and flexibility.

Suitable devices can be FPGAs (field programmable gate arrays) and network processors. Due to their different design methodology and hardware infrastructure these devices have distinct advantages and disadvantages with respect to both their flexibility and their performance. Chapter 2 closes with a presentation and a discussion of current and future packet processing systems based on FPGAs and network processors from industry and academia.

For motivating the new architecture, that is presented in this thesis, the different hardware and software levels of abstraction for the design of digital systems, in particular of systems based on FPGAs and network processors, are analysed at the beginning of Chapter 3. Depending on the specific requirements the hardware design or the software design is to be preferred. Moreover, the two basic arrangements of processing units for parallel processing are studied: the arrangement as a pool and as a pipeline. Both are analysed and compared with respect to throughput, implementation cost and resource requirements. The results of these investigations show that processing in a pipeline arrangement is to be preferred, if guaranteed high throughput, high modularity and low resource requirements are needed.

The objective when designing the new architecture was to maximize throughput and flexibility. The architecture is based on a pipeline structure that guarantees a deterministic processing rate of at least one minimum-sized packet per clock cycle. The pipeline consists of processing modules that can be built up arbitrarily, as long as they meet the pipeline's throughput requirements. So, one can make use of modules with many parallel processor cores and of modules based on programmable logic. Depending on the specific requirements of a processing function one has thus the possibility to realize it at software level or at register-transfer level. Additionally, it is possible to integrate modules with pre-built functions by universities or companies as Open-Source modules or Intellectual-Property modules.

The architectures of various processing modules are discussed and examplary implementations of a processor-pool-module and a dataflow-pipeline-module are presented. The processor-pool-module contains a pool of small RISC processor cores with an instruction set specialized for packet processing. The dataflow-pipeline-module is based on a pipeline of processing units, that each executes on instruction on each data word that passes through. Moreover, application scenarios show how one can flexibly adapt a packet processing system of the new architecture corresponding to the respective requirements, and how appropriate software tools can support that process.

The feasibility and applicability of the new architecture are shown by a prototypical realization of such a packet processing system and the implementation of various protocol processing functions in Chapter 4. The prototype was implemented in VHDL, synthesized for different FPGA families and operated for testing purposes on the universal hardware platform in the author's institute. Functional simulations and tests validate the correct behavior of the prototype. The Place & Route results show that such a system is feasible with current technology and can be used on 100 Gbit/s line cards with deterministic throughput. To demonstrate the applicability of the architecture with its various processing modules, a description of the implementation of the fast path functions of an IPv4 router as well as functions of possible future protocol mechanisms for router-assisted congestion control follows. Here, the functions have been realized in software and in programmable logic using three different types of processing modules.

In summary, this thesis presents a modular architecture for packet processing in switching nodes, which guarantees high throughput and enables the flexible design of processing functions at a problem-specific, suitable level of abstraction.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen und Symbole

Abkürzungen

ALM	Adaptive Lookup Modules	148
ALU	Arithmetic-Logic Unit	121
ALUT	Adaptive LUT	148
ARM	Acorn RISC Machine; *auch*: Advanced RISC Machine	51
ARP	Address Resolution Protocol	145
ARPA	Advance Research Projects Agency	17
ASIC	Application Specific Integrated Circuit	38
B-VID	Backbone-VID	24
Bit	Binary Digit	77
C-VID	Customer-VID	24
CAM	Content Addressable Memory	51
CIDR	Classless Inter-Domain Routing	19
CRS	Carrier Routing System	56
DMA	Direct Memory Access	135
DRAM	Dynamic Random Access Memory	47
DSL	Digital Subscriber Line	14
ECC	Error-Correcting Codes	11
ECN	Explicit Congestion Notification	23
FCS	Frame Check Sequence	16

FEDERICA	Federated E-infrastructure Dedicated to European Researchers Innovating in Computing network Architectures	28
FPGA	Field Programmable Gate Array	38
G-LAB	German Lab	28
Gbit/s	Gigabit pro Sekunde	14
GENI	Global Environment for Network Innovations	28
GMII	Gigabit Media Independent Interface	139
GP	Ganzes-Paket-Variante der neuen Architektur	100
GPP	General Purpose Processor	39
HDL	Hardware Description Language	41
I-VID	Backbone-Service-Instance-VID	24
IC	Integrated Circuit	39
ICMP	Internet Control Message Protocol	145
IEC	International Electrotechnical Commission	12
IEEE	Institute of Electrical and Electronics Engineers	15
IETF	Internet Engineering Task Force	12
IFG	Inter-Frame Gap	16
IP	Internet Protocol	17
IPC	Instructions Per Clock cycle	119
ISO	International Organization for Standardization	12
ITU	International Telecommunication Union	24
ITU-T	Telecommunication Standardization Sector of the ITU	24
JTAG	Joint Test Action Group	135
LAN	Local Area Network	13
LUT	Lookup Table	41
Mbit/s	Megabit pro Sekunde	14
MP/s	Millionen Pakete pro Sekunde	36
MPLS	Multi-Protocol Label Switching	20
MPLS-TP	Multi-Protocol Label Switching – Transport Profile	23

NAT	Network Adress Translation	21
NH	Nur-Header-Variante der neuen Architektur	100
NP	Network Processor	38
OAM	Operation, Administration and Maintenance	24
OSI	Open Systems Interconnnection	12
P/s	Pakete pro Sekunde	35
PB	Provider Bridge	24
PBB	Provider Backbone Bridge	24
PBB-TE	Provider Backbone Bridging – Traffic Engineering	23
PCI	Peripheral Component Interconnect	70
PCS	Physical Coding Sublayer	61
PHY	Physical Layer (Schicht 1)	29
PLL	Phase-Locked Loop	42
PMA	Physical Medium Attachment Sublayer	61
QoS	Quality of Service	18
RCP	Rate Control Protocol	26
RISC	Reduced Instruction Set Computer	37
ROM	Read-Only-Memory	142
RPC	Remote Procedure Call	71
RTL	Register-Transfer Level	77
RTT	Round Trip Time	163
S-VID	Service-VID	24
SDH	Synchronous Digital Hierarchy	14
SFD	Start Frame Delimiter	16
SIMD	Single-Instruction-Multiple-Data	66
SONET	Synchronous Optical Network	14
SOPC	System On a Programmable Chip	43
SPI	System Packet Interface	47

SRAM	Static Random Access Memory	42
Tbit/s	Terabit pro Sekunde	9
TCAM	Ternary Content Addressable Memory	34
TCP	Transmission Control Protocol	21
UART	Universal Asynchronous Receiver/Transmitter	136
UDP	User Datagram Protocol	20
UHP	Universelle Hardware-Plattform	135
VHDL	VHSIC Hardware Description Language	44
VHSIC	Very High Speed Integrated Circuit	44
VID	VLAN-Identifier	24
VLAN	Virtual Local Area Network	23
VPN	Virtual Private Network	20
WAN	Wide Area Network	14
WDM	Wavelength Division Multiplex	9
WLAN	Wireless Local Area Network	9
XAUI	10 Gigabit Attachment Unit Interface	47
XCP	Explicit Control Protocol	26

Symbole

D	Datenrate	35
f	Taktfrequenz	98
G	Abstand zwischen zwei Rahmen oder Paketen	16
G_{min}	minimaler Abstand zwischen zwei Rahmen oder Paketen	16
L	Paketgröße	100
L_{min}	minimale Paketgröße	36
N	Anzahl der Verarbeitungseinheiten	88
p	Anzahl der Verarbeitungseinheiten (Prozessorkerne) in einem Modul	109
P	Paketrate	35
P_{max}	maximale Paketrate	36
$P_{Pipeline}$	Paketrate bei Pipeline-Anordnung	89
$P_{Pipeline,simpl}$	Paketrate bei idealisierter Pipeline-Anordnung	88
$P_{Pipeline,spez}$	Paketrate bei Pipeline mit spezialisierten Verarbeitungseinheiten	90
$\overline{P}_{Pool}$	Mittlere Paketrate bei Pool-Anordnung	90
P_{Pool}	Paketrate bei Pool-Anordnung	89
$P_{Pool,simpl}$	Paketrate bei idealisierter Pool-Anordnung	88
$P_{VE,Pool}$	Paketrate pro Verarbeitungseinheit bei Pool-Anordnung	88
s	Verhältnis der internen Modultaktfrequenz zur Systemtaktfrequenz	110
$T_{S,opt}$	optimale Verarbeitungsdauer je Pipeline-Verarbeitungseinheit	88
$T_{S,i}$	Verarbeitungsdauer in Pipeline-Verarbeitungseinheit i	88
$T_{S,i,spez}$	Verarbeitungsdauer in spezialisierter Pipeline-Verarbeitungseinheit i	90
T_{ges}	Gesamtverarbeitungsdauer	88
$T_{ges,Pipeline}$	Gesamtverarbeitungsdauer bei Pipeline-Verarbeitung	90
$T_{ges,Pool}$	Gesamtverarbeitungsdauer bei Pool-Verarbeitung	89
$\overline{T}_{ges,Pool}$	Mittlere Gesamtverarbeitungsdauer bei Pool-Verarbeitung	90
W	Wortbreite der Pipeline-Verarbeitungsstufen	99
$\lceil \ \rceil$	Aufrunden zur nächsten ganzen Zahl	100

1 Einleitung

Das Internet wandelt sich ständig. – Unablässig wächst die Zahl der Nutzer des Internets ebenso wie das Volumen und die Vielfalt der über das Internet bereitgestellten Informationen, Daten und Dienste. Gleichzeitig steigt unentwegt auch die Anzahl und Nutzung von nicht-öffentlichen landes- oder weltweiten Rechnernetzen großer Firmen, Konzerne und Organisationen. Infolge dieser Entwicklungen setzen Netzbetreiber immer höhere Datenübertragungsraten ein – was wiederum neuartige Dienste und Anwendungen in diesen Netzen ermöglicht.

1.1 Paketverarbeitung in zukünftigen Kommunikationsnetzen

Der Transport der Daten im Internet und in zunehmendem Maße auch in anderen Weitverkehrsnetzen erfolgt *paketvermittelt*. Dies bedeutet, dass die Endgeräte – die Computer, Notebooks und Mobiltelefone – die zu übermittelnden Daten in Datenpakete aufgeteilt versenden. Jedes Paket enthält zur Kennzeichnung des Empfängers und des Absenders sowie für weitere Steuerinformationen einen Paketkopf. Die Vermittlungsknoten im Netz – die *Router* oder *Switches* – verarbeiten diesen Paketkopf zur Ermittlung, in welche Richtung sie das Paket weiter zum Empfänger-Endgerät senden müssen.

Außer der Weitervermittlung der Pakete müssen heutige Vermittlungsknoten immer mehr zusätzliche Funktionen auf den Paketen ausführen. Hierzu zählen zum einen Funktionen zur Klassifizierung der Pakete für eine differenzierte Verarbeitung und Weiterleitung verschiedener Verkehrsklassen. Zum anderen sind dies Funktionen zur Modifikation der Pakete. Dies kann einzelne Felder im Paketkopf betreffen, um die weitere Vermittlung der Pakete zu steuern oder um Rückmeldungen an die Endgeräte zu übermitteln, aber auch die Nutzdaten oder das ganze Paket, wie bei der Umkodierung, Verschlüsselung oder Komprimierung der Paketdaten.

Die Anforderungen an zukünftige Paketverarbeitungssysteme sind nur schwer zu vereinbaren:

- Die steigenden Übertragungsraten erfordern eine **sehr hohe Daten- und Paketrate** bei der Paketverarbeitung. In Weitverkehrsnetzen soll der Durchsatz zusätzlich nach Möglichkeit **garantiert** werden.

- Die Einführung neuer Dienste und Anwendungen erfordert eine **hohe Flexibilität,** um die Paketverarbeitung problemlos und schnell an neue Funktions- und Leistungsanforderungen anpassen zu können.

Programmierbare Logikbausteine und Netzprozessoren sind kommerziell erhältliche elektronische Bauelemente, die versprechen, beide Anforderungen erfüllen zu können. Erstere basieren auf einer Vielzahl konfigurierbarer Abbildungstabellen und Flipflops und erlauben damit die Implementierung beinahe beliebiger Verarbeitungsarchitekturen. Letztere enthalten mehrere Prozessorkerne, Co-Prozessoren und Speicher und werden in Software programmiert. Allerdings weisen beide auch Nachteile hinsichtlich ihrer Leistungsfähigkeit und ihrer Flexibilität auf. Daher erforschen Ingenieure und Informatiker auf der ganzen Welt neue Konzepte und Architekturen mit dem Ziel, diese gegensätzlichen Anforderungen an die Paketverarbeitung zukünftig noch besser erfüllen zu können.

Die Parallelisierung von Verarbeitungseinheiten stellt eine wichtige Maßnahme zum Erreichen eines hohen Durchsatzes bei der Paketverarbeitung dar. Hierbei werden zwei prinzipiell unterschiedliche Anordnungen unterschieden: Zum einen die Anordnung als Pool, bei der eine Empfangseinheit jedem ankommenden Paket eine Verarbeitungseinheit zuweist, die die komplette Verarbeitung ausführt, und zum anderen die Anordnung als Pipeline, bei der die Paketverarbeitung sequentiell durch eine Kette von Verarbeitungseinheiten erfolgt. Auch hier weisen beide Arten der Parallelisierung Vor- und Nachteile auf und beide Anordnungen werden in schnellen Paketverarbeitungssystemen aus Industrie und Forschung eingesetzt.

1.2 Fragestellungen und Beiträge dieser Arbeit

Gegenstand dieser Dissertation ist der Entwurf einer neuen Architektur für zukünftige Paketverarbeitungssysteme in Hochgeschwindigkeitsweitverkehrskommunikationsnetzen, die einen maximalen deterministischen Durchsatz garantiert und gleichzeitig höchste Flexibilität für zukünftige Anpassungen und Erweiterungen bietet.

Hinführend dazu werden folgende Themenbereiche behandelt:

- Die aktuellen und zukünftigen Anforderungen an die Paketverarbeitung in Vermittlungsknoten werden analysiert und diskutiert. Hierbei wird auf den Aufbau, die Mechanismen und Protokolle sowie die Knotenarchitekturen bestehender und kommender Kommunikationsnetze eingegangen.

- Zurzeit erhältliche kommerzielle Netzprozessoren und FPGA-basierte Paketverarbeitungssysteme für höchste Datenraten sowie gegenwärtig von Forschern untersuchte neue Architekturen und Konzepte für die schnelle und flexible Paketverarbeitung werden vorgestellt und diskutiert.

Daran anschließend werden die beiden bei dieser Diskussion ausgemachten Spannungsfelder beim Entwurf einer optimalen Paketverarbeitungsarchitektur eingehend und grundsätzlich untersucht und diskutiert:

- Die verschiedenen Hardware- und Software-Abstraktionsebenen beim Entwurf von Paketverarbeitungsfunktionen auf FPGAs und Netzprozessoren werden hinsichtlich der Leistungsfähigkeit der entworfenen Funktionen und der Flexibilität beim Entwurf analysiert und vergleichend gegenübergestellt.

- Die beiden prinzipiellen Arten der Parallelisierung von Verarbeitungseinheiten, Pool und Pipeline, werden bezüglich ihres erzielbaren Durchsatzes, der Vor- und Nachteile bei der Implementierung von Paketverarbeitungsfunktionen sowie der zur Realisierung dieser Strukturen erforderlichen Ressourcen eingehend untersucht und verglichen. Abschließend wird dabei zusätzlich auf andere Ebenen der Parallelisierung eingegangen.

Diese Untersuchungen motivieren den Ansatz der neuen Architektur für schnelle und flexible Paketverarbeitung.

> *Die neue Architektur basiert auf einer sehr breiten Pipeline mit einem garantierten Durchsatz von einem Paket mit Minimalgröße pro Taktzyklus und besteht aus unterschiedlichen Modulen, mit denen Paketverarbeitungsfunktionen auf einer jeweils dem Problem angepassten Entwurfsebene implementiert werden können.*

Weiterführend werden verschiedene mögliche Verarbeitungsmodule sowie Nutzungsszenarien dieser modularen und flexiblen Architektur behandelt:

- Mögliche Verarbeitungsmodule für die neue Architektur werden vorgestellt, wobei insbesondere auf die Architekturen und auf beispielhafte Implementierungen dreier solcher Module eingegangen wird.

- Anwendungsszenarien zur Nutzung der von der Architektur bereitgestellten Flexibilität werden beschrieben ebenso wie die Schritte zur Integration neuer oder Anpassung vorhandener Paketverarbeitungsfunktionen einschließlich möglicher Werkzeugunterstützung hierbei.

Aufgrund der deterministischen Leistungsfähigkeit der neuen Architektur muss deren Durchsatz nicht durch weitere Untersuchungen, z. B. durch Simulation, überprüft werden. Um die Realisierbarkeit und Anwendbarkeit zu bewerten, werden abschließend die prototypische Implementierung eines Paketverarbeitungssystems nach der neuen Architektur und die Umsetzung verschiedener Paketverarbeitungsfunktionen beschrieben:

- Der Prototyp wurde in VHDL implementiert und auf einer FPGA-basierten Plattform sowohl mit anderen Netzelementen funktional getestet als auch mit internen Paketgeneratoren unter Volllast auf Korrektheit geprüft. Synthese-Ergebnisse für zwei verschiedene FPGA-Modelle von Altera zeigen die Realisierbarkeit auf aktuellen Bauelementen mit einem Durchsatz von über 100 Gbit/s und mehr als 150 Millionen Paketen pro Sekunde.

- Die beispielhafte Umsetzung von aktuellen und möglicherweise zukünftigen, noch experimentellen Router-Funktionen unter Verwendung verschiedener Verarbeitungsmodule des vorgestellten Prototyps zeigt die Anwendbarkeit der neuen Architektur.

Die Ergebnisse dieser Dissertation wurden vom Autor teilweise bereits im Rahmen von wissenschaftlichen Konferenzen veröffentlicht [1–3]. Zudem wurden grundsätzliche Fragestellungen zur schnellen und flexiblen Paketverarbeitung vom Autor zusammen mit anderen Wissenschaftlern bereits in einem Positionspapier [4] diskutiert. Bei folgenden weiteren Veröffentlichungen zu thematisch verwandten Themen ist er ebenfalls Erstautor [5–8] oder Mitautor [9–12]. Der Entwurf und die Implementierung des Prototyps erfolgte zum Teil in Studien- und Diplomarbeiten [13–18] unter der Anleitung des Autors.

1.3 Aufbau der Arbeit

Nachfolgend wird kurz der Aufbau dieser Abhandlung dargelegt. Zusammengefasst beschreibt Kapitel 2 die Grundlagen zum Verständnis dieser Arbeit einschließlich des aktuellen Stands von Forschung und Technik. Kapitel 3 widmet sich der neuen Architektur. Es umfasst einleitende Abschnitte zur Motivation und Zielsetzung, den eigentlichen neuen Architekturansatz und weiterführenden Details sowie Nutzungsszenarien. Kapitel 4 beschreibt den Prototypen und die Umsetzungen verschiedener Paketverarbeitungsfunktionen zur Überprüfung von Realisierbarkeit und Anwendbarkeit. Kapitel 5 gibt eine Zusammenfassung und einen Ausblick.

Kapitel 2 beginnt mit den Grundlagen paketvermittelnder Netze: Den verschiedenen Netzelementen, der Paket- und der Leitungsvermittlung, dem Schichtenmodell und einer Einordnung der verschiedenen Netztypen. Im Anschluss daran werden die wichtigsten heute verwendeten Protokolle auf Schicht 2 bis 4 (insbesondere Ethernet, IP und TCP) sowie (mögliche) zukünftige Protokolle und Dienste eingeführt, wobei auf die dafür notwendigen Verarbeitungsfunktionen in Vermittlungsknoten eingegangen wird. Außerdem werden die Aufgaben und die Architektur aktueller Hochgeschwindigkeitsvermittlungsknoten beschrieben. Unterkapitel 2.2 erörtert im Anschluss daran die Kriterien Funktionalität, Leistungsfähigkeit und Flexibilität von zukünftigen Paketverarbeitungssystemen.

Unterkapitel 2.3 führt die prinzipiell für die Paketverarbeitung geeigneten Bauelemente ein mit einem besonderen Fokus auf FPGAs und Netzprozessoren. Im Anschluss daran stellt Unterkapitel 2.4 die wichtigsten Hochgeschwindigkeitsnetzprozessoren vor und diskutiert deren Architekturen. Außerdem werden Bauelemente und Komponenten sowie Komplettsysteme für die FPGA-basierte Hochgeschwindigkeitspaketverarbeitung detailliert beschrieben und eingeordnet. Unterkapitel 2.5 gibt einen Überblick über die wichtigsten Forschungsarbeiten bezüglich verschiedener Aspekte für die Realisierung schneller und flexibler Paketverarbeitungssysteme auf Basis von FPGAs oder (Netz-)Prozessoren.

Kapitel 3 führt die neue Architektur für schnelle und flexible Paketverarbeitung in Hochgeschwindigkeitskommunikationsnetzen ein. Unterkapitel 3.1 analysiert zunächst die verschiedenen Hardware- und Software-Entwurfsebenen, die für den Entwurf von Paketverarbeitungsfunktionen in Frage kommen, und vergleicht diese hinsichtlich verschiedener Kriterien. Daraufhin definiert Unterkapitel 3.2 die Ziele der neuen Architektur: Einen hohen deterministischen Durchsatz, Modularität und die Möglichkeit, Verarbeitungsfunktionen auf einer jeweils möglichst passenden Entwurfsebene implementieren zu können. Nur durch die Parallelisierung von Verarbeitungseinheiten ist mit heutiger Technologie eine Paketverarbeitung mit einem Durchsatz von 100 Gbit/s möglich. Unterkapitel 3.3 untersucht und vergleicht die Pool- und die Pipeline-Anordnung hinsichtlich Durchsatz, Implementierungsaufwand und Ressourcenbedarf.

Nach diesen einleitenden Unterkapiteln wird in Unterkapitel 3.4 der Entwurf der neuen Architektur beschrieben. Dieser Architektur liegt eine Pipeline-Struktur zu Grunde, welche ausführlich in diesem Unterkapitel beschrieben und analysiert wird. Weiter sieht die Architektur verschiedene Verarbeitungsmodularten vor, die an ihren Schnittstellen den Durchsatz der Pipeline garantieren, intern jedoch eine beliebige Verarbeitungsarchitektur aufweisen können. Unterkapitel 3.5 geht detaillierter auf diese Module ein und diskutiert insbesondere die Architekturen von drei solchen Verarbeitungsmodularten. Unterkapitel 3.6 beschreibt, wie ein System mit der

neuen Architektur verwendet wird und wie so ein System flexibel an neue Anforderungen angepasst werden kann. Zum Abschluss von Kapitel 3 wird in Unterkapitel 3.7 ein Einblick in zwei beispielhafte Implementierungen von zwei Verarbeitungsmodulen gegeben.

Kapitel 4 beschreibt, wie die neue Architektur auf ihre Leistungsfähigkeit, Realisierbarkeit und Anwendbarkeit überprüft wurde. Unterkapitel 4.1 motiviert, warum diese drei Kriterien für die Bewertung der Architektur entscheidend sind. Außerdem wird auf die Zusammenhänge eingegangen, die zeigen, dass die Leistungsfähigkeit der Architektur garantiert ist und nur von wenigen Architekturparametern abhängt. Daher sind keine weiteren Schritte zur Überprüfung dieses Kriteriums notwendig. Unterkapitel 4.2 beschreibt den implementierten Prototypen, dabei wird auf die verwendete Plattform, die Netz- und Management-Schnittstellen, die verschiedenen Verarbeitungsmodule, die funktionalen Tests sowie die Synthese-Ergebnisse eingegangen. Letztere zeigen die Tauglichkeit eines solchen Systems basierend auf einem aktuellen FPGA für 100 Gbit/s-Line-Cards. Unterkapitel 4.3 und 4.4 stellen die durchgeführte Umsetzung der wichtigsten IP-Router-Funktionen sowie Paketverarbeitungsfunktionen zukünftiger Protokolle in Modulen des Prototyps dar und zeigen somit die Anwendbarkeit eines solchen Systems.

Kapitel 5 fasst diese Dissertation nochmals zusammen und gibt einen Ausblick auf mögliche weitere Arbeiten, die auf den erzielten Ergebnissen aufbauen.

2 Paketverarbeitungssysteme in Hochgeschwindigkeits-kommunikationsnetzen

Paketverarbeitungssysteme in zukünftigen Hochgeschwindigkeitskommunikationsnetzen müssen eine Vielzahl von Funktionen auf mehreren Millionen Paketen pro Sekunde durchführen. Zusätzlich sollen sie einfach an neue Anforderungen aufgrund von sich ändernden und neuen Protokollen anpassbar sein. Dieses Kapitel erläutert, aufbauend auf einer Einführung in die Grundlagen von Kommunikationsnetzen und -netzknoten, die Anforderungen an Paketverarbeitungssysteme und beschreibt Technologien, Bauelemente und Architekturen, mit denen versucht wird, diese heute und zukünftig zu erfüllen.

Unterkapitel 2.1 führt in die Thematik paketvermittelnder Kommunikationsnetze und insbesondere in die Aufgaben und die Architektur moderner Vermittlungsknoten ein. Die Anforderungen an Paketverarbeitungssysteme als wesentlicher Teil von Vermittlungsknoten sowie Kriterien zur Bewertung derselben erläutert Unterkapitel 2.2. Unterkapitel 2.3 beschreibt Bauelemente, mit welchen Hochgeschwindigkeits-Paketverarbeitungssysteme realisiert werden können, bevor Unterkapitel 2.4 und 2.5 schließlich aktuelle Paketverarbeitungssysteme vorstellen und diskutieren. Hier wird sowohl auf aktuelle, kommerziell erhältliche Systeme, als auch auf Fragestellungen aktueller Forschungsaktivitäten eingegangen.

2.1 Grundlagen paketvermittelnder Netze

Dieses Unterkapitel gibt in Abschnitt 2.1.1 eine Einführung in die Grundlagen paketvermittelnder Kommunikationsnetze. Im Anschluss daran geben Abschnitt 2.1.2 und Abschnitt 2.1.3 einen Überblick über die wichtigsten, aktuellen Netztechnologien und Protokolle sowie zukünftige Entwicklungen. Darauf aufbauend beschreibt Abschnitt 2.1.4 die Aufgaben und die prinzipielle Architektur von Vermittlungsknoten in heutigen Hochgeschwindigkeitskommunikationsnetzen.

2.1.1 Einführung

Zur Einführung in die Thematik paketvermittelnder Kommunikationsnetze beschreibt dieser erste Abschnitt die drei wichtigsten Elemente eines Kommunikationsnetzes und stellt die beiden wesentlichen Arten der Datenvermittlung vor. Außerdem wird das so genannte Schichtenmodell

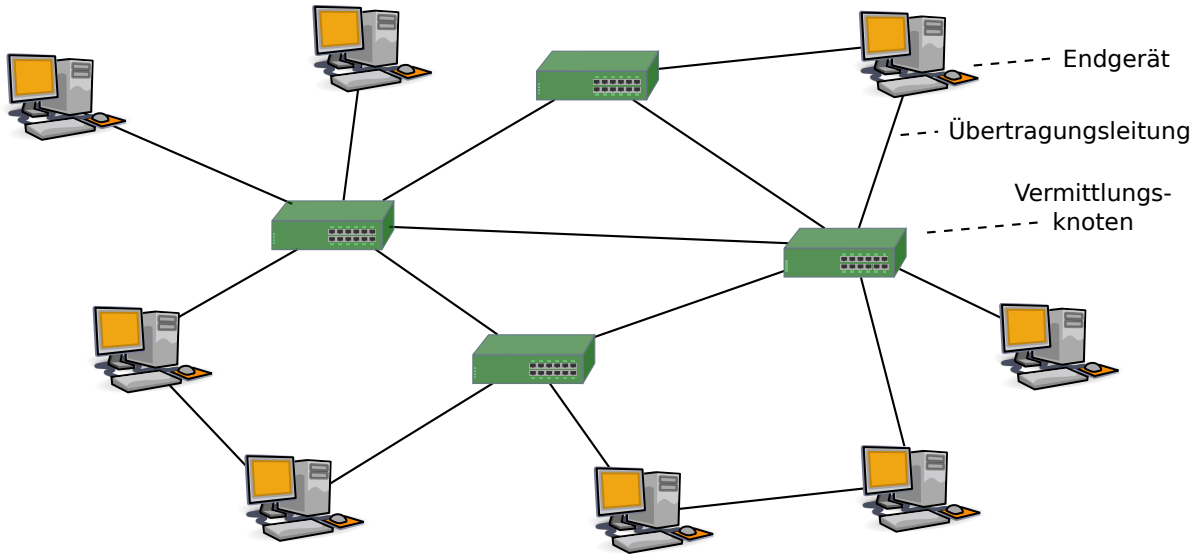

Abbildung 2.1: Kommunikationsnetz bestehend aus Endgeräten, Vermittlungsknoten und Übertragungsleitungen

zur Reduzierung der Komplexität von Kommunikationsnetzen erklärt. Der Abschnitt endet mit einer Einteilung der weltweiten Kommunikationsnetze-Infrastruktur in drei hierarchisch miteinander verbundene Netztypen.

Netzelemente

Kommunikation bezeichnet das Übermitteln einer Botschaft von einer Entität zu einer oder mehreren anderen. Bei der digitalen Kommunikation handelt es sich bei den kommunizierenden Entitäten um elektronische Geräte. Diese werden im Folgenden *Endgeräte* genannt. Diese übermitteln die Botschaften in binär codierter Form, d. h. in Form von Einsen und Nullen. Die Übermittlung dieser Binärdaten kann elektrisch über eine elektrisch leitende Verbindung, optisch über eine Glasfaser oder elektromagnetisch durch Funkwellen erfolgen. Diese verschiedenen Übertragungsmedien werden im Folgenden allgemein als *Übertragungsleitung* zusammengefasst.

Bei Systemen mit vielen kommunizierenden Endgeräten kann aufgrund von Platz-, Ressourcen- und Kostengründen häufig nicht jedes Endgerät direkt mit jedem anderen durch eine Übertragungsleitung verbunden werden. Aus diesem Grund werden die Endgeräte über Vermittlungsgeräte miteinander verbunden. Diese leiten die zu übermittelnden Daten entweder direkt zum Ziel-Endgerät oder zum nächsten Vermittlungsgerät in Richtung des Ziels weiter. Somit entsteht ein Netz, dessen Knoten die Vermittlungsgeräte (sowie je nach Definition auch die Endgeräte) und dessen Kanten die Übertragungsleitungen sind. Die Vermittlungsgeräte werden daher im Folgenden *Vermittlungsknoten* genannt.

Abbildung 2.1 zeigt ein solches Kommunikationsnetz, bestehend aus Endgeräten, Vermittlungsknoten und Übertragungsleitungen. Diese drei fundamentalen Elemente von Kommunikationsnetzen werden im Folgenden im Detail beschrieben:

Endgeräte sind die Systeme, die untereinander über das Netz kommunizieren. Endgeräte sind im Wesentlichen Computer – in Form von PCs, Notebooks und Server-Rechnern – sowie Mobil- und Festnetz-Telefone, wobei die Grenzen zwischen diesen Geräten in den letzten Jahren mehr und mehr verschwimmen. Des Weiteren können auch andere elektronische Geräte, Maschinen und Sensoren als Endgeräte an ein Kommunikationsnetz angeschlossen sein.

Vermittlungsknoten sind die Systeme innerhalb des Netzes, die zu übermittelnde Daten in Richtung des Ziel-Endsystems weiterleiten. Je nach verwendeter Technologie werden sie auch als *Switch* oder *Router* bezeichnet. Über die Datenweiterleitung hinaus übernehmen Vermittlungsknoten in heutigen Kommunikationsnetzen mehr und mehr zusätzliche Verarbeitungsfunktionen. Hierfür werden flexibel an die immer neuen Aufgaben anpassbare Verarbeitungssysteme benötigt. Diese Dissertation beschreibt eine neue Architektur eines solchen Systems für Kommunikationsnetze mit sehr hoher Datenrate.

Übertragungsleitungen transportieren die Daten zwischen den beiden oben genannten Systemen, den Endgeräten und den Vermittlungsknoten. Die höchsten Übertragungsraten erreichen Glasfasern, welche Daten optisch, d. h. in Form von Licht, übertragen. Durch die Verwendung von Licht unterschiedlicher Wellenlänge können bis zu mehrere Terabit pro Sekunde (Tbit/s) übertragen werden (Wellenlängenmultiplex, *Wavelength Division Multiplex*, WDM). Des Weiteren können Daten über elektrisch leitende Verbindungen, z. B. Kupferleitungen, übertragen werden sowie als Funkwellen, wie bei Mobilfunknetzen, Richtfunkstrecken oder lokalen Funknetzen (*Wireless Local Area Network*, WLAN).

Schließlich können auch mehrere Kommunikationsnetze miteinander verbunden werden. Der bekannteste, größte und wichtigste solche Zusammenschluss vieler eigenständiger Kommunikationsnetze ist das weltumspannende Internet. Die unterschiedlichen Netze des Internets werden dabei über so genannte *Router* verbunden. Diese sind die Vermittlungsknoten des Internets. Sie werden zusätzlich auch innerhalb einzelner Netze eingesetzt. Die einzelnen Netze bzw. Netzbereiche zwischen den Routern entsprechen – auf dieser Abstraktionsebene – jeweils mehreren Übertragungsleitungen, die die angeschlossenen Endgeräte und Router miteinander verbinden. Näheres zum Internet und zu Routern wird in Abschnitt 2.1.2 bzw. 2.1.4 dargestellt.

Vermittlungsarten

Die Übermittlung der Daten durch ein Kommunikationsnetz erfolgt entweder *leitungsvermittelt* oder *paketvermittelt*. Das herkömmliche, weltweite Telefonnetz ist das bekannteste leitungsvermittelnde Kommunikationsnetz, während das Internet ein paketvermittelndes Netz ist.

Bei der **Leitungsvermittlung** (*Circuit Switching*) wird vor der Kommunikation eine durchgängige „Leitung“ (*circuit*) zwischen den Kommunikationspartnern aufgebaut. In den ersten Jahren des Telefons baute das „Fräulein vom Amt“ diese Leitung noch physikalisch durch Verbinden von Eingangs- und Ausgangsbuchse in der Vermittlungsstelle auf. Heutzutage erfolgt der Leitungsaufbau elektronisch und mehrere Tausend logische, digitale „Leitungen“ – Kanäle genannt – können durch synchronen Zeitmultiplexbetrieb eine Übertragungsleitung quasi gleichzeitig verwenden. Wesentliche Eigenschaft einer leitungsvermittelten Kommunikationsverbindung ist eine konstante, reservierte Datenübertragungsrate. Diese steht in vollem Umfang

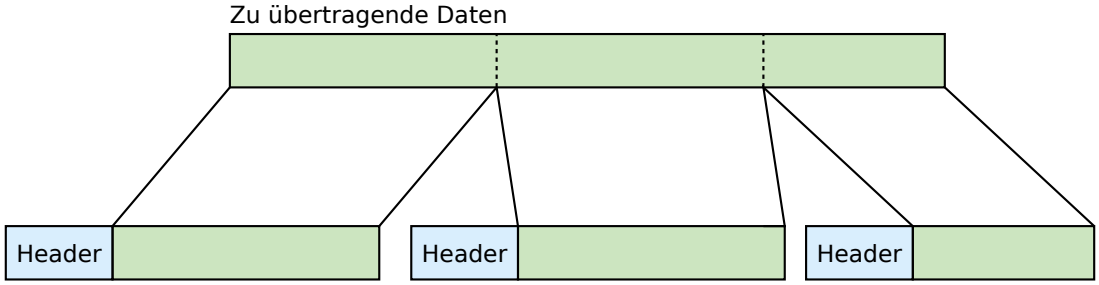

Abbildung 2.2: Aufteilung der Nutzdaten in mehrere Pakete, bestehend aus Header und Daten

den kommunizierenden Endgeräten zur Verfügung unabhängig davon, in welchem Maße sie diese nutzen – selbst dann, wenn sie zeitweise keine Daten übertragen. Nach dem Leitungsaufbau können die Vermittlungsknoten die Daten direkt durchschalten und müssen keinerlei Verarbeitung durchführen.

Bei der **Paketvermittlung** (*Packet Switching*) zerteilt das sendende Endgerät die zu übermittelnden Daten in Datenblöcke. Zusätzlich ergänzt es jeden Datenblock um Steuerinformationen. Diese so erweiterten Datenblöcke nennt man *Pakete* oder *Rahmen*. Die angefügten Zusatzinformationen bezeichnet man als *Paketkopf* bzw. häufiger mit dem englischen Begriff *Header*. Abbildung 2.2 illustriert diese Aufteilung der Daten in Pakete. Die Vermittlungsknoten leiten nun jedes Paket unter Auswertung seiner Steuerinformationen im Header individuell in Richtung seines Ziel-Endknotens weiter. Der Vorteil der Paketvermittlung ist, dass jede Kommunikation zu jeder Zeit nur so viel Bandbreite[1] auf den Übertragungsleitungen verwendet, wie sie tatsächlich benötigt (zuzüglich des Bandbreitebedarfs durch die zusätzlichen Header). Die Verarbeitung der Header-Informationen in den Vermittlungsknoten – insbesondere bei sehr hohen Datenraten – ist zentrales Thema dieser Dissertation und wird in den folgenden Unterkapiteln eingehend behandelt.

Bei der Leitungsvermittlung wird durch den Leitungsaufbau eine feste, so genannte *Verbindung* (*connection*) zwischen den beiden Kommunikationsendgeräten aufgebaut. Bei der Paketvermittlung kann vor der eigentlichen Kommunikation ebenso eine Verbindung in Form einer so genannten *virtuellen* Leitung (*Virtual Circuit*) aufgebaut werden. Dabei muss – wie bei der Leitungsvermittlung – im Vorhinein bei jedem Vermittlungsknoten zwischen den beiden Endgeräten eine eindeutige Zuordnung von Eingangs- zu Ausgangsschnittstelle eingerichtet werden. Diese Vermittlungsart wird als *verbindungsorientierte* Paketvermittlung bezeichnet (*connection-oriented packet switching*). Wie bei der verbindungslosen Paketvermittlung (*connection-less packet switching*) wird weiterhin jederzeit nur so viel Bandbreite belegt, wie benötigt wird. Jedoch ist vor der eigentlichen Kommunikation sichergestellt, über welchen Pfad die Pakete durch das Netz vermittelt werden. Außerdem erleichtert die verbindungsorientierte Vermittlung eine Unterstützung verschiedener Dienstgüteklassen in den Vermittlungsknoten. Des Weiteren ist die Ermittlung der Weiterleitungsinformationen in den Vermittlungsknoten aufgrund kürzerer Weiterleitungstabellen weniger komplex.

[1] Der Begriff *Bandbreite* wird in der Kommunikationstechnik häufig (fälschlicherweise) im Sinne einer Bitrate verwendet und nicht in seiner eigentlichen Bedeutung eines Frequenzbereichs.

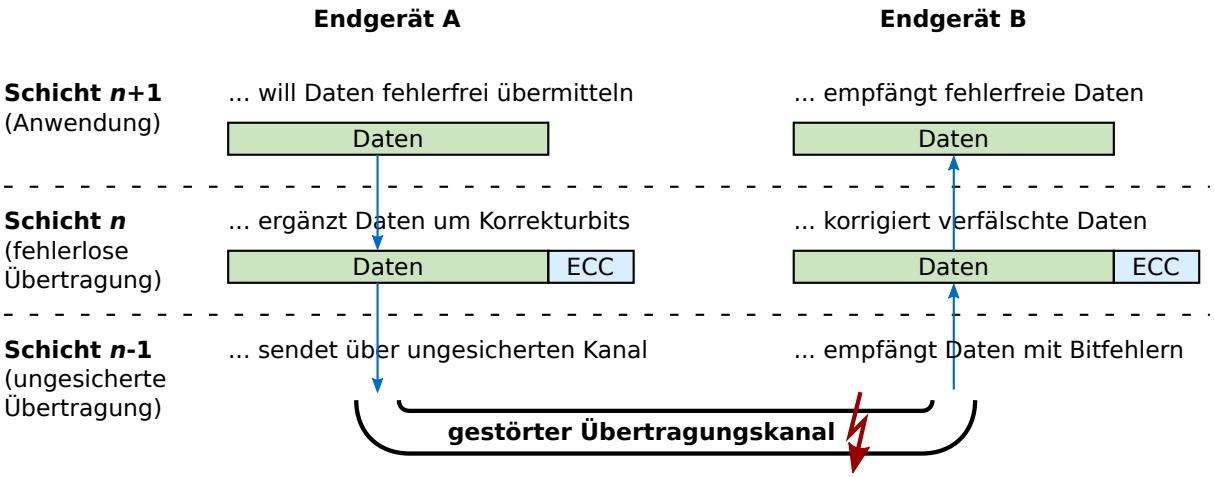

Abbildung 2.3: Beispiel: Schicht $n+1$ nutzt fehlerlose Datenübertragung von Schicht n

Schichtenmodell

Um die Implementierungskomplexität von Kommunikationsnetzen möglichst gering zu halten, sind die verschiedenen für die Kommunikation benötigten Funktionen in eine Hierarchie unterschiedlicher Schichten (*layer*) untergliedert. Diese werden in den Endgeräten und Vermittlungsknoten von unterschiedlichen Software- oder Hardware-Instanzen realisiert. Jede Schicht stellt ihrer nächst höheren Schicht bestimmte Dienste zur Verfügung, abstrahiert jedoch von ihrer tatsächlichen Realisierung. Für die Realisierung dieser Dienste greift sie dabei wiederum auf Dienste ihrer nächst tieferen Schicht zu.

Ein Beispiel soll dies anhand Abbildung 2.3 verdeutlichen: Schicht n stellt der nächst höheren Schicht $n+1$ eine fehlerlose Datenübertragung bereit. Um diese zu realisieren, verwendet Schicht n den Dienst der darunter liegenden Schicht $n-1$, welche eine ungesicherte Übertragung bereitstellt. Wie in Abbildung 2.3 dargestellt, überträgt Schicht n hierfür mit dem Dienst von Schicht $n-1$ beispielsweise zusätzlich zu den eigentlich zu übertragenden Daten Fehlerkorrekturcodes (*Error-Correcting Codes*, ECC). Mit diesen ist es der Zielinstanz auf Schicht n möglich, Übertragungsfehler zu korrigieren. So kann Schicht $n+1$ die fehlerlose Übertragung von Schicht n verwenden, ohne sich um deren Realisierung zu kümmern.

Wie in obigem Beispiel zu beobachten ist, kommunizieren bei der Datenübertragung zwischen zwei Systemen (Endgeräte, Vermittlungsknoten) jeweils Instanzen derselben Schicht miteinander. Die Regeln für die Kommunikation auf einer Schicht definiert jeweils ein *Protokoll*, welches beide Seiten einhalten. Allerdings erfolgt der Nachrichtenaustausch nicht direkt, sondern jede Schicht übergibt zum Versenden die Daten an die jeweils nächst tiefere Schicht, bis auf der untersten Schicht die eigentliche physikalische Übertragung über die Leitung erfolgt. Beim Empfängersystem werden die Daten wieder von Schicht zu Schicht nach oben transportiert.

Bei der Weitergabe der zu übermittelnden Daten an die darunter liegende Schicht fügt jede Schicht diesen Daten i. d. R. Steuerinformationen entsprechend des verwendeten Protokolls hinzu. Bei paketvermittelnden Netzen erfolgt dies in Form eines Headers und/oder eines *Trailers* vor bzw. hinter den Daten (vgl. auch Abbildung 2.3). Beim Empfangen der Daten wertet jede Schicht die für sie bestimmten Header und Trailer aus und entfernt sie vor der Weitergabe der

Abbildung 2.4: Beispiel der hierarchischen Schachtelung der zu übertragenden Nutzdaten mit den Headern und Trailern der verschiedenen Schichten

Daten an die nächst höhere Schicht. Somit enthalten die zu übertragenden Daten auf der Leitung eine hierarchische Schachtelung von Protokoll-Headern und -Trailern. Dies ist exemplarisch in Abbildung 2.4 dargestellt.

In der Norm ISO/IEC 7498–1 [19] (*International Organization for Standardization* / *International Electrotechnical Commission*) ist das Basisreferenzmodell für Offene Kommunikationssysteme (*Open Systems Interconnnection*, OSI) beschrieben. Dieses so genannte ISO/OSI-Basisreferenzmodell definiert eine Hierarchie von folgenden sieben Schichten:

Schicht 1: Bitübertragungsschicht (*Physical Layer*) – zuständig für die physikalische Übertragung von Bits über eine Übertragungsleitung

Schicht 2: Sicherungsschicht (*Data Link Layer*) – zuständig für die sichere Übertragung von Datenpaketen über eine Übertragungsleitung

Schicht 3: Vermittlungsschicht (*Network Layer*) – zuständig für die adressgesteuerte Vermittlung der Pakete durch das Kommunikationsnetz

Schicht 4: Transportschicht (*Transport Layer*) – zuständig für eine effiziente Ende-zu-Ende-Datenkommunikation

Schicht 5: Kommunikationssteuerungsschicht (*Session Layer*) – zuständig für die Verwaltung verschiedener Kommunikationssitzungen

Schicht 6: Darstellungsschicht (*Presentation Layer*) – zuständig für Syntax, Semantik und Codierung der zu übertragenden Daten

Schicht 7: Verarbeitungsschicht (*Application Layer*) – stellt verschiedene Kommunikationsanwendungen, wie E-Mail-Kommunikation, Datei-Transfer und Web-Browsing, bereit.

Das OSI-Modell wurde unabhängig von bestehenden Implementierungen von Kommunikationsnetzen entworfen. Es ist sehr allgemein formuliert und unterscheidet bei jeder Schicht explizit zwischen den drei Basiskonzepten Dienst, Schnittstelle und Protokoll. Allerdings fassten das Modell und seine Protokolle nie richtig Fuß. Gründe hierfür waren deren Komplexität und die einhergehende schwere Implementierbarkeit, einige Schwachpunkte im Modell und den Protokollen sowie politische und zeitliche Randbedingungen [20]. Dennoch werden die sieben Schichten des OSI-Modells bis heute sehr gerne zur Beschreibung und Einordnung von Funktionen und Protokollen in Kommunikationsnetzen verwendet.

Zur Beschreibung des Internets sowie allgemein von Netzen auf Basis des *Internet Protocol* (IP) wird neben dem ISO/OSI-Referenzmodell in zunehmendem Maße das so genannte Internet-Schichtenmodell verwendet. Dieses definierte die IETF (*Internet Engineering Task Force*)

in RFC 1122 und 1123 [21, 22]. Es orientiert sich stark an bestehenden Protokollen. Das Internet-Schichtenmodell besteht aus der Netzzugangsschicht (*host-to-network layer*), der Internet-Schicht (*internet layer*), der Transportschicht (*transport layer*) sowie der Anwendungssschicht (*application layer*).

Die Netzzugangsschicht fasst Schicht 1 und 2 des ISO/OSI-Modells zusammen. Über die Funktion dieser Schicht wird nicht viel ausgesagt, außer dass sie für die Paketübertragung innerhalb eines Netzes zuständig ist. Das wichtigste Protokoll dieser Schicht ist Ethernet. Die Internet-Schicht entspricht Schicht 3 und ist für die Vermittlung von Datenpaketen von einem Sender zu einem Empfänger zuständig, wobei diese auch an unterschiedliche Netze angeschlossen sein können. Das entsprechende Protokoll ist das *Internet Protocol* (IP). Die Transportschicht hat eine ähnliche Funktion wie die entsprechende Schicht 4 des ISO/OSI-Modells und wird durch das zuverlässige, verbindungsorientierte TCP (*Transmission Control Protocol*) oder das unzuverlässige, verbindungslose UDP (*User Datagram Protocol*) realisiert. Die Anwendungsschicht fasst die Schichten 5 bis 7 zusammen und wird durch eine Vielzahl von Protokollen realisiert.

Vermittlungsknoten in Hochgeschwindigkeitskommunikationsnetzen verarbeiten Pakete i. d. R. auf Schicht 2 und 3. Die Header von Schicht 4 wurden ursprünglich nicht in Vermittlungsknoten betrachtet, da diese Schicht eine Ende-zu-Ende-Kommunikation bereitstellt. Allerdings machen neue Entwicklungen zunehmend auch die Verarbeitung dieser Schicht in Routern notwendig (vgl. z. B. [23]). Teilweise werden selbst Protokollelemente oberhalb von Schicht 4 und sogar Nutzdaten in Vermittlungsknoten verarbeitet (vgl. z. B. [24]). Abschnitt 2.1.2 gibt einen kurzen Einblick in die wichtigsten heute verwendeten Protokolle auf Schicht 2 bis 4, namentlich Ethernet, IP, MPLS (*Multi-Protocol Label Switching*) sowie TCP und UDP, und deren Verarbeitung in Vermittlungsknoten. Der darauf folgende Abschnitt 2.1.3 beschreibt neue Protokoll-Entwicklungen, welche zusätzliche bzw. geänderte Verarbeitungsfunktionen in Vermittlungsknoten erfordern und daher eine flexible Verarbeitungsarchitektur erfordern.

Netztypen

Weltweit gibt es eine Vielzahl von Kommunikationsnetzen in verschiedensten Größen und Topologien und unter Verwendung vieler unterschiedlicher Übertragungs- und Netztechnologien. Diese Netze weisen daher auch sehr unterschiedliche Datenraten auf und erfordern unterschiedliche Verarbeitungsfunktionalitäten in ihren Vermittlungsknoten. Sehr viele kleine Netze sind lokal begrenzt und unabhängig von anderen Netzen, Beispiele sind insbesondere eingebettete Netze in Autos, Flugzeugen und anderen Maschinen. Andererseits gibt es auch eine weltweite Infrastruktur miteinander verbundener Datennetze. Hierüber ist das globale Internet realisiert, aber auch Firmennetze mit welt- oder landesweit verteilten Standorten.

Diese weltweite Kommunikationsnetze-Infrastruktur lässt sich grob in folgende drei, hierarchisch miteinander verbundenen Netztypen untergliedern: Zugangsnetze, Metronetze und Kernnetze. Diese werden im Folgenden kurz beschrieben. Abbildung 2.5 stellt die hierarchisch aufgebaute Netzinfrastruktur beispielhaft dar. Die Paketverarbeitung in Hochgeschwindigkeits-Metro und -Kernnetzen ist Schwerpunkt dieser Arbeit.

Zugangsnetze sind lokal begrenzte Netze, über die die Endgeräte und lokalen Netze (*Local Area Network*, LAN) privater Personen sowie kleinerer Firmen an die globale Kommu-

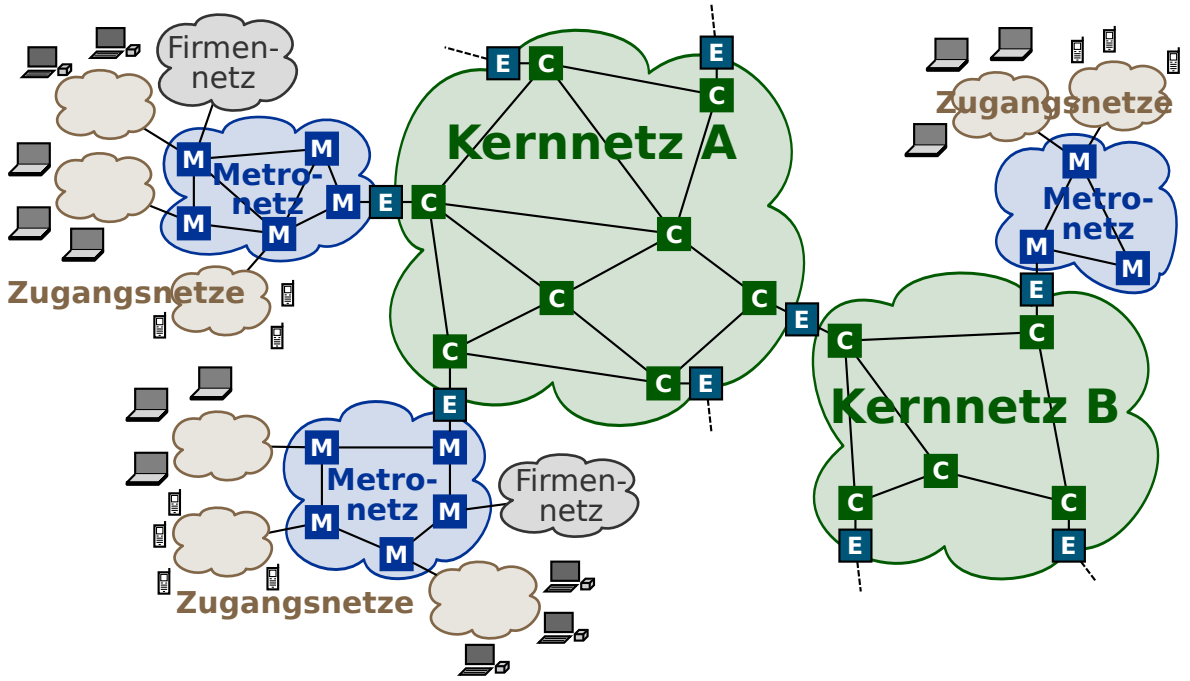

Abbildung 2.5: Netztypen und Vermittlungsknoten in weltweiter Kommunikationsinfrastruktur

nikationsnetze-Infrastruktur angeschlossen sind. Diese Netze basieren auf verschiedenen Technologien. Stationäre Endgeräte sind z. B. mittels des herkömmlichen Telefonanschlusses über DSL (*Digital Subscriber Line*), über das ehemals ausschließlich für Fernsehdienste genutzte Kabelnetz oder über neu installierte optische Netze an das Internet angeschlossen. Mobile Endgeräte verwenden drahtlose Zugangsnetze, wie Mobilfunknetze oder drahtlose, lokale Funknetze (*Wireless Local Areal Network,* WLAN). Die Datenraten, mit denen die Benutzer in einem solchen Netz angebunden sind, betragen heutzutage zwischen 1 Mbit/s und 1 Gbit/s. Die aggregierte Datenrate im Netz beträgt maximal 10 Gbit/s.

Metronetze sind stadtweite Netze, an die die Zugangsnetze sowie Netze größerer Firmen oder anderer Institutionen angeschlossen sind. Metronetze aggregieren den Datenverkehr dieser angebundenen Netze. Typische Datenraten sind 10 bis 40 Gbit/s, wobei in den nächsten fünf Jahren auch bis zu 100 Gbit/s zu erwarten sind. Aufgrund dieser hohen Datenraten basieren diese Netze in der Regel auf optischen Übertragungstechnologien. In den letzten Jahrzehnten wurden hier auf Schicht 1 und 2 im Wesentlichen leitungsvermittelnde Technologien, wie SDH/SONET (*Synchronous Digital Hierarchy/Synchronous Optical Network*), eingesetzt. Zurzeit wird allerdings dazu übergegangen, hier, wie in vielen Firmen- und Heimnetzwerken, paketvermittelnde Technologien wie Ethernet einzusetzen (vgl. hierzu auch Abschnitt 2.1.2 und 2.1.3).

Kernnetze bilden das Rückgrat der weltweiten Kommunikationsnetze-Infrastruktur. Sie verbinden Metronetze einer bestimmten geografischen Region, z. B. eines Landes, miteinander und sind außerdem untereinander verbunden. Aufgrund der großen räumlichen Entfernungen zwischen den Netzknoten wird diese Art von Netz auch als Weitverkehrsnetz (*Wide Area Network*, WAN) bezeichnet. Übliche Datenraten im Kernnetz betragen 10 bis 100 Gbit/s und auch höhere Datenraten, wie z. B. 400 Gbit/s, sind in den nächsten zehn Jahren zu erwarten. Die Übertragung findet auch hier optisch über mehrere Wellenlängen mittels Glasfasern

statt. Ebenso wie in Metronetzen ist auf Schicht 1 und 2 ein Trend von der Leitungsvermittlung (SDH/SONET) zur Paketvermittlung basierend auf Ethernet oder MPLS auszumachen.

In dieser Dissertation wird die schnelle und flexible Paketverarbeitung in Vermittlungsknoten von Metro- und Kernnetzen betrachtet. Diese Vermittlungsknoten lassen sich in Kern- (*Core-*), Rand- (*Edge-*) und Metro-Knoten gliedern. In Abbildung 2.5 sind diese Knoten entsprechend mit **C, E** und **M** bezeichnet. Kern-Vermittlungsknoten übernehmen die schnelle und effiziente Paketweiterleitung in den Kernnetzen. Diese müssen im Wesentlichen auf Geschwindigkeit und Ausfallsicherheit optimiert sein. Rand-Vermittlungsknoten bilden die Grenze zwischen Metronetzen und Kernnetzen sowie auch zwischen unterschiedlichen Kernnetzen. Sie vermitteln und überprüfen die Pakete zwischen den Netzen. Anders als im Bild der Übersichtlichkeit halber dargestellt, sind üblicherweise zwei solcher Vermittlungsknoten zwischen zwei Netzen, von jedem Netzbetreiber einer. Metro-Netzknoten vermitteln und verarbeiten die Pakete innerhalb von Metronetzen. In Rand- und Metro-Vermittlungsknoten ist neben einer hohen Verarbeitungsgeschwindigkeit insbesondere auch eine hohe Flexibilität der Verarbeitungssysteme erforderlich, um neue Anforderungen in Folge von Protokolländerungen oder neuer Dienste schnell realisieren zu können.

2.1.2 Heutige Netze und Protokolle

Dieser Abschnitt führt kurz die wichtigsten, heute verwendeten Protokolle auf Schicht 2 bis 4 ein und erläutert die dafür notwendigen Verarbeitungsfunktionen in den Vermittlungsknoten.

Ethernet – Netztechnologie auf Schicht 1 und 2

Ethernet ist die am weitesten verbreitete Netztechnologie für lokale Netze auf Schicht 1 und 2. Hauptgrund für den Erfolg ist, dass Ethernet im Vergleich zu konkurrierenden LAN-Technologien einfach und flexibel einsetz-, wart- und erweiterbar sowie zuverlässig und kostengünstig ist [20]. Die hohe Attraktivität von Ethernet führte in den letzten Jahren dazu, dass erweiterte Formen dieser Technologie zunehmend auch in anderen Einsatzgebieten verwendet werden, wie z. B. in Industrieanlagen, Flugzeugen und Autos [25]. Zudem wird überlegt, Ethernet aufgrund seiner hohen Verbreitung in Privat- und Firmennetzen sowie seiner kostengünstigen Komponenten auch für paketvermittelnde Weitverkehrsnetze einzusetzen (siehe Abschnitt 2.1.3).

Robert Metcalfe entwickelte die Netztechnologie Ethernet in den 1970er Jahren zusammen mit Kollegen bei der Firma *Xerox*. Im Jahre 1980 definierten *Xerox*, *Intel* und *Digital Equipment* einen Standard für ein 10 Mbit/s-Ethernet. Dieser bildete die Grundlage für den 1985 veröffentlichten IEEE-Standard 802.3 [26] (*Institute of Electrical and Electronics Engineers*). In den letzten zwei Jahrzehnten wurde dieser Standard für höhere Datenraten erweitert: Der Standard für *Fast-Ethernet* mit einer Datenrate von 100 Mbit/s wurde 1995, der für 1 Gbit/s-Ethernet (*Gigabit-Ethernet*) 1998 und der für 10 Gbit/s-Ethernet (*10-Gigabit-Ethernet*) 2002 veröffentlicht. Ein Standard für 40 Gbit/s- und 100 Gbit/s-Ethernet wurde im Juni 2010 verabschiedet. 1 Gbit/s-Ethernet und 10 Gbit/s-Ethernet ist für elektrische und optische Übertragungsleitungen standardisiert. Für die neuen Standards mit noch höheren Datenraten sind vermutlich nur noch optische Übertragungsleitungen realisierbar.

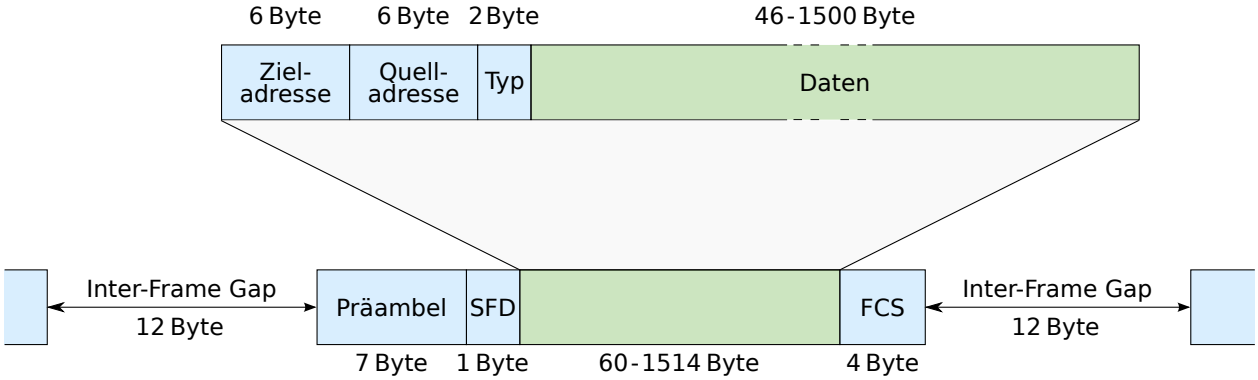

Abbildung 2.6: Das Ethernet-Rahmenformat

Der sehr umfangreiche Ethernet-Standard des IEEE definiert die Details der Datenübertragung von Schicht 2 und Schicht 1 bis hinunter zu Kabel- und Steckerspezifikationen. Im Folgenden wird auf die für die Paketverarbeitung in Hochgeschwindigkeitsnetzknoten relevanten Details des Standards eingegangen.

Abbildung 2.6 zeigt den Aufbau eines Ethernet-Rahmens. Ein Rahmen besteht aus einem Rahmenkopf (*frame header*) und Nutzdaten. Der Header besteht aus Ziel- und Quelladresse sowie einem Typfeld. Die Adressfelder sind jeweils 48 bit breit und erlauben global eindeutige Adressen. Das 16 bit breite Typfeld gibt i. d. R. an, welche Art von Daten im Nutzdatenbereich sind, z. B. bedeutet der Hexadezimalwert 0x0800 im Typfeld, dass die Nutzdaten ein IPv4-Paket (vgl. nächster Unterabschnitt) darstellen. Teilweise wird das Typfeld auch zur Spezifikation der Rahmenlänge verwendet. Der Nutzdatenbereich muss zwischen 46 und 1500 Byte lang sein.

Vor der Übertragung eines Ethernet-Rahmens erzeugt der Leitungsschnittstellen-Chip (*PHY-Chip*) eine 7 Byte lange Präambel mit abschließendem 1 Byte großem Rahmenanfangsbegrenzer (*Start Frame Delimiter*, SFD). Diese 8 Byte verwendet der Empfänger, um sich auf den ankommenden Bitstrom zu synchronisieren. Zum Schluss, nach Senden der Nutzdaten, wird eine 32 bit breite Rahmenprüfzeichenfolge (*Frame Check Sequence*, FCS) übertragen, mit welcher der Empfänger Bitfehler erkennen kann.

Zwischen der Übertragung zweier kompletter Ethernet-Rahmen (inklusive Präambel, SFD und FCS) muss ein zeitlicher Abstand entsprechend der Sendedauer von 12 Byte eingehalten werden. Dieser Abstand wird *Inter-Frame Gap* (IFG) genannt. Die Zeitdauer dieser Sendepause hängt somit von der jeweiligen Übertragungsrate ab. Um Aussagen unabhängig von der Übertragungsrate machen zu können, wird der Abstand zwischen zwei Rahmen häufig direkt durch die entsprechende Anzahl an Bytes angegeben. Ein solcher Rahmen- bzw. Paketabstand in Byte wird im Folgenden mit G bezeichnet. Unter Berücksichtigung des Inter-Frame Gap ist somit zwischen dem Ende der Rahmenprüfzeichenfolge und dem Beginn der Zieladresse des folgenden Rahmens ein minimaler (Byte-) Abstand G_{min} von 20 Byte. Dieser Abstand ist für die Berechnung der maximalen Rahmen- bzw. Paketrate wichtig (vgl. Abschnitt 2.2.2).

Beim ursprünglichen 10 Mbit/s-Ethernet waren alle Endgeräte an ein gemeinsam genutztes Kabel angeschlossen und es gab keine Vermittlungsknoten. Später wurde jedes Endgerät über ein eigenes Kabel an einen so genannten *Hub* angeschlossen. Dieser verarbeitet die ankommenden Rahmen jedoch nicht, sondern leitet die ankommenden physikalischen Signale jedes an-

geschlossenen Kabels lediglich an alle anderen Kabel weiter. Bei diesen beiden Verkabelungsschemata mussten sich alle Endgeräte die verfügbare Datenrate teilen, da alle Daten immer an alle Endgeräte übertragen werden.

Eine effizientere Art der Verkabelung verwendet einen so genannten *Switch* (früher auch *Bridge* genannt). Solch ein Vermittlungsknoten arbeitet auf Rahmenebene. Ein Switch empfängt einen ankommenden Rahmen, ermittelt anhand der Zieladresse und seiner Weiterleitungstabelle, über welche seiner Ausgangsschnittstellen der Rahmen weitergeleitet werden muss, und sendet ihn anschließend ausschließlich in Richtung des Empfänger-Endgeräts. Somit werden nur die für diese Übertragung notwendigen Leitungsressourcen verwendet. Zudem überprüft der Switch jeden Rahmen mit Hilfe der Rahmenprüfzeichenfolge auf Fehler und verwirft fehlerhafte Rahmen.

Das Internet – TCP/IP-Protokolle auf Schicht 3 und 4

Das Internet ist ein weltweiter Zusammenschluss einer Vielzahl unterschiedlicher Kommunikationsnetze. Wie das Wort *Internet* – zusammengesetzt aus den Worten *inter* (zwischen) und *net* (Netz) – bereits andeutet, ermöglicht dieser Zusammenschluss die Kommunikation zwischen verschiedenen Netzen.

a. Einführung und Geschichte

Das Internet entstand aus dem 1969 von der amerikanischen Forschungsorganisation für Verteidigungsprojekte ARPA (*Advance Research Projects Agency*) aufgebauten ARPANET, einem der ersten paketvermittelnden Netze weltweit. Es war entwickelt worden, um die Rechnersysteme von Universitäten und Forschungseinrichtungen in den USA untereinander zu vernetzen. Um dieses Netz auch mit anderen paketvermittelnden Netzen, wie einem paketvermittelnden Satellitennetz und einem paketvermittelnden Funknetz, verbinden zu können, entwickelten Vint G. Cerf und Robert E. Kahn 1973 das *Transmission Control Program* [27]. Dieses wurde 1977 in zwei Protokolle aufgeteilt: Das *Internet Protocol* (IP) [28], das im Wesentlichen für die Paketadressierung und -weiterleitung zuständig ist, und das *Transmission Control Protocol* (TCP) [29], das sich um Paketverluste und die Flussregelung kümmert. Näheres zur Geschichte des Internets ist in [30] zu finden.

Das so entstandene TCP/IP-Internet verwirklicht folgende Grundprinzipien (vgl. [30]):

- Zusammenschluss autonomer Netze
- Paketvermittlung ohne Garantien, nach dem so genannten *Best-Effort*[2]-Prinzip
- Vermittlungsknoten mit einfacher Funktionalität zwischen den Netzen
- Keine globale oder zentrale Steuerung

Abbildung 2.7 zeigt eine einfache Darstellung des Internets, bzw. allgemein eines IP-basierten Netzes, bestehend aus mehreren unabhängigen Netzen. Diese sind u. U. mit unterschiedlichen

[2] *best effort* bedeutet *nach bestem Bemühen*

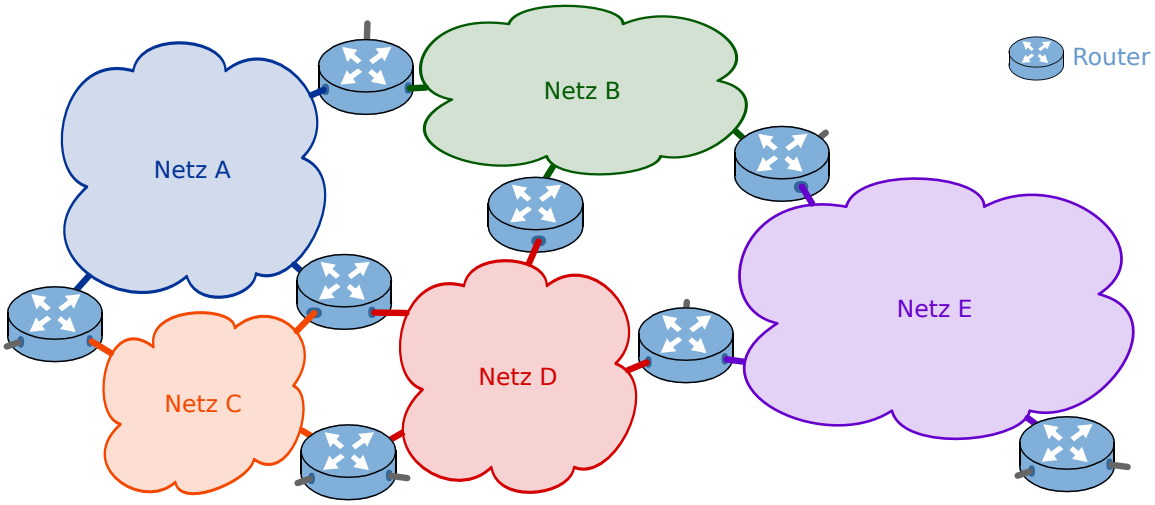

Abbildung 2.7: Internet bestehend aus verschiedenen Netzen und Routern dazwischen

Technologien bzw. unterschiedlichen Schicht-2-Protokollen realisiert. *Router* (ursprünglich *Gateways* genannt) verbinden die Netze miteinander und vermitteln Pakete zwischen ihnen. Hierfür greifen die Router auf entsprechende Header-Felder des Internet-Protokolls (IP) zu. Dieses wird im nächsten Unterabschnitt beschrieben.

b. IPv4 – das Internet Protokoll (Version 4)

Bis heute wird hauptsächlich das Internet Protokoll in Version 4 (IPv4) verwendet. Allerdings soll dieses bis in etwa 10 bis 20 Jahren vollständig von Version 6 (IPv6) abgelöst werden. Aufgrund der enormen Verbreitung von IPv4 soll hier dennoch zunächst die Definition dieser Version betrachtet werden. IPv6 wird nachfolgend in Abschnitt 2.1.3 vorgestellt.

IPv4 definiert den in Abbildung 2.8 dargestellten Paketkopf. Um zum einen die Funktionsweise dieses Protokolls näher zu erläutern und zum anderen einen ersten Einblick in die pro Paket benötigten Verarbeitungsschritte eines IP-Routers zu erlauben, werden im Folgenden die Funktionen der einzelnen Felder kurz dargestellt:

Version – Hieran erkennt der Router, welche IP-Version das Paket verwendet, hier also IPv4.

Header-Länge – Daran kann der Router erkennen, ob der Header Optionen (s. u.) enthält und wo der Nutzdatenbereich beginnt. Ohne Optionen beträgt die Header-Länge 20 Byte.

Diensttyp – Hiermit kann der Router bei der Verarbeitung und Weiterleitung des Pakets zwischen verschiedenen Kriterien von Dienstgüte (*Quality of Service*, QoS) differenzieren, z. B. geringe Verzögerung, hoher Durchsatz, hohe Zuverlässigkeit.

Gesamtlänge – Damit kann der Router überprüfen, ob das Paket vollständig empfangen worden ist.

Identifikation, Flags und Fragment-Versatz – Falls ein Paket größer ist als die maximal unterstützte Paketgröße des nächsten Netzes, kann der Router das Paket in kleinere Fragmente

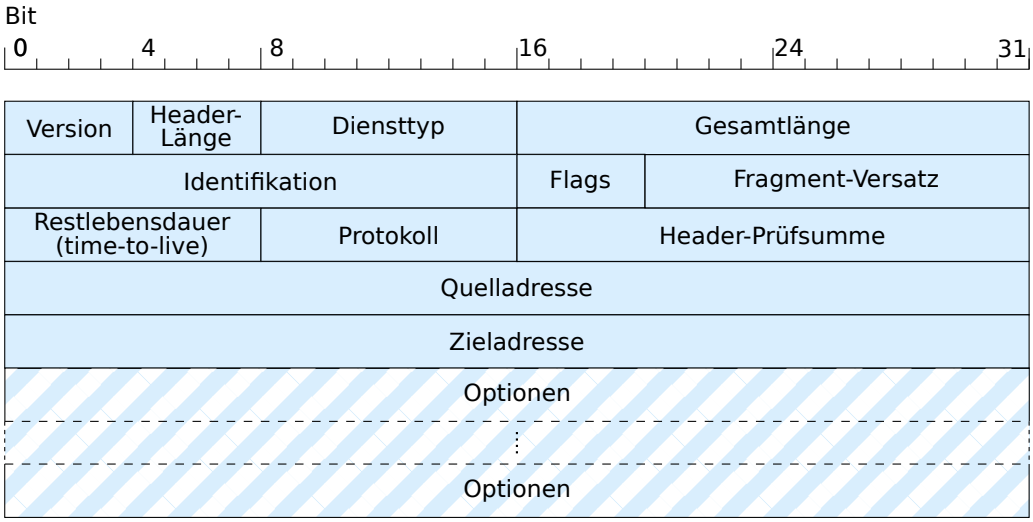

Abbildung 2.8: Format eines IPv4-Paketkopfs

zerteilen. Das Ziel-Endgerät oder ein nachfolgender Router kann mit Hilfe dieser Informationen das Paket wieder zusammensetzen.

Rest-Lebenszeit (*time-to-live*) – Diese muss von jedem Router jeweils um eins verringert werden. Ist die Restlebenszeit Null verwirft er das Paket, damit für längere Zeit fehlgeleitete Pakete nicht weiterhin das Netz belasten. Tatsächlich handelt es sich somit nicht um eine Zeit, sondern um die noch zulässige Anzahl von Routern bis zum Ziel-Endgerät.

Protokoll – Damit kann das Empfänger-Endgerät feststellen, wie die Daten auf Schicht 4 zu verarbeiten sind. Router verwenden diese Information zur Klassifizierung von Paketen.

Header-Prüfsumme – Zur Erkennung von Bitfehlern muss der Router eine Prüfsumme über die Header-Felder berechnen und mit dieser empfangenen Prüfsumme vergleichen. Bei Ungleichheit wird das Paket verworfen. Nach Änderung von Header-Feldern muss der Router die Prüfsumme aktualisieren.

Quelladresse – Diese Information identifiziert den Sender und kann vom Router zur Klassifizierung des Pakets verwendet werden.

Zieladresse – Diese Information identifiziert den Empfänger und wird vom Router verwendet, um das Paket in dessen Richtung weiterzuleiten. Eine IP-Adresse lässt sich in eine Netz- und eine Endgerätadresse zerteilen, wobei beliebig viele Bits der IP-Adresse als Netzadresse verwendet werden können[3]. Dies erfordert bei der Suche nach den Weiterleitungsinformationen das Finden des Eintrags mit dem längsten übereinstimmenden Netzpräfix in der Routing-Tabelle.

Optionen – Damit können im Header weitere Informationen für zusätzliche Funktionen angegeben werden.

Näheres zur Verarbeitung von IPv4-Paketen in Routern ist in RFC 1812 [31] beschrieben.

[3]Dieses *Classless Inter-Domain Routing* (CIDR) Addressierungsschema ersetzte aufgrund der explosionsartigen Zunahme der Internet-Nutzer Anfang der 1990er Jahre ein klassenbasiertes Adressierungsschema.

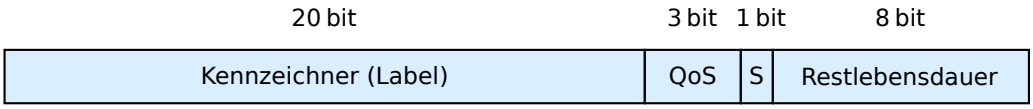

Abbildung 2.9: Format eines MPLS-Header

c. Multi-Protocol Label Switching – ein verbindungsorientiertes Protokoll unterhalb IP

Das verbindungsorientierte *Multi-Protocol Label Switching* (MPLS) ist in den RFCs 3031 und 3032 [32, 33] definiert. Ursprünglich wurde es entwickelt, um die aufwändige IP-Adresssuche in Routern zu vermeiden. Durch Einfügen eines Kennzeichners (*Label*) sollte die Ermittlung der Weiterleitungsinformation durch einfache Indizierung einer Tabelle möglich sein. Aufgrund sehr schneller, Hardware-basierter IP-Adresssuchmodule ist dies allerdings nicht der Grund, weshalb MPLS heute sehr häufig verwendet wird. Einsatzzweck ist zum einen die gezielte Verkehrssteuerung und -verwaltung (*traffic engineering*) innerhalb eines Netzes und zum anderen die Einrichtung virtueller privater Netze (*Virtual Private Network*, VPN) auf Schicht 2 oder 3 mit Hilfe von Tunnels.

In einem MPLS-Netz bilden die Rand-Router die Pakete, die durch das Netz transportiert werden sollen, entsprechend ihrer Zieladresse sowie u. U. anderer Kriterien auf Weiterleitungs-Äquivalenzklassen ab und weisen ihnen entsprechende Kennzeichner für einen vorher eingerichteten Pfad (Verbindung) durch das Netz zu. Die nachfolgenden Router innerhalb des Netzes können damit sehr effizient die Weiterleitungsinformationen der Pakete ermitteln. Wenn die Pakete das Netz wieder verlassen, entfernt der jeweilige Rand-Router den Kennzeichner wieder.

Die Kennzeichner sind jeweils nur lokal auf einer Punkt-zu-Punkt-Verbindung gültig. Daher muss jeder Router entsprechend den Informationen in seiner Weiterleitungstabelle den aktuellen Kennzeichner durch einen neuen ersetzen. Um MPLS-Pfade auch hierarchisch ineinander verschachteln zu können, z. B. zur Realisierung von Tunnels, können mehrere MPLS-Kennzeichner über einander „gestapelt“ werden, man spricht dann von einem *MPLS-Label-Stack*. Die Einrichtung von MPLS-Pfaden in einem Netz kann sowohl „von Hand“ als auch über verschiedene Steuerprotokolle erfolgen.

Abbildung 2.9 zeigt den Aufbau eines MPLS-Header. Dieser wird bei IP-Paketen zwischen den Schicht-2- und den IP-Header geschoben, entsprechend wird dieses Protokoll häufig als ein Protokoll der Schicht 2,5 bezeichnet. Der Header enthält den 20 bit breiten Kennzeichner (*Label*), ein 3 bit breites Feld zur Differenzierung der Dienstgüte (QoS), ein Bit zur Markierung des untersten Kennzeichners eines Stapels (S) sowie eine 8 bit breite Restlebensdauer. Letztere dekrementiert jeder Router wie das entsprechende Feld im IP-Header, wodurch bei einem Wert von Null lange Zeit fehlgeleitete Pakete erkannt und verworfen werden können.

d. User Datagram Protocol – ein einfaches Transportschicht-Protokoll

Das *User Datagram Protocol* (UDP) ist ein einfaches, ungesichertes, verbindungsloses Transportschicht-Protokoll. Es bietet eine Ende-zu-Ende-Kommunikation zwischen zwei Endgeräte-Anwendungen. Diese werden über Port-Nummern auf dem jeweiligen Endgerät identifiziert. Hauptsächliche Einsatzbereiche sind die Übertragung von Multimedia-Daten, bei denen eine

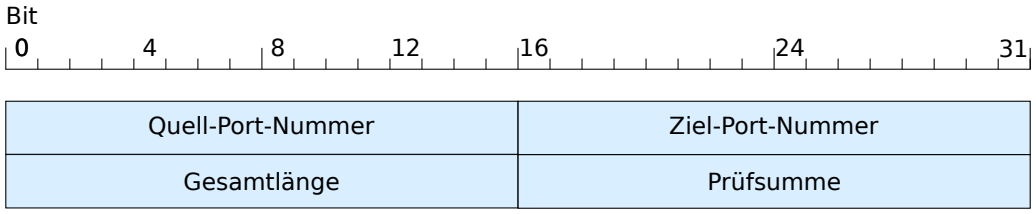

Abbildung 2.10: Format eines UDP-Header

pünktliche Zulieferung wichtiger ist als die Zuverlässigkeit, sowie kurze Datenübertragungen, bei denen das Aufsetzen einer Verbindung einen übermäßig großen Aufwand darstellen würde (z. B. bei der Auflösung eines Domain-Namens in eine IP-Adresse). Das UDP-Protokoll ist in RFC 768 [34] spezifiziert.

Abbildung 2.10 stellt den Header von UDP dar. Er enthält die 16 bit breite Quell-Port-Nummer, die 16 bit breite Ziel-Port-Nummer, ein 16 bit breites Feld mit der Gesamtpaketlänge inklusive des UDP-Headers sowie ein 16 bit breites Prüfsummenfeld. Die Prüfsumme wird hierbei über die Nutzdaten sowie einen Pseudo-Header berechnet. Dieser besteht aus dem UDP-Header, den IP-Quell- und -Zieladressen, der UDP-Protokollnummer aus dem IP-Header sowie (einem zweiten Mal) der Gesamtlänge aus dem UDP-Header.

Router verarbeiteten den UDP-Header ursprünglich nicht, da es sich bei UDP um ein Schicht-4-Protokoll zwischen den Endgeräten handelt. Allerdings verwenden Router heute die UDP-Quell- und Ziel-Ports in zunehmendem Maße zur Klassifikation von Paketen zur differenzierten Verarbeitung sowie zur Adressumsetzung am Rande lokaler Netze (*Network Adress Translation*, NAT).

e. Transmission Control Protocol – ein zuverlässiges Transportschicht-Protokoll

Das am häufigsten verwendete Transportschicht-Protokoll im Internet und neben IP eines der beiden zentralen Protokolle des Internets ist das *Transmission Control Protocol* (TCP). Es stellt Endgeräten eine zuverlässige, verbindungsorientierte, bidirektionale Ende-zu-Ende-Kommunikation zur Verfügung. Zur eindeutigen Identifizierung der kommunizierenden Anwendungen oder Prozesse auf den Endgeräten verwendet es dieselben Port-Nummern wie UDP. Das Protokoll wurde in RFC 793 [29] spezifiziert.

Eine Kommunikation mit TCP läuft folgendermaßen ab: Nach Aufbau der Verbindung zwischen den beiden Endsystemen durch einen Drei-Wege-*Handshake*-Mechanismus, beginnt der Sender mit der Übertragung von Daten. Um das Netz und die Vermittlungsknoten nicht zu überlasten, wird hierbei ein fensterbasierter Überlastregelungsmechanismus (*congestion control mechanism*) verwendet. Der Sender darf hierbei nur Daten entsprechend der Größe seines aktuellen Sendefensters versenden, bis diese vom Empfänger durch Antwortpakete bestätigt wurden. Zu Beginn einer Übertragung, in der so genannten *Slow-Start*-Phase (langsame Startphase), beginnt der Sender mit einer sehr kleinen Fenstergröße und vergrößert diese mit der Zeit exponentiell. Ab einem bestimmten Schwellwert geht er in die *Congestion-Avoidance*-Phase (Überlastvermeidungsphase) über, in der er das Sendefenster vorsichtiger erhöht. Erkennt der Sender Paketverluste, verringert er die Fenstergröße sehr stark, um die Überlastsituation zu

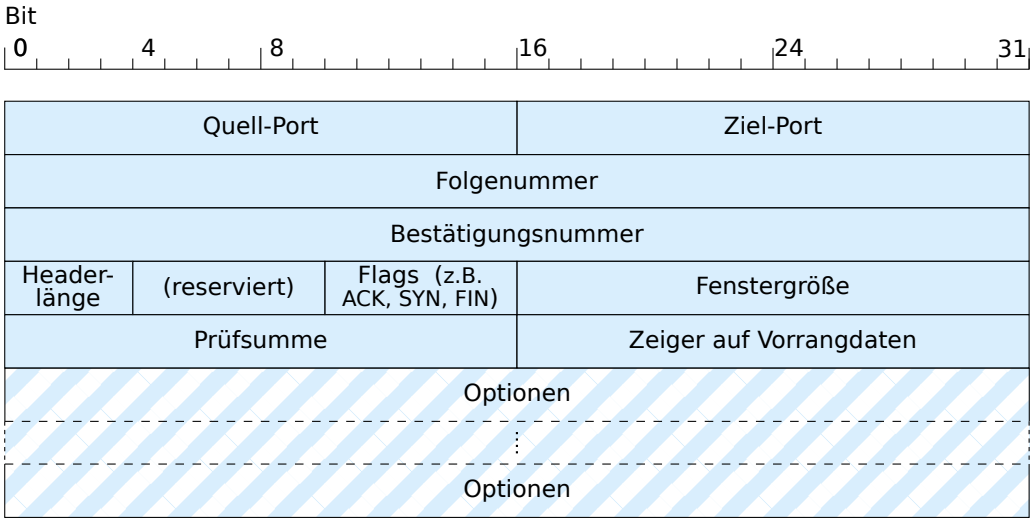

Abbildung 2.11: Format eines TCP-Header

entschärfen. Anschließend erhöht er die Fenstergröße wieder vorsichtig. Forscher diskutieren diesen Überlastmechanismus seit einigen Jahren intensiv und schlagen Änderungen und Erweiterungen von TCP sowie neue Protokolle vor, da insbesondere bei Verbindungen mit hoher Kapazität diese aufgrund des konservativen Verhaltens von TCP oft nicht optimal ausgenutzt werden. Abschnitt 2.1.3 stellt entsprechende Vorschläge vor, welche u. a. Eingriffe in die Verarbeitung von Routern erfordern.

Abbildung 2.11 zeigt den Aufbau eines TCP-Headers. Neben Quell- und Ziel-Port enthält er eine Folgenummer zur Reihenfolgesicherung, eine Bestätigungsnummer zur Bestätigung von empfangenen Paketen sowie eine Empfangsfenstergröße zur Flussregelung (*flow control*). Über die Header-Länge kann man erkennen, ob und wie viele Optionen der Header enthält, und somit, wo die Nutzdaten beginnen. Des Weiteren enthält der Header sechs Flags, u. a. ein Bestätigungsbit ACK, das die Bestätigungsnummer für gültig erklärt, sowie ein SYN- und ein FIN-Bit zur Kennzeichnung eines Verbindungsauf- bzw. -abbaus. Das Prüfsummenfeld wird wie bei UDP über den Header, die Nutzdaten sowie einen Pseudo-Header bestehend aus Quell- und Ziel-IP-Adressen, TCP-Protokollnummer sowie der Gesamtlänge des TCP-Pakets berechnet.

Als Ende-zu-Ende-Protokoll sollte TCP, ebenso wie UDP, in Vermittlungsknoten eigentlich keine Rolle spielen. Allerdings werden zunehmend einzelne Header-Felder, insbesondere die Port-Nummern sowie das SYN-Feld, zur Klassifizierung und Erkennung von Verkehrsflüssen in Routern verwendet. Mit diesen Informationen können die Vermittlungsknoten dann unterschiedliche Flüsse unterschiedlich verarbeiten und somit beispielsweise zwischen verschiedenen Dienstgüteklassen differenzieren oder für verschiedenen Flüsse unterschiedliche Dienste bereitstellen.

f. Explicit Congestion Notification – eine Erweiterung von TCP/IP

Um Überlastsituationen im Netz zu erkennen, benötigt das sendende Endgerät eine diesbezügliche Rückmeldung aus dem Netz. TCP verwendet Paketverluste als impliziten Indikator für eine solche Überlastung des Netzes und verringert daraufhin sein Sendefenster. Dieser Mechanis-

mus ist nicht optimal, da zum einen Paketverluste auch andere Gründe haben können und zum anderen eine Reaktion auf Überlast, *bevor* Pakete verloren gehen, zu bevorzugen ist.

Aus diesem Grund führte die IETF im Jahre 2001 in RFC 3168 [35] die *Explicit Congestion Notification* (ECN, explizite Überlastbenachrichtigung) ein. Dieser Mechanismus sieht vor, dass überlastete Router, bevor sie Pakete verwerfen müssen, einzelne Pakete durch Setzen eines bestimmten Bits im IP-Diensttyp-Header-Feld markieren. Das empfangende Endgerät sendet diese Mitteilung durch Setzen eines der sechs bis dahin reservierten Bits im TCP-Header zum Sender-Endgerät. Daraufhin reagiert der Sender wie bei einem Paketverlust und reduziert seine Senderate. Zusätzlich bestätigt er die Reduktion seines Sendefensters durch Setzen eines anderen der sechs bis dahin reservierten Bits im TCP-Header. Somit kann das Netz entlastet werden, ohne dass zuvor Pakete verloren gehen müssen.

2.1.3 Zukünftige Netze und Protokolle

Die Kommunikationsnetze und insbesondere das Internet ändern sich ständig. Zur Bereitstellung neuer oder verbesserter Dienste sowie zur Anpassung der bestehenden Protokolle an neue Randbedingungen werden neue Protokolle definiert, alte ersetzt, erweitert oder modifiziert. Entsprechend muss die Verarbeitung in Vermittlungsknoten flexibel angepasst werden können. Dieses Kapitel zeigt aktuelle Beispiele von sich ändernden und neuen Protokollen, für deren Umsetzung die Verarbeitung in Hochgeschwindigkeitsvermittlungsknoten angepasst werden muss.

Paketvermittelnde Transportnetze

Wie bereits in Abschnitt 2.1.1 erwähnt, wird der Datentransport auf Schicht 1 und 2 in Weitverkehrsnetzen zunehmend von leitungsvermittelnden auf paketvermittelnde Technologien umgestellt. Der Grund hierfür ist, dass aufgrund des ständig zunehmenden Datenverkehrs von paketvermittelnden Firmennetzen sowie des Internets hauptsächlich Paketverkehr über diese Netze transportiert wird. Die beiden konkurrierenden Technologien sind hierbei *Provider Backbone Bridging – Traffic Engineering* (PBB-TE) [36] und *Multi-Protocol Label Switching – Transport Profile* (MPLS-TP) [37].

a. Ergänzungen zum Ethernet-Standard: PBB-TE

PBB-TE (Provider Backbone Bridging – Traffic Engineering) basiert auf mehreren Ergänzungen des Ethernet-Standards, die in den letzten Jahren standardisiert wurden, um Ethernet tauglich für die Verwendung in Weitverkehrsnetzen zu machen. Die häufig benutzte englische Bezeichnung für ein solches für Netzbetreiber angepasstes Ethernet ist *Carrier-Grade Ethernet*.

Eine wichtige Anforderung an Carrier-Grade Ethernet ist die Möglichkeit, den Verkehr verschiedener Kunden getrennt voneinander behandeln zu können. Dies ist mit den bereits 1998 im Standard IEEE 802.1Q [38] definierten virtuellen LANs (VLANs, *Virtual Local Area Networks*) möglich, welche durch eine VLAN-Markierung (*VLAN-Tag*) unterschieden werden. Diese Markierung wird zwischen die Quelladresse und das Typfeld des ursprünglichen Ethernet-Rahmens geschrieben (siehe Abbildung 2.12 Zeile 2) und besteht aus der Ethernet-Typbezeichnung für

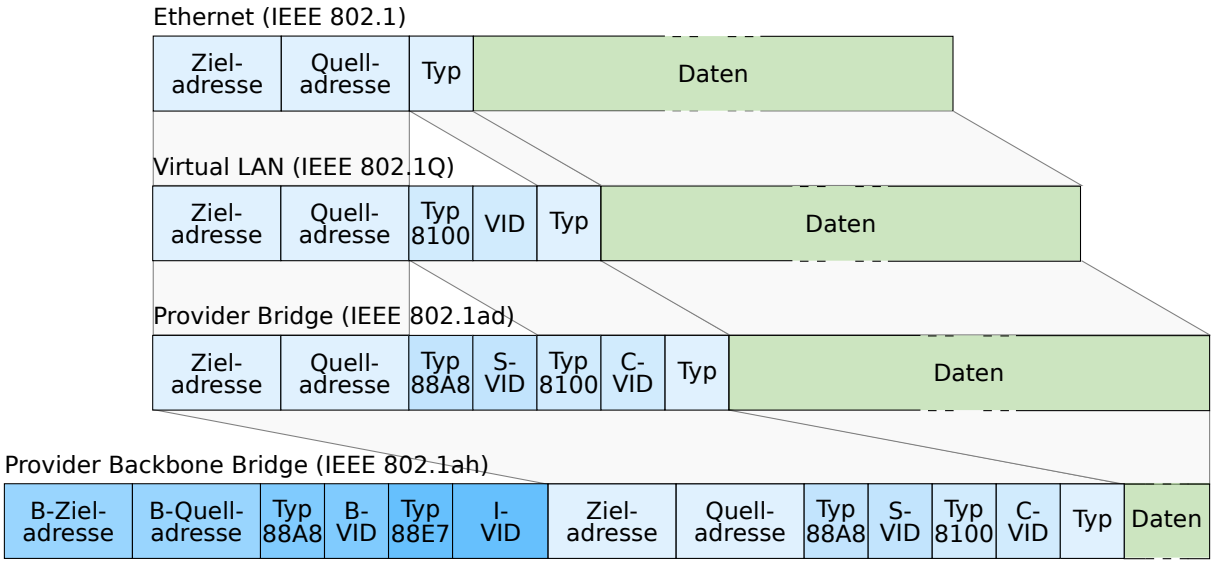

Abbildung 2.12: Erweitertes Ethernet-Rahmenformat für den Einsatz in Weitverkehrsnetzen

VLANs 0x8100, 4 Steuerbits (in der Abbildung nicht dargestellt) und einer 12 bit breiten VLAN-Kennung (*VLAN-Identifier*, VID).

Damit Netzbetreiber auch Rahmen, welche schon vom Kunden mit VLAN-Markierungen versehen wurde, separieren können, wurde 2006 die *Provider Bridge*-Erweiterung (PB) im Standard IEEE 802.1ad [39] definiert. Hiermit kann vor die Kunden-VLAN-Markierung (Typ 0x8100 und C-VID) eine so genannte *Service*-VLAN-Markierung (Typ 0x88A8 und S-VID) eingefügt werden (siehe Abbildung 2.12 Zeile 3).

Der 2008 eingeführte IEEE-Standard 802.1ah [40] beschreibt die *Provider Backbone Bridge*-Erweiterung (PBB). Er definiert einen eigenen *Backbone*-Ethernet-Header, der vor den eigentlichen Ethernet-Rahmen-Header gesetzt wird (siehe Abbildung 2.12 Zeile 4). Dieser Header hat eigene Adressfelder und zwei eigene VLAN-Markierungen (*Backbone*-VLAN-Markierung, entspricht *Service*-VLAN-Markierung: Typ 0x88A8 und B-VID; *Backbone-Service-Instance*-Markierung: Typ 0x88E7 und 24 bit breiter I-VID). Damit können die Vermittlungsknoten innerhalb eines Transportnetzes einen getrennten Adressraum verwenden, was zu kleineren Weiterleitungstabellen führt, und der Netzbetreiber kann durch die mit 24 bit doppelt so große I-VID sehr viele Kunden unterscheiden.

PBB-TE (IEEE 802.1Qay) [36] definiert einen verbindungsorientierten Betrieb eines solchen Netzes und der Standard IEEE 802.1ag [41] beschreibt die Mechanismen für Betrieb, Verwaltung und Instandhaltung (*Operation, Administration and Maintenance*, OAM) des Netzes.

b. Anpassung von MPLS: MPLS-TP

MPLS-TP (Multi-Protocol Label Switching – Transport Profile) [37] wird auf der Basis von MPLS gemeinsam von der IETF und der (*Telecommunication Standardization Sector of the ITU*, *International Telecommunication Union*) standardisiert. Das Ziel ist hierbei, MPLS für den Betrieb in einem paketvermittelnden Weitverkehrsnetz anzupassen.

MPLS-TP übernimmt die Dienstgüte-Mechanismen von MPLS, beschränkt dieses ausschließlich auf verbindungsorientierte Kommunikation und fügt Funktionen für OAM-Funktionalität hinzu, wie sie in Weitverkehrsnetzen üblich sind. Das Aufsetzen und Beenden von Pfaden ist außer über die üblichen MPLS-Steuerprotokolle ebenso über eine Management-Ebene möglich, so dass Pfade auch manuell angelegt werden können [42].

c. Schlussfolgerung

Die Vorstellung dieser beiden konkurrierenden Protokolle für paketvermittelnde Weitverkehrsnetze zeigt die Notwendigkeit, die Paketverarbeitung in Vermittlungsknoten flexibel an sich ändernde Anforderungen anpassen zu können. Zum einen ist lange Zeit nicht klar, welches der beiden Protokolle sich schließlich durchsetzen wird, so dass die Vermittlungsknoten in der Lage sein sollten, flexibel an beide Standards angepasst zu werden. Zum anderen sollen diese Protokolle oftmals bereits unterstützt werden, bevor die Standardisierung abgeschlossen ist, so dass auch zu einem späteren Zeitpunkt noch Änderungen in den Protokollen möglich sind.

IPv6 – Internet Protokoll Version 6

IPv6 wurde bereits 1998 im RFC 2460 [43] spezifiziert, da bereits damals abzusehen war, dass die 32 bit breiten Adressen von IPv4 ausgehen werden. Aktuelle Schätzungen gehen davon aus, dass innerhalb der nächsten ein bis zwei Jahre alle vier Milliarden IPv4-Adressen vergeben sein werden. IPv6 wurde daher so entworfen, dass die Adressen nie ausgehen werden. Wie in Abbildung 2.13 zu sehen, sind die IPv6-Adressen 128 bit breit. Zusätzlich zu dem enorm großen Adressraum erlauben diese breiten Adressen auch eine bessere hierarchische Strukturierung des Adressraums, so dass die IP-Adresssuche vereinfacht wird.

Eine weitere Vereinfachung der Verarbeitung bringt die kleinere Anzahl der Header-Felder, indem selten gebrauchte Felder von IPv4 in Optionen gewandelt wurden. Neben den Quell- und Zieladressen enthält der Header die Versionsnummer 6, ein 8 bit breites Verkehrsklassenfeld sowie 20 bit für eine Flusskennung. Ein 8 bit breites Feld spezifiziert den nächsten Header und ersetzt das Protokollfeld des IPv4-Headers. Dieses neue Feld gibt entweder den folgenden Schicht-4-Header an, oder im Falle von IPv6-Optionen deren Erweiterungs-Header. Somit muss das Vorhandensein von Optionen nicht mehr aus der Header-Länge abgeleitet werden und der Zugriff auf die Optionen wird vereinfacht. Das Restlebenszeitfeld (TTL) wird durch das Feld Teilstreckenlimit mit derselben Funktion ersetzt. Die Header-Prüfsumme wurde entfernt, da davon auszugehen ist, dass die Integrität der Daten auch auf Schicht 2 und/oder auf Schicht 4 überprüft wird. Des Weiteren müssen Router keine Fragmentierung und Defragmentierung von IP-Paketen mehr durchführen.

IPv6-Unterstützung wird schon seit mehreren Jahren von allen großen Router-Herstellern in ihre Produkte integriert. Dennoch wird sich in den nächsten Jahren zeigen, in wie weit diese auch für große IPv6-Verkehrsströme dimensioniert sind. Des Weiteren ist zu erwarten, dass durch den ersten großen, flächendeckenden Einsatz in den nächsten Jahren sicher viele Erweiterungen und Verbesserungen vorgeschlagen werden, sowohl was die Protokolle betrifft als auch was die konkrete Verarbeitung von IPv6-Paketen in Routern betrifft. Daher ist eine Paketverarbeitungsarchitektur, die flexible Anpassungen ermöglicht, in Routern zwingend erforderlich.

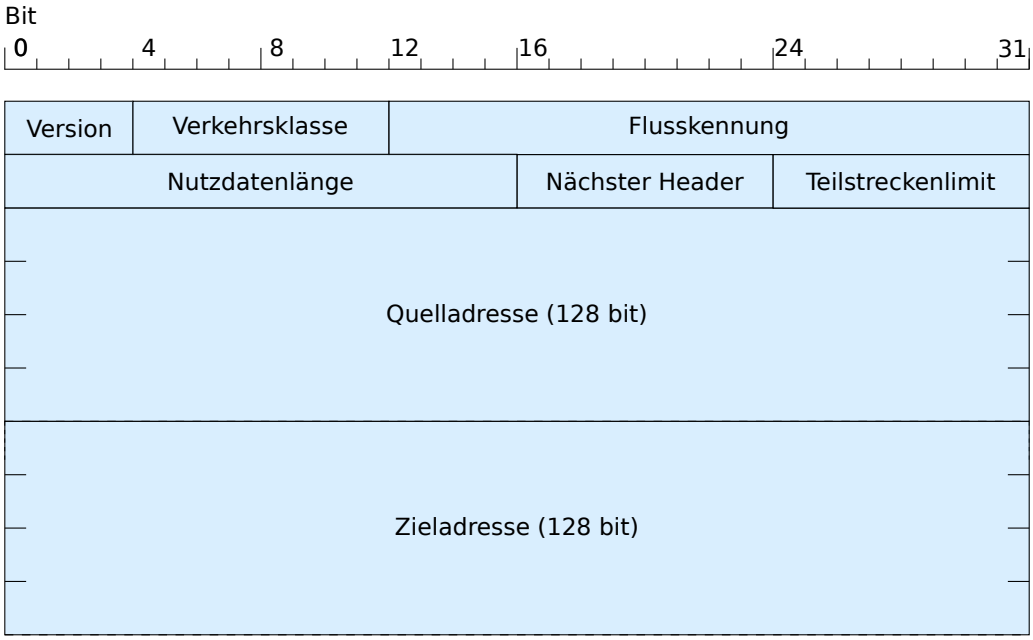

Abbildung 2.13: Format eines IPv6-Paketkopfs

Router-unterstützte Überlastregelung

Der Überlastregelungsmechanismus von TCP versucht, wie bereits in Abschnitt 2.1.2 dargestellt, eine optimale Netzauslastung durch kontinuierliches, vorsichtiges Erhöhen der Senderate und Zurücksetzen der Rate bei Paketverlust zu erreichen. Jedoch können auf diese Weise TCP-Verbindungen bei großen Umlaufzeiten, wie sie bei Interkontinental-Verbindungen vorkommen, große verfügbare Bandbreitekapazitäten, z.B. auf schwach ausgenutzten Verbindungen, nur sehr langsam nutzen. Daher wurden in den letzten Jahren verschiedene TCP-Erweiterungen sowie Protokolle mit alternativen Überlastregelungsmechanismen vorgeschlagen, die eine explizite Router-Rückmeldung zur Einstellung der optimalen Senderate am Endgerät vorsehen.

Die vorne beschriebene ECN-Erweiterung verwendet bereits eine explizite Rückmeldung der Router bezüglich Überlastsituationen über ein einzelnes gesetztes Bit im IP-Header. Mehrere aktuelle Forschungsaktivitäten untersuchen Protokolle, die eine umfangreichere Rückmeldung der Router vorsehen. Diese soll über zusätzliche Protokoll-Felder direkt eine maximal erlaubte Senderate oder eine erwünschte Änderung der Senderate an die Endgeräte übermitteln. Die wichtigsten Vorschläge stellen in diesem Zusammenhang *Quick-Start* TCP [44], das *Explicit Control Protocol* (XCP) [45] und das *Rate Control Protocol* (RCP) [46] dar.

a. Quick-Start TCP

Quick-Start TCP [44] ist eine experimentelle TCP-Erweiterung, die es Endsystemen ermöglicht, den Routern entlang ihrer Verbindung eine explizite Bandbreiteanforderung zu stellen. Die Router, die das Paket weiterleiten, werten diese Bandbreitenanforderung aus, akzeptieren sie, verringern die geforderte Bandbreite auf einen für sie akzeptablen Wert oder lehnen die Anforderung ab. Das Ergebnis, die von allen Routern akzeptierte Bandbreite oder die Ablehnung,

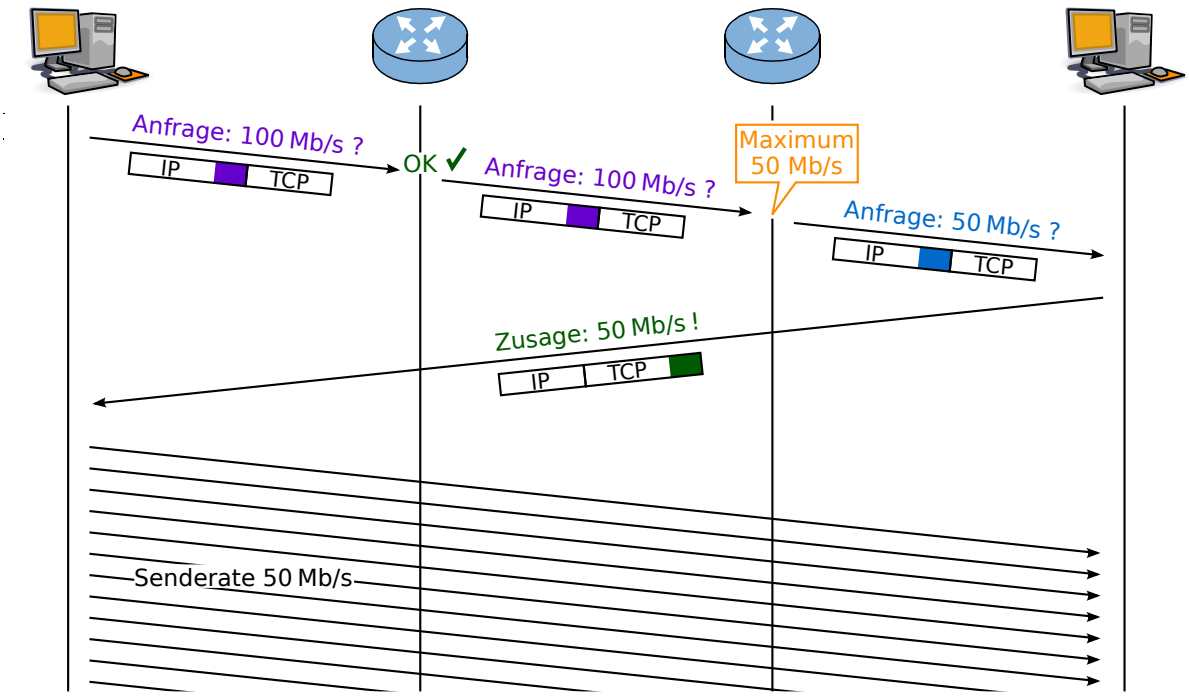

Abbildung 2.14: Ablauf einer Quick-Start Anfrage von 100 Mbit/s. Der zweite Router verringert die zugesagte Rate auf 50 Mbit/s.

sendet das empfangende Endsystem an den Sender zurück. Dieser kann dann mit der entsprechenden Senderate senden und anschließend die normale TCP-Überlastregelung anwenden.

Quick-Start wurde insbesondere dafür entworfen, den langsamen Slow-Start zu Beginn einer TCP-Übertragung zu ersetzen, und somit mittellange Übertragungen von einigen Mega-Byte zu beschleunigen. Der Mechanismus kann jedoch auch nach Sendepausen zur Ermittlung einer neuen Senderate verwendet werden. Nach welcher Strategie die Router bestimmen, welche Senderate sie erlauben, ist nicht spezifiziert.

Abbildung 2.14 zeigt den Ablauf einer Bandbreiteanforderung während des Verbindungsaufbaus. Die gewünschte Bandbreite wird als IP Option in ein TCP/IP-Paket (in diesem Fall in das SYN Paket) geschrieben. Die Router verarbeiten diese Option und leiten das Paket an den Empfänger weiter. Dieser sendet das Ergebnis nun über eine TCP-Option zurück. Diese wird von Routern üblicherweise nicht betrachtet.

b. XCP und RCP

Das *Explicit Control Protocol* (XCP) [45] und das *Rate Control Protocol* (RCP) [46] sind beide als eigenständige Protokolle zwischen IP und TCP mit dem Ziel definiert, die Überlastregelung vollständig von TCP zu übernehmen. Daher findet die Regelung kontinuierlich statt, so dass jedes übertragene Paket den entsprechenden XCP- bzw. RCP-Header enthält. Somit ist i. d. R. auch eine höhere Paketverarbeitungsleistung in den Routern erforderlich.

Der Ablauf zur Festlegung einer Senderate in den Endgeräten erfolgt ähnlich wie bei Quick-Start TCP. Bei allen drei Protokollen stellen die Endgeräte eine Senderatenanforderung an die Router und diese akzeptieren oder verringern diese. Nachdem das Paket alle Router auf dem Weg zum Empfänger passiert hat, ist somit die maximale von allen Routern akzeptierte Rate im Header enthalten. Der Empfänger sendet dieses Ergebnis mit dem nächsten Paket zum Sender, so dass dieser seine Senderate entsprechend einstellen kann.

Allerdings unterscheiden sich die Protokolle im genauen Typ der Anfrage. Während XCP die Senderate der Endsysteme über Ratenänderungen anpasst, bestimmt RCP ebenso wie Quick-Start eine absolute Senderate. Des Weiteren ist die Ermittlung der von den Routern akzeptierten Rate unterschiedlich komplex. XCP erfordert einen relativ komplexen Algorithmus zur Ermittlung der für jedes Paket erlaubten Ratenänderung. Bei RCP muss eine in regelmäßigen Abständen berechnete Maximalrate lediglich mit der geforderten Rate verglichen werden und diese, falls sie zu groß ist, entsprechend angepasst werden.

c. Schlussfolgerung

Alle oben beschriebenen Überlastregelungsmechanismen erfordern Rückmeldungen von den Routern. Daher muss die Paketverarbeitung in den Routern, falls sich zukünftig ein solcher Mechanismus durchsetzt, entsprechend angepasst und erweitert werden. In Unterkapitel 4.4 wird eine prototypische Umsetzung dieser neuen Protokolle für die in dieser Arbeit vorgestellte, neue Paketverarbeitungsarchitektur beschrieben. Damit soll gezeigt werden, dass und in wie weit ein System mit dieser Architektur auch für zukünftige Protokolle und Protokollerweiterungen anpass- und erweiterbar ist.

Experimentelle Netze zu Forschungszwecken

Das heutige Internet hat prinzipielle Einschränkungen hinsichtlich Aspekten wie Mobilität, Sicherheit, Zuverlässigkeit, Energieverbrauch, u. v. m. In den letzten Jahren wird daher vermehrt über die Erforschung grundlegend neuer Internet-Architekturen diskutiert, ohne dabei die Einschränkungen der heutigen Protokolle und Mechanismen berücksichtigen zu müssen. Dies wird als *Clean Slate*-Ansatz bezeichnet. [47]

Um auch die Implementierung und den Betrieb der verschiedenen Elemente solcher neuartigen Netzarchitekturen und -technologien testen zu können, werden weltweit Experimental-Netzinfrastrukturen aufgebaut, wie z. B. GENI (*Global Environment for Network Innovations*) [48], FEDERICA (*Federated E-infrastructure Dedicated to European Researchers Innovating in Computing network Architectures*) [49] oder G-LAB (*German Lab*) [50]. Bei diesen Netzen wird großer Wert auf die Programmierbarkeit und Anpassbarkeit der Vermittlungsknoten gelegt, um damit verschiedene neue Mechanismen und Dienste testen zu können.

In diesem Zusammenhang wird auch über Netzvirtualisierung (*Network Virtualization*) nachgedacht. Dies bedeutet, dass eine physikalische Netzinfrastruktur, bestehend aus Endgeräten, Vermittlungsknoten und Übertragungsleitungen für mehrere unterschiedliche, logische Netze verwendet werden kann. Zur Unterscheidung der Pakete unterschiedlicher logischer Netze an den Vermittlungsknoten wurde der so genannte *Open Flow*-Standard entwickelt [51]. Dieser

sieht eine Klassifizierung aller ankommender Pakete anhand ihrer Header-Felder der Protokolle auf Schicht 2 bis 4 sowie eine Modifizierung dieser Header und die Weiterleitung dieser Pakete entsprechend des Klassifizierungsergebnisses. Zusätzlich spezifiziert der Standard eine einheitliche Management- und Steuerungsschnittstelle zur Anpassung der Klassifizierungsregeln sowie zur Verarbeitung unbekannter Pakete.

2.1.4 Vermittlungsknoten: Gestern, Heute und Morgen

In den letzten Abschnitten wurde Grundlegendes über Kommunikationsnetze und die wichtigsten Protokolle heutiger und zukünftiger paketvermittelnder Netze dargestellt. Eines der drei fundamentalen Elemente eines solchen Netzes sind die Vermittlungsknoten (vgl. Abschnitt 2.1.1), deren Aufgabe die Weiterleitung von eingehenden Paketen in Richtung des adressierten Ziel-Endgeräts ist. Ein Vermittlungsknoten auf Schicht 2 wird i. d. R. als Switch, einer auf Schicht 3 als Router bezeichnet.

In diesem Abschnitt werden die Aufgaben von Paketvermittlungsknoten ausführlicher erklärt. Anschließend wird die prinzipielle Architektur eines modernen Hochgeschwindigkeitsvermittlungsknotens vorgestellt, mit welcher die heute und zukünftig benötigten, hohen Leistungsanforderungen erreicht werden können. Näheres hierzu kann auch in [52] und [53] nachgelesen werden.

Aufgaben der Datenebene

Die Hauptaufgabe eines Vermittlungsknotens ist die schnelle Weiterleitung von Paketen. Diese Aufgabe lässt sich in verschiedene Teilaufgabengebiete untergliedern: Den Empfang und das Versenden der Pakete an den Leitungsschnittstellen, die Verarbeitung der Protokoll-Header, das Puffern jedes Pakets, bis die Ausgangsschnittstelle frei ist, sowie der Transport der Pakete innerhalb des Vermittlungsknotens.

Empfang und Versand – Ein Leitungsschnittstellensystem (PHY-Chip, *Physical-Layer*-Chip) empfängt die auf der Leitung übertragenen Signale jedes Pakets. Je nach Übertragungstechnologie wandelt es diese in elektrische Signale um und dekodiert diese. Außerdem prüft das System den so erhaltenen Bitstrom auf Integrität. Anschließend werden die Daten zu Worten parallelisiert dem Paketverarbeitungssystem übergeben. Beim Versand wandelt das Leitungsschnittstellensystem entsprechend jedes Paket wieder zurück und versendet es über die angeschlossene Übertragungsleitung.

Paketverarbeitung – Ein Paketverarbeitungssystem analysiert die Protokoll-Header-Felder jedes Pakets und bestimmt damit, über welche Ausgangsschnittstelle dieses Paket weitervermittelt werden muss. Weitere Aufgaben sind je nach Einsatzgebiet des Vermittlungsknotens die Veränderung einzelner Protokollfelder, das Aufzeichnen von Statistiken sowie weitergehende Analysen und Operationen auf den Header-Feldern und teilweise auch auf den Nutzdaten des Pakets. Aufgrund ständig neuer zu unterstützender Protokolle und Dienste muss eine Paketverarbeitungssystem heute flexibel anpassbar sein. In Unterkapitel 2.2 wird tiefergehend auf die Anforderungen heutiger und zukünftiger Paketverarbeitungssysteme eingegangen.

Pufferung – Die Paketspeicher puffern die zu vermittelnden Pakete in logischen Warteschlangen, bis sie die geforderte Ausgangsschnittstelle versenden kann. Heutige Hochgeschwindigkeitsvermittlungsknoten verwalten viele unterschiedliche Warteschlangen zur Unterscheidung verschiedener Verkehrsklassen mit unterschiedlichen Anforderungen.

Interne Vermittlung – Schließlich müssen interne Kommunikationssysteme die Pakete zwischen den verschiedenen Einheiten eines Vermittlungsknotens transportieren. Wie weiter unten erläutert werden wird, erfolgte dies früher über Bussysteme, während heute komplexe Hochgeschwindigkeitsschaltnetzwerke eingesetzt werden.

Diese Aufgaben muss ein Vermittlungsknoten bei jedem ankommenden Paket durchführen. Er muss sie daher so schnell durchführen können, dass sich keine Pakete aufgrund fehlender Verarbeitungsleistung dauerhaft aufstauen und verworfen werden müssen.[4] Man bezeichnet diese als Aufgaben der **Datenebene.** Die Gesamtheit der Module, die diese Aufgaben durchführen, werden als **Datenpfad** oder aufgrund ihrer schnellen und optimierten Verarbeitung als ***Fast Path*** (schneller Pfad) zusammengefasst.

Aufgaben der Steuerungs- und Managementebene

Neben den oben beschriebenen zeitkritischen Aufgaben muss ein Vermittlungsknoten noch Aufgaben der **Steuerungs- und Managementebene** (*Control Plane* und *Management Plane*) erfüllen. Diese Aufgaben muss er nur bei bestimmten Paketen und somit weniger häufig, unabhängig von ankommenden Paketen in regelmäßigen Abständen oder auf externe Anfragen hin durchführen. Diese Aufgaben sind weniger zeitkritisch, weshalb die entsprechenden Verarbeitungsmodule den ***Slow Path*** (langsamer Pfad) bilden.

Die wichtigste Aufgabe der Steuerungsebene ist der Aufbau und die regelmäßige Aktualisierung der Weiterleitungstabellen. Hierfür tauschen die Vermittlungsknoten über spezielle Protokolle Routing- bzw. Topologie-Informationen aus. Ebenso gehört hierzu die Adressauflösung und Zuordnung von Adressen auf Schicht 2 und 3 sowie der Auf- und Abbau von Verbindungen in verbindungsorientierten Netzen, wie z. B. in einem MPLS-Netz. Die Managementebene stellt Funktionen zur Überwachung, Konfiguration und Wartung des Netzes und des Vermittlungsknotens bereit. Hierfür verfügt das entsprechende Modul über Konfigurations- und Auslese-schnittstellen zu den Modulen des Fast Path. Eine weitere Aufgabe des Slow Path ist schließlich die Verarbeitung von Paketen mit nur sehr selten durchzuführenden Verarbeitungsfunktionen, die aus Effizienzgründen nicht im Fast Path durchgeführt werden. Die Trennung von Datenebene und Steuerungs- und Managementebene in IP-Routern beschreibt auch RFC 3654 [54].

Architektur

Wichtigste Leistungsanforderung an einen Vermittlungsknoten ist, dass er den gesamten ankommenden Datenverkehr schnell genug empfangen, verarbeiten und – so weit möglich – wei-

[4] Paketverluste aufgrund dauerhafter Überlastung einzelner Ausgangsschnittstellen können Vermittlungsknoten natürlich nicht verhindern, da dies unendlich große Paketpuffer erfordern würde. Solche Situationen müssen durch Überlastregelungsmechanismen verhindert bzw. aufgelöst werden (vgl. TCP, XCP und RCP in Abschnitt 2.1.2 und Abschnitt 2.1.3).

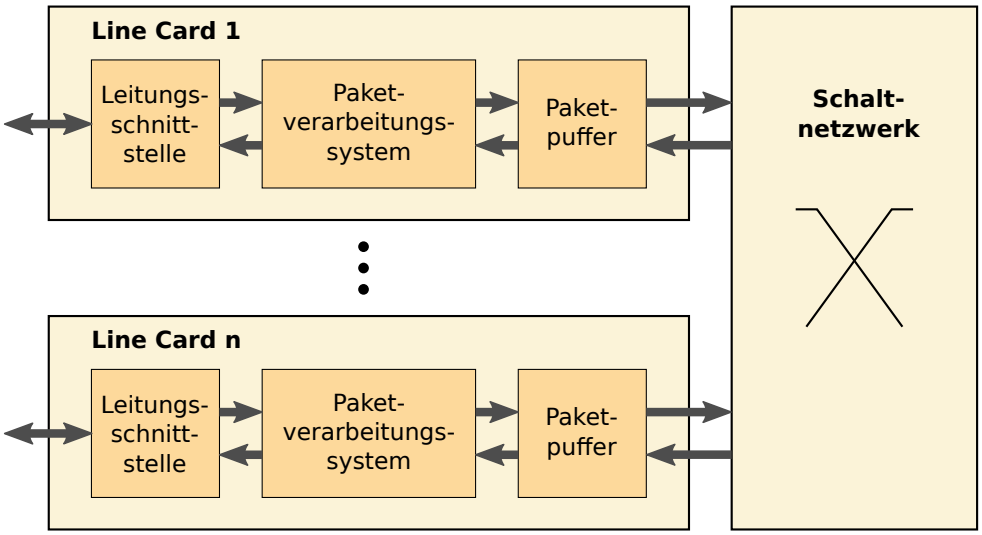

Abbildung 2.15: Architektur eines Hochleistungs-Vermittlungsknoten

terversenden kann. Diese Anforderung hat die Architektur von Vermittlungsknoten im Laufe der Jahre stark beeinflusst [55–57].

Die ersten Vermittlungsknoten in den 1960er und 1970er Jahren basierten auf Computern mit einfachen Netzwerkkarten. Die Netzwerkkarten enthielten lediglich einfache Schaltungen zum Empfangen und Versenden von Paketen. Die empfangenen Pakete wurden über den internen Bus des Computers in den Hauptspeicher übertragen. Die gesamte Verarbeitung erfolgte Software-gesteuert im Prozessor des Computers. Anschließend wurde das Paket wiederum über den Bus zur Ausgangs-Netzwerkkarte übertragen, welche das Paket daraufhin versendete. Solche Software-Vermittlungsknoten waren damals aufgrund der noch sehr niedrigen Übertragungsraten und der geringen Auslastung noch überall möglich. Heute werden PC-basierte Software-Router noch für lokale 10 Mbit/s-, 100 Mbit/s- und 1 Gbit/s-Netze eingesetzt.

Nachteile dieser Computer-basierten Software-Router – insbesondere bei vielen Netzanschlüssen – waren und sind noch immer zwei Flaschenhälse innerhalb des Computers: Zum einen die Übertragungskapazität des Busses, über den jedes Paket mindestens zweimal übertragen werden muss, und zum anderen die Verarbeitungsgeschwindigkeit des Prozessors. Bei vielen angeschlossenen Übertragungsleitungen und voller Auslastung derselben hat der Prozessor für die Verarbeitung eines Paketes nämlich nur wenige Taktperioden zur Verfügung. Beispielsweise ist es bei 1 Gbit/s-Ethernet möglich, dass pro Schnittstelle alle 672 ns ein neues Paket ankommt. Bei acht Schnittstellen verringert sich die Zeit pro Paket somit auf 84 ns (vgl. Abschnitt 2.2.2), was für herkömmliche Computer nicht für eine vollständige Paketverarbeitung ausreicht.

Um diese Flaschenhälse zu entlasten bzw. zu umgehen, verfügen heutige Hochleistungs-Vermittlungsknoten über folgende zwei wesentliche Merkmale:

- Dezentrale Verarbeitung der Pakete direkt auf den Netzwerkkarten
- Hochgeschwindigkeits-Schaltnetzwerk zwischen den Netzwerkkarten

Abbildung 2.15 zeigt den prinzipiellen Aufbau eines heutigen Hochleistungs-Vermittlungsknotens. Der Knoten besteht aus einem Schaltnetzwerk (*Switch Fabric*) und mehreren (im Bild *n*) Netzwerkkarten. Diese Netzwerkkarten sind wesentlich leistungsfähiger (sowie größer und teurer) als herkömmliche Computer-Netzwerkkarten. Zur Unterscheidung bezeichnet man daher die Netzwerkkarten eines Hochleistungs-Vermittlungsknoten meist mit dem englischen Begriff *Line Card*.

Die Verarbeitung der Pakete erfolgt hier direkt in einem Paketverarbeitungssystem auf den Line Cards. Somit muss nicht ein zentraler Prozessor alle Pakete verarbeiten, sondern jedes Paketverarbeitungssystem muss nur noch die Pakete verarbeiten, die über dessen Line Card empfangen (und versendet) werden. Außerdem kann jedes Paket direkt von seiner Eingangs- zu seiner Ausgangs-Line-Card übermittelt werden. Dies entlastet die verwendete Kommunikationsinfrastruktur. Durch den gleichzeitigen Einsatz eines Hochgeschwindigkeits-Schaltnetzwerks (mit Raum-Multiplex-Betrieb) anstatt eines einfachen Busses können heutige Vermittlungsknoten bis zu mehrere Dutzend Line Cards unterstützen.

Jede Line Card empfängt und versendet Pakete über eine (wie im Bild dargestellt) oder mehrere Übertragungsleitungen. Die das Paket empfangende Karte wird als Eingangs- bzw. *Ingress*-Line-Card, die versendende Karte als Ausgangs- bzw. *Egress*-Line-Card bezeichnet.[5] Empfangene Pakete werden wie beschrieben auf der Karte verarbeitet und dann (falls nötig) über das Schaltnetzwerk zu der bei der Verarbeitung ermittelten Ausgangs-Line-Card weitergeleitet, welche diese dann versendet.

Die wesentlichen Elemente auf einer Line Card sind, wie abgebildet, eine Leitungsschnittstelle, ein Paketverarbeitungssystem und ein Paketpuffer. Die Leitungsschnittstelle empfängt und versendet die Pakete. Das daran anschließende Paketverarbeitungssysstem verarbeitet die angekommenen Pakete und reiht sie schließlich in die ermittelte Ausgangs-Warteschlange ein. Die Warteschlangen sind im Paketpuffer realisiert. Dort wird das Paket so lange gespeichert, bis es über das Schaltnetzwerk zur Ausgangs-Line-Card übermittelt werden kann. Dort wird das Paket wiederum gepuffert, bis es über die Leitungsschnittstelle versendet werden kann. Zuvor können im Paketverarbeitungssystem der Ausgangskarte noch verschiedene Statistiken aktualisiert oder weitere, ausgangsspezifische Modifikationen am Paket vorgenommen werden. Die Aufgaben des Slow Path werden bei solchen Architekturen teilweise lokal auf jeder Line Card, teilweise verteilt auf allen Karten zusammen und teilweise auf speziellen Steuerungs- und Managementkarten durchgeführt.

2.1.5 Zusammenfassung

Dieses Einführungsunterkapitel beschrieb grundlegende Konzepte moderner, paketvermittelnder Kommunikationsnetze. Die Paketvermittlung in einem Kommunikationsnetz erfolgt mit Hilfe einer Hierarchie von Protokollschichten, welche für unterschiedliche Aufgaben, wie Punkt-zu-Punkt-Übertragung, Fehlersicherung und Ende-zu-Ende-Kommunikation zuständig sind. Die Protokollinformationen werden in Headern mit den eigentlichen Nutzdaten übertragen und in

[5] Diese Unterscheidung macht bei der Betrachtung der Verarbeitung einzelner Pakete Sinn, global gesehen ist aufgrund von bidirektionalen Übertragungsleitungen meist jede Line Card sowohl Eingangs- als auch Ausgangs-Line-Card, da jede sowohl Pakete empfängt als auch versendet.

den Vermittlungsknoten ausgewertet und verarbeitet. Anhand der Vorstellung der wichtigsten, aktuellen Protokolle auf Schicht 2 bis 4, wurde ein Einblick in die pro Paket notwendige Paketverarbeitung gegeben. Die Einführung einiger (möglicher) zukünftiger Protokolle zeigte die Notwendigkeit flexibel anpassbarer Verarbeitungsarchitekturen in Netzknoten auf. Abschließend wurde die Architektur moderner Vermittlungsknoten dargestellt, die die Paketverarbeitung direkt auf den jeweiligen Line Cards realisieren, um die benötigte schnelle Verarbeitung gewährleisten zu können,

2.2 Anforderungen

In Abschnitt 2.1.4 wurde das Paketverarbeitungssystem als wesentliches Subsystem einer Line Card innerhalb eines Hochgeschwindigkeitsvermittlungsknotens eingeführt. Die drei wesentlichen Kriterien bei der Klassifizierung und Bewertung von Paketverarbeitungssystemen sind ihre *Funktionalität*, ihre *Leistungsfähigkeit* und ihre *Flexibilität*. In den folgenden Abschnitten werden die Anforderungen bezüglich dieser Kriterien diskutiert.

2.2.1 Funktionalität

Ein Paketverarbeitungssystem muss pro Paket verschiedene Aufgaben durchführen. Diese lassen sich in folgende Kategorien einteilen:

- Klassifizierung und Bestimmung der Weiterleitungsinformationen
- Berechnungen und Modifizierung
- Warteschlangen-Einreihung, Verkehrsanpassung und Zuweisung des Sendezeitpunkts

Abbildung 2.16 gibt eine Übersicht über die verschiedenen Aufgabenkategorien. Diese werden im Folgenden näher dargestellt. Zusätzlich zeigt es erforderliche Schnittstellen zum Slow-Path (vgl. Abschnitt 2.1.4), über die Pakete der Steuerungsebene und Daten, wie z. B. Klassifizierungs- und Weiterleitungstabellen sowie Statistiken, ausgetauscht werden können.

Klassifizierung und Bestimmung der Weiterleitungsinformationen – Ein Paketverarbeitungssystem klassifiziert jedes ankommende Paket und ermittelt alle für die Weiterleitung benötigten Informationen. Falls die Klassifizierung ergibt, dass das Paket nicht im Fast Path verarbeitet werden kann, wird es zur weiteren Verarbeitung in den Slow Path übergeben. Fehlerhafte und unerwünschte Pakete werden verworfen.

Je nach Art und Aufgabe des Vermittlungsknotens erfolgt diese Funktionalität anhand eines einzigen Header-Feldes, anhand eines Tupels von Header-Feldern oder sogar auch anhand der Nutzdateninhalte. Beispielsweise erfolgt im Falle von Ethernet und MPLS die Klassifizierung ausschließlich durch die Zieladresse und eine eventuelle VLAN-Markierung bzw. durch die MPLS-Markierung. Die Weiterleitungsinformationen können durch einfache Indizierung einer Weiterleitungstabelle ermittelt werden. Im Gegensatz dazu analysiert ein Paketverarbeitungssystem zur Klassifizierung von TCP-Flüssen ein Fünf-Tupel bestehend aus den IP-Quell- und Zieladressen, dem eingesetzten Transportschicht-Protokoll und den dafür

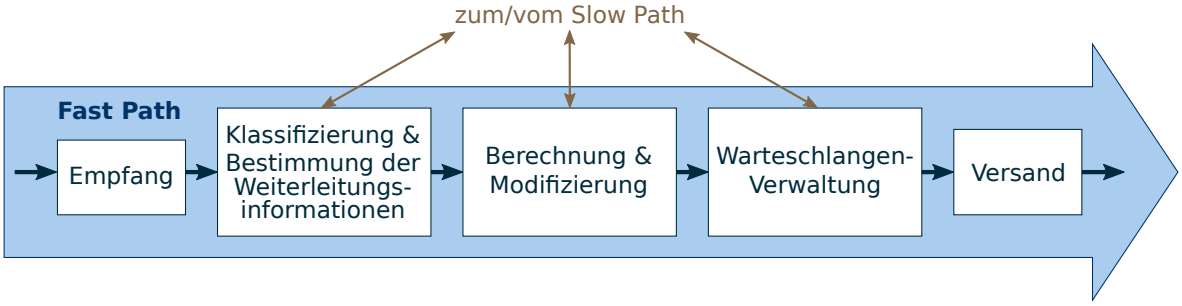

Abbildung 2.16: Fast-Path-Aufgaben eines Paketverarbeitungssystems

verwendeten Quell- und Ziel-Ports. Die Bestimmung der Weiterleitungsinformationen bei IP-Paketen erfordert eine aufwändige Suche nach dem längsten übereinstimmenden Adresspräfix. Für diese Aufgaben führen Paketverarbeitungssysteme aufwändige Traversierungen von Suchbäumen mit mehreren Speicherzugriffen durch [58, 59] oder verwenden ressourcenaufwändige und energieintensive Schaltungen, so genannte dreiwertige inhaltsadressierte Speichermodule (*Ternary Content Addressable Memory*, TCAM) [58, 60]. Schließlich muss ein Paketverarbeitungssystem für eine anwendungsspezifische Aufteilung des Verkehrs (*load balancing*) sowie zum Scannen des Nutzdateninhalts zur Abwehr unerwünschter Pakete, bei der so genannten *Deep Packet Inspection* (tiefgehende Paketdurchsuchung), auf Protokolle oberhalb von Schicht 4 und den Nutzdatenbereich der Pakete zugreifen [24].

Berechnungen und Modifizierung – Entsprechend der Klassifizierung und der gefundenen Weiterleitungsinformationen berechnet und verändert das Paketverarbeitungssystem einzelne Header-Felder oder auch das komplette Paket. Diese Funktionalität umfasst Funktionen wie das Dekrementieren des Time-to-Live-Felds im IP-Header, die Berechnung von Prüfsummen, wie z. B. die Header-Prüfsumme von IP, sowie das Einfügen, Ersetzen oder Entfernen von Header-Feldern, wie z. B. einer VLAN- oder MPLS-Markierung. Teilweise müssen Vermittlungsknoten die kompletten Nutzdaten eines Pakets ändern, z. B. ver- bzw. entschlüsselt ein Router am Rand eines Firmennetzes aus- bzw. eingehenden Verkehr, oder ein Router am Rand eines niederbitratigen Funknetzes transkodiert einen Videostrom in einen mit geringerer Auflösung. Ebenso beinhaltet diese Funktionalität die Aktualisierung von knoteninternen Statistiken.

Warteschlangen-Einreihung, Verkehrsanpassung und Zuweisung des Sendezeitpunkts – Ein Verkehrsverwaltungsmodul (*traffic management module*) reiht das Paket entsprechend seiner Klassifizierung und seiner Weiterleitungsinformationen in eine Warteschlange eines Paketpuffers ein. Um bestimmte Verkehrscharakteristika einzuhalten, die z. B. vertraglich mit einem Kunden vereinbart wurden, kann das Modul auch Pakete verzögern oder verwerfen. Der Fachbegriff hierfür ist *Traffic Shaping* (Verkehrsformung). Schließlich bestimmt das Verkehrsmanagementmodul, wann das Paket versendet wird. Die Festsetzung der Sendezeitpunkte bezeichnet man als *Scheduling* (Zeitplanung). Die Verkehrsverwaltung wird häufig als eigenständiges Modul neben dem Paketverarbeitungssystem realisiert.

Welche konkreten Funktionen ein Paketverarbeitungssystem durchführen können muss, hängt davon ab, wo der Vermittlungsknoten eingesetzt wird. Paketverarbeitungssysteme im Kernnetz müssen (bisher) nur relativ einfache Funktionen auf den Paket-Headern der Schichten 2 bis 4

durchführen. Am Rand eines Kernnetzes ist häufig eine intensivere Klassifizierung der Pakete und ein komplexeres Verkehrsmanagement erforderlich. In zunehmendem Maße sind hier und auch im Metronetz – ebenso wie in Zugangsnetzen – Funktionen notwendig, die auch die Paketnutzdaten analysieren bzw. modifizieren.

Entsprechend dieser unterschiedlichen Einsatzgebiete kann man Paketverarbeitungssysteme danach unterscheiden, auf welchem Datenbereich eines Pakets sie Funktionen ausführen können. Es gibt Systeme, die Funktionen nur auf einem begrenzten ersten Teil, der die Protokoll-Header enthält, ausführen können, und es gibt Systeme, die auf allen Daten eines Pakets Funktionen ausführen können.

2.2.2 Leistungsfähigkeit

Ein Paketverarbeitungssystem muss die ankommenden Pakete so schnell verarbeiten können, wie diese beim System ankommen. Damit keine Pakete verloren gehen, muss das System für den schlimmstmöglichen Fall ausgelegt sein. Die wichtigsten beiden Metriken sind hierbei die *Datenrate* und die *Paketrate*.

Datenrate D ist definiert als Datenmenge pro Zeiteinheit. Sie wird in der Kommunikationstechnik i. d. R. in Bit pro Sekunde (bit/s) angegeben, weshalb auch der Begriff *Bitrate* gebräuchlich ist. Mit Bezug auf die Übertragung der Daten ist auch von der *Übertragungsrate* die Rede.

Paketrate P ist definiert als Anzahl von Paketen pro Zeiteinheit. Sie wird in Paketen pro Sekunde (P/s) angegeben.

Ist der Verarbeitungsaufwand für alle Pakete ungefähr gleich, wie dies bei typischen Header-Verarbeitungen von Protokollen auf Schicht 2 bis 4 der Fall ist, so ist die unterstützte Paketrate entscheidend. Der Grund hierfür ist, dass sie angibt, wie viele Header pro Sekunde ankommen und somit verarbeitet werden müssen. Ihr Kehrwert sagt aus, in welchen zeitlichen Abständen die Pakete ankommen und somit wie viel Zeit für eine Verarbeitung pro Paket zur Verfügung steht, falls keine Parallelverarbeitung stattfindet.

Ist hingegen der Verarbeitungsaufwand proportional zur Paketgröße, dann ist die unterstützte Datenrate die kritische Metrik. Dies ist beispielsweise bei der Verschlüsselung der Nutzdaten oder dem Scannen der Nutzdaten nach bösartigem Inhalt der Fall. Hier muss jedes Bit des Pakets verarbeitet werden. Daher ist entscheidend, wie viele Bits pro Sekunde ankommen.

Besitzt eine Line Card mehrere Leitungsschnittstellen, so muss das Paketverarbeitungssystem die *aggregierte* Daten- bzw. Paketrate verarbeiten können, d. h. die Summe der Daten- bzw. Paketraten aller angeschlossenen Leitungsschnittstellen. Sollen auf der Line Card die ein- und die ausgehenden Pakete im selben Paketverarbeitungssystem verarbeitet werden, verdoppeln sich die geforderten Raten entsprechend.

Aktuelle Datenraten in und am Rand von Kernnetzen liegen aktuell bei 10 bis 40 Gbit/s pro Übertragungsleitung und werden zurzeit auf 100 Gbit/s erhöht. Vergleichbare Datenraten werden in Rechenzentren eingesetzt.

Die Paketrate hängt von der Datenrate ab. Die maximale Paketrate P_{max} einer Übertragungsleitung berechnet sich als Quotient aus deren Datenrate D und der minimalen Paketgröße L_{min}. Muss auf der Leitung ein minimaler Abstand G_{min} (in Byte) zwischen den Paketen eingehalten werden, wie z. B. bei Ethernet 12 Byte Inter-Frame Gap und 8 Byte Präambel, so muss der Nenner des Quotienten um die entsprechende Byteanzahl erhöht werden:

$$P_{max} = \frac{D}{L_{min} + G_{min}} \tag{2.1}$$

Bei Ethernet-basierten Netzen ist die minimale Paketgröße 64 Byte. Somit ergibt sich z. B. für eine Line Card mit einer 100 Gbit/s-Ethernet Leitungsschnittstelle (d. h. $D = 100\,Gbit/s$, $L_{min} = 64\,Byte$, $G_{min} = 20\,Byte$) eine maximale Paketrate P_{max} von ungefähr 149 MP/s (Millionen Pakete pro Sekunde).

Bei anderen Schicht-2-Protokollen ohne minimale Paketgröße und minimalem Paketabstand (z. B. *Packet over SONET/SDH*) wird häufig 40 Byte als minimale Paketgröße angenommen, was der Größe eines TCP/IP-Bestätigungspakets ohne Nutzdaten entspricht (20 Byte IPv4-Header + 20 Byte TCP-Header). Im Vergleich zu solchen Netzen ist damit die maximale Paketrate bei Ethernet-basierten Netzen bei gleicher Datenrate nur etwa halb so groß. Tabelle 2.1 gibt eine Übersicht über die geforderten Paketraten bei verschiedenen Netzkonfigurationen. Anzumerken ist, dass in reinen IPv6-Netzen aufgrund der größeren minimalen IPv6-Header-Größe von 40 Byte entsprechend $L_{min} = 74\,Byte$ bei Ethernet bzw. $L_{min} = 60\,Byte$ bei SDH/SONET ist und somit um 11% bzw. 33% niedrigere Paketraten erreicht werden müssen.

Paketverarbeitungssysteme in Kern- und Metronetzen sollten den geforderten Durchsatz deterministisch erreichen. Bei der Einordnung von Paketverarbeitungssystemen kann zwischen Systemen unterschieden werden, welche einen deterministischen Durchsatz aufgrund ihrer Architektur garantieren, und solchen Systemen, bei welchen nur durch geeignete Konfiguration bzw. Programmierung ein deterministischer Durchsatz erreicht werden kann. Erstere Systeme sind natürlich vorzuziehen.

Die Verzögerung innerhalb eines Paketverarbeitungssystems ist i. d. R. nicht kritisch. Die Übertragungszeiten in Weitverkehrsnetzen liegen im Bereich von Millisekunden. Beispielsweise benötigen Daten in einer Glasfaser für eine Strecke von 1000 km etwa 5 ms. Daher macht sich

Tabelle 2.1: Geforderte Paketraten (pro Richtung) bei verschiedenen Datenraten

D	P_{max} für Ethernet-Minimalrahmen ($L_{min} = 64\,Byte$, $G_{min} = 20\,Byte$)	P_{max} für TCP-Ack über SDH/SONET ($L_{min} = 40\,Byte$, $G_{min} = 0$)
10 Gbit/s	14,9 MP/s	31,3 MP/s
40 Gbit/s	59,5 MP/s	125,0 MP/s
100 Gbit/s	148,8 MP/s	312,5 MP/s
160 Gbit/s	238,1 MP/s	500,0 MP/s

eine zusätzliche Verzögerung der Pakete durch die Verarbeitung in den Vermittlungsknoten im Mikrosekunden-Bereich kaum bemerkbar. Beispielsweise benötigt die Ausführung von 10 000 Befehlen nur etwa 10 μs auf einem einfachen 1- GHz-RISC-Prozessor (*Reduced Instruction Set Computer*, Rechner mit reduziertem Befehlssatz).

2.2.3 Flexibilität

Aufgrund der großen Beliebtheit des Internets steigt das Verkehrsvolumen in den Netzen kontinuierlich an und es werden immer mehr Dienste über das Netz angeboten (vgl. Abschnitt 2.1.3). Damit das Netz den Verkehr dennoch weiterhin effizient vermitteln sowie zusätzlich neue Qualitäts- und Dienstmerkmale bereitstellen kann, passen die verschiedenen Standardisierungsgremien bestehende Protokolle an, erweitern sie oder führen neue Protokolle ein (vgl. [56]). Beispiele für solche Änderungen sind die Einführung von IPv6 oder des erweiterten Ethernet-Standards für Weitverkehrsnetze sowie die mögliche Einführung eines neuen Protokolls zur Router-unterstützen Überlastregelung (vgl. Abschnitt 2.1.3). Genauso sollten beispielsweise auch Scheduling- oder Klassifizierungsmodule an neue Funktions- oder Leistungsanforderungen anpassbar sein.

Aus diesem Grund ist Flexibilität eine der wichtigsten Anforderungen an aktuelle und zukünftige Paketverarbeitungssysteme. Die Verarbeitung soll zum einen anpassbar sein an neue Leistungs- oder Funktionsanforderungen, d. h. das System soll eine hohe *Anpassbarkeit* aufweisen. Zum anderen soll man diese Änderungen möglichst einfach und schnell realisieren können. Dies wird im Folgenden als die *Einfachheit*, Änderungen durchzuführen, bezeichnet.

$$\textbf{Flexibilität} = \textbf{Anpassbarkeit} \times \textbf{Einfachheit}$$

Flexibilität bezeichnet somit die **rasche und einfache Anpassbarkeit** des Verarbeitungssystems an sich ändernde Randbedingungen. Um dies erfüllen zu können, muss ein System also einerseits viele Möglichkeiten bieten, neue Anforderungen in ein bestehendes System zu integrieren. Auch soll es möglich sein, ganze Funktionen hinzuzufügen, auszutauschen oder zu entfernen. Andererseits sollen sich Ingenieure zügig in die Methodik einarbeiten können, um solche Anpassungen vornehmen zu können. Dies kann z. B. durch bekannte Programmiersprachen oder -modelle erreicht werden. Des Weiteren kann das eigentliche Implementieren bzw. Ändern von Funktionen durch einen geeigneten Befehlssatz sowie vordefinierte, parametrisierbare Funktionen bzw. Module vereinfacht und beschleunigt werden. Zusätzlich darf eine Änderung keinen aufwändigen Verifikations- und Fertigungsprozess von mehreren Monaten erfordern.

2.2.4 Zusammenfassung

Dieses Unterkapitel erörterte die Anforderungen an aktuelle und zukünftige Paketverarbeitungssysteme in Vermittlungsknoten. Neben den Anforderungen bezüglich der Funktionalität und der Leistungsfähigkeit ist dies Flexibilität.

Paketverarbeitungssysteme in Kernnetzvermittlungsknoten müssen eher einfache Funktionen auf den Paket-Headern von Paketen ausführen und dies bei Paketraten von mehr als 100 MP/s. An den Rändern der Kernnetze müssen vermehrt komplexe Funktionen sowie Funktionen auf dem gesamten Paket ausgeführt werden. Hier ist die zu verarbeitende Datenrate entscheidend, welche bisher bei 10 bis 40 Gbit/s liegt, jedoch wie die Kernnetze bald auf 100 Gbit/s aufgerüstet wird. Da sich die bestehenden Protokolle in paketvermittelnden Kommunikationsnetzen ständig ändern und kontinuierlich neue Protokolle und Dienste eingeführt werden, müssen neue Paketverarbeitungssysteme flexibel an neue Anforderungen bezüglich ihrer Funktion und ihrer Leistung anpassbar sein.

2.3 Technologie und Bauelemente

Dieses Unterkapitel stellt verschiedene Bauelemente vor, die für die Realisierung von Paketverarbeitungssystemen verwendet wurden und werden. Insbesondere der Einsatz von Netzprozessoren und programmierbaren Logikbausteinen bietet die Möglichkeit, die im vorigen Unterkapitel beschriebenen Anforderungen bezüglich Funktionalität, Leistungsfähigkeit und Flexibilität zu erreichen.

2.3.1 Überblick

Die Verarbeitung von Paketen in einem Vermittlungsknoten kann von jedem der folgenden Bauelemente durchgeführt werden:

ASICs (*Application Specific Integrated Circuits*, Anwendungsspezifische integrierte Schaltungen) sind elektronische Bauelemente, die eigens für die durchzuführenden Verarbeitungsfunktionen entwickelt wurden. Dadurch sind mit diesen Bauelementen die höchsten zu erreichenden Durchsätze bei einem sehr geringen Bedarf an Chipfläche und minimalem Energieverbrauch möglich. Allerdings lassen sich die einmal hergestellten Chips nicht mehr ändern und der Entwurfs- und Fertigungsprozess eines neuen Bausteins ist sehr teuer und zeitaufwändig.

FPGAs (*Field Programmable Gate Arrays*, Programmierbare Logikbausteine) sind konfigurierbare Bauelemente. Hardware-Entwickler können mit diesen Bausteinen mit vergleichsweise geringem Aufwand beliebige, digitale Systeme implementieren. Durch diese hohe Anpassbarkeit bei gleichzeitig hohen Verarbeitungsraten werden FPGAs in den letzten Jahren zunehmend für die Realisierung leistungsstarker, anpassbarer Paketverarbeitungsfunktionen für Kommunikationsnetzknoten eingesetzt. Aufgrund der Rekonfigurierbarkeit ist die erreichbare Taktfrequenz jedoch geringer als bei ASICs und es wird eine größere Chipfläche benötigt.

Netzprozessoren (NPs, *Network Processors*) sind Prozessoren, die speziell für die Verarbeitung von Paketen in Vermittlungsknoten entwickelt wurden. Sie verfügen daher über einen angepassten Spezialbefehlssatz und über schnelle Paketschnittstellen. Um den erforderlichen hohen Durchsatz erreichen zu können, erfolgt die Verarbeitung mittels vieler paralleler

Prozessorkerne. Hochleistungs-NPs unterstützen sehr hohe Paketraten und sind dennoch flexibel in Software programmierbar. Aufgrund architektureller Randbedingungen stoßen sie allerdings insbesondere bei komplexen Verarbeitungsfunktionen und sehr hohen Durchsatzanforderungen teilweise an ihre Grenzen.

Standardprozessoren (GPPs, *General Purpose Processors*) wurden und werden für die Paketverarbeitung in PC-basierten Vermittlungsknoten insbesondere bei niedrigen Übertragungsraten eingesetzt. Entwickler müssen über keine speziellen Programmierkenntnisse verfügen, da sie Verarbeitungsfunktionen in gängigen Hochsprachen implementieren können. Aufgrund ihrer hohen Verbreitung sind Standardprozessoren sowohl kostengünstig als auch hoch optimiert, so dass höchste Taktraten möglich sind. Aus diesem Grund wird immer wieder über die Eignung von Standardprozessoren für die Paketverarbeitung in Hochgeschwindigkeitsnetzen diskutiert.

Alle diese vier Bauelemente – ASICs, FPGAs, NPs und GPPs – basieren auf digitalen integrierten Schaltungen (*integrated circuits*, ICs). Daher werden im folgenden Abschnitt 2.3.2 die wichtigsten Grundlagen bezüglich dieser Technologie dargestellt, insbesondere ihre rasante Entwicklung zu immer kleineren Strukturen und die daraus entstandenen Vor- und Nachteile hinsichtlich Leistungsfähigkeit und Kosten. Daran anschließend werden die vier Bauelementtypen im Detail vorgestellt.

2.3.2 Grundlagen integrierter Schaltungen

Eine integrierte Schaltung ist eine elektronische Schaltung, deren Elemente auf einem gemeinsamen Substrat, in der Regel Silizium, gefertigt wurde. Die ersten ICs wurden im Jahre 1958 von Jack Kilby [61] und 1959 von Robert Noyce [62] unabhängig von einander gebaut.

Die Größe der Transistoren und anderer Bauelemente auf ICs hat sich seitdem kontinuierlich verkleinert, so dass immer mehr Transistoren auf einen Chip integriert werden konnten. Gordon Moore, einer der Gründer von *Intel*, sagte im Jahre 1965 voraus, dass sich in den folgenden Jahren die Anzahl der Transistoren pro IC alle zwölf Monate verdoppeln werde [63]. Zehn Jahre später korrigierte er den Zeitraum für die Verdopplung auf 24 Monate [64]. Nicht zuletzt angespornt durch diese bald als *Moore's Law* (Moores Gesetz) bekannte Vorhersage, erreichte die Halbleiterindustrie in den darauf folgenden Jahrzehnten bis heute immer höhere Integrationsdichten. Abbildung 2.17 dokumentiert die ständig schrumpfenden Strukturgrößen in logarithmischer Darstellung. Auch für die kommenden Jahre prognostizieren Experten der Halbleiterindustrie in der alle zwei Jahre herausgegebenen *International Technology Roadmap for Semiconductors* (ITRS) [65] weitere Verringerungen der Strukturgrößen und zunehmende Transistorzahlen.

Die immer kleineren Strukturen ermöglichen eine Erhöhung der Schaltgeschwindigkeiten der Transistoren und somit höhere Verarbeitungsraten. Außerdem kann die Leistungsaufnahme durch Senkung der Betriebsspannung gesenkt werden. Andererseits haben die heutigen sehr kleinen Strukturgrößen einen sehr komplexen Entwurfsprozess zur Folge. Entwickler und Entwicklungswerkzeuge müssen mehr und mehr physikalische Effekte wie Signallaufzeiten und Übersprechen berücksichtigen. Dies hat zur Folge, dass der Entwurf einer komplexen integrier-

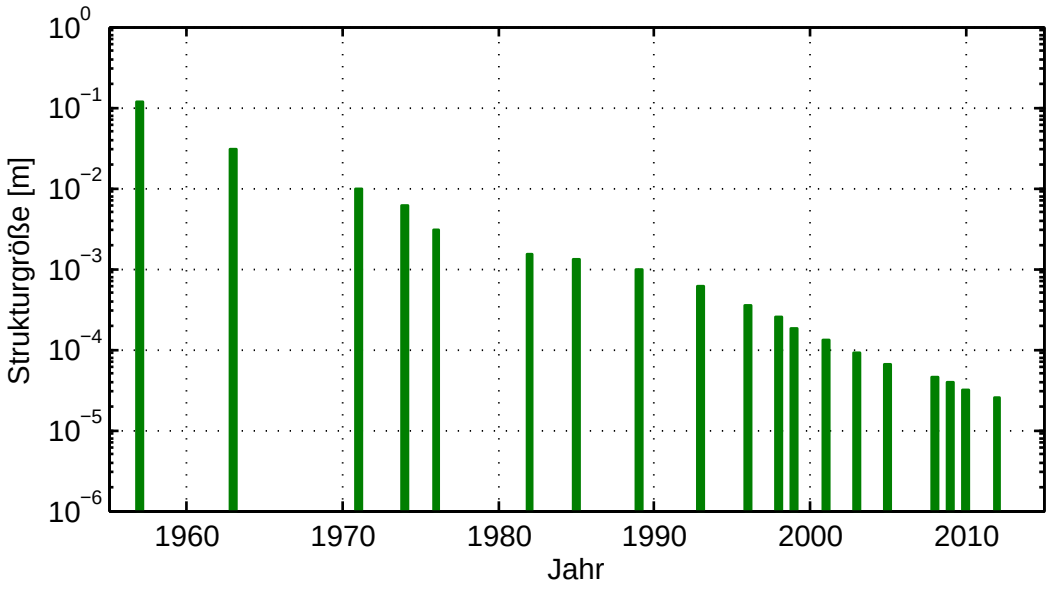

Abbildung 2.17: Halbleitertechnologie-Strukturgrößen (logarithmisch) im Laufe der Jahre

ten Schaltung heute mehrere zehn Millionen US-Dollar kostet [66]. Außerdem benötigt der Entwurfs- und Fertigungsprozess eines neuen ICs heute bis zu 18 Monate [67].

2.3.3 Anwendungsspezifische integrierte Schaltungen (ASICs)

Anwendungsspezifische integrierte Schaltungen, kurz ASICs, sind integrierte Schaltungen, die speziell für einen bestimmten Einsatzzweck entwickelt und hergestellt werden. Aufgrund ihrer hohen Leistungsfähigkeit werden sie seit Ende der 1990er Jahre und teilweise bis heute auch für die Paketverarbeitung in Hochleistungs-Routern verwendet. Nachteil dieser Bauelemente ist, dass sie nicht flexibel an neue Anforderungen anpassbar sind, da hierfür der Chip neu gefertigt werden müsste. Hierfür ist, wie oben erwähnt, ein bis zu 18 Monate dauernder Entwicklungs- und Herstellungsprozess notwendig mit Kosten bis zu mehreren Millionen US-Dollar [66, 67].

Die höchsten Verarbeitungsraten sind mit so genannten *full-custom* ASICs erzielbar. Das sind integrierte Schaltungen, bei denen jeder einzelne Transistor speziell nach Kundenwunsch platziert wird. Allerdings sind solche Schaltungen bei den heutigen Entwurfskosten von mehreren zehn Millionen US-Dollar nur bei sehr hohen Abnahmezahlen rentabel.

Daher werden häufig so genannte *semi-custom* ASICs verwendet. Diese basieren auf fertig entwickelten und getesteten Standardzellen, welche nur noch nach den spezifischen Kundenwünschen zusammengeschaltet werden müssen. Vorgefertigte Standardzellen sind beispielsweise Dekoderschaltungen, Zähler, Register aber auch größere Elemente wie Speicher, Prozessoren oder programmierbare Logik-Module. Die Kosten für solche Standardzellen-basierten ASICs liegen zwischen 250 000 und einer Million US-Dollar. [68]

Eine andere Alternative sind so genannte *Gate-Arrays*. Diese Bausteine basieren auf einer vorgefertigten Matrix von einfachen Logikgattern. Die Verdrahtung dieser Gatter erfolgt kunden-

spezifisch je nach gewünschter Funktion. Die Herstellungskosten solcher ASICs betragen lediglich 10 000 bis 100 000 US-Dollar. Im Vergleich zu full- bzw. semi-custom ASICs benötigen Gate-Array Schaltungen allerdings eine größere Chipfläche und unterstützen nicht so hohe Taktraten. [68]

Der Entwurfsprozess eines ASICs umfasst folgende Schritte: Die gewünschte digitale Schaltung wird auf Register-Transfer-Ebene (vgl. Abschnitt 3.1.1) modular mit Hilfe einer Hardware-Beschreibungssprache (*Hardware Description Language*, HDL) beschrieben und ihre Funktion mit taktgenauen Simulationswerkzeugen validiert. Anschließend übersetzt ein Synthesewerkzeug diese HDL-Beschreibung in auf der Zieltechnologie verfügbare Elemente. Bei Full-Custom ASICs sind dies einzelne Logikgatter oder Transistoren, bei Semi-Custom ASICs die vorgefertigten Standardzellen und bei Gate-Arrays die verfügbaren Logikgatter. Diese Elemente müssen danach auf dem Chip platziert und miteinander verbunden werden. Dies geschieht aufgrund der riesigen Menge an Elementen ebenfalls rechnerunterstützt. Die Fertigung der Chips erfolgt in Reinsträumen mit Hilfe von Verfahren der Halbleitertechnik, wie Epitaxie-, Foto-Litographie- und Dotierverfahren.

2.3.4 Programmierbare Logikbausteine

Programmierbare Logikbausteine sind Bauelemente, deren Funktionalität jederzeit beliebig konfigurierbar ist. Die am vielseitigsten einsetzbaren und am häufigsten verwendeten programmierbaren Logikbausteine sind FPGAs. Die Abkürzung FPGA steht hierbei für *Field Programmable Gate Array*. Dieser Name beschreibt bereits, dass die Grundstruktur dieser Bausteine eine Matrix aus „Gattern“ bildet und dass dessen Funktion am Einsatzort „programmiert“ werden kann. Genau genommen ist diese Abkürzung jedoch in zwei wichtigen Punkten irreführend: Zum einen enthält ein FPGA keine (Logik-) Gatter, sondern emuliert diese durch kleine Abbildungstabellen (*Lookup Tables*, LUTs). Zum anderen wird ein FPGA nicht programmiert, denn er führt kein Programm, also keine Sequenz von Befehlen, aus. Vielmehr müssen die einzelnen Elemente des FPGAs, insbesondere die LUTs, sowie deren Verschaltung miteinander vor dem Betrieb *konfiguriert* werden. Die Konfiguration erfolgt über Steuerbits, welche auf den FPGA geladen werden.

Architektur

Das Grundelement eines FPGAs ist der konfigurierbare Logikblock, der kleine Abbildungstabellen (LUTs) zur Realisierung logischer Funktionen (s. o.) sowie Flipflops enthält. Moderne FPGAs enthalten eine Matrix von bis zu mehreren 100 000 solcher Grundelemente, welche über eine Vielzahl konfigurierbarer Verbindungsleitungen miteinander verschaltet werden können.

Ein konfigurierbarer Logikblock enthält meist mehrere Abbildungstabellen (LUTs) für Logikfunktionen mit drei bis sechs Eingangsbits, zwei Volladdiererschaltungen, zwei oder mehr Flipflops sowie eine Vielzahl von Multiplexern. Abbildung 2.18 zeigt den prinzipiellen Aufbau eines solchen konfigurierbaren Logikblocks. Die Abbildungstabellen sind durch sehr kleine Speicher realisiert, welche für jede mögliche Eingangsbitkombination einen Ausgangswert enthalten. Zum Beispiel verfügt eine Abbildungstabelle für 4 Eingangsbits über $2^4 = 16$ Speich-

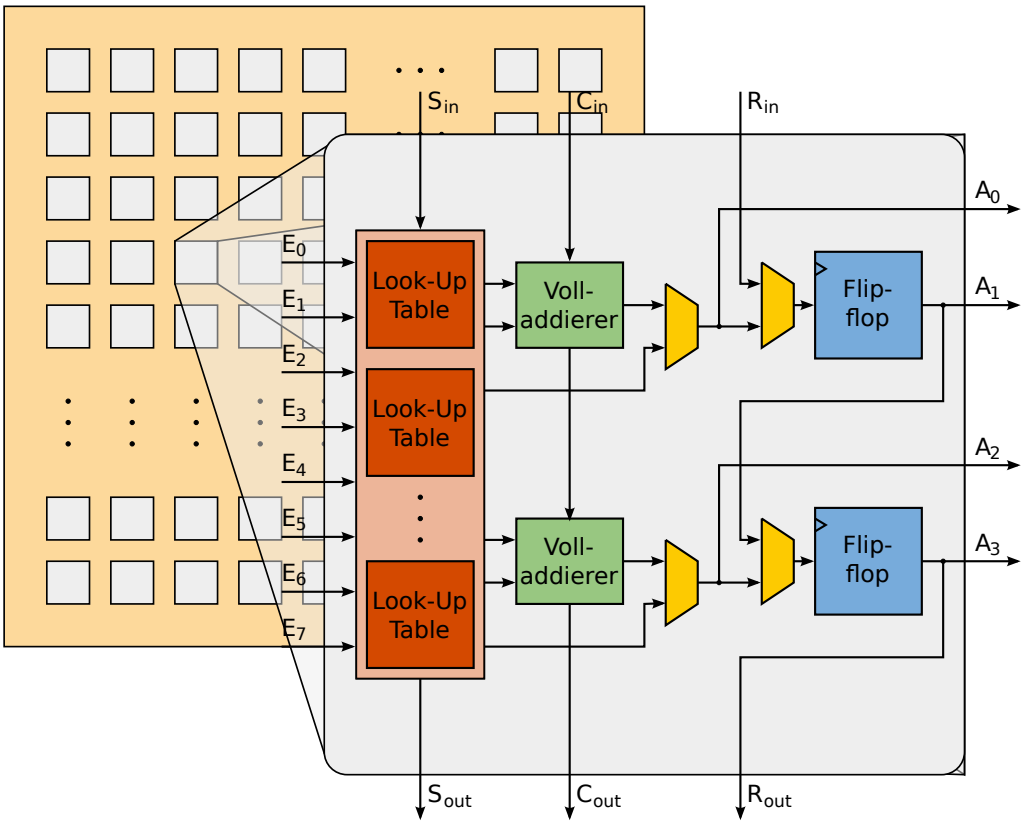

Abbildung 2.18: Prinzipieller Aufbau eines konfigurierbaren Logikblocks. Ein FPGA ist eine Matrix solcher Logikblöcke.

erbits. Die Eingangssignale eines konfigurierbaren Logikblocks können je nach benötigten Logikfunktionen auf verschiedene Arten mit den Abbildungstabellen verbunden werden, so dass auch Funktionen durch mehrere Tabellen gemeinsam realisiert werden können. Die Ausgänge der Abbildungstabellen können als Eingänge von Volladdierern verwendet werden. Diese sind über mehrere Logikblöcke hinweg über eine schnelle Übertragskette (*Carry Chain*) miteinander verbunden, um schnelle Additionen und Subtraktionen realisieren zu können. Schließlich können die Ergebnisse des Logikblocks in den Flipflops gespeichert werden. Über diverse Multiplexer lassen sich auch einzelne Elemente des Logikblocks umgehen.

Zusätzlich enthalten moderne FPGAs i. d. R. SRAM-Blöcke (*Static Random Access Memory*) verschiedener Größe zur Realisierung von Speichern innerhalb eines digitalen Systems. Des Weiteren sind festverdrahtete Multiplizierer integriert für die effiziente Durchführung von Signalverarbeitungsalgorithmen. Einige FPGAs enthalten sogar einzelne, festverdrahtete RISC-Prozessorkerne. Diese Elemente könnten auch mit Hilfe der vorhandenen Logikblöcke realisiert werden, jedoch beschleunigen die festverdrahteten Elemente diese Funktionen und benötigen weniger Chipfläche. Zusätzlich sind PLLs (*Phase-Locked Loops*) vorhanden, um beliebige Taktsignale zu generieren.

Einsatzgebiete

FPGAs gibt es seit etwa 25 Jahren. In den ersten Jahren enthielten sie nur wenige hundert bis tausend Abbildungstabellen und Flipflops und man realisierte damit hauptsächlich Logikmodule zur Schnittstellenanpassung (so genannte *Glue Logic*) zwischen verschiedenen Chips auf einer Leiterplatte. Mit zunehmender Anzahl an Logikblöcken sowie Speicherblöcken und Multiplizierern wurden FPGAs im Laufe der Jahre mehr und mehr für die Realisierung größerer digitaler Systeme eingesetzt. Die Verwendung von FPGAs ist hierbei insbesondere bei der Herstellung von Produkten mit geringer Stückzahl interessant sowie für die schnelle, prototypische Implementierung neuartiger Systeme, bevor diese als festverdrahtete integrierte Schaltung realisiert werden (so genanntes *Rapid Prototyping*).

Heute werden FPGAs zur digitalen Signalverarbeitung, zur Paketverarbeitung auf Schicht 2 und teilweise Schicht 3 sowie zur Beschleunigung von Prozessorkernen, insbesondere bei gut parallelisierbaren Funktionen wie bei der Ver- und Entschlüsselung von Datenströmen und bei der Multimediadatenverarbeitung verwendet. In den letzten Jahren wurde außerdem damit begonnen, FPGAs im Betrieb umzukonfigurieren, um die Funktionalität dynamisch an äußere Randbedingungen anpassen zu können (siehe auch Unterkapitel 2.5). Ein weiteres aktuelles Anwendungsgebiet ist die Realisierung ganzer Systeme bestehend aus Prozessorkernen, Speicher-Controller und anderen Schnittstellenmodulen auf einem FPGA. Bei einem solchen „System auf einem programmierbaren Chip“ (*System On a Programmable Chip*, SOPC) hat man die Freiheit, während des Entwurfs zu bestimmen, welche Funktionen in Software von den integrierten Prozessorkernen ausgeführt und welche Funktionen in Hardware realisiert werden sollen. Dieser kombinierte Entwurf wird auch als *Hardware-/Software-Codesign* bezeichnet.

Vergleich mit anderen Bauelementen

Kuon [69] vergleicht Chipflächenbedarf, maximale Taktfrequenz und dynamischen Energiebedarf von FPGAs und *semi-custom* ASICs. Beide Bauelemente wurden mit einer vergleichbaren Prozesstechnologie gefertigt. Kuon ermittelt, dass FPGAs im Schnitt eine um den Faktor 35 größere Chipfläche für dieselben Funktionen benötigen. Dieser Faktor kann jedoch durch den Einsatz von Spezialblöcken wie Speicher und Multiplizierern auf durchschnittlich 18 gesenkt werden. FPGAs sind um den Faktor 3 bis 4 langsamer und haben eine etwa 14-mal so hohe dynamische Leistungsaufnahme. Letztere kann wiederum durch den Einsatz der Spezialblöcke verringert werden.

Klarer Nachteil von ASICs sind deren sehr hohe Kosten für die Fertigung. Außerdem sind FPGAs heute so gefragt, dass neue FPGAs stets mit den aktuellsten IC-Prozesstechnologien gefertigt werden [70, 71]. Dieser Technologievorsprung kann die oben genannten Vorteile von ASICs bezüglich Fläche, Geschwindigkeit und dynamischer Leistungsaufnahme teilweise abschwächen.

Im Vergleich zu Standardprozessoren, aber auch zu Netzprozessoren, weisen FPGAs aufgrund ihrer Rekonfigurierbarkeit ebenfalls eine etwa um den Faktor 10 geringere Taktfrequenz auf. Dennoch lassen sich viele Funktionen auf FPGAs schneller ausführen als auf prozessorbasierten Systemen. Guo [72] analysiert diesen Geschwindigkeitsvorteil und begründet ihn mit der

hohen möglichen Parallelisierung von Subfunktionen sowie mit einer höheren Effizienz. Letztere ergibt sich bei FPGAs aufgrund mehrerer Faktoren: Zum einen werden keine Takte für die Steuerung des Daten- und Programmflusses benötigt, zum anderen können große Datenmengen effizient verteilt abgelegt werden. Außerdem können einige Operationen, wie Schiebeoperationen sowie Operationen, die auf Prozessoren durch mehrere Befehle emuliert werden müssen, auf FPGAs sehr einfach und effizient realisiert werden.

Entwurf und Implementierung digitaler Systeme

Der Entwurf und die Implementierung digitaler Systeme auf FPGAs, aber auch auf ASICs, gliedert sich in mehrere Schritte. Diese wurden im letzten Abschnitt bereits teilweise kurz erwähnt: Die Beschreibung des Systems in einer Hardware-Beschreibungssprache, die Simulation des Systems anhand dieser Beschreibung, die Umsetzung der Beschreibung in eine Netzliste, die Platzierung der Elemente dieser Netzliste auf dem Chip und das letztendliche Konfigurieren des Chips.

Beschreibung – Zur Beschreibung eines digitalen Systems verwendet man eine Hardware-Beschreibungssprache. Die zwei bekanntesten und am weitesten verbreiteten Sprachen sind *Verilog* und VHDL (*VHSIC Hardware Description Language*, wobei VHSIC die Abkürzung für *Very High Speed Integrated Circuit* ist). Erstere Sprache wird hauptsächlich im US-amerikanischen Raum verwendet, letztere hauptsächlich in Europa. Mit diesen Sprachen kann man sowohl das Verhalten einzelner Blöcke beschreiben, als auch die Struktur eines Systems durch die hierarchische Instanziierung und Verschaltung solcher Blöcke. Die Beschreibung erfolgt im Wesentlichen auf Register-Transfer-Ebene, es können jedoch auch einerseits einzelne Gatter und andererseits abstrakte Datentypen für die Beschreibung verwendet werden (vgl. auch Abschnitt 3.1.1).

Simulation – Um die korrekte Funktion von Teilblöcken sowie des Gesamtsystems zu überprüfen, simuliert man diese mit Hilfe von Testumgebungen (*test beds*), welche das zu testende System mit Eingangssignalen stimulieren und die Ausgangssignale überprüfen. Die Testumgebungen sind häufig ebenfalls eine HDL-Beschreibung. Die Simulation erfolgt taktgenau und es kann jedes einzelne Signal innerhalb des Systems aufgezeichnet und überprüft werden.

Synthese – Die durch Simulation validierte Systembeschreibung muss anschließend in eine Netzliste mit Elementen umgesetzt werden, mit denen das System später auf dem Chip realisiert werden kann. Bei FPGAs sind dies die Abbildungstabellen, Flipflops, Speicherblöcke und sonstigen Spezialblöcke, bei ASICs Gatter und Standardzellen. Diese Umsetzung, auch Synthese genannt, führt eine spezialisierte Software durch. Nach diesem Schritt können bereits erste Aussagen zum Ressourcenbedarf und zur maximalen Taktfrequenz gemacht werden.

Platzierung und Verbindung – In diesem vorletzten Schritt versucht nun eine weitere Software, die benötigten Elemente aus der übergebenen Netzliste möglichst effizient auf dem Chip zu platzieren und miteinander zu verschalten. Der Fachbegriff für diesen Schritt ist *Place & Route*. Als Ausgabe erhält man eine Konfigurationsdatei mit den Steuerinformation, wie alle Logikblöcke, Verbindungsstrukturen und Spezialblöcke konfiguriert werden

müssen. Des Weiteren erstellt die Place & Route-Software diverse Ergebnislisten, die den Ressourcenverbrauch des Systems und das zeitliche Verhalten der Signale – insbesondere an kritischen Stellen – dokumentieren.

Konfiguration – Schließlich kann man die erstellte Datei mit den Konfigurationsdaten sowie mit Initialisierungswerten auf den FPGA laden. Anschließend realisiert der FPGA das beschriebene digitale System und kann mit externer Peripherie getestet und eingesetzt werden.

2.3.5 Netzprozessoren

Netzprozessoren versuchen, die goldene Mitte zwischen der Leistungsfähigkeit von ASICs und der Flexibilität von Standardprozessoren zu sein. Wie in Abschnitt 2.3.3 dargestellt, verfügen ASICs über die höchste erreichbare Verarbeitungsleistung. Allerdings sind sie nicht flexibel an neue Anforderungen anpassbar. Andererseits ermöglichen Standardprozessoren (vgl. Abschnitt 2.3.6) eine programmgesteuerte Paketverarbeitung, die einfach an neue Anforderungen angepasst werden kann. Sie erreichen jedoch nicht den benötigten Durchsatz. Bei Netzprozessoren verfolgt man daher den Ansatz, einerseits durch Spezialisierung der Hardware an die benötigte Funktionalität eine hohe Verarbeitungsleistung zu erzielen, andererseits durch Programmierbarkeit die gewünschte Flexibilität bereitzustellen.

Geschichte

Netzprozessoren blicken auf eine noch sehr kurze Vergangenheit von nur etwa zehn Jahren zurück. In Folge der *dot-com*-Euphorie in den Jahren 1998 bis 2000 erwarteten viele ein explosionsartiges Wachstum des Internetverkehrs. So prognostizierte auch das Technologie-Marktforschungsunternehmen *In-Stat*, dass die Einnahmen durch Netzprozessoren in den folgenden Jahren jährlich um durchschnittlich 63% auf 2,9 Milliarden US-Dollar im Jahre 2004 ansteigen würden [73]. Daher entwickelten Anfang des Jahrtausends mehrere Dutzend Firmen, darunter sehr viele Neugründungen, eigene Netzprozessoren. Entsprechend viele unterschiedliche Architekturen wurden entworfen. Shah [74] gibt einen Überblick über Netzprozessoren 19 verschiedener Hersteller, die im Jahre 2001 auf dem Markt bzw. angekündigt waren.

Inzwischen sind die meisten dieser Unternehmen bzw. Unternehmenszweige wieder vom Markt verschwunden oder wurden von anderen Firmen aufgekauft (vgl. [75, 76]). Heute bieten noch sieben Firmen Netzprozessoren an: *Bay Microsystems*, *Broadcom* (ehemals *Sandburst*), *EZchip*, *LSI* (ehemals *Agere*), *Netronome* (ehemals *Intel*), *Wintegra* und *Xelerated* [75]. Zusätzlich verwenden die Router-Hersteller *Cisco*, *Juniper* und *Alcatel-Lucent* auch intern entwickelte Netzprozessoren. Dennoch ist der Markt für Netzprozessoren kontinuierlich angewachsen und sie erzeugten 2008 Gesamteinnahmen von 305 Millionen US-Dollar [75]. Die Gründe des nur langsamen Erfolgs von Netzprozessoren werden am Ende dieses Abschnitts diskutiert.

Architektur

Abbildung 2.19 zeigt eine generische Netzprozessorarchitektur. Netzprozessoren bestehen im Wesentlichen aus folgenden Elementen:

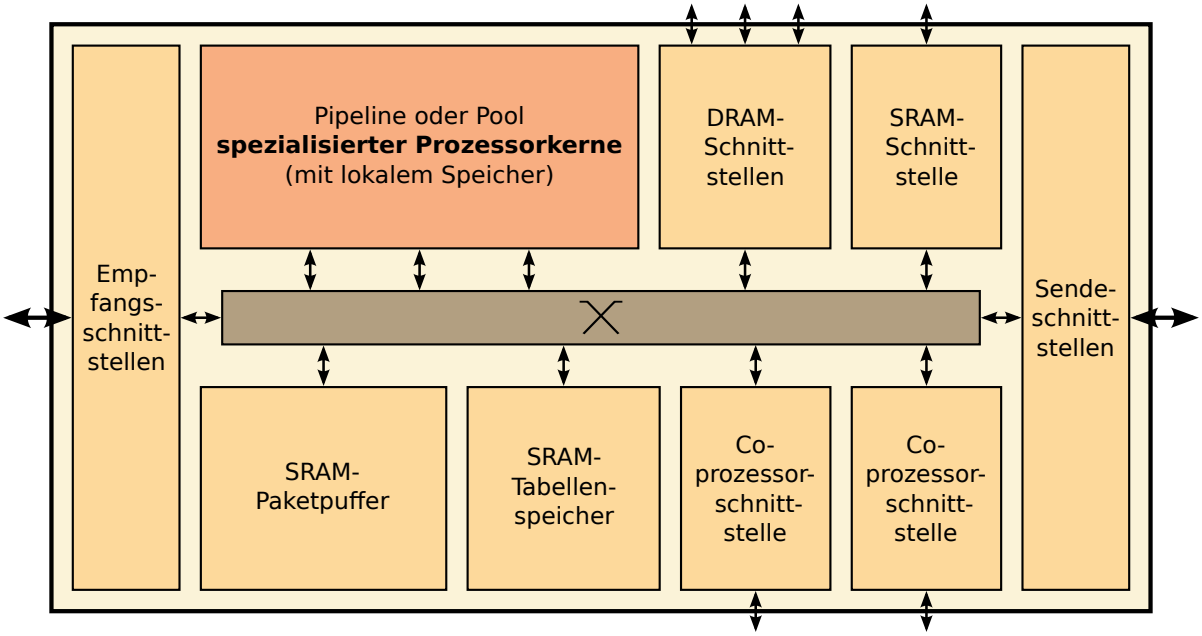

Abbildung 2.19: Generische Architektur eines Netzprozessors

- mehrere gleiche oder unterschiedliche Prozessorkerne in Pool- oder Pipeline-Konfiguration
- verschiedene lokale (*on-chip*) Coprozessoren für Spezialaufgaben sowie Schnittstellen zu externen (*off-chip*) Coprozessoren
- verschiedene lokale (*on-chip*) Speichermodule zur Ablage von Paketdaten, Tabellen und Metadaten sowie Schnittstellen zu externen (*off-chip*) Speichermodulen
- verschiedene Kommunikationsschnittstellen zu einem Schaltnetzwerk (*Switch Fabric*) und zu den Leitungsschnittstellen-Chips auf Schicht 1 oder 2 (PHY-Chips).

Außerdem ist bei manchen Modellen noch ein Standardprozessor integriert. Dieser ist für Aufgaben der Steuerungs- und Management-Ebene zuständig.

Die Prozessorkerne sind je nach Hersteller untereinander unterschiedlich organisiert. Es gibt zwei grundlegende Organisationsstrukturen, die teilweise auch kombiniert werden:

Pool von Prozessoren – Hier werden die zu verarbeitenden Pakete reihum den Prozessorkernen zugeordnet und jeder Prozessor verarbeitet sein Paket vollständig. Alle Prozessorkerne sind identisch.

Pipeline von Prozessoren – Jedes Paket wird sequentiell von jedem Prozessor verarbeitet. So übernimmt jeder Prozessor nur eine bestimmte Aufgabe. Hierbei können sowohl ausschließlich gleichartige Prozessorkerne verwendet werden, als auch auf ihre jeweilige Funktion spezialisierte Prozessoren, wie z. B. spezielle Klassifizierungsprozessoren.

Auf die Leistungsfähigkeit dieser beiden Organisationsstrukturen wird in Unterkapitel 3.3 näher eingegangen und ihre Vor- und Nachteile diskutiert. Es werden auch beide Strukturen kombiniert, z. B. als Pipeline mit jeweils einem Pool von Prozessoren pro Stufe.

Die **Prozessorkerne** selbst sind meist einfache Prozessoren – häufig mit RISC-Architektur – und verfügen über einen auf Paketverarbeitung spezialisierten Befehlssatz. Damit die Prozessoren Protokollfelder in Paket-Headern effizient verarbeiten können, enthält solch ein Befehlssatz beispielsweise Befehle, um auf einzelne Bits oder Bitgruppen eines Datenwortes zugreifen zu können. Außerdem sind Befehle enthalten, um die verschiedenen Speicher und Coprozessoren zu adressieren. Teilweise werden auch so genannte *superskalare* Prozessorkerne und VLIW-(*Very Long Instruction Word*)-Prozessorkerne eingesetzt, die mehrere parallele Verarbeitungseinheiten aufweisen und so mehrere Operationen pro Takt ausführen können.

Da der Zugriff auf Speichermodule und Coprozessoren i. d. R. Wartetakte zur Folge hat, unterstützen die Prozessorkerne meist Hardware-seitig optimiertes *Multi-Threading*. Damit kann der Prozessor bei Wartetakten ohne Taktverlust in einen anderen *Thread* (Faden) mit separatem Registersatz umschalten, um währenddessen ein anderes Paket zu verarbeiten. Somit ist es im Idealfall möglich, den Prozessor trotz dieser Latenzen zu 100% auszulasten.

Zusätzlich zu den Prozessorkernen sind häufig **Coprozessoren** auf dem Netzprozessor-Chip integriert. Coprozessoren werden für besonders aufwändige oder rechenintensive Aufgaben eingesetzt, die sehr oft benötigt werden. So gibt es beispielsweise häufig Coprozessoren zur Berechnung der Prüfzeichen des Ethernet-Rahmens, für die IP-Adresssuche oder die Klassifizierung von Paketen. Spezielle Netzprozessoren enthalten außerdem Coprozessoren für die Ver- oder Entschlüsselung von Daten oder De- und Enkoder zur Verarbeitung von Multimedia-Daten.

Netzprozessoren verfügen i. d. R. über eine große Anzahl verschiedener **Speicher:** Schnelle, kleine, Prozessor-lokale Speicher können zum kurzzeitigen Puffern von Paketdaten oder zur Ablage häufig benötigter lokaler Variablen verwendet werden. Größere, von mehreren oder allen Prozessoren gemeinsam genutzte Speicher auf dem Chip können zur Ablage von Weiterleitungstabellen oder Klassifizierungsregeln von wenigen MByte Größe genutzt werden, aber auch zur Kommunikation zwischen verschiedenen Prozessorkernen. Diese On-Chip-Speicher werden als SRAM realisiert. Größere Tabellen können bei einigen Netzprozessoren auch in externem SRAM gespeichert werden. Als Paketpuffer sowie zur Speicherung großer Tabellen oder sonstiger Datenmengen wird in der Regel externer DRAM (*Dynamic Random Access Memory*, dynamischer Speicher mit wahlfreiem Zugriff) verwendet.

Um Pakete effizient empfangen und weiterleiten zu können, verfügen Netzprozessoren i. d. R. über **Hochgeschwindigkeitskommunikationsschnittstellen** wie XAUI (*10 Gigabit Attachment Unit Interface*) [26] oder SPI-4.2 (*System Packet Interface*) [77] für Schnittstellen mit 10 Gbit/s oder Interlaken [78] für Schnittstellen mit 100 Gbit/s Datenrate.

Programmierung

Die Programmierung von Netzprozessoren ist von Hersteller zu Hersteller verschieden. Sie kann anhand mehrerer Charakteristika unterschieden werden:

Abstraktion: Sehr Hardware-nahe Sprachen bzw. eine Assembler-Sprache bieten den Vorteil, dass alle vorhandenen Hardware-Ressourcen direkt und somit sehr effizient verwendet werden

können. Dies führt zu hohen Verarbeitungsraten. Abstraktere, höhere Programmiersprachen erleichtern und beschleunigen dagegen die Implementierung der gewünschten Funktionen.

Sprache: Manche Netzprozessor-Hersteller entwickeln eigene Spezialsprachen zur Programmierung der Prozessoren. Beispielsweise hat *Agere* eine eigene Sprache zur Programmierung ihres Klassifizierungsprozessorkerns entwickelt [79]. Solche Spezialsprachen sind optimal auf die gegebene Hardware abgestimmt und ermöglichen so einen kurzen Programmcode. Andererseits hat die Verwendung einer bekannten Sprache, wie z. B. der Sprache C, den Vorteil, dass Entwickler sie nicht komplett neu erlernen müssen und unter Umständen vorhandene Bibliotheken weiterverwenden können.

Programmiermodell: Viele Netzprozessoren werden mit Hilfe eines so genannten *Single-Processor-Single-Thread*-Modells programmiert (auch *Run-to-Completion*-Modell genannt). Bei diesem Modell programmiert der Entwickler den Netzprozessor so, als ob er nur einen Prozessorkern mit einem einzigen Thread programmiert. Die Compiler-Software verteilt den Code dann automatisch auf die verschiedenen Prozessorkerne und Threads. Dies hat den Vorteil einer einfachen Programmierung. Allerdings können u. U. nicht alle Ressourcen des Prozessors optimal genutzt werden. Das gegenteilige Programmiermodell ist das *Multi-Processor-Multi-Thread*-Modell. Hier muss jeder Thread eines jeden Prozessors vom Entwickler einzeln programmiert werden.

Diskussion

Netzprozessoren werden heutzutage in die meisten neuen Hochgeschwindigkeits-Router eingebaut. Allerdings hatten sie nicht so einen durchschlagenden Erfolg wie er zur Jahrtausendwende prognostiziert wurde. Zu einem großen Teil ist dies natürlich durch die völlig übertriebenen Erwartungen während der so genannten Internet-Blase zu erklären. Doch auch nach dem Ende dieser Euphorie, in den Jahren 2002 bis 2004, wurde Netzprozessoren ein weitaus größeres Wachstum vorhergesagt [80], als tatsächlich eingetreten ist. Gründe für den nur langsamen Erfolg von Netzprozessoren liegen in der Schwierigkeit, die gewünschte Flexibilität hinsichtlich ihrer beiden Komponenten, der Einfachheit und der Anpassbarkeit (vgl. Abschnitt 2.2.3), mit diesen Bauelementen zu erreichen.

Die Programmierung von Netzprozessoren ist nicht so einfach wie die Programmierung eines Standardprozessors. Für Netzprozessoren gibt es bisher kein einheitliches Programmiermodell und auch keine universell verwendbare Programmiersprache. Somit müssen Entwickler, wenn sie einen Netzprozessor programmieren wollen, zunächst die entsprechende Sprache erlernen. Selbst wenn sich die Sprache, wie oben erwähnt, an einer bekannten Sprache wie C orientiert, müssen neue Sprachkonstrukte erlernt werden, mit denen die auf dem Netzprozessor integrierten Elemente, wie die verschiedene Coprozessoren oder Speichertypen, verwendet werden können.

Des Weiteren muss der Programmierer ein im Vergleich zu Standardprozessoren wesentlich komplexeres Modell der Hardware-Architektur kennen, damit der Netzprozessor einen hohen Paketdurchsatz erreichen kann. Hu [81] zeigt in seiner Veröffentlichung zu einer Implementierung auf einem Intel Netzprozessor ausführlich die vielen Möglichkeiten, die gegebene Hardware-Architektur möglichst effizient einzusetzen. Auch Kögel beschreibt in [12, 82] Einschrän-

kungen durch die Hardware-Architektur eines Intel Netzprozessors und entsprechende Lösungen bei der Implementierung einer Aggregationseinheit in Assembler. In manchen Fällen ist eine effiziente Programmierung einer Funktion nur in Assembler möglich, da die Compiler nicht leistungsfähig genug sind, die verfügbare schnelle Hardware wirklich effizient nutzen zu können. Manche Hersteller empfehlen sogar, Funktionen, die kritisch für die Leistung des Netzprozessors sind, in Assembler zu programmieren. Letztendlich führt die schwierige Programmierung von Netzprozessoren zu langen Entwicklungszeiten und schwierig erweiterbarem Code.

Eine anderes Problem von Netzprozessoren liegt darin, dass ihre Programmierbarkeit zwar eine hohe Anpassbarkeit bietet, komplexe Funktionen jedoch häufig nicht effizient genug implementiert werden können, um den benötigten hohen Paketdurchsatz erreichen zu können. Beispiele für solche komplexen Funktionen sind die IP-Adresssuche, die Paketklassifikation sowie Deep Packet Inspection. Um solche Funktionen dennoch realisieren zu können, werden daher häufig Spezialbefehle sowie Coprozessoren verwendet, welche die gewünschte Funktion festverdrahtet realisieren. Der Nachteil davon ist jedoch, dass diese nur in sehr geringem Umfang an neue Anforderungen angepasst werden können. Und früher oder später treten immer neuartige Funktionen auf, die nicht durch Spezialbefehle oder Coprozessoren realisiert werden können. Hier tritt somit eine Zielkonflikt auf, einerseits möglichst allgemein zu sein, um alle zukünftigen Funktionen realisieren zu können, und andererseits möglichst spezialisierte Funktionen bereitzustellen, die eine schnelle Paketverarbeitung erlauben. Was diesen Aspekt angeht, sind FPGA-basierte Systeme zu bevorzugen oder auch Prozessoren mit rekonfigurierbarem Befehlssatz.

Ähnliche Probleme kann es durch die feste, unveränderliche Architektur eines Netzprozessors geben. Unterkapitel 3.3 wird ausführlich die Vor- und Nachteile von Pool- und Pipeline-Verarbeitungsstrukturen diskutieren. Auch die Anbindung an andere Ressourcen, wie Speicher und Coprozessoren, kann auf unterschiedliche Weisen erfolgen, z. B. durch eine lokale Punkt-zu-Punkt-Verbindung, über einen oder mehrere gemeinsame Busse oder über eine Schaltmatrix. Je nach den geforderten Randbedingungen ist in manchen Fällen die eine Architektur und in anderen Fällen eine andere optimal. Da diese jedoch i. d. R. unveränderbar ist, hat die Flexibilität von Netzprozessoren auch hier ihre Grenzen und Systeme, bei denen auch die Hardware-Architektur an die benötigte Funktionalität anpassbar ist, sind hier von Vorteil.

2.3.6 Standardprozessoren

Prozessoren sind Bauelemente, mit deren Hilfe einfache, grundlegende Operationen programmgesteuert ausgeführt werden können. Standardprozessoren (*General Purpose Processors*, GPPs) werden millionenfach als zentrale Ausführungseinheit (*Central Processing Unit*, CPU) in Servern, PCs und Notebooks eingesetzt. Außer Standardprozessoren gibt es eine Vielzahl von Spezialprozessoren, deren Architektur für einen speziellen Einsatzzweck optimiert ist. Beispiele hierfür sind Grafikprozessoren, digitale Signalprozessoren, Netzprozessoren (siehe oben) sowie eingebettete Prozessoren in Mobiltelefonen, Druckern, Waschmaschinen oder Autos.

Moderne Standardprozessoren basieren auf einer *RISC*-Architektur (*Reduced Instruction Set Computer*, Rechner mit reduziertem Befehlssatz). Entsprechend *Amdahls Gesetz* (*Amdahl's Law*) [83] weisen RISC-Prozessoren einen auf häufig verwendete Befehle reduzierten Befehlssatz auf, der sehr schnell ausgeführt werden kann. Die Ausführung der Befehle erfolgt über-

lappend in einer Pipeline-Struktur. Hierdurch kann ein Befehlsdurchsatz von beinahe einem Befehl pro Taktzyklus erreicht werden. Um noch höhere Befehlsdurchsätze erzielen zu können, verwenden heutige Prozessoren weitere Optimierungen, wie Sprungvorhersage, Befehlsumsortierung und Multi-Threading. *Superskalare* Prozessoren und VLIW-(*Very Long Instruction Word*)-Prozessoren verfügen über mehrere parallele Verarbeitungseinheiten und können so mehrere Operationen pro Takt ausführen.

Zur Erhöhung der Leistungsfähigkeit der Prozessoren wurden die Taktraten dieser Bauelemente im Laufe der Jahre immer weiter erhöht. Dies führte jedoch auch zu einem immer höheren Energieverbrauch und somit einer immer höheren Abwärme der Prozessoren. Daher werden seit etwa 2005/2006 keine höher getakteten Standardprozessoren mehr entwickelt. Stattdessen ging der Trend zu *Mehrkernprozessoren* (*Multi-core Processors*). Mehrkernprozessoren beinhalten zwei, vier oder mehr Prozessoren, jetzt Prozessorkerne genannt, auf einem Chip. Dadurch können unabhängige Software-Prozesse, abgesehen von Speicher-Zugriffen, vollständig nebenläufig ausgeführt werden.

Aufgrund der sehr hohen Nachfrage nach Standardprozessoren durch den Server-, PC- und Notebook-Markt sind dies hoch-optimierte integrierte Schaltungen, die stets mit den aktuellsten Halbleitertechnologien und den jeweils kleinsten Transistorgrößen gefertigt werden. Aktuelle Standardprozessoren arbeiten mit Taktfrequenzen von 1 bis 4 GHz und verfügen über bis zu acht Kerne, die jeweils bis zu vier Befehle pro Takt ausführen können. Somit können unter optimalen Bedingungen mehrere 10 Milliarden Befehle pro Sekunde (*Giga-Instructions per Second*, GIPS) erreicht werden. Die Leistungsaufnahme liegt je nach Modell zwischen 5 und 150 Watt.

Heute wie früher werden Standardprozessoren für die Paketverarbeitung im Datenpfad von Vermittlungsknoten bei niederen bis mittleren Paketraten eingesetzt (z. B. [84–86]). Aufgrund des oben beschriebenen hohen Befehlsdurchsatzes bei noch moderater Leistungsaufnahme werden immer wieder Überlegungen angestellt, ob solche hochoptimierten Standardprozessoren auch für die Paketverarbeitung in Hochleistungs-Routern verwendet werden können [87]. Eine zu beobachtende zunehmende Integration von schnellen Kommunikations- und Speicherschnittstellen direkt auf dem Prozessor-Chip begünstigt diese Entwicklung. Für die Durchführung von Aufgaben der Steuerungs- und Managementebene sind Standardprozessoren üblich.

2.4 Aktuelle kommerzielle Systeme

Kommerzielle Paketverarbeitungssysteme, welche sowohl einen hohen Durchsatz als auch hohe Flexibilität aufweisen, sind entweder Netzprozessoren oder basieren auf programmierbarer Logik. Die folgenden beiden Abschnitte geben einen Überblick über die verschiedenen kommerziell erhältlichen Systeme und diskutieren deren Eigenschaften.

2.4.1 Netzprozessoren

Die prinzipielle Architektur von Netzprozessoren wurde in Abschnitt 2.3.5 beschrieben. In diesem Abschnitt sollen nun konkrete Netzprozessoren verschiedener Hersteller vorgestellt und

diskutiert werden. Die Netzprozessoren von Intel/Netronome werden in Industrie und Forschung weltweit am häufigsten eingesetzt. Die schnellsten, kommerziell erhältlichen Netzprozessoren sind von den Firmen Xelerated, EZchip und Bay Microsystems, außerdem verwenden die großen Router-Hersteller wie Cisco, Juniper und Alcatel-Lucent häufig selbst entwickelte Netzprozessorsysteme. Im Folgenden werden die wichtigsten Merkmale dieser Netzprozessorarchitekturen dargestellt und am Ende gegenüberstellend diskutiert. Netzprozessorarchitekturen für Vermittlungsknoten in Zugangsnetzen, wie die von LSI/Agere, Wintegra und Broadcom werden hier nicht diskutiert.

Intel/Netronome – Konfigurierbarer Pool von RISC-Prozessorkernen

Die Netzprozessoren von Intel sind die an Universitäten am häufigsten verwendeten Netzprozessoren. Der Grund hierfür ist Intels für diese Technologie groß angelegtes Universitätsförderprogramm sowie die Offenlegung vieler Details ihrer Hardware- und Software-Architektur. Intels Hochgeschwindingkeitsnetzprozessoren sind der IXP2800 und IXP2850. Im Jahre 2007 verkaufte Intel die Lizenzen für diese Netzprozessorlinie an Netronome, welche diese weiter unterstützt und weiterentwickelt. 2010 beginnt Netronome mit dem Verkauf der NFP3240-Chips [88], der nächsten Generation dieser Netzprozessorfamilie. Einsatzgebiet der IXP28xx sowie der NFP3240 Netzprozessoren sind die „intelligente“ Paketverarbeitung bei einem Durchsatz von 10 bzw. 20 Gbit/s in Routern und Switches am Rand von Kernnetzen. Dies beinhaltet sowohl die schnelle Weiterleitung von Paketen auf Schicht 2 und 3 als auch die Verarbeitung oder Analyse der Pakete auf den Schichten 4 bis 7.

Die Paketverarbeitungsarchitektur [89] basiert auf einem Pool von RISC-Prozessorkernen, den so genannten *Microengines*. Diese sind für die Verarbeitung von Paketen spezialisiert. Die IXP28xx-Netzprozessoren besitzen 16 solcher Microengines, die neuen NFP3240 Netzprozessoren 40. Interne Busse ermöglichen eine schnelle Kommunikation mit den Paketschnittstellen, Schnittstelleneinheiten zu externem SRAM- und DRAM und zu einem externen Coprozessor, einem internen Speichermodul, einer Hash-Wert-Berechnungseinheit, einem Steuer- und Signalisierungsregistersatz sowie (in speziellen Versionen) einer Kryptografieeinheit. Für Steuerungs- und Managementaufgaben steht ein ARM-basierter (*Acorn RISC Machine*) XScale-Prozessor zur Verfügung. Außerdem ermöglichen die neuen Netronome-Chips eine hochbitratige Kommunikation mit Intel Standardprozessoren mit geringer Latenz zur kombinierten Paketverarbeitung in beiden Prozessortypen. Abbildung 2.20 gibt einen Überblick über die verwendete Architektur.

Die RISC-Prozessorkerne unterstützen einen für die Paketverarbeitung spezialisierten Befehlssatz mit effizienten Verarbeitungs-, Verschiebe- und Steuerbefehlen auf Bit-, Byte- und Wortebene. Die Prozessoren verfügen jeweils über einen separaten, lokalen Befehlsspeicher, einen separaten, lokalen Datenspeicher und einen kleinen CAM (*Content Addressable Memory*). Die Prozessor-Hardware unterstützt außerdem jeweils bis zu acht Threads. Durch das Wechseln des aktiven Threads können Zugriffslatenzen auf externe Module, wie Speicher oder Coprozessoren, verborgen werden. Ein Thread-Wechsel muss dabei explizit durch das Programm angewiesen werden.

Zusätzlich zum lokalen Datenspeicher, kann jeder der RISC-Prozessoren auf einen lokalen Registersatz, den gemeinsam genutzten On-Chip-Speicher sowie das externe SRAM und DRAM

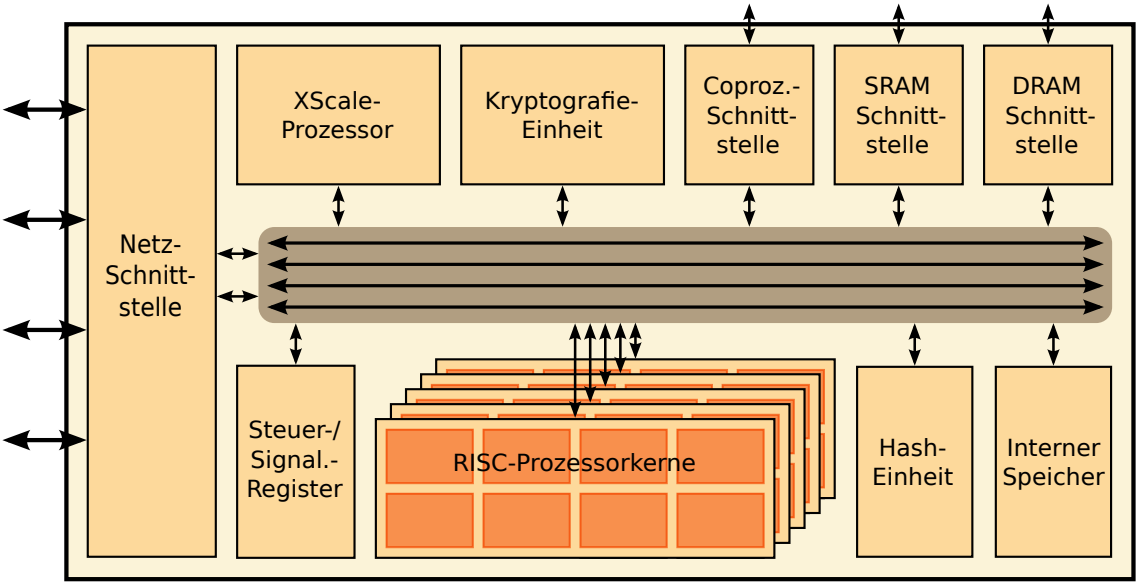

Abbildung 2.20: Architektur der Intel/Netronome Netzprozessoren

zugreifen. Die Kommunikation zwischen den RISC-Kernen kann sowohl über gemeinsam genutzte Register als auch über die gemeinsam genutzten Speicher erfolgen – mit entsprechenden Unterschieden in Volumen, Durchsatz und Latenz. Die Architektur erleichtert hierbei die Kommunikation durch Hardware-Unterstützung für Ringspeicherstrukturen, synchronisierte, atomare Speicheroperationen sowie Synchronisationssignale.

Die flexible Kommunikationsinfrastruktur zwischen den RISC-Prozessoren ermöglicht eine beliebige funktionale Aufteilung der Verarbeitung auf die Prozessorkerne. Die Prozessoren können die Verarbeitung als funktionale Pipeline hintereinander geschaltet ausführen, als Pool jeweils alle die komplette Verarbeitung ausführen oder in einer Mischform dieser beiden Ansätze die Paketverarbeitung ausführen.

Die Programmierung der RISC-Spezialprozessoren kann in einer Assembler-ähnlichen, sehr Hardware-nahen Programmiersprache oder bei Durchsatz-unkritischen Funktionen auch in einer angepassten C-Sprache erfolgen. Dabei muss je nach Verarbeitungsorganisation jeder Prozessorkern einzeln programmiert werden.

Xelerated – Pipeline gleichartiger VLIW-Prozessorkerne

Die X10- und X11-Netzprozessoren von Xelerated zählen seit ihrer Einführung in den Jahren 2003 und 2005 zu den Durchsatz-stärksten kommerziellen Netzprozessoren. Die seit 2010 verfügbaren HX-Netzprozessoren [90] können Carrier-Grade Ethernet-Verkehr mit bis zu 150 MP/s und 100 Gbit/s verarbeiten. Einsatzgebiet der Hochleistungsnetzprozessoren von Xelerated sind die Paketverarbeitung auf Schicht 2 bis 4 in Vermittlungsknoten am Rand und im Kern von paketbasierten Metro- und Kernnetzen, aber auch in hochbitratigen Glasfaser-Zugangsnetzen.

Die Verarbeitung in den Xelerated Netzprozessoren erfolgt Datenfluss-orientiert – im Gegensatz zur Programmfluss-orientierten Verarbeitung in herkömmlichen Prozessoren. In herkömm-

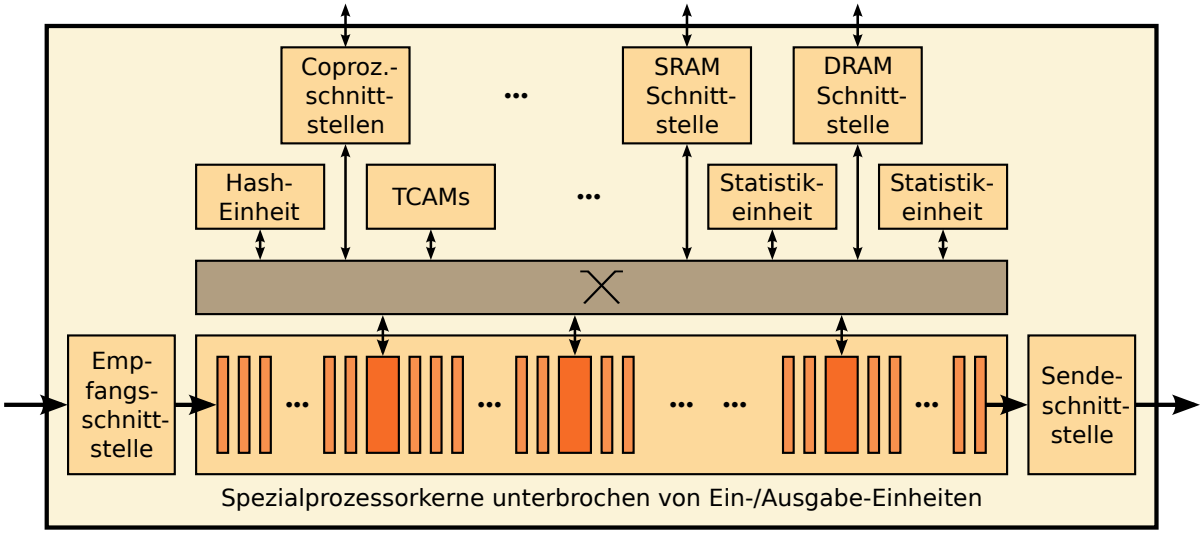

Abbildung 2.21: Architektur der Xelerated Netzprozessoren

lichen, Programmfluss-orientierten Prozessoren steuert ein Programm die Verarbeitung der Daten, indem es die Daten lädt, sie dann verarbeitet und anschließend wieder abspeichert. Bei der Datenfluss-orientierten Verarbeitung bewirken die im Prozessor ankommenden Daten das Laden eines Befehls, welchen der Prozessor dann auf den Daten ausführt.

Die Architektur [91, 92] basiert auf einer sehr langen Pipeline von mehreren Hundert gleichartigen, spezialisierten Prozessorkernen – die zurzeit leistungsfähigsten Netzprozessoren der HX3xx-Reihe verfügen über 500 solcher Kerne. In regelmäßigen Abständen unterbrechen Ein-/Ausgabeeinheiten die Pipeline. Diese ermöglichen den Zugriff auf externen SRAM und DRAM, externe Coprozessoren sowie interne Coprozessoren und TCAMs. Die Netzprozessoren verfügen über fünf TCAMs, einen Coprozessor zur schnellen algorithmischen Suche in Tabellen, eine Hash-Wert-Berechnungseinheit, sowie Einheiten zur Aufzeichnung verschiedener Statistiken. Spezielle Versionen beinhalten außerdem ein Verkehrsverwaltungsmodul (*Traffic Manager*). Abbildung 2.21 zeigt den prinzipiellen Aufbau dieser Netzprozessorarchitektur.

Für die Datenfluss-gesteuerte Verarbeitung der Paketdaten werden empfangene Pakete in 64 Byte große Datenworte aufgeteilt und hintereinander in die Pipeline geschoben. Die Datenworte wandern nun jeden zweiten Takt von einem Prozessor zum nächsten, zusammen mit Status- und Steuerinformationen wie Eingangs-Port, Zwischenergebnissen, Klassifizierungsergebnissen und einem Befehlsfolgezähler. Die Prozessoren führen jeweils nur einen Befehl pro Paket aus, welcher aufgrund einer VLIW-Architektur aus jeweils vier parallelen Operationen besteht. Das unterbrechungsfreie Weiterschieben der Paket-Datenworte alle zwei Takte bewirkt, dass die Netzprozessoren einen deterministischen Durchsatz garantieren können.

Während des Zugriffs auf externen Speicher oder Coprozessoren in den Ein-/Ausgabeeinheiten durchlaufen die Paketdaten einen FIFO-Speicher. Dieser ist so dimensioniert, dass die Daten die Einheit mit einer konstanten Verzögerung wieder verlassen. Diese Zeit ist ausreichend, um die auftretenden Speicherzugriffslatenzen bzw. die Verarbeitungszeit der Coprozessoren zu überbrücken. Durch eine feste Zuweisung der Ressourcen an die verschiedenen Ein-/Ausga-

beeinheiten vor der Inbetriebnahme des Netzprozessors gibt es keine Zugriffskonflikte. Dies garantiert weiterhin eine konstante Latenz und einen deterministischen Durchsatz.

Aufgrund der festen Länge der Pipeline ist auch die Anzahl der Befehle pro Paket begrenzt. Falls der Prozessor nicht mit Maximaldurchsatz betrieben werden muss, können Pakete für eine komplexere Verarbeitung auch mehrmals die Pipeline durchlaufen. Schleifen können nur aufgerollt ausgeführt werden, doch stellt dies für Paketverarbeitungsfunktionen auf Schicht 2 bis 4 i. d. R. keine große Einschränkung dar. Die Programmierung der Xelerated Netzprozessoren erfolgt in einer Spezialsprache über ein einfaches Programmiermodell eines Einzelprozessors mit einem Verarbeitungskontext.

EZchip – Pipeline von Pools spezialisierter Prozessorkerne

Die EZchip-Netzprozessoren NP-x zählen seit der Einführung des NP-1 Chips im Jahre 2002 zu den Netzprozessoren mit dem höchsten Durchsatz. Der NP-4, der seit 2010 vertrieben wird, kann Datenraten bis zu 100 Gbit/s verarbeiten [93]. Einsatzgebiet dieser Netzprozessoren ist die Paketverarbeitung bis zu Schicht 7 in sowie am Rand von Metro- und Kernnetzen.

Die Architektur aller NP-x Netzprozessoren von EZchip basieren auf einer Pipeline von Pools heterogener, spezialisierter Prozessorkerne, den so genannten *Task Optimized Processors* (kurz TOPs), und – seit dem NP-2 – einem Verkehrsverwaltungsmodul (Traffic Manager). Zusätzlich verfügt der NP-4 über zwei Standard-RISC-Prozessoren zur Konfiguration, Steuerung und Überwachung der Paketverarbeitungsprozessoren und des Verkehrsverwaltungsmoduls. Ein internes Koppelnetz erlaubt die Kommunikation dieser vier Einheiten untereinander sowie über Schnittstellenmodule mit externen Komponenten. Solche über Schnittstellenmodule angebundenen Elemente sind zum einen DRAM-Speicher zur Paketpufferung und für Statistik- sowie Klassifizierungs- und Weiterleitungstabellen, zum anderen TCAMs für zusätzliche nicht intern durchführbare, komplexe Klassifikationsaufgaben. Abbildung 2.22 zeigt ein Blockschaltbild dieser Architektur.

Die Paketverarbeitung erfolgt in fünf funktionalen Pipeline-Stufen [94]. Die erste Stufe enthält spezielle Prozessorkerne (*TOPparse*) zur Analyse und Dekodierung der Paket-Header und der Nutzdaten sowie zur Extraktion von Header-Feldern oder sonstigen Paketausschnitten. Diese Funktionalität erfolgt teilweise festverdrahtet durch eine Hardware-Einheit und teilweise programm-gesteuert. Die Prozessorkerne in der zweiten Stufe (*TOPsearch I*) verwenden die extrahierten Paketfelder zur Klassifizierung des Pakets und zum Heraussuchen von Weiterleitungsinformationen. Die unabhängigen Suchen können dabei parallel in mehreren Prozessorkernen erfolgen. Neben dem direkten Zugriff auf Tabellenwerte werden auch die Suche in Hash-Tabellen sowie die Suche in Baumstrukturen von speziellen, integrierten Hardware-Einheiten unterstützt. Die Tabellen können dabei sowohl in internem Speicher als auch in externem DRAM abgelegt sein. Zusätzlich kann ein externes TCAM-Modul verwendet werden. Die dritte Stufe enthält Spezialprozessorkerne (*TOPresolve*), die die Suchergebnisse sowie die Ergebnisse der ersten Stufe auswerten und unter zusätzlicher Berücksichtigung von Dienstgütekriterien entscheiden, wie das Paket weiter verarbeitet werden soll. Die vierte Pipeline-Stufe enthält ähnliche Prozessorkerne (*TOPsearch II*) wie die zweite Stufe. Hier können weitere, von den bisherigen Erkenntnissen abhängige Suchen vorgenommen werden. Die letzte Stufe enthält spezialisierte

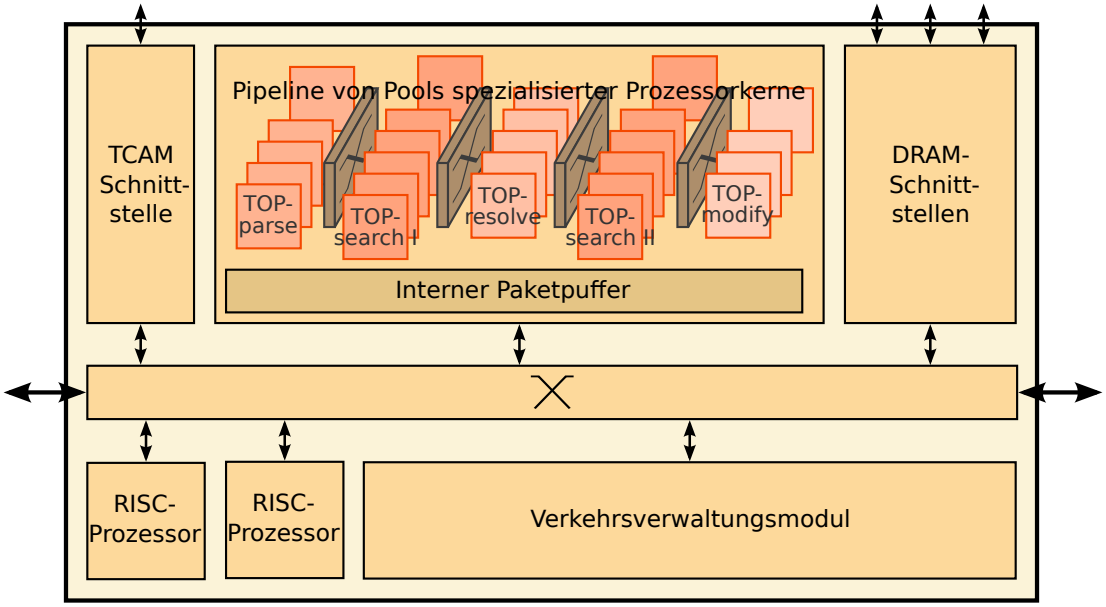

Abbildung 2.22: Architektur der EZchip NP-x Netzprozessoren

Prozessorkerne (*TOPmodify*) zur Modifikation des Pakets. Diese Prozessoren ändern Header-Felder und Inhalte bis zu Schicht 7 entsprechend der Ergebnisse der vorhergehenden Stufen.

Die Spezialprozessorkerne sind als vierstufige Pipelines realisiert, die einen Befehl pro Takt verarbeiten können. Der Befehlssatz der verschiedenen Prozessortypen ist ähnlich, jedoch jeweils an die verwendeten Spezial-Hardware-Einheiten angepasst. Alle Prozessorkerne einer Stufe führen jeweils den gleichen Programmcode aus, sind jedoch völlig eigenständig. Während der Verarbeitung ist das Paket in einem internen Paketpuffer abgelegt. Eine Verteileinheit weist jedes angekommene Paket einem freien Prozessorkern der ersten Stufe zu. Zwischen den Stufen gibt es ähnliche Einheiten, die wiederum fertig verarbeitete Paketkontexte jeweils einem oder bei Suchanfragen mehreren freien Prozessorkernen der nachfolgenden Stufe übergeben. Somit werden in jeder Stufe i. d. R. mehrere Pakete parallel verarbeitet. Umsortierungen von Paketen sowie Schreib-Lese-Abhängigkeiten werden von der Hardware transparent für den Programmierer aufgelöst.

Die Programmierung der Spezialprozessorkerne erfolgt in einer Assembler-ähnlichen Sprache, welche Spezialbefehle für den Zugriff auf die jeweiligen Hardware-Einheiten enthält. Das Programmiermodell besteht aus fünf einzelnen Prozessoren, deren Programme sequentiell für ein einzelnes Paket ausgeführt werden. Der Datentransport zwischen den Pipeline-Stufen, die Aufrechterhaltung der Reihenfolge zwischen Paketen sowie der synchronisierte Zugriff auf geteilte Ressourcen sind für den Programmierer nicht sichtbar.

Bay Microsystems – Pipeline spezialisierter VLIW-Prozessorkerne

Die Netzprozessorarchitektur von Bay Microsystems garantiert einen konstant hohen Durchsatz und eine feste Latenz bei der Verarbeitung von Paketen. Der seit 2006 erhältliche Netzprozessor Chesapeake [95] verspricht eine Datenrate von 40 Gbit/s und eine Paketrate von 122 MP/s –

bei 40 Byte-Paketen ohne Paketabständen. Die interne Datenrate beträgt beachtliche 125 Gbit/s. Aufgrund dieser Leistungszahlen zählen diesen Netzprozessor der Hersteller aber auch viele Branchenkenner zu den 100 Gbit/s-Ethernet-fähigen Paketverarbeitungssystemen (122 MP/s bei minimalen Ethernetpaketen entsprechen 82 Gbit/s). Das Einsatzgebiet der Bay Microsystems Netzprozessoren sind die Paketverarbeitung inklusive Verkehrsverwaltung in Vermittlungsknoten am Rand und innerhalb von Metro- und Kernnetzen, jedoch auch in Aggregationsnetzen.

Die Paketverarbeitungsarchitektur der Netzprozessoren von Bay Microsystems basiert auf einer funktionalen Pipeline sehr breiter und heterogener Verarbeitungsstufen [96]. Jede dieser Verarbeitungsstufen besteht aus vielen parallelen, spezialisierten Verarbeitungseinheiten. Die Einheiten einer Stufe verarbeiten parallel jeweils ein Paket. Um einen deterministischen Paketdurchsatz zu erreichen, ist die Verarbeitungsdauer in jeder Pipeline-Stufe gleich. Jede der jeweils gleichzeitig arbeitenden Verarbeitungseinheiten hat eine spezielle Funktionalität, die in Abhängigkeit des zu verarbeitenden Pakets verwendet oder deaktiviert werden kann. Es gibt in den verschiedenen Stufen der Pipeline jeweils mehrere spezialisierte Verarbeitungseinheiten für Dekodierfunktionen, für Klassifizierungsaufgaben, für Paketmodifizierungen sowie für Verkehrsverwaltungsfunktionen.

Der Netzprozessor besitzt außer seinen Hochgeschwindigkeitsnetzschnittstellen eine Schnittstelle zu einem externen Chip zur Klassifizierung oder IP-Adresssuche sowie DRAM-Schnittstellen für den Paketpuffer sowie zur Ablage von Tabellen. Die Programmierung erfolgt abstrahiert von der verwendeten Hardware mit Spezialbefehlen und mit einem Ein-Prozessor-Modell.

Cisco – Pool von RISC-Prozessorkernen

Cisco verwendet in seinen leistungsfähigsten Routern firmeneigene Netzprozessoren. Der Kernnetzrouter CRS-1 (*Carrier Routing System*), der seit 2004 vertrieben wird, verwendet den *Silicon Packet Processor* für die Paketverarbeitung. Dessen 2010 vorgestellter Nachfolger der CRS-3 verwendet den *QuantumFlow Array Processor*. Dies ist eine erweiterte Version des 2008 mit dem Randknoten-Router ASR-1000 vorgestellten *QuantumFlow Processor*.

Die 2004 eingeführte Architektur des Silicon Packet Processor basiert auf einem Pool von 188 für die Paketverarbeitung spezialisierten RISC-Prozessorkernen [97]. Diese sind in 16 Untergruppen mit jeweils zwölf Kernen strukturiert, wobei sich jede Untergruppe einen gemeinsamen Befehlsspeicher sowie eine gemeinsame Anbindung an Chip-weit geteilte Ressourcen teilt. Jeder Prozessor verfügt über einen Befehlscache sowie einen kleinen lokalen Datenspeicher. Jeder Prozessorkern verarbeitet ein Paket vollständig, wobei ihm automatisch zunächst die ersten hundert Bytes des Pakets mit den Protokoll-Headern übergeben werden. Bei der Verarbeitung kann er auf verschiedene, gemeinsam genutzte Speicher mit Tabellen, den internen Paketpuffer, spezielle Hardwareeinheiten für rechenintensive Operationen sowie TCAMs zur Klassifizierung zugreifen. Zur IP-Adresssuche steht ein spezielles Suchmodul zur Verfügung, welches den *Tree Bitmap*-Algorithmus [59] implementiert. Nach der Verarbeitung wird der Kopf des Pakets wieder mit dem Rest kombiniert.

Die QuantumFlow Netzprozessorarchitektur wurde 2008 vorgestellt und für den Rand-Aggregationsrouter ASR 1000 verwendet. Sie verwendet zwei Chips, einen für die Paketverarbeitung und einen für die Verkehrsverwaltung, welche hier nicht näher betrachtet wird. Der 2010

vorgestellte CRS-3 Kernnetzrouter verwendet Line Cards mit einer Paketverarbeitung in einer skalierten Version derselben Architektur. Der dort verwendete QuantumFlow Array Prozessor besteht aus einem Satz von sechs Chips.

Ziel der QuantumFlow-Architektur war eine skalierbare, leistungsstarke und flexibel programmierbare Paketverarbeitung von Schicht 2 bis 7 [98]. Sie basiert wie die Architektur des Silicon Packet Processor auf einem Pool von spezialisierten RISC-Prozessorkernen. Der Netzprozessor des ASR 1000 enthält 40 solcher Kerne, die Anzahl beim CRS-3 wurde bisher nicht veröffentlicht. Jeder der Prozessorkerne unterstützt Hardware-seitig vier Threads (der Silicon Packet Processor unterstützte kein Multi-Threading). Damit kann der Prozessor während Wartezeiten beim Zugriff auf Speicher und Coprozessoren die Verarbeitung eines anderen Pakets fortsetzen. Die Pakete liegen während der Verarbeitung in einem internen Paketpuffer. Je nach den benötigten Verarbeitungsfunktionen kann jeder Prozessor auf eine Vielzahl von internen Coprozessoren für Adresssuchen, Klassifizierungen, Hash-Wert-Berechnungen und andere verarbeitungsintensive Aufgaben zugreifen. Nach der Verarbeitung wird das Paket dem Verkehrsverwaltungsmodul übergeben, welches es in einer Warteschlange in externem Speicher ablegt. Die Programmierung erfolgt in C mit einem einfachen Ein-Prozessor-Programmiermodell.

Andere Router-Hersteller

Vermittlungsknotenhersteller wie Alcatel-Lucent und Juniper Networks, aber auch Huawei Technologies und ZTE, verwenden ebenfalls flexible Paketverarbeitungssysteme in ihren Routern für den Rand und innerhalb von Kern- und Metronetzen. Ebenso wie Cisco verwenden sie neben kommerziell erhältlichen auch selbst entworfene Netzprozessoren. Jedoch veröffentlichen sie kaum oder keine Informationen über die verwendeten Architekturen.

Alcatel-Lucent verwendet einen selbst entwickelten 100 Gbit/s-Netzprozessor-Chipsatz mit dem Namen *FP2* in seinen Hochleistungs-Randknoten-Routern [99]. Dieser besteht aus einem Paketverarbeitungs-Chip, einem Verkehrsverwaltungs-Chip und einem Switch-Fabric-Schnittstellen-Chip. Für die Ingress-Paketverarbeitung auf einer 100 Gbit/s-Line-Card werden aufgrund der hohen Anzahl von benötigten Verarbeitungsfunktionen zwei Paketverarbeitungs-Chips als funktionale Pipeline hintereinander geschaltet, in Egress-Richtung genügt ein solcher Chip. Die interne Paketverarbeitungsarchitektur eines solchen Chips basiert auf 112 Prozessorkernen, angeordnet in 16 Reihen zu je sieben Kernen [100]. Jede der Prozessorreihen führt eine andere Gruppe von Verarbeitungsfunktionen bis Schicht 7 aus und die Pakete verschiedener Flüsse oder Klassen werden entsprechend der benötigten Funktionen verteilt.

Juniper setzt in seinen neusten Randknoten-Routern einen Netzprozessor-Chipsatz namens *Junos Trio* ein [101]. Der Chipsatz besteht aus vier Chips, einem Schnittstellen-Prozessor zur Anbindung an paket- und verbindungsorientierte Kommunikationstechnologien auf Schicht 2, wie Carrier-Grade Ethernet oder SONET/SDH, einem Chip zur eigentlichen Paketverarbeitung von Schicht 2 bis 7 basierend auf einer hoch-parallelen Mehrkern-Architektur mit Multi-Threading sowie zwei Chips zur Verkehrsverwaltung und Pufferung. In seinen Kernnetz-Routern verwendet Juniper ebenfalls selbst entwickelte programmierbare ASICs [102] mit einer ähnlichen Chip-Aufteilung wie beim Junos Trio.

Diskussion und Zusammenfassung

Die Vorstellung der Architekturen der verschiedenen Netzprozessorhersteller zeigt, dass sich bis heute – etwa ein Jahrzehnt nachdem die ersten Netzprozessoren auf den Markt kamen – noch keine einheitlich verwendete Architektur für Hochgeschwindigkeitsnetzprozessoren etabliert hat. Dennoch kann man Vor- und Nachteile verschiedener Architekturmerkmale ausmachen und Schlussfolgerungen daraus ziehen. Tabelle 2.2 listet einige wesentliche Unterscheidungsmerkmale der vorgestellten Netzprozessorarchitekturen zusammengefasst auf.

Das wichtigste Unterscheidungsmerkmal ist die Organisation der Prozessorkerne untereinander (vgl. Abschnitt 2.3.5 und Unterkapitel 3.3). Die Prozessorkerne in den Netzprozessoren von Xelerated und Bay Microsystems sind als Pipeline organisiert, die Prozessorkerne in den Netzprozessoren von Intel/Netronome und Cisco als Pool und die Prozessorkerne der EZchip Netzprozessoren als Kombination dieser beiden Strukturen, als Pipeline mehrerer Pools (vgl. Abbildung 2.23).

Vorteil der Pipeline-Organisation ist die wesentlich einfachere Kommunikationsinfrastruktur zwischen den Komponenten auf dem Chip. Die Paketdaten bzw. -kontexte müssen lediglich über Punkt-zu-Punkt-Verbindungen kommuniziert werden, ebenso ist der Zugriff auf Coprozessoren oder interne Speicherblöcke meist auf ein oder wenige Kommunikationspartner begrenzt. Durch die über die Pipeline verteilte Verarbeitung, beispielsweise bei den Xelerated Netzprozessoren, wird auf spezielle Coprozessoren oder TCAMs jeweils nur an bestimmten Stellen in der Pipeline und zusätzlich hintereinander zugegriffen.

Bei einem Pool von Prozessorkernen müssen empfangene Pakete zum einen auf alle Prozessorkerne verteilt werden und nach deren Verarbeitung wieder eingesammelt werden, und zum anderen benötigt auch jeder Prozessorkern eine Zugriffsmöglichkeit auf gemeinsam genutzte Ressourcen wie interne und insbesondere externe Speicher und Coprozessoren. Ciscos QuantumFlow Netzprozessor mildert dieses Problem durch den Einsatz sehr vieler gleichartiger, interner Coprozessoren, so dass weniger Kerne um einen Coprozessor konkurrieren bzw. die Anfragen auf mehrere Einheiten verteilt werden können. EZchip mildert das Zugriffsproblem

Tabelle 2.2: Vergleich der betrachteten, kommerziellen Netzprozessorarchitekturen

Netzprozessor	Prozessorkerne		Anzahl von Coprozessoren	Deterministischer Durchsatz
Intel/Netronome	Pool	homogen	wenige	nein
Xelerated	Pipeline	homogen	einige	ja
Bay Microsystems	Pipeline	heterogen	integriert	ja
EZchip	Pipeline von Pools	heterogen	integriert	nein
Cisco	Pool	homogen	viele	nein

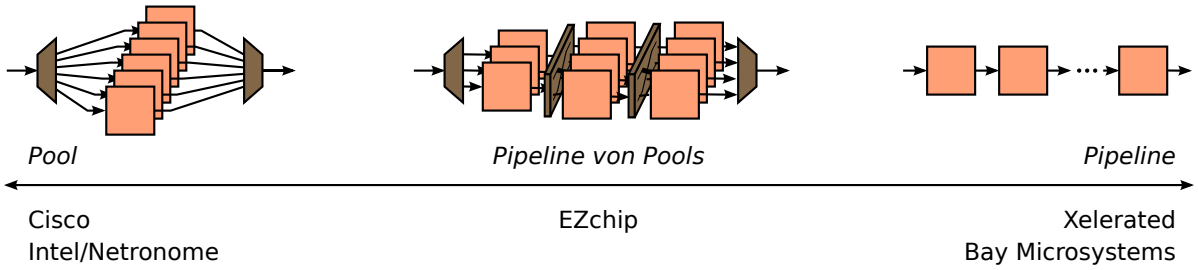

Abbildung 2.23: Organisation der Prozessorkerne

dadurch, dass jeweils nur die Einheiten einer Pipeline-Stufe um gemeinsame Ressourcen konkurrieren.

Die Netzprozessorarchitekturen von Xelerated und Bay Microsystems versprechen einen deterministischen Durchsatz. Dies können sie aufgrund der Tatsache, dass dank ihrer Pipeline-Struktur jeweils nur eine feste Anzahl von Einheiten gleichzeitig auf eine Komponente zugreifen und sie somit deterministische Zugriffszeiten auf Speicher und Coprozessoren garantieren können. Die Pool-Architekturen von Cisco und Intel können dies nicht garantieren, da im schlimmsten Fall alle Einheiten um eine geteilte Ressource konkurrieren. Andererseits können in Pool-Architekturen längere Verarbeitungszeiten einzelner Pakete durch kürzere Verarbeitungszeiten anderer Pakete ausgeglichen werden. Somit können diese „im Normalfall“ ebenfalls häufig sehr hohe Datenraten unterstützen, diese allerdings (im schlimmsten Fall) nicht garantieren.

Ein weiteres Unterscheidungsmerkmal ist, insbesondere bei Netzprozessoren mit Pipeline-Verarbeitung, ob die Prozessorkerne gleichartig (homogen) oder heterogen für ihre jeweilige Aufgabe spezialisiert sind (vgl. Abbildung 2.24). Vorteil spezialisierter Kerne ist, dass sie ihre Verarbeitungsfunktion effizienter durchführen können, sowohl hinsichtlich der Geschwindigkeit, als auch hinsichtlich des Ressourcen- und Energieverbrauchs. Nachteil ist die geringere Flexibilität, da Funktionen, die nicht vorgesehen wurden, nur sehr umständlich oder gar nicht realisiert werden können. Außerdem werden zur Programmierung für die verschiedenen Prozessorkerne unterschiedliche Programmiersprachen bzw. unterschiedliche Befehle benötigt. Beispiele für Systeme mit heterogenen Prozessorkernen sind die Netzprozessoren von Bay Microsystems und EZchip.

Die geringere Verarbeitungsleistung aufgrund ihrer gleichartigen, weniger spezialisierten Prozessorkerne gleichen die Netzprozessoren von Cisco und Xelerated durch eine Vielzahl von Coprozessoren aus, welche beliebig angesprochen werden können. Somit kombinieren sie die Flexibilität gleichartiger, weniger spezialisierter Prozessorkerne mit der zuschaltbaren Leistungsfähigkeit spezialisierter Coprozessoren. Die Netzprozessoren von Intel/Netronome, welche lediglich über kleine Prozessorkern-lokale TCAMs sowie einen gemeinsamen Verschlüsselungs-Coprozessor verfügen, erreichen nur einen geringeren Durchsatz.

Anzumerken ist außerdem, dass die Pool-Netzprozessorsysteme von Cisco, sowie auch die von Alcatel-Lucent und Juniper, aus mehreren Chips bestehen, während die Pipeline-Netzprozessoren von Xelerated und Bay Microsystems sowie der Netzprozessor von EZchip jeweils nur einen Chip (oder zwei bei eigenem Verkehrsverwaltungschip) pro Richtung benötigen. Grund hierfür könnte der höhere Chipflächenbedarf bei Pool-Architekturen sein, sowohl für die be-

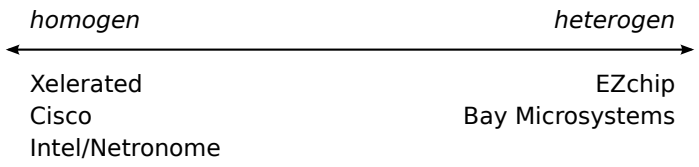

Abbildung 2.24: Homogenes oder heterogenes Mehrprozessorsystem

nötigten komplexen Verbindungsstrukturen als auch für die vielen internen Coprozessoren und Speichermodule, um Ressourcenengpässe zu vermeiden.

Zusammenfassend kann man feststellen, dass ein hoher Paketverarbeitungsdurchsatz am günstigsten mit einer Pipeline-Organisation von Prozessorkernen erreicht werden kann. Grund hierfür ist, dass diese Anordnung Ressourcenkonflikte minimiert und somit konstante Zugriffszeiten erreicht werden können. Dies erlaubt die Garantie eines deterministischen Durchsatzes. Pool-Konfigurationen weisen einen höheren Ressourcenbedarf auf, welcher durch eine komplexere Kommunikationsinfrastruktur und mehr redundante Verarbeitungs- und Speicherressourcen begründet ist. Schließlich ist festzustellen, dass spezialisierte Prozessorkerne einerseits die Verarbeitung beschleunigen sowie den Chipflächen- und Energiebedarf verringern, andererseits jedoch auch mit einer geringeren Flexibilität einhergehen.

2.4.2 Systeme auf Basis von Programmierbaren Logikbausteinen

Programmierbare Logikbausteine werden aufgrund ihrer hohen Flexibilität in zunehmendem Maße für Paketverarbeitungssysteme anstatt von ASICs oder Netzprozessoren verwendet und nicht mehr wie früher lediglich zur Anpassung der Schnittstellen zwischen ASIC-Chips oder zur Fertigung von Prototypen. FPGAs werden aufgrund ihrer hohen Verkaufszahlen stets mit den aktuellsten Halbleiterfertigungstechnologien hergestellt, was sich positiv auf ihre Leistungsfähigkeit und ihren Energieverbrauch auswirkt [70, 71]. Die FPGA-Hersteller integrieren außerdem stets sehr leistungsfähige *Transceiver* (kombinierte Sende- und Empfangsmodule), so dass auch der Empfang und das Versenden von Paketen bei hohen Datenraten über wenige Pins möglich ist.

Im Folgenden werden programmierbare Logikbausteine vorgestellt, welche sich für Hochgeschwindigkeitspaketverarbeitung eignen. Anschließend wird auf kommerziell vertriebene Paketverarbeitungssysteme und -komponenten auf FPGA-Basis eingegangen, bevor eine Architektur vorgestellt wird, welche als Kombination aus FPGA und Netzprozessor angesehen werden kann.

Hochgeschwindigkeits-FPGAs zur Paketverarbeitung

Die beiden großen FPGA-Hersteller Xilinx und Altera bieten verschiedene FPGAs an, welche für Hochgeschwindigkeitspaketverarbeitungssysteme geeignet sind.

Xilinx – Die TXT-Modelle der Hochleistungs-FPGAs *Virtex-5* von Xilinx verfügen über bis zu 48 Transceiver mit einer unterstützten Datenrate bis zu 6,5 Gbit/s. In [103] zeigt Xilinx, wie

diese zur Anbindung des FPGAs über drei externe Vierfach-PHY-Chips an ein optisches 100 Gbit/s-Modul verwendet werden können. Bei der neuen ab 2010 verfügbaren *Virtex-6*-Generation verfügen alle Modelle über diese 6,5 Gbit/s-Transceiver und die Spitzenmodelle zusätzlich über 24 11,2 Gbit/s-Transceiver, sodass zehn Transceiver zur Anbindung an ein 100 Gbit/s-Modul ausreichen. Die Xilinx-Transceiver verfügen jeweils über PMA- (*Physical Medium Attachment Sublayer*) und PCS- (*Physical Coding Sublayer*) Schicht-1-Module. Somit müssen diese Funktionsblöcke nicht mit Logikzellen realisiert werden, was sowohl Chip-Ressourcen als auch Energie einspart. Die ebenfalls in den Xilinx FPGAs enthaltenen, wenigen, festverdrahteten Gigabit-Ethernet-MAC-Module sind in Hochgeschwindigkeitspaketverarbeitungsmodulen kaum verwendbar.

Altera – Die *Stratix-IV* FPGAs von Altera verfügen in den GX-Modellen über bis zu 32 Transceiver mit einer maximalen unterstützten Datenrate von 8,5 Gbit/s und 16 zusätzlichen 6,5 Gbit/s-Transceivern. Die GT-Modelle weisen sogar bis zu 32 Stück der 11,3 Gbit/s-Transceiver auf. [104] beschreibt, wie erstere GX-Modelle – wie die Xilinx FPGAs oben – über drei externe Vierfach-PHY-Chips und letztere GT-Modelle über ihre 11,3 Gbit/s-Transceiver direkt mittels zehn Transceivern an ein 100 Gbit/s-Modul angebunden werden können. Wie die Xilinx-Transceiver verfügen auch diese über integrierte PMA- und PCS-Module.

Die für 2011 angekündigten *Stratix-V* FPGAs, welche in 28 nm Technologie gefertigt werden sollen, werden zusätzlich über 28 Gbit/s-Transceiver verfügen [105]. Somit sind nur noch vier Transceiver zur Anbindung an 100 Gbit/s-Schnittstellenmodule notwendig. Die Stratix-V FPGAs können somit auch an mehrere 100 Gbit/s-Module sowie an zukünftige 400 Gbit/s-Module angebunden werden [106]. Die Stratix-V FPGAs werden außerdem über doppelt so viele Flipflops pro Logikzelle verfügen. Dies sowie verteilte Speicherblöcke mit integrierter Registerstufe sind besonders für die schnelle Paketverarbeitung in einer Pipeline-Struktur mit sehr breiten Datenworten nützlich.

Außer den beiden großen FPGA-Herstellern gibt es einige neu gegründete Firmen, welche mit neuen Technologien und Architekturen schnellere, ressourceneffizientere, energieeffizientere oder kostengünstigere programmierbare Logikbausteine entwickeln und anbieten. Nachfolgend werden drei solcher neuartigen FPGA-Typen vorgestellt, die sich für die schnelle Paketverarbeitung in Vermittlungsknoten eignen.

Achronix – Achronix entwickelte FPGAs, welche eine Taktfrequenz von bis zu 1,5 GHz unterstützen sollen [107]. Das ist dreimal so viel wie die FPGAs von Altera oder Xilinx erreichen, wobei Achronix nur einen 65 nm Prozess für die Herstellung verwenden, im Gegensatz zu den 40 nm bei Alteras aktuellen FPGAs. Achronix erreicht diese hohe Verarbeitungsrate durch seine interne *picoPIPE*-Architektur. Die gewünschten Logikfunktionen realisiert hierbei eine sehr fein-granulare Pipeline-Struktur, deren einzelne Pipeline-Stufen die zu verarbeitenden Daten untereinander asynchron über einen Handshake-Mechanismus weiterleiten. Hierdurch wird ein sehr hoher Durchsatz erreicht. Nach außen verhält sich der FPGA wie ein normaler, synchron getakteter FPGA, außer dass aufgrund der internen hohen Verarbeitungsrate eine wesentlich höhere Taktrate unterstützt werden kann. Laut [108] wird dies jedoch durch eine geringere Logikdichte und einen vergleichsweise hohen Energieverbrauch erkauft. Achronix bietet jedoch einerseits die Möglichkeit, die FPGAs mit geringerer Betriebsspannung zu betreiben und somit einen geringeren Durchsatz durch eine geringere

Leistungsaufnahme aufzuwiegen, und andererseits die Möglichkeit, bei einem langsameren externen Takt die internen Logikelemente im Zeitmultiplexbetrieb mehrmals pro externem Taktzyklus zu verwenden und somit die effektive Logikdichte zu erhöhen.

Die FPGAs sind in vier Versionen mit unterschiedlicher Logikkapazität erhältlich, wobei die größte FPGA-Version vierzig 10,3 Gbit/s-Transceiver enthält. Dies ermöglicht den Einsatz dieser FPGAs für Hochgeschwindigkeitspaketverarbeitungssysteme. Dies ist laut Achronix auch einer der Zielanwendungsbereiche ihrer Produkte, was durch die Bereitstellung einer Evaluationsplatine mit einem Full-Duplex-Durchsatz von 120 Gbit/s für Infiniband/Ethernet-Anwendungen unterstrichen wird [109].

Tabula – Tabula vertreibt ABAX FPGAs mit einer so genannten *SpaceTime*-Architektur [110]. Der FPGA wird intern mit einer Frequenz von 1,6 GHz betrieben und kann in jedem Taktzyklus die Konfiguration der Lookup-Tabellen und Verbindungsleitungen ändern. So kann jedes Logikelement bei einer externen Frequenz von 200 MHz bis zu achtmal genutzt werden. Dieses Zeitmultiplex emuliert eine wesentlich höhere Logikdichte. Verbindungen zwischen verschiedenen Logikelementen sind wesentlich kürzer, da sie entweder in derselben Logikzelle in verschiedenen Zeitschlitzen durchgeführt wird oder aufgrund der somit kleineren Chipfläche örtlich näher sind. Die FPGAs verfügen über eine große Anzahl von platzsparenden SRAM-Zellen mit nur einem Port im Gegensatz zu den sonst üblichen doppelt so großen Dual-Port-Speichern. Da auch diese im Zeitmultiplex betrieben werden, stehen dem eigentlichen digitalen System bis zu acht Ports zur Verfügung. Die FPGAs verfügen über 48 Transceiver mit einer Datenrate bis zu 6,5 Gbit/s. Tabula empfiehlt in [111] diese FPGAs für Paketverarbeitungssysteme in Mobilkommunikationsnetzen sowie in Carrier-Grade Ethernet-Metronetzen, wofür sie aufgrund ihrer „virtuell“ hohen Logikdichte bei einer Frequenz von 200 MHz auch gut geeignet erscheinen. Allerdings ist fraglich, wie diese Technologie skalieren wird, falls höhere Taktraten benötigt werden.

Tier Logic – Tier Logic vertreibt FPGAs mit einer neuen so genannten 3D-Technologie [112]. Bei diesen FPGAs sind die SRAM-Zellen zur Konfiguration nicht zwischen den Logikblöcken angeordnet, sondern in einer Schicht über den Logikblöcken. Somit können die Logikblöcke dichter gepackt werden. Dadurch wird weniger Chipfläche benötigt und – aufgrund kürzerer Verbindungsleitungen – kann der FPGA schneller und energieeffizienter betrieben werden. Diese Eigenschaften machen diese FPGAs zu geeigneten Kandidaten für Hochgeschwindigkeitspaketverarbeitungssysteme. Durch Ersetzen der beschriebenen Konfigurations-SRAM-Schicht durch eine Metalllage kann Tier Logic auf einfache und schnelle Weise eine ASIC-Version des FPGAs mit exakt denselben Eigenschaften erstellen. Tier Logic wurde im Sommer 2010 aufgrund fehlender Investoren aufgelöst.

FPGA-Paketverarbeitungssysteme und -komponenten

TPACK und Ethernity Networks vertreiben Paketverarbeitungssysteme auf Basis von FPGAs. Diese werden nachfolgend näher beschrieben. Daran anschließend wird auf kommerziell vertriebene Paketverarbeitungskomponenten eingegangen, welche in FPGA-Anwendungen verwendet werden können.

TPACK – TPACK verkauft Schicht-2-Paketverarbeitungssysteme auf Basis von Altera FPGAs. Die Hauptanwendungsgebiete sind die Paketverarbeitung in Carrier-Grade Ethernet-Swit-

ches sowie das Umsetzen von Paketverkehr – im Wesentlichen Ethernet-Verkehr – auf SONET/SDH- oder OTN-Kanäle in Aggregations- und Metronetzknoten. Bei Verwendung der leistungsfähigsten FPGAs erreicht der TPX4004 einen deterministischen Full-Duplex-Durchsatz von bis zu 80 Gbit/s [113] und ein 100 Gbit/s-Full-Duplex-Paketverarbeitungssystem wurde bereits angekündigt [114]. (Die Netzprozessor-Chips von Xelerated oder EZchip unterstützen jeweils nur *simplex* 100 Gbit/s, enthalten dafür jedoch ein komplexeres Verkehrsverwaltungsmodul.)

Das Vertriebskonzept von TPACK heißt *SoftSilicon* [115]. Dabei bietet TPACK seinen Kunden Paketverarbeitungssysteme auf FPGA-Basis mit verschiedenen vordefinierten Merkmalen an. Der Kunde kann dieses System anhand von Konfigurationsoptionen an seine Bedürfnisse anpassen lassen sowie auch eigene Verarbeitungsmodule mit auf dem FPGA integrieren lassen. TPACK bietet nach Auslieferung der Systeme zeitnahe Aktualisierungen der Verarbeitungsfunktionen an, sobald sich die Anforderungen aufgrund von neuen Standards ändern bzw. der Kunde neue funktionale Anforderungen hat. Der Kunde bekommt keinen Einblick in den Quellcode der Hardware-Beschreibung des Systems. Eine begrenzte Konfigurations- und Steuerungsmöglichkeit wird über eine Software-Schnittstelle bereitgestellt. TPACK wurde im Sommer 2010 von AppliedMicro aufgekauft. Diese wollen TPACKs Produkte weiterhin verkaufen und weiterentwickeln.

Ethernity Networks – Ethernity Networks bieten Paketverarbeitungssysteme für Vermittlungsknoten im Zugangsnetz an [116]. Hierfür verwenden sie FPGAs von Altera. Ihre beiden Produktreihen ENET3xxx und ENET4xxx garantieren einen deterministischen Durchsatz von bis zu 10 Gbit/s bzw. 15 Gbit/s, außerdem verspricht eine angekündigte Version einen Durchsatz von 40 Gbit/s [117].

Der deterministische Durchsatz wird durch eine Pipeline von spezialisierten Verarbeitungseinheiten realisiert. Der Datenpfad durch die Pipeline hat in der leistungsfähigsten Version eine Breite von 256 bit, also 32 Byte, wobei von jedem Paket jeweils nur die ersten 128 Byte die Pipeline durchlaufen. Dies ermöglicht bei „normalem“ Verkehr, der auch größere Pakete enthält, einen höheren als den minimal garantierten Durchsatz. Die einzelnen Verarbeitungseinheiten können auf verschiedene externe DRAM-Speicher zugreifen, welche für Suchtabellen, Paketpuffer und die Pufferverwaltung genutzt werden.

Die einzelnen Paketverarbeitungseinheiten bieten eine begrenzte Programmierbarkeit bzw. Konfigurierbarkeit. Diese wird durch änderbare Konfigurationsregister sowie anpassbare Such- und Weiterleitungstabellen erreicht, welche über eine Schnittstelle zu einem externen Standardprozessor angesprochen werden können. Die Konfigurierbarkeit der FPGAs wird lediglich von Ethernity Networks selbst zur Anpassung der Netzschnittstellen entsprechend der Bedürfnisse der Kunden genutzt.

IP-Cores (*Intellectual-Property-Cores*) sind optimierte Hardware-Beschreibungen für vielfach benötigte Funktionsblöcke. Diese bieten die beiden großen FPGA-Hersteller Altera und Xilinx sowie mehrere kleinere Firmen an – teilweise als VHDL- oder Verilog-Quelltext und teilweise als bereits übersetzte Netzlisten. In [118, 119] sind Auflistungen aller von verschiedenen Firmen angebotenen IP-Cores für Altera und Xilinx FPGAs, darunter auch viele für Paketverarbeitungssysteme benötigte Funktionsblöcke wie Ethernet-MAC-Module, SONET/SDH-Schnittstellenmodule, Verschlüsselungs- und sonstige Codiermodule, SRAM- und DRAM-Speicher-Ansteuermodule (*Controller*) sowie verschiedene Mikroprozessoren.

MoreThanIP und **Sarance** – Hervorzuheben sind MoreThanIP und Sarance, die PHY- und MAC-Module für 100 Gbit/s-Ethernet als IP-Cores anbieten [120, 121]. Sarance vertreibt außerdem auch Hardware-Beschreibungen für Klassifikationsmodule, CAMs sowie Module für Verkehrsverwaltungsaufgaben [121].

FPGA-Netzprozessor-Kombination

Cswitch – Cswitch entwickelte eine Paketverarbeitungsarchitektur, die eine Kombination aus Netzprozessor und FPGA darstellt [122]. Die Firma vertrieb lediglich die in 90 nm Technologie gefertigten und für einen Durchsatz von bis zu 40 Gbit/s dimensionierte CS90-Chipfamilie, bevor sie im Jahre 2009 von der chinesischen Firma Agate-Logic aufgekauft wurde.

Die von Cswitch entwickelte, so genannte *Configurable Switch Array* Architektur basiert auf einem sehr leistungsstarken internen Kommunikationsnetz das verschiedene rekonfigurierbare Logikstrukturen, interne Speicherblöcke sowie mehrere schnelle Netz- und Speicherschnittstellen verbindet. Das interne Kommunikationsnetz besteht aus einem Array von 20 bit breiten 40 Gbit/s-Verbindungen. Zusätzlich verfügt der Chip über flexible All-Zweck-Verbindungsstrukturen, wie man sie von Standard-FPGAs kennt.

Zur Paketverarbeitung stehen verschiedene Module zur Verfügung: Spezielle, in C programmierbare Parser-Prozessoren können Paket-Header analysieren und dabei Header-Felder extrahieren. Der Chip verfügt über 56 solcher Prozessoren, wobei 28 zusammen einen Durchsatz von 40 Gbit/s bei Ethernet-Verkehr mit minimalen Paketen erreichen. Es gibt mehrere, kleine, rekonfigurierbare CAMs, welche man für die Paketklassifizierung und das Heraussuchen von Weiterleitungsinformationen nutzen kann. Des Weiteren enthält der Chip spezielle rekonfigurierbare arithmetisch-logische Einheiten. Diese bestehen im Wesentlichen aus Addierern, Multiplizierern, logischen Operationswerken sowie Register- und Multiplexerstrukturen. Die Einheiten wurden so spezialisiert, dass sie effizient Prüfsummenberechnungen, Verschlüsselungsoperationen sowie spezielle Galois-Feld-Operationen durchführen können. Schließlich verfügt die Cswitch-Architektur noch über FPGA-übliche Logikmodule, bestehend aus LUTs, Flipflops und Speicherzellen.

Die Cswitch-Chips verfügen über bis zu 18 Mbit internen Speicher, der für kleine Paketpuffer sowie zur Speicherung von Zählern und Tabellen genutzt werden kann. Außerdem kann man über spezialisierte konfigurierbare Speicher-Schnittstellen inklusive Ansteuerungsmodul (Controller) auf externe Speichermodule zugreifen. Für die Paketschnittstellen stehen bis zu 24 Transceiver mit einer unterstützten Datenrate von bis zu 6,4 Gbit/s sowie bis zu sechs festverdrahtete 10 Gbit/s-Ethernet-MAC-Module zur Verfügung.

Zusammenfassung

Auf den letzten Seiten wurden aktuelle und angekündigte FPGAs vorgestellt, welche gut für Paketverarbeitungssysteme geeignet sind. Gründe für die gute Eignung von FPGAs ist deren inhärente Flexibilität bis auf Gatterebene sowie deren in den letzten Jahren ständig steigende Leistungsfähigkeit. Letztere ist damit begründet, dass FPGAs aufgrund ihrer hohen Verkaufszahlen

stets mit den kleinsten technisch-möglichen Strukturgrößen hergestellt werden. Dadurch können trotz der Rekonfigurierbarkeit sehr viele und schnelle Logikzellen auf einem Chip integriert werden, wodurch viele Funktionen mit einem hohen Verarbeitungsdurchsatz realisiert werden können. Die hohe Integrationsdichte erlaubt außerdem, viele festverdrahtete Module zusätzlich zu den Logikzellen auf dem Chip zu realisieren. Heutige FPGAs enthalten daher viele für die Paketverarbeitung nützliche Komponenten wie Speicherblöcke für Paketpuffer und Tabellen zur Paketklassifikation und Statistikaufzeichnung, Prozessorkerne für Software-gesteuerte Funktionen insbesondere für Steuerungs- und Verwaltungsaufgaben sowie Hochgeschwindigkeits-Transceiver für den Empfang und das Versenden von Paketen über wenige Pins. Durch neue Technologien werden FPGAs außerdem auch zunehmend energieeffizient.

Firmen wie TPACK und Ethernity Networks entwickeln und verkaufen unter Verwendung solcher aktueller FPGAs Paketverarbeitungskomplettsysteme, welche einfach an neue Anforderungen von Seiten der Kunden oder der Standardisierungsgremien angepasst werden können. Allerdings bleibt hierbei die Möglichkeit der Rekonfigurierung der FPGAs aus wirtschaftlichen Interessen auf die anbietenden Firmen begrenzt. Die Firma Cswitch demonstrierte mit ihrer CS90-Architektur, wie ein speziell für Paketverarbeitung optimierter FPGA aussehen kann.

Eine andere Möglichkeit für die Realisierung schneller Paketverarbeitungssysteme auf FPGA-Basis ist, hierfür bereits implementierte und optimierte Module mit häufig benötigten Funktionalitäten zu verwenden und nur spezielle Funktionsblöcke selbst zu entwickeln. Sowohl die beiden FPGA-Hersteller Altera und Xilinx sowie insbesondere Sarance und MoreThanIP bieten verschiedene nützliche IP-Cores für die Paketverarbeitung bei 100 Gbit/s an. Dies sind z. B. schnelle Speicheransteuerungsmodule, 100 Gbit/s-Ethernet-MACs, CAMs oder Verkehrsverwaltungsmodule.

2.5 Stand der Wissenschaft

Weltweit forschen Wissenschaftler an Universitäten und anderen Forschungseinrichtungen daran, den Durchsatz sowie die Flexibilität von Paketverarbeitungssystemen weiter zu erhöhen. Flexibilität wird dabei i. d. R. durch die Software in Prozessorkernen oder durch programmierbare Logik in FPGAs erzielt. Forschungsarbeiten zu der Frage, wie diese Flexibilität weiter erhöht bzw. besser nutzbar gemacht werden kann, stellt Abschnitt 2.5.2 vor. Zunächst werden jedoch in Abschnitt 2.5.1 architekturelle Ansätze diskutiert, wie der Durchsatz solcher Paketverarbeitungssysteme erhöht werden kann.

2.5.1 Hoher Durchsatz

Ein hoher Paketdurchsatz lässt sich durch verschiedene architekturelle Ansätze erreichen. Zum einen kann ein hoher Durchsatz durch eine hoch-parallele Verarbeitung erreicht werden. Eine weitere Möglichkeit ist, die Verarbeitungsleistung durch zusätzliche Module zu erhöhen, welche die Durchführung häufig benötigter Funktionen beschleunigen. Schließlich können die Verarbeitungsmodule selbst für die zur Verarbeitung benötigten Funktionen spezialisiert werden. Im Folgenden werden verschiedene Forschungsarbeiten, welche diese drei unterschiedlichen Kategorien von Beschleunigungsansätzen verfolgen, vorgestellt und diskutiert.

Hohe Parallelisierung

Eine hohe Verarbeitungsrate kann erreicht werden, indem man möglichst viele parallele Verarbeitungseinheiten einsetzt. Daher werden in der Prozessortechnik in den letzten Jahren vermehrt Systeme entwickelt, welche mehr und mehr Prozessorkerne integrieren.

Saif [123] berichtet von der Implementierung eines IP-Routers auf dem 16-Kern-Prozessor *RAW* des *Massachussetts Institute of Technology*. Die skalierbare RAW-Architektur verfügt über ein hochoptimiertes Netzwerk von gleichartigen Prozessorkernen. Sowohl die Verarbeitung in den Prozessoren als auch die Art der Inter-Prozessor-Kommunikation sind vollständig Software-gesteuert. Die Paketverarbeitung in den untersuchten Router-Implementierungen erfolgt in vier parallelen, funktionalen Pipelines, bestehend jeweils aus vier hintereinander geschalteten Prozessorkernen. Die beste Router-Implementierung erzielt einen Durchsatz von bis zu 9,75 MP/s, ausreichend für sechs 1 Gbit/s-Ethernet-Verbindungen.

Niemann [124] stellt ein skalierbares Mehrprozessorsystem vor und zeigt dessen Leistungsfähigkeit anhand einer Implementierung der Fast-Path-Paketverarbeitungsfunktionen eines IP-Routers für Zugangsnetze. Das System besteht aus einem Netzwerk von jeweils vier eng gekoppelten Prozessorkernen. Zusätzlich können je nach Zielanwendung auch Coprozessoren in das System integriert sein. Niemann berichtet, dass eine Gruppe aus vier Prozessorkernen und einem Coprozessor zur Prüfsummenberechnung einen Durchsatz von über 5 MP/s bei 40 Byte großen Paketen erzielt. Ein System aus 20 solcher Gruppen, also ein System aus 80 Prozessorkernen, erreicht somit bei 40 Byte-Paketen eine Rate von 100 MP/s. Dies entspricht, so Niemann, einem um etwa zwei Größenordnungen höheren Durchsatz als ihn Standardprozessoren mit vergleichbarer Chipfläche aufweisen, bei einem um zwei Größenordnungen geringeren Energiebedarf.

Kritisch bei den beiden oben beschriebenen Mehrkern-Architekturen ist deren Leistungsfähigkeit, wenn alle oder mehrere Prozessorkerne auf eine gemeinsam genutzte Ressource zugreifen müssen. In [124] müssen sich allerdings jeweils nur vier Prozessorkerne einen Coprozessor teilen, wodurch hier noch kein Flaschenhals entsteht. Ebenso greifen in [123] maximal vier Prozessorkerne auf gemeinsam genutzten Speicher zu. Die in den beiden Arbeiten implementierten Router gehen jeweils von unabhängig verarbeitbaren Paketen aus, wodurch keine Daten geteilt werden müssen. So wird auf gemeinsame Daten, z. B. die Weiterleitungstabellen, nur lesend zugegriffen und geschrieben wird nur auf separate Speicherbereiche. Somit kommt es zu keinen Konsistenzproblemen.

Rai [125] untersucht die Verwendung eines *Stream*-Prozessors für die Implementierung von IP-Router-Funktionen sowie zur Verschlüsselung von Paketdaten. Stream-Prozessoren werden hauptsächlich in modernen Grafikprozessoren verwendet und bestehen aus SIMD-Prozessorkernen (*Single-Instruction-Multiple-Data*). Ein Befehl wird hier jeweils parallel in mehreren Verarbeitungseinheiten auf Datenelemente derselben Art angewendet. Der Vorteil bei der Verwendung einer solchen Stream-Architektur zur Paketverarbeitung im Vergleich zu anderen Mehrprozessorarchitekturen ist der durch die Architektur vorgegebene, synchronisierte, sequentielle Zugriff auf den Speicher. Dadurch nutzt das System die verfügbare Speicherbandbreite sehr gut und es gibt keine Konflikte beim Zugriff auf gemeinsam genutzte Daten.

Optimierung für Standardfälle

Neben dem Einsatz von mehreren parallelen Prozessoren oder programmierbaren Verarbeitungseinheiten gibt es auch andere Möglichkeiten, den Durchsatz eines Paketverarbeitungssystems zu erhöhen. Im Folgenden werden Forschungsarbeiten vorgestellt, welche durch zusätzliche Einheiten die Verarbeitungsdauer der meisten Pakete wesentlich verkürzen können.

Okuno und Ishida schlagen eine Cache-basierte Netzprozessorarchitektur vor [126, 127]. Ein solches System soll die zeitliche Lokalität von Paketen mit identischer Quell- und Zieladresse ausnutzen, um bis zu 80% der Verarbeitungszeit einzusparen. Ein Analyse-Modul extrahiert aus jedem ankommenden Paket einen Merkmalsvektor mit für die Verarbeitung relevanten Informationen, wie z. B. die verwendeten Protokolle, die Quell- und Zieladresse und die TCP-Port-Nummern. Ein Cache-Modul überprüft daraufhin, ob ein Paket mit demselben Merkmalsvektor schon einmal verarbeitet wurde. Ist dies der Fall, so liest es die entsprechenden modifizierten Header-Felder aus dem Cache aus und ein Modifizierungsmodul ändert das Paket entsprechend. Findet das Cache-Modul keinen passenden Eintrag, so führt einer von mehreren Prozessorkernen eine vollständige Verarbeitung des Pakets durch und speichert die veränderten Header-Felder mit dem entsprechenden Merkmalsvektor anschließend im Cache-Modul. Messungen unter Verwendung von Aufzeichnungen von echtem Kernnetzpaketverkehr zeigen, dass die Prozessorkerne nur noch etwa 10% bis 35% der Pakete verarbeiten müssen. Der realisierte Prototyp auf einem FPGA erreicht einen Durchsatz von 2 Gbit/s. Eine mögliche ASIC-Implementierung kann jedoch nach Ansicht der Autoren 100 Gbit/s unterstützen.

Ohlendorf und Meitinger verfolgen eine andere Strategie. Sie schlagen eine Architektur namens *FlexPath NP* vor [128, 129], welche es ermöglicht, für verschiedene Klassen von Paketen unterschiedliche Verarbeitungspfade durch das Verarbeitungssystem zu bestimmen. So können normale IP-Pakete, welche keine spezielle Verarbeitung benötigen, direkt über optimierte, nicht- oder nur begrenzt-programmierbare Weiterleitungsmodule verarbeitet werden. Pakete, die z. B. ver- oder entschlüsselt werden sollen, werden mittels eines Kryptografie-Moduls verarbeitet. Andere Pakete, für die es keine speziellen Coprozessoren gibt, verarbeiten programmierbare RISC-Prozessoren. Der realisierte FPGA-basierte Prototyp erreicht einen Durchsatz von 3,2 Gbit/s und die Autoren erwarten einen Durchsatz von 10 Gbit/s und mehr von einer Realisierung als ASIC.

Andere Forscher schlagen vor, die Rekonfigurierbarkeit von FPGAs auszunutzen, um die Funktionalität eines Paketverarbeitungssystems zur Laufzeit an den aktuellen Verkehr anzupassen. Sie argumentieren, dass sich – insbesondere in Zugangsnetzen – die Verarbeitungsanforderungen in den Vermittlungsknoten im Laufe des Tages ändern. Während z. B. tagsüber ein Großteil des Verkehrs aufgrund von virtuellen privaten Netzen (*Virtual Private Network*, VPN) verschlüsselt werden muss, müssen abends verstärkt Multimediadaten transkodiert werden. Durch die Rekonfigurierbarkeit benötigen diese Systeme weniger Chipfläche, als wenn sie statisch über alle Verarbeitungsmodule verfügen müssten. Außerdem können auch im Nachhinein neue Funktionen realisiert werden.

Albrecht *et al.* stellen so einen FPGA-basierten, rekonfigurierbaren Coprozessor für Netzprozessoren vor [130]. Der Netzprozessor sendet Pakete zum Coprozessor, die eine komplexere Verarbeitung, wie z. B. Komprimierung, Verschlüsselung oder Deep Packet Inspection, erfordern. Der Coprozessor übernimmt diese Verarbeitung und sendet das Paket anschließend zum

Netzprozessor zurück. Ein Rekonfigurationsmodul kann die Funktionalität des Coprozessors zur Laufzeit entsprechend des aktuellen Verkehrs anpassen, indem es durch teilweise Rekonfiguration einzelne Verarbeitungsmodule austauscht.

Des Weiteren schlägt Kachris eine Architektur eines FPGA-basierten, rekonfigurierbaren Netzprozessors für Zugangs- und Firmennetze vor [131]. Die Architektur besteht aus einem Pool von Soft-Core-Prozessoren und mehreren Coprozessoren. Ein Rekonfigurationsmodul kann entsprechend der aktuellen Verkehrsverteilung und den benötigten Verarbeitungsfunktionen die Anzahl der Prozessorkerne variieren sowie aus einer Bibliothek unterschiedlicher Coprozessoren auswählen. Zusätzlich verfügt das System über ein rekonfigurierbares Warteschlangen-Verwaltungsmodul sowie einen konfigurierbaren Transaktionsspeicher.

Spezialisierung und Einschränkung der Flexibilität

Eine weitere Möglichkeit, einen hohen Durchsatz in einem Paketverarbeitungssystem zu erreichen, ist, die Flexibilität des Systems einzuschränken und damit das System stärker auf die benötigte Funktionalität zu spezialisieren.

Karras arbeitet an einer Pipeline-basierten Architektur für ein Paketverarbeitungssystem für Carrier-Grade 100 Gbit/s-Ethernet-Netze [132, 133]. Um den benötigten Durchsatz zu erreichen sollen die Verarbeitungsblöcke nur über eine begrenzte Programmierbarkeit verfügen. Diese soll ausreichen, um das System an Änderungen in den entsprechenden Standards für MPLS-TP und PBB-TE anpassen zu können. Die Verarbeitung ist dabei auf Schicht 2 bzw. 2,5 (MPLS) begrenzt.

Die Architektur basiert auf einer Pipeline von Subfunktionsblöcken, welche Funktionen wie das Auslesen von Header-Feldern, das Ermitteln von Weiterleitungsinformationen sowie das Einfügen, Löschen oder Ändern von Header-Feldern realisieren. Diese Blöcke sind in geringem Umfang programmierbar. Um den geforderten Durchsatz von 100 Gbit/s erreichen zu können, ist vorgesehen, ein Paket pro Taktzyklus zu verarbeiten, was auch innerhalb der Subfunktionsblöcke eine Pipeline-Struktur erfordert. Außerdem soll nur der Paket-Header durch die Pipeline transportiert werden, während der Nutzdatenbereich außerhalb der Pipeline gepuffert wird.

Ein weiteres Beispiel für eine Architektur mit eingeschränkter Flexibilität ist die Paketverarbeitungsarchitektur von Lai [134]. Diese besteht – in Anlehnung an herkömmliche ASIC-Architekturen – aus einer Vielzahl sequentieller Netzwerke, die jedoch programmierbar sind. Dies wird durch den Einsatz äußerst einfacher RISC-Prozessoren realisiert, die sowohl Verarbeitungs- als auch Steueraufgaben übernehmen. Die Interaktion mit anderen Prozessoren und Modulen erfolgt über spezielle Ein- und Ausgabe-Register.

2.5.2 Hohe Flexibilität

Architekturen für eine flexible Paketverarbeitung basieren entweder auf Prozessorkernen oder auf programmierbarer Logik. Die Flexibilität kann hierbei sowohl auf architektureller bzw. modularer Ebene als auch auf feingranularer Ebene zu finden sein.

Prozessorbasierte Systeme

Die Flexibilität prozessorbasierter Systeme ist hauptsächlich in der Programmierbarkeit ihrer Prozessoren zu finden. Das Programmiermodell und der Befehlssatz sind dabei i. d. R. fest von der Prozessorarchitektur vorgegeben. Zusätzliche Flexibilität kann durch einen rekonfigurierbaren Befehlssatz erreicht werden [135]. Des Weiteren gibt es diverse Architektur-unabhängige Arbeiten, wie die Programmierung von Netzprozessoren vereinfacht werden kann. Hier gibt es Forschungsaktivitäten mit dem Ziel spezielle Betriebssysteme, Paketverarbeitungsprogrammiersprachen und Compiler für Netzprozessoren zu entwickeln [136–139]. Beispielsweise beschreibt Kohler in [137] die Sprache *Click* zur effizienten Beschreibung von Paketverarbeitungsabläufen in Software-basierten Routern. Shah [138] erweitert diese Sprache für den Einsatz in Netzprozessoren, insbesondere für den Intel Netzprozessor IXP1200.

Zusätzlich gibt es auch einige Beispiele, Flexibilität auf modularer Ebene zu integrieren. Die IXP2xxx Netzprozessoren von Intel bzw. Netronome [88, 89] erlauben die logische Anordnung der Prozessorkerne beliebig anzupassen (vgl. Abschnitt 2.4.1). Bei Kachris' rekonfigurierbarer Netzprozessorarchitektur [131] ist es möglich, die Anzahl der verwendeten Prozessorkerne zu ändern, und sowohl bei ihm als auch bei Albrecht [130] kann man die verwendeten Coprozessormodule beliebig austauschen. Diese Möglichkeit wurde zur Entlastung der Prozessoren und zur Steigerung des Durchsatzes vorgesehen (vgl. Abschnitt 2.5.1), jedoch steigert sie auch die Flexibilität.

Systeme basierend auf programmierbarer Logik

Systeme basierend auf programmierbarer Logik sind prinzipiell sehr gut geeignet, ein sehr hohes Maß an Flexibilität zu ermöglichen. Der Grund hierfür ist, dass sie Adaptivität nicht nur auf *funktionaler* bzw. algorithmischer Ebene ermöglichen, sondern auch auf mikro- und makro*architektureller* Ebene.

a. FPGA-basierte Systeme

In den letzten Jahren wurden vermehrt Paketverarbeitungseinheiten auf FPGA-Basis entworfen und implementiert. Mutter [140] berichtet z. B. von einem Aggregationsrandknoten für ein zukünftiges paketvermittelndes, verbindungsorientiertes, so genanntes *Frame Switching*-Netz. Die Verarbeitung ist durchgängig als Pipeline realisiert und erreicht eine Paketrate von 10 MP/s.

Chen und Turner entwarfen eine Pipeline-basierte Verarbeitungseinheit für Vermittlungsknoten in einem *Optical Burst Switching*-Vermittlungsknoten [141]. Die Einheit verarbeitet jeden zweiten Takt ein neues Paket, so dass die FPGA-Implementierung einen Durchsatz von 50 MP/s erzielt.

b. Plattformen für FPGA-basierte Systeme

Die vielen Anpassungsmöglichkeiten auf architektureller als auch auf funktionaler Ebene sind

auch ein Nachteil von FPGAs gegenüber Netzprozessoren. Zum Entwurf einer Paketverarbeitungsfunktion auf einem FPGA sind nämlich vielerlei Kenntnisse erforderlich: Kenntnisse im Hardware-Entwurf, im Architektur-Entwurf und im Entwurf und in der Realisierung der eigentlichen Funktionalität bzw. eines Algorithmus. Wie jedoch in Abschnitt 2.2.3 dargestellt, ist auch die Einfachheit, Änderungen zu integrieren, wesentlicher Bestandteil der Flexibilität eines Systems und somit eine wichtige Anforderung an neue Paketverarbeitungssysteme. Daher gibt es verschiedene Forschungsarbeiten die versuchen, Plattformen zu entwickeln, die den Entwurf FPGA-basierter Paketverarbeitungssysteme erleichtern.

Der *Field Programmable Port Extender* [142] des *Washington University Gigabit Router* ermöglicht es Forschern, Paketverarbeitungsmodule für Line Cards eines Hochgeschwindigkeits-Routers zu realisieren und im Betrieb zu testen. Die Architektur basiert auf rekonfigurierbaren Modulen mit definierten Paket- und Speicherschnittstellen und vereinfacht so die prototypische Realisierung neuer Paketverarbeitungsfunktionen. Beispielsweise implementierte auf dieser Plattform Taylor *et al.* [143] eine prototypische Realisierung des *Tree Bitmap* IP-Adresssuchalgorithmus [59].

Die Universelle Hardware-Plattform (UHP) [144] des Institut für Kommunikationsnetze und Rechnersysteme der Universität Stuttgart, dem Institut des Autors, ist eine hierarchisch aufgebaute Plattform bestehend aus miteinander kombinierbaren Platinen mit verschiedenen FPGAs, Speichermodulen und Kommunikationsschnittstellen. Sie wird für die prototypische Realisierung und den Test von zukünftigen Vermittlungsknotenfunktionen (z. B. [11, 140, 145]) sowie anderer digitaler Systeme eingesetzt. Eine genauere Beschreibung dieser Plattform folgt in Abschnitt 4.2.2, da auf dieser Plattform auch eine prototypische Implementierung der im folgenden Kapitel vorgestellten Paketverarbeitungsarchitektur aufgebaut wurde.

NetFPGA [146, 147] ist eine relativ neue und weltweit sehr erfolgreich eingesetzte FPGA-basierte Plattform für Forschung und Lehre der *Stanford University*. Mit ihr kann man auf schnelle und einfache Weise neue Paketverarbeitungsfunktionen implementieren und testen. Die Plattform besteht aus der NetFPGA-Hardware, Bibliotheken mit den Hardware-Beschreibungen der Verarbeitungsmodule sowie Software für Steueraufgaben auf einem angeschlossenen PC. Die NetFPGA-Hardware ist eine speziell gefertigte Platine mit einem FPGA, vier 1 Gbit/s-Ethernet-Schnittstellen, verschiedenen Speichermodulen sowie einer PCI-Schnittstelle (*Peripheral Component Interconnect*). Die prinzipielle Architektur für die Paketverarbeitung im FPGA basiert auf einer Pipeline unterschiedlicher Verarbeitungsmodule. Paketdaten und Metadaten werden über 64 bit breite FIFO-Puffer von einem Modul zum nächsten übermittelt. Nachdem die Plattform bisher für eine Datenrate von 4 Gbit/s ausgelegt ist, wird zurzeit an einer Version mit vier 10 Gbit/s Schnittstellen und somit bis zu 40 Gbit/s Durchsatz gearbeitet. Es gibt weltweit eine Vielzahl von prototypischen Implementierungen unter Verwendung dieser Plattform, u. a. existiert eine Implementierung der RCP-Verarbeitung in Routern [148] sowie eine Realisierung eines *OpenFlow*-Routers [149].

c. Entwurfsmethodik für FPGA-basierte Systeme

Zur Fragestellung, wie FPGAs von Nicht-Hardware-Experten und mit niedrigem Entwicklungsaufwand zur Realisierung von anpassbaren Paketverarbeitungssystemen verwendet werden können, veröffentlichte Brebner verschiedene Forschungsarbeiten. In [150] weist er auf die „Hyper-

Programmierbarkeit“ von FPGAs auf Software-Ebene (durch Prozessorkerne auf dem FPGA), auf Register-Transfer-Ebene und auf Gatter-Ebene (vgl. auch Unterkapitel 3.1) und somit auf ihre Veränderbarkeit in funktionaler als auch in architektureller Hinsicht hin. Er empfiehlt, alle auf FPGAs verwendbaren Elemente effizient zu nutzen, also außer den konfigurierbaren Lookup-Tabellen auch z. B. Speichermodule, Multiplizierer oder integrierte Prozessorkerne.

Brebner schlägt eine prinzipielle, logische Architektur vor, bestehend aus unabhängigen Modulen, die parallel auf den Paketdaten arbeiten können [150]. Diese Module können über anwendungsspezifische Hochsprachen beschrieben werden oder auch über Bibliotheken eingebunden werden. In [151] stellen Keller und Brebner eine XML-basierte Sprache zur Beschreibung solcher Verarbeitungsmodule vor und präsentieren die Implementierung eines RPC (*Remote Procedure Call*)-Servers inklusive Ethernet-, IP- und UDP-Verarbeitung. Zusammen mit Attig untersucht Brebner die automatische Erzeugung verschieden granularer Paketverarbeitungs-Pipelines [152] zur Auswahl einer optimalen Pipeline hinsichtlich Durchsatz, Latenz und Ressourcenverbrauch. Des Weiteren veröffentlichte er zusammen mit Kulkarni [153] eine Umsetzung der Paketverarbeitungs-Beschreibungssprache Click [137] für FPGA-basierte Systeme. Damit können Datenfluss-Graphen bestehend aus elementaren Verarbeitungsfunktionen in ein entsprechendes Verarbeitungssystem übersetzt werden.

Aufbauend auf diesen Arbeiten entwickelten Brebner *et al.* eine Hochsprache G für die einfache Beschreibung von Paketverarbeitungsfunktionen [154]. Ein Compiler setzt diese Beschreibung unter Berücksichtigung der geforderten Leistungsanforderungen dann in eine VHDL-Beschreibung einer Paketverarbeitungs-Pipeline um. Die erzeugten Komponenten können modular zusammengeschaltet werden, um so größere Paketverarbeitungssysteme zu erzeugen. Ein beispielhaft so entwickelter MPLS-Router unterstützte Datenraten oberhalb 40 Gbit/s, wobei der Ressourcenbedarf etwa doppelt so hoch war wie ein entsprechendes von Hand entworfenes System.

Schelle [155] veröffentlichte eine Click-ähnliche Skriptsprache zur Beschreibung von Paketverarbeitungsfunktionen zur Umsetzung in ein FPGA-basiertes System. Die Verarbeitung im FPGA verläuft dabei in hohem Maße spekulativ, wobei die Auswirkungen fälschlich ausgeführter Funktionen im Nachhinein wieder rückgängig gemacht werden können. Schelle berichtet von einem so implementierten System mit 1 Gbit/s Durchsatz.

Ein Beschreibungsmodell zur Synthese von Hochgeschwindigkeits-Paketverarbeitungs-Pipelines beschreibt Soviani in [156, 157]. Der Entwickler beschreibt die Verarbeitungsfunktionen in Form eines so genannten *Packet Editing Graph*, einem azyklischen, gerichteten Graph. Funktionen können direkt auf beliebige Bytes des Paketkopfs zugreifen. Die Umsetzung in eine Hochleistungs-Pipeline mit einem Durchsatz von einem Wort pro Takt erfolgt anschließend automatisiert. Soviani berichtet von einem so implementierten System mit einem 128 bit breiten Datenpfad und einem Durchsatz von 40 Gbit/s.

d. FPGA-basierter Netzprozessor

Ravindran *et al.* zeigen, wie Multi-Prozessor-Paketverarbeitungssysteme auf Basis von FPGAs realisiert werden können [158]. Sie argumentieren, dass FPGA-basierte Prozessorsysteme zwar einen niedrigeren Durchsatz als optimierte Netzprozessoren erreichen und i. d. R. auch etwas

teurer in der Anschaffung sind, jedoch keine teuren einmaligen Entwicklungs- und Herstellungskosten aufweisen. Somit können z. B. schnell neue Prozessor- oder Coprozessor-Funktionsblöcke integriert werden sowie unterschiedliche Prozessorkonfigurationen getestet werden.

Die Autoren realisierten einen Router mit zwei 1 Gbit/s-Ethernet-Ports. Die Prozessoren führen die komplette Fast-Path-IP-Verarbeitung Software-gesteuert durch, wobei die IP-Adresssuche über einen Multibit-Suchbaum mittels bis zu sechs Speicherzugriffen erfolgt. Die Weiterleitungstabelle ist dabei auf dem FPGA mit einer Kapazität von etwa 5000 Einträgen realisiert. Eine optimierte Konfiguration bestehend aus zehn Prozessoren erreicht einen Durchsatz von 1,9 Gbit/s bei Betrachtung von Minimalpaketen.

2.5.3 Schlussfolgerungen

Ziel der vorgestellten Forschungsarbeiten ist, Paketverarbeitungssysteme einerseits schneller und andererseits flexibler zu machen.

Parallelisierung ist eine häufig gewählte und gute Möglichkeit den Durchsatz eines Systems zu erhöhen. Im ersten Abschnitt dieses Unterkapitels wurden verschiedene Arbeiten vorgestellt, bei denen Prozessorsysteme mit bis zu 80 Prozessoren für die Paketverarbeitung parallel eingesetzt wurden. Dieser Ansatz skaliert sehr gut, solange die Verarbeitung unabhängig von anderen Prozessorkernen erfolgen kann. Daher werden hier sehr hohe Paketraten berichtet. Falls sich diese vielen Verarbeitungseinheiten jedoch einen gemeinsamen Speicher oder Coprozessor teilen müssen, da deren Zustand aus Konsistenzgründen nicht ebenfalls vervielfacht werden kann, kommt es zu massiven Problemen. Der Zugriff auf die geteilten Ressourcen muss synchronisiert werden und der Vorteil der hohen Parallelisierung verschwindet (vgl. [5]). Ebenso werden sehr komplexe Kommunikationsinfrastrukturen zwischen den Verarbeitungseinheiten untereinander und zu den verwendeten Ressourcen sowie zu den Ein- und Ausgängen benötigt.

Eine Erhöhung des Verarbeitungsdurchsatzes entsprechend Amdahls Gesetz, den häufigen Anwendungsfall zu beschleunigen, ist eine andere mehrmals gewählte Methode, die Verarbeitung von Paketverarbeitungssystemen zu beschleunigen. So bringt der Einsatz eines Ergebnis-Caches sowie spezialisierter Coprozessoren im statistischen Mittel eine Erhöhung des Durchsatzes. Allerdings sind solche Maßnahmen nicht geeignet, einen bestimmten Verarbeitungsdurchsatz zu garantieren, da im ungünstigsten Fall, gerade nicht diese Verarbeitungsfunktionen benötigt werden, die beschleunigt wurden. Systeme ohne garantierten Durchsatz sind jedoch nicht für den Einsatz in Hochgeschwindigkeitsvermittlungsknoten in Metro- und Kernnetzen geeignet.

Schließlich kann die Verarbeitungsleistung durch Spezialisierung erhöht werden, indem beispielsweise nur spezielle Protokolle bzw. nur eine feste Header-Länge verarbeitet werden kann. Solche Einschränkungen laufen allerdings dem zweiten Ziel, der Erhöhung der Flexibilität, entgegen.

Flexibilität wurde in Abschnitt 2.2.3 als einfache und schnelle Anpassbarkeit definiert. Sie besitzt also die Komponenten Anpassbarkeit und Einfachheit, die erfüllt werden müssen.

Die Anpassbarkeit eines (Netz-)Prozessorsystems beschränkt sich meist auf dessen Programmierbarkeit mittels des gegebenen Befehlssatzes. Teilweise kann auch das System selbst ange-

passt werden, z. B. durch Änderung der logischen Anordnung von Modulen, durch das Austauschen von Modulen oder das Anpassen des Befehlssatzes. FPGAs bieten Anpassbarkeit auf vielen unterschiedlichen Ebenen: Einerseits in funktionaler Hinsicht durch Änderung der realisierten Funktionen auf Gatter-, Register-Transfer- oder Software-Ebene, und andererseits in architektureller Hinsicht durch Anpassungen an der realisierten Verarbeitungsarchitektur. So können hier neue Module hinzugefügt werden, bestimmte Verarbeitungsschritte parallelisiert werden und Funktionen überlappend als Pipeline realisiert werden.

Allerdings ist es nicht einfach, die gerade beschriebene Vielfalt an Anpassungsmöglichkeiten beim Entwurf digitaler FPGA-basierter Systeme zu nutzen. Um die Realisierung von Paketverarbeitungsfunktionen in Hardware zu fördern, wurden daher verschiedene Plattformen realisiert, welche prototypische Implementierungen erleichtern. Außerdem gibt es mehrere Forschungsarbeiten mit dem Ziel, eine Entwurfssprache zu entwickeln, mit der die Realisierung von Paketverarbeitungsfunktionen auf FPGAs erleichtern wird. Ähnliche Aktivitäten gibt es auch für Netzprozessor-Systeme, da auch hier die effiziente Nutzung aller Hardware-Ressourcen eines Netzprozessors häufig Schwierigkeiten bereitet.

Wie die Vorteile von FPGAs und Netzprozessoren kombiniert werden könnten, wurde zum Abschluss des letzten Abschnitts gezeigt. Dort wurde eine Arbeit vorgestellt, die mehrere Prozessorkerne auf einem FPGA integriert, um so Paketverarbeitungsfunktionen in Software realisieren zu können. Der vorgestellte Prototyp erreicht jedoch nur eine vergleichsweise geringe Paketrate.

Die neue Paketverarbeitungsarchitektur, die im folgenden Kapitel vorgestellt wird, kombiniert ebenfalls die Vorteile von FPGA-basierten Paketverarbeitungssystemen mit den Vorteilen von Software-programmierten Netzprozessoren. Die neue Architektur erlaubt die Kombination verschiedenster Verarbeitungsmodule, die sowohl in Software implementiert als auch in einer Hardware-Beschreibungssprache realisiert sein können. Zusätzlich garantiert die Architektur einen sehr hohen Durchsatz durch die Verwendung einer extrem breiten Pipeline-Struktur, womit die Nachteile der vorgestellten Pool-Anordnungen umgangen werden konnten.

2.6 Zusammenfassung

Dieses Kapitel führte in die Grundlagen paketvermittelnder Kommunikationsnetze ein und beschrieb die Anforderungen an zukünftige Paketverarbeitungssysteme in Vermittlungsknoten am Rand und innerhalb von Metro- und Kernnetzen. Solche Systeme müssen neben der geforderten Funktionalität einerseits einen sehr hohen Paketdurchsatz unterstützen und andererseits eine hohe Flexibilität für zukünftige Änderungen aufweisen.

Netzprozessoren und FPGAs sind Bauelemente, welche prinzipiell in der Lage sind, diese beiden gegensätzlichen Anforderungen zu realisieren. Die Analyse aktueller kommerzieller sowie akademischer Systeme zur Paketverarbeitung zeigte, wie durch massive Parallelisierung hohe Durchsätze erreicht werden können und dass Flexibilität sowohl durch Software in Prozessorsystemen als auch durch programmierbare Logik auf FPGAs erreicht werden kann. Insbesondere scheint die Parallelisierung in Form einer Pipeline Vorteile gegenüber einer Pool-Anordnung aufzuweisen. Diese unterschiedlichen Arten der Parallelisierung werden im nächs-

ten Kapitel nochmals tiefergehend untersucht. Des Weiteren wurde klar, dass Funktionen in Software zwar einfach und schnell auf einem Netzprozessor implementiert werden können, sehr hohe Durchsätze jedoch teilweise aufgrund ungeeigneter Befehlssätze oder Verarbeitungsarchitekturen nicht erreicht werden können. FPGA-basierte Systeme können andererseits sowohl in funktionaler als auch in architektureller Hinsicht entworfen und nachträglich angepasst werden. Diese unterschiedlichen Entwurfsebenen für die Realisierung von Paketverarbeitungsfunktionen sowie deren Vor- und Nachteile werden ebenfalls im folgenden Kapitel nochmals tiefergehend analysiert. Darauf aufbauend wird dann eine neue Paketverarbeitungsarchitektur für zukünftige Vermittlungsknoten vorgestellt, die einen maximalen Durchsatz und maximale Flexibilität bietet.

3 Eine neue Architektur für schnelle und flexible Paketverarbeitung

Vermittlungsknoten in Hochgeschwindigkeitskommunikationsnetzen benötigen sehr schnelle und flexible Paketverarbeitungssysteme. Kapitel 2 führte die Randbedingungen für solche Systeme ein und diskutierte bereits kommerziell erhältliche Systeme sowie aktuelle Forschungsarbeiten zu diesem Thema. Dabei zeigte sich, dass hierfür sowohl FPGA-basierte Systeme als auch Netzprozessoren geeignete Plattformen sind. Bei ersteren wird die Funktionalität auf Gatter- und Register-Transfer-Ebene mittels Hardware-Beschreibungssprachen implementiert, während bei letzteren die Funktionalität durch Software realisiert wird. Um einen möglichst hohen Durchsatz zu erreichen, wenden diese Systeme verschiedene Arten der Parallelschaltung ihrer Verarbeitungseinheiten an.

In diesem Kapitel wird eine neue Architektur für ein schnelles und flexibles Paketverarbeitungssystem vorgestellt. Dieses erlaubt die Verwendung von Entwurfsverfahren auf unterschiedlichen Abstraktionsebenen und basiert auf einer Pipeline von Verarbeitungsmodulen mit sehr hohem Durchsatz, ohne die Verwendung anderer Arten der Parallelisierung auszuschließen.

Unterkapitel 3.1 diskutiert zunächst den Entwurf von Verarbeitungsfunktionen auf den verschiedenen Abstraktionsebenen digitaler Systeme. Unterkapitel 3.2 formuliert daraufhin die Ziele, welche für den Entwurf der neuen Architektur maßgeblich waren. Daran anschließend werden in Unterkapitel 3.3 die verschiedenen Arten der Parallelisierung genauer analysiert. Diese einleitenden Abschnitte führen schließlich zu Unterkapitel 3.4 hin, welche den grundlegenden Aufbau der neuen Architektur vorstellen. Unterkapitel 3.5 erklärt die unterschiedlichen Arten von Verarbeitungsmodulen, welche in der neuen Architektur verwendet werden können, und Unterkapitel 3.6 geht darauf ein, wie ein System mit der vorgestellten Architektur schließlich verwendet wird, um flexible Hochgeschwindigkeitspaketverarbeitung zu realisieren. Das Kapitel schließt mit zwei Implementierungsbeispielen von Verarbeitungsmodulen in Unterkapitel 3.7 und einer Zusammenfassung in Unterkapitel 3.8.

3.1 Entwurfsebenen

Der Entwurf und die Realisierung von Paketverarbeitungsfunktionen bzw. allgemein von Funktionen in digitalen Systemen kann auf verschiedenen Ebenen der Abstraktion stattfinden. Der Entwurf von Hardware-Modulen kann auf Transistorebene, Gatterebene, Register-Transfer-

Ebene oder Verhaltensebene erfolgen. Ebenso kann man Software auf Mikrocode-Ebene, Assembler-Ebene oder auf Hochsprachenebene entwerfen und implementieren. Jede Ebene verbirgt dabei jeweils komplexe Details der darunter liegenden Ebene, die man in dem jeweiligen Entwurfskontext nicht beachten muss. Dieser Prozess des Heraussonderns des Wesentlichen und des Verbergens von Unwichtigem heißt **Abstraktion** und ist ein wesentlicher und notwendiger Schritt für den Entwurf komplexer Systeme. Allerdings kann sich eine zu hohe bzw. falsche oder unpassende Abstraktion auch negativ auswirken, wenn durch das Verbergen gewisser Details der Entwurf komplizierter oder sogar unmöglich wird.

In den folgenden Abschnitten werden zunächst die üblichen Abstraktionsebenen beim Hardware-Entwurf und anschließend die beim Software-Entwurf vorgestellt. Daraufhin werden die für die Paketverarbeitung relevanten Entwurfsebenen hinsichtlich Leistungsfähigkeit, Einfachheit und Anpassbarkeit diskutiert.

3.1.1 Hardware

Der Entwurf digitaler Hardware lässt sich auf vier unterschiedlichen Abstraktionsebenen durchführen. Diese werden im Folgenden vorgestellt. Insbesondere die Register-Transfer-Ebene wird heutzutage für die Realisierung von Paketverarbeitungsfunktionen in Hardware verwendet.

Transistorebene

Die Transistorebene ist die Abstraktionsebene für den Entwurf von hochoptimierten, integrierten Schaltungen (vgl. Abschnitt 2.3.2). Elemente dieser Ebene sind im Wesentlichen Transistoren sowie andere elektrische Bauelemente wie Dioden, Widerstände, Spulen oder Kondensatoren. Letztere können teilweise auch durch eine entsprechende Beschaltung von Transistoren realisiert werden. Das Verhalten der Schaltung wird durch die physikalischen Größen Spannung und Strom beschrieben.

Der Entwurfsprozess gliedert sich in den Schaltungsentwurf, die Umsetzung dieser Schaltung auf einem Chip sowie die Simulation der entworfenen Schaltung zur Validierung und Evaluierung. Der Schaltungsentwurf erfolgt meist grafisch in speziellen Software-Entwicklungsumgebungen, dabei werden Schaltungen mit Hilfe spezieller Schaltzeichen für die verschiedenen Bauelemente entworfen. Nach Umsetzung der eingesetzten Schaltzeichen in konkret dimensionierte Bauelemente einer bestimmten Fertigungstechnologie müssen die Bauelemente auf der Chipfläche möglichst platzsparend und dennoch störungssicher platziert werden. Die Funktion, Dimensionierung und Leistungsfähigkeit der Schaltung kann zwischen diesen Schritten jeweils mit entsprechenden mathematischen Simulationsprogrammen überprüft werden.

Diese Abstraktionsebene ist geeignet, um z. B. schnelle Analog-/Digital-Wandler, Taktsynchronisationsschaltungen oder auch spezielle Speichermodule zu entwickeln. Für die Realisierung kompletter Paketverarbeitungsfunktionen ist diese Entwurfsebene jedoch zu detailliert und würde bei Weitem zu viel Zeit in Anspruch nehmen.

Gatterebene

Die Gatterebene abstrahiert von den physikalischen Größen Spannung und Strom und verwendet nur noch die binären Werte (binary digits, bits) '0' und '1' zur Beschreibung von Eingangs- und Ausgangssignalen.[1] Außerdem stellt sie dem Entwickler logische Gatter, wie NICHT-, UND-, ODER- oder EXKLUSIV-ODER-Gatter, sowie Flipflops als Entwurfselemente zur Verfügung. Dabei verbirgt sie deren tatsächliche Realisierung durch eine Transistorschaltung.

Diese Entwurfsebene kann zur Realisierung spezieller oder sehr schneller, digitaler Schaltungen verwendet werden, insbesondere solcher, welche auf Bitebene operieren. Beispiele sind spezielle Kodier- und Dekodierschaltungen (z. B. mittels rückgekoppelter Schieberegister) und Schaltungen zur Bitmusteranalyse oder -manipulation.

Der Entwurf solcher Schaltungen erfolgt entweder mit Hilfe grafischer Entwicklungswerkzeuge unter Verwendung der entsprechenden Gatter- und Flipflop-Symbole, oder sehr häufig mittels der Beschreibung durch Logikfunktionen und Wahrheitstabellen. Zur Beschreibung solcher Logikfunktionen verwendet man Hardware-Beschreibungssprachen wie VHDL oder Verilog. So genannte Synthese-Werkzeuge setzen die Beschreibung in eine Netzliste von Elementen um, die auf der zu verwendenden ASIC- bzw. FPGA-Technologie verfügbar sind. Bei FPGAs sind dies kleine Lookup-Tabellen und Flipflops, bei ASICs Standardgatter- bzw. -flipflopzellen. Diese werden anschließend, meist ebenfalls automatisiert, auf dem Chip bzw. dem FPGA platziert und miteinander verbunden (vgl. hierzu auch Abschnitt 2.3.4, letzter Unterabschnitt).

Register-Transfer-Ebene

Die Register-Transfer-Ebene (häufig mit RTL für *Register-Transfer Level* abgekürzt) ist die wichtigste Entwurfsebene für die Realisierung von Paketverarbeitungsfunktionen in Hardware. Auf dieser Abstraktionsebene wird nicht mehr auf einzelnen Bits operiert sondern auf Bitvektoren. Bitvektoren mit einer bestimmten, festen Bitanzahl werden auch als Bytes, Worte und Doppelworte bezeichnet. Außerdem wird den Bitvektoren in vielen Fällen auch eine Interpretation zugeordnet, so dass diese Zahlen, Buchstaben, Protokollfelder oder andere Aufzählungstypen (z. B. zur Realisierung von Automatenzuständen) repräsentieren. Die wichtigsten Elemente dieser Ebene sind Multiplexer und Datentore zur Steuerung des Datenflusses, Rechenwerke, Logikoperationswerke und Dekodierwerke zur Manipulation und Auswertung von Bitvektoren sowie Register und Speichermodule zur Speicherung dieser Bitvektoren.

Für den Entwurfsprozess verwendet man, wie bereits in Abschnitt 2.3.4 eingeführt, eine Hardware-Beschreibungssprache wie VHDL oder Verilog. Mit diesen Sprachen kann man beliebige, abstrakte Datentypen sowie Steuerstrukturen, wie die Prüfung von Bedingungen (*if, case*), Wiederholungen durch Schleifen (*for, while*) oder Funktionsaufrufe (*function*) verwenden. Zusätzlich können grafische Entwicklungsumgebungen eingesetzt werden, beispielsweise zur Verknüpfung atomarer Elemente oder auch größerer Komponenten, welche selbst auf Register-Transfer-Ebene entworfen wurden. Ebenso unterstützen Entwicklungsumgebungen auch den

[1] Zum Teil werden zusätzlich die Werte 'Z' für hochohmige Signale sowie 'L' und 'H' für Signale, die durch einen *Pull-Down*- bzw. *Pull-Up*-Widerstand auf '0' bzw. '1' gezogen werden, verwendet.

grafischen Entwurf endlicher Automaten. Die so erstellte HDL-Beschreibung setzt anschließend ein Synthese- sowie ein Place & Route-Werkzeug, wie beschrieben, für die Realisierung auf einem FPGA oder ASIC um.

Verhaltensebene

Die Verhaltensebene abstrahiert den Entwurf digitaler Systeme weiter von schaltungstechnischen Details und nun auch zunehmend von architekturellen Details. Somit kann sich der Entwickler auf das Verhalten bzw. die Funktion oder den zu realisierenden Algorithmus des Systems konzentrieren. Diese Entwurfsebene wird daher auch als algorithmische Ebene sowie als elektronische Systemebene (*electronic system level*) bezeichnet. Der Entwurf auf dieser Abstraktionsebene wird seit mehreren Jahrzehnten erforscht, setzt sich allerdings erst langsam z. B. bei der Realisierung von Videokodierungs-Chips durch.

Auf der Verhaltensebene beschreibt man den Algorithmus bzw. das Verhalten des zu entwerfenden Systems wie in einer Software-Programmierhochsprache. Hierfür werden Sprachen wie Standard ANSI C++ sowie *SystemC, SystemVerilog* oder *CatapultC* verwendet. Bei der Beschreibung muss zunächst nicht auf eine Realisierung in Hardware Rücksicht genommen werden. Anforderungen bezüglich des zu erreichenden Durchsatzes, der maximalen Latenz oder der zur Verfügung stehenden Ressourcen werden einem Synthese-Werkzeug separat als Randbedingungen übergeben. Dieses wandelt die abstrakte Verhaltensbeschreibung intern in einen Steuer- und Datenflussgraphen um, weist die benötigten Operationen entsprechenden Rechenwerken zu und setzt den Entwurf so schließlich in eine Hardware-Architektur auf Register-Transfer-Ebene um. Diese kann mit den oben beschriebenen RTL-Synthese-Werkzeugen für eine Realisierung auf einem FPGA oder ASIC weiterverarbeitet werden. [159, 160]

3.1.2 Software

Software ist eine Abfolge von Operationen, die von einer gegebenen Hardware-Infrastruktur ausgeführt werden. Die gegebene Hardware-Infrastruktur wird hierbei i. d. R. als Prozessor bezeichnet. Der Entwurf von Software kann ebenfalls auf verschiedenen Abstraktionsebenen durchgeführt werden. Diese werden im Folgenden kurz vorgestellt.

Mikrocode-Ebene

Die Mikrocode-Ebene ist die niederste Software-Abstraktionsebene. Sie basiert auf Mikrooperationen. Dies sind einfache Operationen und benötigen eine Taktperiode zur Ausführung. Die ausführenden Hardware-Elemente entsprechen i. d. R. denen der Register-Transfer-Ebene. Daher wird die Mikrocode-Ebene auch häufig als die der Register-Transfer-Ebene entsprechende Software-Abstraktionsebene bezeichnet wird. Beispiele für Mikrooperationen sind `D0 + D1` $\rightarrow$ `AC` („Addiere die Werte aus Register D0 und D1 und schreibe das Ergebnis in das Register AC“) und `AC` $\rightarrow$ `D3` („Schreibe den Wert aus Register AC in das Register D3“). Da Mikrooperationen häufig nur einen Teil der zur Verfügung stehenden Hardware-Infrastruktur nutzen, ist es möglich, voneinander unabhängige Operationen parallel ausführen zu lassen.

Ein Programm aus Mikrooperationen setzt eine Mikrocode-Assembler-Software schließlich in so genannten Mikrocode um. Diesen verwendet ein Steuerwerk zur taktgenauen Ansteuerung der verschiedenen Elemente der Hardware-Infrastruktur. Hierbei kann man unterscheiden zwischen horizontalem Mikrocode, der für jede Steuerleitung direkt ein Bit enthält, und vertikalem Mikrocode, der zur Erzeugung der Steuersignale zunächst dekodiert werden muss.

Die einzelnen Assembler-Befehle von CISC-Prozessoren werden aus Abfolgen von teilweise mehreren parallelen Mikrooperationen realisiert. Heute wird Mikrocode zur Steuerung von Spezialprozessoren, wie rekonfigurierbaren Prozessoren [161] und Coprozessoren [162], verwendet sowie zur Anpassung und Konfiguration von sonst unflexibler Spezial-Hardware [11, 134] oder ASICs [163].

Assembler-Ebene

Die Assembler-Ebene ist die Ebene, in der häufig die Programmierung von Netzprozessoren erfolgt. Diese Ebene abstrahiert von der verwendeten Hardware-Infrastruktur sowie, falls vorhanden, von der Mikrocode-Ebene. Auf dieser Ebene verwendet der Programmierer einen fest vorgegebenen Befehlssatz sowie ein so genanntes Programmier- und Speicher-Modell, das nur die für die Programmierung wesentlichen Elemente der Prozessor-Hardware enthält. Diese Elemente sind die Register des Registersatzes, der Befehlsfolgezähler, eventuell vorhandene Status-*Flags*, der adressierbare Speicher sowie darauf abgebildete Ein-/Ausgabegeräte. Bei RISC-Prozessoren ist diese Ebene die niederste Software-Ebene.

Der Befehlssatz enthält diverse elementare arithmetische und logische Operationen sowie Schiebeoperationen, Operationen zur Initialisierung von Registerwerten sowie zum Verschieben der Werte zwischen Registern, Lade- und Speicheroperationen sowie einfache Programmsteuerbefehle, wie bedingte und unbedingte Sprünge. Spezialprozessoren, wie Netzprozessoren oder Grafikprozessoren, enthalten außerdem für das jeweilige Anwendungsgebiet optimierte Befehle, wie Befehle zur Extraktion von Bitfeldern. Aufgrund des – insbesondere bei RISC-Prozessoren – sehr eingeschränkten Befehlssatzes müssen komplexe Operationen häufig durch eine Folge von einfachen Operationen emuliert werden. So müssen bei RISC-Prozessoren häufig Divisionen und andere komplexe arithmetische Operationen, das Kopieren von Speicherbereichen sowie Steuerstrukturen, wie Schleifen und Unterprogrammaufrufe, durch mehrere Befehle nachgebildet werden.

Die Programmierung auf Assembler-Ebene erfolgt mit Hilfe von so genannten *Mnemonics,* kurzen, symbolischen Anweisungen. Eine Assembler-Software setzt diese Sequenz von Mnemonics in Maschinensprache um, d. h. in binär codierten Code (so genannten *opcode, operation code*), der vom Steuer- bzw. Dekodierwerk des Prozessors interpretiert werden kann. Bei dieser Umsetzung überprüft die Assembler-Software, ob die Anweisungen syntaktisch korrekt sind, und berechnet ausgehend von vom Programmierer verwendeten Sprungmarken die erforderlichen relativen Sprungdistanzen sowie absoluten Sprungzieladressen.

Hochsprachen-Ebene

Die Hochsprachen-Ebene abstrahiert von der Prozessor-Hardware und dem Assembler-Befehlssatz und versucht die Programmierung stärker an die menschlichen Denkweisen anzupassen. In einer Hochsprache geschriebene Programme operieren auf Variablen von abstrakten Datentypen, wie ganzen Zahlen, Fließkommazahlen oder Aufzählungstypen, und auf Instanzen bzw. Objekten umfangreicher Datenstrukturen, wie Feldern, Verbünden oder Klassen. Die Ausführung kann durch komplexere Steuerstrukturen, wie Schleifen, mehrfache Verzweigungen sowie statische und dynamische Unterprogrammaufrufe beeinflusst werden. Zusätzlich erleichtert die Einbindung umfangreicher Bibliotheken mit Funktionen für bestimmte Anwendungsgebiete die Programmierung komplexer Software-Systeme.

Der Software-Entwurf in einer Hochsprache erfolgt i. d. R. in Textform, kann jedoch durch mächtige Entwicklungsumgebungen mit Syntax-Hervorhebungen, Textvervollständigungen und nebenläufiger Fehlererkennung und -hervorhebung erleichtert werden. Teilmodule sowie das Gesamtsystem können in *Debug*-Umgebungen getestet und auf Fehler untersucht werden. Ein *Compiler* (Übersetzungswerkzeug) setzt schließlich den Hochsprachen-Quellcode in Maschinensprache um. Hierfür muss er die verwendeten Variablen und Objekte in Speicherstrukturen umsetzen sowie die verwendeten Befehle und Steuerstrukturen mit Assembler-Befehlen nachbilden.

3.1.3 Diskussion

Abbildung 3.1 zeigt ein Gesamtbild der vorgestellten Entwurfsebenen. Für jede Abstraktionsebene sind die für den Entwurf zur Verfügung stehenden Elemente angegeben und die Entwurfsfreiheiten, die man bei der Verwendung dieser Elemente hat. Das Bild illustriert zudem, wie die Software-Entwurfsebenen eine Hardware-basierte Infrastruktur erfordern, und es verdeutlicht die vergleichbare Abstraktion der Hochsprachenebene auf Software-Seite und der Verhaltensebene auf Hardware-Seite.

In diesem Abschnitt sollen die verschiedenen Entwurfsebenen bezüglich ihrer Eigenschaften hinsichtlich Flexibilität und Leistungsfähigkeit untersucht werden. Die Diskussion wird hierbei sowohl den eigentlichen Entwurfsprozess auf den vorgestellten Abstraktionsebenen umfassen als auch die entsprechenden zu verwendenden Bauelemente, beim Hardware-Entwurf FPGAs und beim Software-Entwurf (Netz-)Prozessoren.

Wie in Unterkapitel 2.2 dargestellt, sind Flexibilität und Leistungsfähigkeit die wichtigsten nicht-funktionalen Anforderungen an die hier betrachteten Paketverarbeitungssysteme. Flexibilität wurde dort als einfache und schnelle Anpassbarkeit definiert. Damit werden zwei Eigenschaften gefordert: Zum einen *Einfachheit,* ein System zu ändern, und zum anderen *Anpassbarkeit*, also die Möglichkeit, ein System entsprechend jeglicher die Funktion oder die Leistung betreffender Anforderungen zu modifizieren. Außerdem bedeutet eine hohe *Leistungsfähigkeit,* dass ein System möglichst viele Funktionen in möglichst kurzer Zeit erfüllen kann.

Abbildung 3.1: Gesamtbild der Entwurfsebenen

Einfachheit

Abstraktion erleichtert im Allgemeinen den Entwurfsprozess. Sie blendet unwesentliche Details aus, sodass komplexe Systeme auf schnellere und einfachere Weise realisiert werden können. Anstatt z. B. auf Gatterebene eine Addierschaltung entwerfen zu müssen, abstrahiert die Register-Transfer-Ebene von diesen Implementierungsdetails und bietet ein Addierwerk als Basiselement an. Ebenso ist z. B. auf Assembler-Ebene die Umwandlung von als Zeichenkette abgelegten Zahlenwerten in binär codierte Zahlen sehr aufwändig, während dies in Hochsprachen meist durch Standardfunktionen realisiert wird.

Andererseits können durch die Abstraktion auch zu viele bzw. die falschen Details einer darunter liegenden Entwurfsschicht ausgeblendet werden. Somit kann es passieren, dass Funktionen, die auf einer niedereren Abstraktionsebene einfach möglich waren, auf einer höheren Ebene nicht mehr direkt möglich sind und umständlich umschrieben werden müssen. Beispielsweise sind komplexe Bitmanipulationen auf Gatterebene problemlos möglich, müssen auf Register-Transfer-Ebene sowie auf den Software-Ebenen jedoch häufig umständlich durch Sequenzen mehrerer Logikoperationen emuliert werden. Des Weiteren können allerdings auch wiederum höhere Entwurfsebenen solche Funktionen wieder direkt zur Verfügung stellen und diese dann transparent für den Entwickler emulieren.

FPGAs basieren, wie in Abschnitt 2.3.4 beschrieben, im Wesentlichen auf einem Netz von konfigurierbaren Abbildungstabellen zur Emulation von Gattern, Flipflops und konfigurierbaren Verbindungsleitungen dazwischen. Somit werden alle Funktionen letztlich auf Gatterebene realisiert. Durch die Verwendung von Hardware-Beschreibungssprachen wie VHDL oder Verilog können jedoch auch Funktionen auf höheren Abstraktionsebenen realisiert werden. So kann man zusätzlich zu Funktionen und Strukturen auf Gatterebene auch Module bestehend aus Elementen der Register-Transfer-Ebene entwerfen sowie mehrere Module zu größeren Systemen integrieren. Weiterhin unterstützen diese Sprachen auch abstrakte Datentypen sowie Steuerstrukturen zur Realisierung bedingter und wiederholter Funktionen. Somit sind mit VHDL oder Verilog auch abstrakte Strukturbeschreibungen sowie abstrakte, allerdings taktgenaue Verhaltensbeschreibungen möglich. Diese Möglichkeiten vereinfachen den Entwurf.

Prozessoren basieren auf einer Verarbeitungs-Hardware, die auf ihren (abstrakten und begrenzten) Befehlssatz optimiert wurde und nicht änderbar ist. Alle benötigten Funktionen müssen durch Kombination dieser Befehle realisiert werden. Der Entwurf von Mikrocode und Assembler-Programmen erfolgt sehr Hardware-nah, was aufgrund der Begrenztheit der zur Verfügung stehenden Befehle, Adressierungsarten und Register mühselig ist. Hochsprachen vereinfachen diesen Entwurf, indem sie von der Hardware des Prozessors abstrahieren. Hochsprachen bieten viele Datentypen, Funktionen und Steuerstrukturen an, wodurch auf einfache Weise komplexe Systeme implementiert werden können.

Die Vorteile des Entwurfs in einer Hochsprache – völlig unabhängig von der darunter liegenden Hardware – versucht man seit Langem auch beim Hardware-Entwurf zu nutzen. Der Entwurf auf Verhaltensebene sieht vor, die zu realisierenden Funktionen zunächst Takt- und Architekturunabhängig in einer Hochsprache zu beschreiben und zu testen. Erst nachfolgende Syntheseschritte mit entsprechenden Randbedingungen erzeugen schließlich eine Netzliste aus Gattern und Flipflops, die auf einem FPGA realisiert werden. Bisher hat sich diese Art des Hardware-Entwurfs jedoch noch kaum durchgesetzt.

Anpassbarkeit

Die Anpassbarkeit eines Systems auf einer bestimmten Entwurfsebene bestimmt sich aus den Möglichkeiten und Freiheiten, dieses System beliebig zu ändern, und aus den Einschränkungen und Abhängigkeiten, die vom Entwickler nicht verändert werden können. Ein umfangreicher Befehlssatz erhöht die Anpassbarkeit, während eine feste und somit begrenzte Anzahl von zur Verfügung stehenden Registern sowie deren feste Bitbreite die Anpassbarkeit einschränkt.

Da die höheren Abstraktionsebenen stets nur von den niedereren Ebenen Details ausblenden, kann die Anpassbarkeit nach oben nicht zunehmen. Schließlich kann jede Anpassung auf einer Abstraktionsebene auch auf den darunter liegenden Ebenen realisiert werden. Lediglich die Einfachheit, diese Anpassungen vorzunehmen, nimmt mit der Abstraktion i. d. R. zu (siehe Unterabschnitt *Einfachheit*). Somit ist die Anpassbarkeit beim Hardware-Entwurf auf Transistor- und Gatter-Ebene am höchsten und beim Software-Entwurf auf Mikrocode- und Assembler-Ebene. Hier können Anpassungen auf Detail-Ebene durchgeführt werden.

Wenn auf höheren Abstraktionsebenen (Detail-) Anpassungen vorgenommen werden sollen, müssen dabei häufig umständliche (und somit leistungsschwache oder ressourcenintensive) Emulationen in Kauf genommen werden. In Fällen schlecht durchdachter Abstraktion – bei denen, wie oben erwähnt, von zu vielen Details abstrahiert wird – kann es auch vorkommen, dass bestimmte Funktionen gar nicht realisiert werden können. Beispielsweise kann es vorkommen, dass der beschränkte Befehlssatz eines sehr einfachen Prozessors nur eine direkte und keine indirekte Adressierung des Speichers vorsieht. Dies bedeutet, es ist nicht möglich, eine zur Laufzeit berechnete Adresse zur Speicheradressierung zu verwenden. In so einem Fall kann dieses Verhalten auch durch Sequenzen anderer Befehle nicht nachgebildet werden – selbst wenn die Hardware dies durch richtiges Setzen der Steuerleitungen eigentlich könnte.

Beim Vergleich der Anpassbarkeit beim Hardware- und Software-Entwurf ist festzustellen, dass beim Hardware-Entwurf eine höhere Anpassbarkeit vorliegt. Während beim Software-Entwurf alle Änderungen durch die vorgegebene Hardware-Infrastruktur und deren Befehlssatz (auf Assembler- oder Mikrocode-Ebene) realisiert werden müssen, kann beim Hardware-Entwurf sowohl die Funktionalität von Modulen als auch deren interner Aufbau und Verschaltung bis hinauf zur Register-Transfer-Ebene angepasst werden.

Leistungsfähigkeit

Intuitiv vermutet man, dass eine höhere Abstraktionsebene zwar den Entwurf eines Systems vereinfacht und beschleunigt, dass man sich dies allerdings durch eine geringere Leistungsfähigkeit erkaufen muss. Diese Vermutung ist in den Fällen richtig, in denen spezielle Funktionen durch viele einzelne Operationen emuliert werden müssen, welche auf einer niedereren Entwurfsebene direkt hätten implementiert werden können. Beispiele sind, wie oben, Bitmanipulationen, welche auf Gatterebene sehr effizient und auf höheren Ebenen eher umständlich zu realisieren sind. Auch Operationen, die nicht durch mehrere Operationen emuliert werden müssen, können auf niedereren Ebenen häufig durch spezialisierte Befehle effizienter durchgeführt werden. Beispielsweise kann die Addition von 4 bit breit codierten Zahlenwerten zwar mit einem 32-bit-Addierwerk realisiert werden, allerdings ist ein auf Gatterebene implementierter 4-bit-Addierer

wesentlich ressourceneffizienter und schneller. Des Weiteren ist die Hardware-nahe Programmierung in Assembler häufig effizienter als ein von einem Compiler übersetzter Hochsprachen-Programmcode – doch es gibt auch gegenteilige Beispiele.

Allerdings muss die Frage nach der Leistungsfähigkeit beim Einsatz von Netzprozessoren und FPGAs differenzierter betrachtet werden. Die Hardware der Prozessorkerne von Netzprozessoren wurde für die Ausführung ihres Befehlssatzes, die Verwendung ihrer Register sowie die Kommunikation mit anderen Prozessorkernen, Coprozessoren und Speichermodulen optimiert. Diese Operationen können sie sehr schnell ausführen. Im Gegensatz dazu benötigen FPGAs für die Ausführung derselben Operationen aufgrund ihrer hohen Konfigurierbarkeit auf Gatter- und Flipflop-Ebene bis zu zehnmal so lange. Somit ist die höhere Entwurfsebene von Netzprozessoren so lange zu bevorzugen, wie zur Realisierung der auszuführenden Funktionen der optimierte Befehlssatz des Netzprozessors effizient genutzt werden kann. Diese hohe Leistungsfähigkeit bei der Ausführung dieser spezialisierten Befehle motivierte, wie in Abschnitt 2.3.5 bereits dargestellt, auch die ursprüngliche Entwicklung von Netzprozessoren. Falls Operationen vom Prozessor nicht bereitgestellt und daher emuliert werden müssen, bringt die hohe Konfigurierbarkeit von FPGAs Leistungsvorteile. FPGAs können jede beliebige Funktion auf Gatterebene atomar und damit sehr leistungsstark realisieren.

Die Anpassbarkeit auf architektureller Ebene beim Hardware-Entwurf auf FPGAs bringt auch Leistungsvorteile mit sich. Wiederum erbringt die Netzprozessorarchitektur höchste Leistungen, solange ihre Architektur für die geforderte Funktion passt. Anderenfalls muss jedoch häufig eine aufwändige und somit leistungsschwache Realisierung implementiert werden. Prinzipiell können alle Aspekte einer Netzprozessorarchitektur für spezielle Anwendungen einschränkend sein: Die Anzahl an Registern kann zu gering sein, die Anbindung an interne und externe Speichermodule und Coprozessoren kann unpassend ausgelegt sein oder die Kommunikationsinfrastruktur zwischen den Prozessorkernen kann für die benötigte Funktion ungeeignet sein. Auf FPGAs kann die Architektur der Verarbeitungseinheiten hingegen speziell an bestimmte Anforderungen angepasst werden.

Beispielhaft wird eine Funktion betrachtet, bei der aus Konsistenzgründen auf eine geteilte Ressource nur sequentiell von verschiedenen Verarbeitungseinheiten zugegriffen werden darf. Somit ist eine parallele Verarbeitung an dieser Stelle sinnlos und erfordert u. U. zusätzliche, aufwändige Synchronisationsmechanismen zwischen den Verarbeitungseinheiten, um das gewünschte Zugriffsmuster zu garantieren. Bei einer Verarbeitung auf einem FPGA kann für diese Funktion eine spezielle Einheit für den Zugriff auf die geteilte Ressource realisiert werden. Diese kann so das gewünschte Zugriffsmuster hocheffizient durch ihre Architektur garantieren. Andere unkritische Verarbeitungsschritte können für eine schnelle Verarbeitung davor und danach nach Belieben parallel durchgeführt werden. Bei Netzprozessoren ist die Hardware dagegen unveränderlich, so dass bei einer Verarbeitung in einem Prozessor-Pool Synchronisationsmechanismen zwischen den Einheiten unvermeidlich sind. In [5] zeigen wir, wie die Leistungsfähigkeit in so einem Fall (bei einer Quick-Start Router-Implementierung) stark durch die Netzprozessor-Architektur eingeschränkt wird, und eine vergleichende FPGA-Implementierung wesentlich höhere Verarbeitungsraten erreicht. Ein weiteres Beispiel ist die hochparallele Analyse von Paketnutzdaten bei der Deep Packet Inspection. Herkömmliche Netzprozessoren sind für so eine hochparallele Verarbeitung eines Pakets i. d. R. nicht ausgelegt und müssen die Ver-

arbeitung sequentiell realisieren. Eine Hardware-Implementierung auf einem FPGA kann die parallele Analyse andererseits problemlos und hoch-effizient realisieren.

Zusammenfassung

Dieses Unterkapitel gab einen kurzen Überblick über die verschiedenen Entwurfsebenen beim Hardware- und Software-Entwurf und diskutierte deren Eigenschaften bezüglich Einfachheit, Anpassbarkeit und Leistungsfähigkeit.

Eine höhere Abstraktionsebene ist einer niedereren vorzuziehen, da sie einen einfacheren und schnelleren Entwurf ermöglicht. Allerdings können Funktionen in einer höheren Abstraktionsebene oftmals weniger effizient realisiert werden. Dies ist dann der Fall, wenn bestimmte Funktionen in Software durch mehrere Befehle emuliert werden müssen und auf einer Hardwarenäheren Sprach-Ebene bzw. auf der Hardware-Ebene direkt implementiert werden können. Des Weiteren sind Anpassungen auf einer niedereren Entwurfsebene teilweise auch einfacher implementierbar. Insbesondere architekturelle Anpassungen sind nur auf den Hardware-Entwurfsebenen realisierbar. Andererseits erlaubt auch die Software-Realisierung von Funktionen auf der optimierten Prozessor-Hardware eine sehr hohe Verarbeitungsgeschwindigkeit, solange die optimierte Architektur und der optimierte Befehlssatz effizient ausgenutzt werden kann.

Aufgrund dieser Ausgeglichenheit der Vor- und Nachteile beim Entwurf auf den unterschiedlichen Hardware- und Software-Abstraktionsebenen ist eines der Ziele der im Folgenden vorgestellten Architektur, die Implementierung von Paketverarbeitungsfunktionen auf beliebigen Entwurfsebenen zu erlauben. Ein Paketverarbeitungssystem mit dieser Architektur soll sowohl Funktionen, die in Software implementiert wurden, als auch Funktionen, die in Hardware realisiert wurden, auf den zu verarbeitenden Paketen ausführen können.

3.2 Ziele

Der Entwurf der in dieser Dissertation vorgestellten Architektur zur Paketverarbeitung war durch das Bestreben bestimmt,

Leistungsfähigkeit und Flexibilität zu maximieren.

Entsprechend lassen sich die Ziele für den Entwurf der neuen Paketverarbeitungsarchitektur in Ziele hinsichtlich der Leistungsfähigkeit und Ziele hinsichtlich der Flexibilität unterteilen.

Das wichtigste Kriterium bei der Leistungsfähigkeit ist der Durchsatz. Die Architektur soll daher die folgenden den Durchsatz betreffenden Ziele erreichen:

- **Sehr hoher Durchsatz** – Ein Paketverarbeitungssystem dieser Architektur soll mit heutiger Technologie für den Einsatz auf einer 100 Gbit/s-Ethernet Line Card geeignet sein, d. h. es soll eine Paketrate von 150 MP/s unterstützen.

- **Deterministischer Durchsatz** – Ein Paketverarbeitungssystem dieser Architektur soll einen deterministischen Durchsatz aufweisen, so dass das System unter jeglichen Verkehrsbedingungen den geforderten Durchsatz garantieren kann.

Um maximale Flexibilität zur Anpassung an zukünftige Anforderungen zu gewährleisten, soll die Architektur folgende Ziele erreichen:

- **Modularität** – Ein Paketverarbeitungssystem dieser Architektur soll so aufgebaut sein, dass auf einfache Weise Module mit neuen Funktionen integriert sowie veraltete Module entfernt oder ausgetauscht werden können.

- **Freie Wahl der Entwurfsebene** – Die Funktionen eines Paketverarbeitungssystems dieser Architektur sollen auf einer beliebigen Hardware- oder Software-Entwurfsebene implementiert werden können, so dass die Abstraktionsebene entsprechend der jeweiligen Anforderungen ausgewählt werden kann.

Die in Unterkapitel 2.4 und Unterkapitel 2.5 vorgestellten Architekturen für schnelle und flexible Paketverarbeitungssysteme aus Wissenschaft und Technik erreichen jeweils nur einzelne dieser vier Ziele, jedoch nie alle zusammen. Insbesondere ist dem Autor keine Architektur bekannt, die eine freie Wahl der Entwurfsebene bei der Implementierung von Verarbeitungsfunktionen erlaubt und gleichzeitig den geforderten hohen Durchsatz garantieren kann.

Das nächste Unterkapitel untersucht und beschreibt nochmals tiefgehend die Parallelisierung als wichtigste Maßnahme für einen hohen Durchsatz, bevor in dem darauf folgenden Unterkapitel 3.4 der grundlegende Ansatz der neuen Architektur dargestellt wird, welche alle vier hier angeführten Ziele erreicht. Weitergehende Unterkapitel vertiefen daraufhin diese Beschreibung.

3.3 Parallelisierung

Eines der Entwurfsziele der neuen Paketverarbeitungsarchitektur ist ein hoher Durchsatz. Der Durchsatz, d. h. die Paketrate bzw. die Datenrate, ist umgekehrt proportional zur benötigten Verarbeitungszeit für ein Paket bzw. ein Byte. Allerdings kann diese Zeit aufgrund steigender Verarbeitungsanforderungen und begrenzter Verarbeitungsleistung einzelner Einheiten nicht beliebig verkürzt werden. Um dennoch den angestrebten hohen Durchsatz erreichen zu können, setzen alle aktuellen Paketverarbeitungssysteme auf verschiedene Arten der Parallelisierung.

Zwei gegensätzliche Arten der Parallelisierung werden dabei angewendet: Die parallele Verarbeitung in einem Pool von Verarbeitungseinheiten und die parallele Verarbeitung in einer Pipeline von Verarbeitungseinheiten. In Abschnitt 2.3.5 und Abschnitt 2.4.1 wurden diese beiden Konfigurationen bereits kurz im Zusammenhang mit Netzprozessoren vorgestellt und diskutiert. [130, 158, 164] diskutieren ebenfalls Vor- und Nachteile beider Parallelisierungskonzepte anhand konkreter Netzprozessorarchitekturen.

In diesem Kapitel werden beide Parallelisierungsarten wesentlich allgemeiner und grundsätzlicher beschrieben, verglichen und bewertet. Die Darstellung ist nicht auf Netzprozessoren begrenzt, sondern gilt allgemein, wobei aufgrund der hier betrachteten Themenstellung besonders

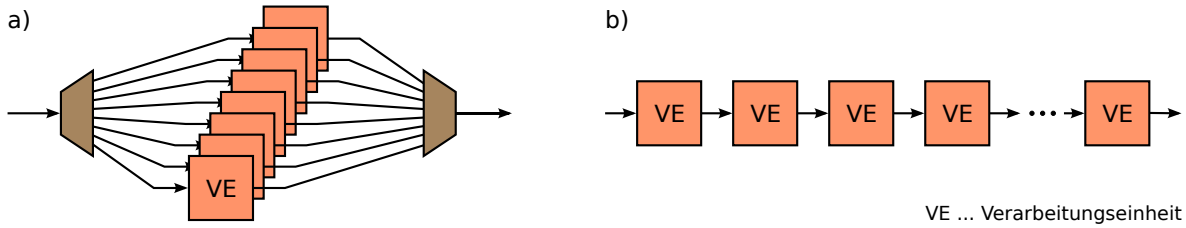

Abbildung 3.2: a) Pool-Konfiguration, b) Pipeline-Konfiguration

auf die Parallelisierung in Paketverarbeitungssystemen eingegangen wird. Hierbei wird allgemein von Verarbeitungseinheiten die Rede sein. Im Kontext von Netzprozessoren entsprechen diese den integrierten Prozessorkernen, im Kontext von FPGA-basierten Systemen den konfigurierten Verarbeitungsmodulen.

Im Folgenden werden zunächst nochmals der Aufbau und die Funktionsweise der beiden Parallelisierungsarten Pool und Pipeline dargestellt. Daran anschließend findet ein detaillierter Vergleich beider Konfigurationen bezüglich ihres Durchsatzes statt. Daraufhin wird der Aufwand für die Implementierung von Paketverarbeitungsfunktionen auf Systemen mit diesen Parallelisierungsarten diskutiert sowie der jeweilige Ressourcenbedarf. Schließlich wird im letzten Abschnitt dieses Unterkapitels noch auf kombinierte Konfigurationen sowie auf andere Ebenen der Parallelisierung in Paketverarbeitungssystemen eingegangen.

3.3.1 Pool und Pipeline

Pool und Pipeline bezeichnen die zwei grundsätzlichen Konfigurationsmöglichkeiten zur zeitlich überlappenden Durchführung einer gegebenen Menge von Verarbeitungsfunktionen mittels mehrerer, paralleler Verarbeitungseinheiten. Im Folgenden werden der Aufbau und der entsprechende Verarbeitungsablauf dieser beiden Konfigurationen allgemein beschrieben.

Abbildung 3.2 a) zeigt den prinzipiellen Aufbau der Pool-Konfiguration. Alle Verarbeitungseinheiten sind logisch nebeneinander angeordnet. Eine Verteileinheit weist jedes ankommende Paket (oder allgemein jeden Datenblock) einer Verarbeitungseinheit zu. Diese Einheit führt daraufhin die komplette Verarbeitung des Pakets bzw. der Daten durch. Ist die Verarbeitung abgeschlossen, übergibt die Verarbeitungseinheit das Paket wiederum einer zentralen Sammeleinheit (bzw. einer von mehreren). Diese führt die verarbeiteten Pakete von den parallelen Einheiten wieder zusammen, sortiert diese u. U. wieder und leitet sie an das nächste System (z. B. ein Verkehrsverwaltungsmodul) weiter. Teilweise wird anstatt Pool auch der Begriff *Cluster* verwendet.

Der prinzipielle Aufbau der Pipeline-Konfiguration ist in Abbildung 3.2 b) dargestellt. Hier sind alle Verarbeitungseinheiten logisch hintereinander angeordnet. Die Gesamtheit der durchzuführenden Verarbeitungsfunktionen ist in so viele Gruppen aufgeteilt, wie Verarbeitungseinheiten vorhanden sind. Die Durchführungsdauer jeder Gruppe von Verarbeitungsfunktionen ist optimaler Weise jeweils gleich lang. Ankommende Pakete (bzw. Datenblöcke) werden nun sequentiell von jeder der Verarbeitungseinheiten verarbeitet, wobei jede Verarbeitungseinheit jeweils eine Gruppe von Funktionen der Gesamtverarbeitung durchführt. Bei der Pipeline-Anordnung

sind keine Verteil- und Sammeleinheiten notwendig. Die komplett verarbeiteten Pakete werden nach der letzten Einheit zum nächsten System weitergeleitet.

Zusammenfassend kann man die parallele Verarbeitung in einem Pool von Verarbeitungseinheiten als unabhängige, überlappende Verarbeitung bezeichnen, da jede Einheit die Verarbeitung unabhängig von den anderen Einheiten durchführt. Die Verarbeitung in einer Pipeline ist hingegen eine kooperierende, überlappende Verarbeitung, da die Verarbeitung jeweils zusammen von allen Einheiten durchgeführt wird. Dennoch verarbeiten im Optimalfall beide Konfigurationen jeweils so viele Datenblöcke gleichzeitig wie Verarbeitungseinheiten zur Verfügung stehen.

3.3.2 Durchsatz

Pool- und Pipeline-Konfigurationen verwenden zwei völlig unterschiedliche Arten der Aufteilung der durchzuführenden Verarbeitung. Daher soll im Folgenden der Durchsatz beider Anordnungen untersucht werden. Zunächst wird dabei von einem vereinfachten Fall ausgegangen, anschließend werden diese vereinfachenden Randbedingungen schrittweise fallen gelassen und betrachtet, wie sich dies auf den Durchsatz der beiden Konfigurationen auswirkt.

Vereinfachter Fall

Für den Vergleich sollen vereinfachend zunächst folgende Annahmen gelten:

1. Die Pool- und die Pipeline-Anordnung bestehen aus jeweils N Verarbeitungseinheiten.
2. Alle Verarbeitungseinheiten seien identisch.
3. Alle Pakete erfahren dieselbe Verarbeitung.
4. Die Gesamtverarbeitungszeit pro Paket für alle durchzuführenden Verarbeitungsfunktionen zusammen sei T_{ges}.
5. Die Verarbeitung sei gleichmäßig auf alle N Pipeline-Verarbeitungseinheiten aufgeteilt. Die Verarbeitungsdauer $T_{S,i}$ je Verarbeitungseinheit $i \in \{1,...,N\}$ ist somit $T_{S,i} = T_{S,opt} = T_{ges}/N$
6. Das Verteilen und Einsammeln der Pakete in der Pool-Konfiguration und die Übergabe der Pakete von einer Einheit zur nächsten in der Pipeline-Konfiguration benötige keine Zeit.

Bei der Pool-Konfiguration benötigt jede Verarbeitungseinheit die Zeit T_{ges} zur Verarbeitung eines Pakets. Somit ist die Paketrate pro Verarbeitungseinheit $P_{VE,Pool} = 1 / T_{ges}$ und die Gesamt-Paketrate der Pool-Konfiguration, bestehend aus N Verarbeitungseinheiten, ist:

$$P_{Pool,simpl} = N \cdot P_{VE,Pool} = \frac{N}{T_{ges}} \tag{3.1}$$

Der Durchsatz der Pipeline-Konfiguration ist unter den getroffenen Annahmen gleich dem Kehrwert der Verarbeitungsdauer $T_{S,opt}$ einer Pipeline-Verarbeitungseinheit und somit:

$$P_{Pipeline,simpl} = \frac{1}{T_{S,opt}} = \frac{N}{T_{ges}} \tag{3.2}$$

Man sieht somit, dass unter den oben getroffenen, simplifizierenden Annahmen beide Konfigurationen den gleichen Durchsatz aufweisen:

$$P_{Pool,simpl} = P_{Pipeline,simpl} \tag{3.3}$$

Realer Fall

Allerdings sind die oben getroffenen Annahmen in der Realität teilweise nicht einzuhalten, teilweise jedoch auch zu pessimistisch. Im Folgenden sollen diese Annahmen daher schrittweise überprüft und fallengelassen werden, um zu sehen, wie sich dies auf den Durchsatz von Pool und Pipeline auswirkt.

Annahme 5, dass bei einer Pipeline-Anordnung die Verarbeitungsdauern $T_{S,i}$ in allen Pipeline-Stufen $i \in \{1,2,...,N\}$ genau gleich groß sind, ist i. d. R. nicht möglich. Daher ist der Durchsatz i. A. durch den Kehrwert der größten Verarbeitungsdauer begrenzt. Es gilt:

$$\max_i\{T_{S,i}\} \geq T_{S,opt} = \frac{T_{ges}}{N} \qquad \Longrightarrow \qquad P_{Pipeline} = \frac{1}{\max_i\{T_{S,i}\}} \leq P_{Pipeline,simpl} \tag{3.4}$$

Jedoch auch ***Annahme 6*** ist, insbesondere bei Pool-Anordnungen, nicht realistisch. Während eine Pipeline-Anordnung nämlich jeweils nur einfache Punkt-zu-Punkt-Verbindungen zwischen ihren Verarbeitungseinheiten benötigt, müssen Pool-Anordnungen über aufwändige Punkt-zu-Mehrpunkt- bzw. Mehrpunkt-zu-Punkt-Verbindungen vor und hinter den Verarbeitungseinheiten verfügen. Sollen zudem externe Ressourcen, wie Speichermodule oder Coprozessoren, angesprochen werden können, so sind auch hier bei der Pool-Anordnung Mehrpunkt-zu-Punkt-Verbindungen notwendig. Bei der Pipeline-Anordnung werden hier wiederum aufgrund der verteilten Funktionalität häufig nur zwischen jeweils einer oder wenigen Verarbeitungseinheiten und externen Ressourcen Kommunikationsverbindungen benötigt. Dies führt i. d. R. zu einer längeren Gesamtverarbeitungszeit $T_{ges,Pool}$ bei der Pool-Konfiguration und einer entsprechenden Verringerung des Durchsatzes P_{Pool}.

$$T_{ges,Pool} > T_{ges} \qquad \Longrightarrow \qquad P_{Pool} = \frac{N}{T_{ges,Pool}} < P_{Pool,simpl} \tag{3.5}$$

Annahme 1 legte die Anzahl an Verarbeitungseinheiten auf N fest. Aus demselben Grund wie gerade kann diese Anzahl bei einer Pool-Anordnung nicht beliebig erhöht werden – der Aufwand für die 1-zu-N-Vermittlungssysteme und somit auch deren Vermittlungsdauern steigen dabei überproportional. Somit skaliert der Durchsatz einer Pool-Anordnung durch Erhöhen von N nicht beliebig.

Im Gegensatz dazu führt eine Verlängerung der Pipeline-Anordnung zunächst zu keinen Problemen. Der Durchsatz ist jedoch auch hier begrenzt, da ab einer gewissen Anzahl an Verarbeitungseinheiten die Funktionen nicht mehr in sinnvolle kleinere Teilfunktionen aufteilbar sind. Jedoch ist das Hinzufügen von neuen Verarbeitungsfunktionen bei gleich bleibendem Durchsatz problemlos durch zusätzliche Pipeline-Stufen machbar.

Annahmen 3 und 4 legten fest, dass alle Pakete dieselbe Verarbeitung mit einer Dauer von T_{ges} erfahren. Häufig benötigen jedoch verschiedene Klassen von Paketen unterschiedliche Verarbeitungsfunktionen, welche auch unterschiedlich aufwändig sind. In einer Pool-Anordnung kann daher nicht von einer einheitlichen Verarbeitungsdauer T_{ges} ausgegangen werden, sondern manche Pakete sind bereits schneller fertig verarbeitet und können die Verarbeitungseinheit entsprechend früher wieder freigeben. Somit sinkt die mittlere Verarbeitungsdauer $\overline{T}_{ges,Pool}$, was den Durchsatz im Mittel erhöht:

$$\overline{T}_{ges,Pool} \leq T_{ges,Pool} \qquad \Longrightarrow \qquad \overline{P}_{Pool} = \frac{N}{\overline{T}_{ges,Pool}} \geq P_{Pool} \tag{3.6}$$

Allerdings kann diese Rate nicht garantiert werden, da nicht ausgeschlossen werden kann, dass für einen längeren Zeitraum nur Pakete mit den aufwändigsten Verarbeitungsanforderungen ankommen.

Laut ***Annahme 2*** sind alle Verarbeitungseinheiten identisch. Falls diese Einschränkung fallengelassen wird, können bei der Pool-Anordnung für die verschiedenen Klassen von Paketen unterschiedliche Verarbeitungseinheiten eingesetzt werden. Jede dieser Einheiten kann dann jeweils auf die benötigten Funktionen spezialisiert sein, was zu einer Verkürzung der jeweiligen Verarbeitungsdauer führt.[2] Allerdings muss nun das Verhältnis der Anzahl der verschiedenen Spezialverarbeitungseinheiten (zumindest im Mittel) genau auf die Verkehrsklassenverteilung im Hochlastfall angepasst sein. Andernfalls kann kein höherer sondern vermutlich sogar ein verringerter Durchsatz die Folge sein. Für eine Durchsatzgarantie ist solch eine Konfiguration somit ungeeignet. (Die Netzprozessoransätze von [130, 131] versuchen durch Rekonfiguration der Verarbeitungseinheiten diese an die Verkehrsklassenverteilung anzupassen, vgl. Unterkapitel 2.5.)

Bei einer Pipeline-Anordnung führt die Existenz von unterschiedlichen Verarbeitungsdauern bei unterschiedlichen Verkehrsklassen ($\neq$ ***Annahmen 3 und 4***) zunächst zu keiner Änderung des Durchsatzes. Der Grund hierfür ist, dass die Pakete jeweils alle Einheiten synchron durchwandern müssen, auch wenn sie nicht in allen Einheiten eine Verarbeitung erfahren. Der Durchsatz ist somit weiterhin, wie beschrieben, durch den Kehrwert der größten Verarbeitungsdauer bestimmt. Bei Volllast sind stets alle Einheiten belegt, so dass keine Einheiten übersprungen werden können, und auch das prinzipiell mögliche vorzeitige Verlassen der Pipeline würde keine Durchsatzsteigerung, sondern lediglich eine Verkürzung der Latenz bringen.

Allerdings kann aufgrund der verteilten Verarbeitung in den N Verarbeitungseinheiten der Pipeline jede Einheit für ihre benötigten Funktionen spezialisiert[3] sein ($\neq$ ***Annahme 2***). Somit ist i. d. R. eine geringere maximale Verarbeitungsdauer bei den Pipeline-Verarbeitungseinheiten $T_{S,i,spez}$ sowie eine geringere Pipeline-Gesamtverarbeitungsdauer $T_{ges,Pipeline}$ zu erwarten, was den Durchsatz entsprechend erhöht:

$$\max_i\{T_{S,i,spez}\} < \max_i\{T_{S,i}\} \qquad \Longrightarrow \qquad P_{Pipeline,spez} > P_{Pipeline} \tag{3.7}$$

Somit kann – bei einer geforderten Durchsatz-Garantie – eine Pipeline mit einer einigermaßen gleichmäßigen Verteilung der Verarbeitungsfunktionen auf N jeweils auf ihre Aufgaben

[2] Eine falsche Spezialisierung z. B. aufgrund von neuen oder geänderten Anforderungen kann jedoch zu umständlichen Implementierungen und einem geringen Durchsatz führen.

[3] siehe Fußnote 2

spezialisierte Verarbeitungseinheiten einen höheren Durchsatz garantieren als ein Pool von N gleichartigen Verarbeitungseinheiten:

$$P_{Pipeline,spez} > P_{Pool} \tag{3.8}$$

Zum Abschluss dieses Unterabschnitts bleibt somit festzuhalten, dass unter vereinfachten Annahmen der Durchsatz vergleichbarer Pool- und Pipeline-Konfigurationen identisch ist. In Realität, ohne diese Vereinfachungen, wirken sich die komplexen Verbindungsstrukturen bei der Pool-Anordnung negativ auf deren Durchsatz aus. Allerdings kann der mittlere Durchsatz dieser Anordnung gesteigert werden, wenn Pakete mit kürzerer Verarbeitungsdauer ihre Verarbeitungseinheit freigeben, sobald die Verarbeitung abgeschlossen ist. Allerdings kann dieser Durchsatz nicht garantiert werden. Der Einsatz von spezialisierten Einheiten in Pool-Konfigurationen beschränkt den Durchsatz unter ungünstigen Verkehrsverteilungen.

Der Durchsatz von Pipeline-Konfigurationen ist stets durch den Durchsatz der Verarbeitungseinheit mit der längsten Verarbeitungsdauer begrenzt. Daher ist eine gleichmäßige Aufteilung der Verarbeitungsfunktionen sehr wichtig. Diese Aufteilung ermöglicht den Einsatz von spezialisierten Einheiten, was zu einem höheren garantierten Gesamtdurchsatz führt. Unterschiedliche Verarbeitungsdauern unterschiedlicher Verkehrsklassen führen hier nicht zu einer Erhöhung des mittleren Durchsatzes. Wird also ein hoher *mittlerer* Durchsatz gewünscht, ist eine Pool-Konfiguration vorzuziehen. Soll ein hoher Durchsatz *garantiert* werden, ist eine Pipeline-Konfiguration besser. Tabelle 3.1 auf Seite 93 fasst alle Vor- und Nachteile nochmals zusammen.

3.3.3 Implementierungsaufwand

Im Folgenden soll der Aufwand für die Implementierung von Verarbeitungsfunktionen auf Systemen mit den beiden betrachteten Konfigurationen, Pool und Pipeline, verglichen werden. Hierbei werden stets Implementierungen in Software auf einem Netzprozessor und Implementierungen in Hardware auf einem FPGA betrachtet.

Die Implementierung von Funktionen bei **Pool**-Konfigurationen ist vergleichsweise einfach. Bei Prozessor-basierten Systemen handelt es sich hier um eine so genannte *run-to-completion*-Programmierung, d. h. der Entwickler muss nur ein (langes) Programm für die komplette Verarbeitung programmieren, welche dann ein Prozessorkern vollständig ausführt. Auch bei FPGA-Verarbeitungssystemen muss sich der Entwickler keine Gedanken über eine Unterteilung der Funktionen in mehrere Module machen.

Allerdings muss der Zugriff auf von den Verarbeitungseinheiten des Pools geteilte externe Ressourcen untereinander abgestimmt werden. Entsprechende Synchronisationsmechanismen kann ein Compiler in die Implementierung integrieren. Falls jedoch ein sehr hoher Durchsatz benötigt wird, muss der Programmierer selbst solche Mechanismen realisieren oder die Verarbeitungs-Hardware-Infrastruktur muss ein entsprechendes Synchronisations-Modul bereitstellen.

Aufgrund möglicher, unterschiedlich langer Verarbeitungszeiten in einer Pool-Konfiguration können die verarbeiteten Pakete in einer anderen Reihenfolge fertig bearbeitet sein, als sie ankamen. Daher muss entweder der Entwickler durch die Implementierung oder ein entsprechendes

Modul in der Verarbeitungsinfrastruktur dafür sorgen, die Pakete wieder in ihre ursprüngliche Reihenfolge umzusortieren.

Der Vorteil bei der Realisierung von Funktionen für eine **Pipeline**-Verarbeitung ist deren Modularität. Die Funktionen jeder Verarbeitungseinheit können die Entwickler jeweils getrennt und unabhängig von anderen Einheiten implementieren. Der Zugriff auf externe Einheiten erfolgt meist nur von einer Verarbeitungseinheit, wodurch keine Synchronisation notwendig ist. Die Paketreihenfolge kann sich bei einer Pipeline nicht ändern.

Nachteilig wirkt sich beim Software- und beim Hardware-Entwurf für Pipeline-Systeme die Notwendigkeit aus, dass man die benötigten Funktionen bezüglich ihrer Verarbeitungsdauer möglichst gleichmäßig auf die Pipeline-Stufen verteilen muss. Um einen hohen Durchsatz erzielen zu können, werden hierfür teilweise mehrere Entwurfsiterationen benötigt.

Bei Systemen mit spezialisierter Verarbeitungsinfrastruktur, **unabhängig ob in Pool- oder Pipeline**-Konfiguration, kann die geforderte Funktionalität meist einfach, schnell und effizient entworfen und realisiert werden. Allerdings müssen sich die Entwickler in verschiedene Architekturen und Basisfunktionalitäten einarbeiten. Außerdem können diese Einheiten für unvorhergesehene Funktionsanforderungen falsch spezialisiert und somit ungeeignet sein, was die Implementierung erheblich erschweren kann. Tabelle 3.1 fasst auch diese Vor- und Nachteile zusammen.

3.3.4 Ressourcenaufwand

Abschließend wird der Ressourcenaufwand von Pool- und Pipeline-Konfiguration verglichen. Ein schon in Abschnitt 3.3.2 angesprochenes Problem der **Pool**-Konfiguration ist, dass diese aufwändige Punkt-zu-Mehrpunkt-Kommunikationsstrukturen benötigt. Diese werden nicht nur für das Verteilen und Einsammeln der Pakete auf bzw. von den Prozessorkernen benötigt, sondern auch für den Zugriff auf gemeinsam genutzte Ressourcen, wie Coprozessoren oder Speicher. Die Realisierung der hierfür benötigten Arbitrierungs-, Multiplexer- und Verbindungsstrukturen ist insbesondere bei vielen Verarbeitungseinheiten sehr ressourcenaufwändig.

Zusätzlich können, wie oben beschrieben, unterschiedlich lange Verarbeitungszeiten ein Umsortierungsmodul notwendig machen. Dieses benötigt ebenfalls zusätzliche Ressourcen. Von Vorteil ist jedoch, dass bei unterschiedlichen Verarbeitungsklassen durch die teilweise vorzeitige Freigabe der Verarbeitungseinheiten weniger Einheiten benötigt werden, falls das System keinen Mindest-Durchsatz garantieren muss.

Eine **Pipeline** benötigt für die Kommunikation der Prozessoren untereinander, mit Coprozessoren und Speichermodulen sowie mit dem Empfangs- und Sendemodul i. d. R. jeweils nur einfache Punkt-zu-Punkt-Verbindungen. Diese benötigen nicht viele Ressourcen. Ein Umsortierungsmodul wird aufgrund der sequentiellen Verarbeitung nicht benötigt. Außerdem erlaubt die Aufteilung der Funktionalität auf die Verarbeitungseinheiten die Verwendung kleiner Befehlsspeicher bei Prozessor-basierten Systemen.

Negativ anzumerken bei der Pipeline-Verarbeitung ist, dass Pakete, auf welche nicht alle Funktionen angewendet werden müssen, dennoch synchron mit den anderen Paketen durch alle Ver-

arbeitungseinheiten transportiert werden müssen. Dadurch werden Einheiten teilweise nicht genutzt, was unnötig bereitgestellte Ressourcen bedeutet.

Sowohl **Pool- als auch Pipeline**-Verarbeitungssysteme aus identischen Verarbeitungseinheiten haben den Vorteil, dass bei der Entwicklung dieser Einheiten ein starker Aufwand betrieben werden kann. Somit können diese Einheiten sehr ressourceneffizient realisiert werden. Besteht ein Pool oder eine Pipeline jedoch aus verschiedenen, spezialisierten Verarbeitungseinheiten, so können diese aus Kostengründen häufig nicht so stark optimiert werden. Andererseits bringt diese Spezialisierung den Vorteil, dass nur wirklich benötigte Funktionsblöcke realisiert wer-

Tabelle 3.1: Vergleich der Eigenschaften von Pool und Pipeline

	Pool	Pipeline
Durchsatz	$\ominus$ Punkt-zu-Mehrpunkt-Verbindungen	$\oplus$ Punkt-zu-Punkt-Verbindungen
	$\oplus$ unterschiedlich lange Verarbeitungsdauern erhöhen mittleren Durchsatz	$\oplus$ garantierter Durchsatz, unabhängig von Verkehrsklassenverteilung
	—	$\ominus$ Stufen mit ungleich langen Verarbeitungszeiten
	$\ominus$ Spezialisierung nur geeignet bei bekannter statischer Verkehrsklassenverteilung	$\oplus/\ominus$ Spezialisierung erhöht garantierten Durchsatz, aber nur falls sie zur Problemstellung passt
Implementierung	$\oplus$ einfach, *run-to-completion*	$\oplus$ modulare, unabhängige Funktionen
	$\oplus$ keine (zwanghafte) Unterteilung in Submodule erforderlich	$\ominus$ möglichst gleichmäßige Aufteilung in Funktionsgruppen
	$\ominus$ evtl. Synchronisation durch Implementierung	$\oplus$ meist keine Synchronisation notwendig
	$\ominus$ evtl. Reihenfolge-Sortierung durch Implementierung	$\oplus$ keine Sortierung notwendig
	$\oplus/\ominus$ einfacher, schneller und effizienter bei spezialisierten Verarbeitungseinheiten, aber nur falls Spezialisierung zur Problemstellung passt	
Ressourcen	$\ominus$ Punkt-zu-Mehrpunkt-Verbindungen	$\oplus$ Punkt-zu-Punkt-Verbindungen
	$\ominus$ evtl. Umsortierungsmodul notwendig	$\oplus$ kein Umsortierungsmodul notwendig
	$\oplus$ Ressourceneinsparungen durch kurze Verarbeitungsdauern, falls kein garantierter Durchsatz notwendig	$\ominus$ schlechte Ressourcennutzung, falls Pakete nicht in jeder Einheit verarbeitet werden müssen
	$\oplus/\ominus$ spezialisierte Verarbeitungseinheiten ressourceneffizienter, aber weniger optimiert als identische Verarbeitungseinheiten	

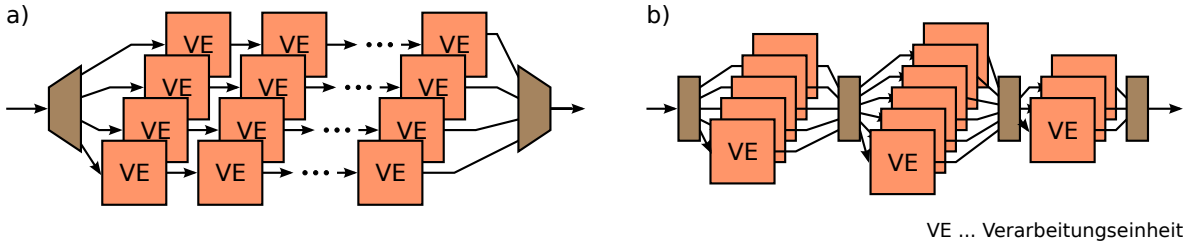

Abbildung 3.3: a) Pool von Pipelines, b) Pipeline aus Stufen in Pool-Konfiguration

den müssen, was somit zu ressourceneffizienten Einheiten führt. Bei Systemen aus identischen Einheiten müssen diese möglichst vielseitig sein, was einer ressourceneffizienten Realisierung entgegensteht. Alle diese Vor- und Nachteile sind, ebenso wie die bezüglich Durchsatz und Implementierungsaufwand, in Tabelle 3.1 auf Seite 93 zusammengefasst dargestellt.

3.3.5 Kombinationen verschiedener Parallelisierungsarten

Die beiden vorgestellten Arten der Parallelisierung von Verarbeitungseinheiten können auch kombiniert werden. Hierauf geht der folgende Unterabschnitt ein. Daran anschließend werden noch kurz andere Arten und Ebenen der Parallelisierung vorgestellt. Diese werden ebenfalls in Paketverarbeitungssystemen eingesetzt, um einen hohen Durchsatz zu erreichen.

Kombination von Pipeline und Pool

Die beiden Konfigurationen Pool und Pipeline lassen sich auf unterschiedliche Weisen kombinieren. Zum einen kann man, wie in Abbildung 3.3 a) dargestellt, mehrere Pipeline-Anordnungen als Pool betreiben. Zum anderen können einzelne oder alle Stufen einer Pipeline als Pool realisiert werden. Dies ist in Abbildung 3.3 b) abgebildet. Solche Kombinationen vereinen auf unterschiedliche Weise sowohl einige der positiven als auch der negativen Eigenschaften beider Anordnungen.

In einem Pool von identischen Pipelines (Abbildung 3.3 a) können die einzelnen Stufen jeweils aller Pipelines spezialisierte Verarbeitungseinheiten enthalten, was den Durchsatz erhöht. Zugriffe auf gemeinsam genutzte Ressourcen sind in dieser Konfiguration jeweils auf die Anzahl der parallelen Pipelines begrenzt. Ein Pool bestehend aus unterschiedlichen jeweils für eine Paketklasse spezialisierten Pipelines ist wie ein Pool mit spezialisierten Einzel-Einheiten ungeeignet, falls die Klassenverteilung schwankt.

Pool-Konfigurationen innerhalb von Pipeline-Stufen (Abbildung 3.3 b) können verschiedene Funktionen erfüllen. Zum einen können in einer Pipeline-Struktur identische Verarbeitungseinheiten einer Stufe mehrmals parallel nebeneinander geschaltet sein, wie in Abbildung 3.4 a) dargestellt. In diesem Fall darf bei m parallelen Einheiten in diesem Pool jede Einheiten die m-fache Zeit für die Verarbeitung benötigen wie eine einzelne Verarbeitungseinheit einer Pipeline-Stufe. Somit kann dort eine komplexere Verarbeitung stattfinden. Falls die Einheiten des Pools jedoch auf gemeinsame Ressourcen zugreifen müssen, hat dies die bekannten Nachteile. Die Netzprozessoren von EZchip implementieren diese Art von Pipeline. Bei der flexiblen

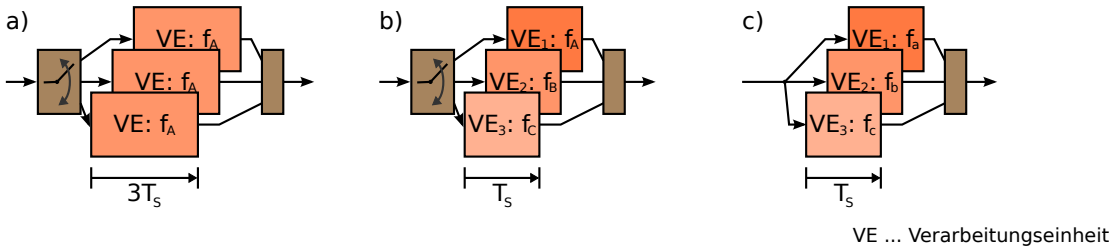

Abbildung 3.4: Pipeline-Stufen: a) Pool dreier identischer Verarbeitungseinheiten mit dreifacher Verarbeitungsdauer, b) Pool mit Verarbeitungseinheiten mit exklusiven Funktionalitäten für unterschiedliche Paketklassen, c) Pool mit Verarbeitungseinheiten mit mehreren Funktionen für jedes Paket

Kommunikationsinfrastruktur zwischen den Verarbeitungseinheiten der Intel/Netronome Netzprozessoren, wird diese Anordnung ebenfalls häufig verwendet.

Eine weitere Möglichkeit ist, in einer Pipeline-Stufe unterschiedliche Verarbeitungseinheiten als Pool parallel zu schalten, wie in Abbildung 3.4 b) dargestellt. Ankommende Pakete werden dann je nach erforderlichem Verarbeitungsschritt von der einen oder der anderen Einheit verarbeitet. Um weiterhin einen garantierten Durchsatz zu unterstützen, darf jede Einheit maximal die einfache Verarbeitungszeit einer Pipeline-Stufe benötigen. Diese Anordnung ermöglicht eine kürzere Latenz im Vergleich zu einer Anordnung, bei der diese Einheiten hintereinander geschaltet worden wären und dennoch pro Paket jeweils exklusiv nur eine Funktionalität genutzt würde.

Abbildung 3.4 c) zeigt eine Anordnung, bei der die verschiedenen Einheiten des Pools innerhalb einer Pipeline-Stufe alle parallel dasselbe Paket verarbeiten. Somit kann die durchzuführende Funktionalität einer Stufe zusätzlich auf mehrere Einheiten modularisiert werden. Die Netzprozessorarchitektur von Bay Microsystems verwendet eine Kombination von dieser und der zuvor beschriebenen Form der Pipeline-Konfiguration. Diese Art der Pool-Verarbeitung ist anders als die bisher betrachteten, bei der ein Paket nur in einer Einheit verarbeitet wurde und die Parallelität genutzt wurde, um mehrere Pakete gleichzeitig zu verarbeiten. Hier handelt es sich um eine andere Ebene der Parallelität. Dies ist keine Parallelität auf Paketebene sondern auf Funktionsebene. Im nächsten Abschnitt wird noch weitergehend auf andere Arten und Ebenen der Parallelisierung eingegangen.

Andere Ebenen der Parallelisierung

Bisher wurde Parallelisierung auf Paketebene betrachtet. Zusätzlich können weitere Arten der Parallelisierung auf niederer Ebene genutzt werden. Die oben vorgestellte Anordnung in Abbildung 3.4 c) nutzt Parallelisierung auf Funktionsebene.

Andere Möglichkeiten der Parallelisierung sind aus der Prozessortechnik wohl bekannt. Hier kommt zusätzlich Parallelisierung auf Datenebene, auf Befehlsebene und auf Bitebene zum Einsatz. Parallelisierung auf Datenebene verwendet parallele Verarbeitungseinheiten zur Ausführung derselben Operationen auf einer großen Menge von Daten, wie z. B. in Grafikprozessoren. [125] zeigt die Verwendung eines solchen Daten-parallelen Prozessors für zwei Paket-

verarbeitungsanwendungen (vgl. Unterkapitel 2.5). Parallelität auf Befehlsebene (*instruction level parallelism*) nutzt parallele Recheneinheiten innerhalb eines Prozessorkerns zur gleichzeitigen Ausführung mehrerer Maschinenbefehle. Beispiele sind superskalare Prozessorarchitekturen und VLIW-Architekturen, wie sie z. B. in den Verarbeitungseinheiten der Xelerated Netzprozessoren vorkommen (vgl. Abschnitt 2.4.1). Parallelisierung auf Bitebene bezeichnet schließlich die Verarbeitung von 16-, 32- oder 64-bit-Vektoren innerhalb eines Rechenwerks. Diese Art der Parallelisierung ist heute Standard und wurde in Abschnitt 3.1.1 im Zusammenhang mit der Register-Transfer-Ebene eingeführt.

Coprozessoren stellen eine weitere Dimension der Parallelisierung dar. Hier lagert eine Verarbeitungseinheit aufwändige Berechnungen aus und kann so gleichzeitig andere Operationen durchführen. Coprozessoren sind i. d. R. nicht oder kaum programmierbar. Sie können exklusiv an eine Verarbeitungseinheit angebunden sein, oder als geteilte Ressource von mehreren Einheiten genutzt werden. Letztere Möglichkeit bringt entsprechende Zugriffskonflikte und zusätzliche Wartezeiten mit sich. Coprozessoren ergänzen häufig Pool- und Pipeline-Anordnungen in Hochleistungsnetzprozessoren, insbesondere wenn alle Verarbeitungseinheiten identisch und somit wenig spezialisiert sind (vgl. die Netzprozessoren von Intel/Netronome, Cisco und Xelerated in Abschnitt 2.4.1).

Multi-Threading parallelisiert die Verarbeitung nur indirekt. Diese Technik wird verwendet, um Wartezeiten z. B. beim Zugriff auf Speicher oder Coprozessoren zu überbrücken. Dafür speichert die Verarbeitungseinheit den aktuellen Verarbeitungszustand ab und fährt mit der Verarbeitung eines anderen Pakets – in einem anderen Thread – fort. Da hierbei innerhalb der Verarbeitungseinheit keine gleichzeitige Verarbeitung stattfindet, ist dies eigentlich keine Parallelisierung. Vielmehr wird die Verarbeitung in einem Zeitmultiplex betrieben. Bei der Betrachtung des Gesamtsystems tritt jedoch tatsächlich Parallelität auf, schließlich verarbeitet die Verarbeitungseinheit ein Paket, während außerhalb der Einheit – parallel dazu – beispielsweise ein Speicherzugriff oder eine Berechnung in einem Coprozessor stattfindet. Multi-Threading wird in vielen Hochleistungsnetzprozessoren eingesetzt, z. B. in den Prozessorkernen der Intel/Netronome- und Cisco-Netzprozessoren sowie in den Ein-/Ausgabeeinheiten innerhalb der Pipeline-Struktur der Xelerated Netzprozessoren.

3.4 Architekturansatz

In diesem Unterkapitel wird der neue Architekturansatz vorgestellt, mit welchem es möglich ist, alle in Unterkapitel 3.2 formulierten Ziele zu erreichen. Zunächst motiviert Abschnitt 3.4.1 die grundlegenden Konzepte des Ansatzes. Abschnitt 3.4.2 beschreibt und argumentiert die zu Grunde liegende Struktur des Architekturansatzes. Anschließend führt Abschnitt 3.4.3 die wichtigsten Merkmale der Verarbeitungsmodule der neuen Architektur ein. Das darauf folgende Unterkapitel 3.5 diskutiert diese Module dann im Detail. Die Grundzüge dieser Architektur wurden vom Autor in [1] bereits einem wissenschaftlichen Fachpublikum vorgestellt.

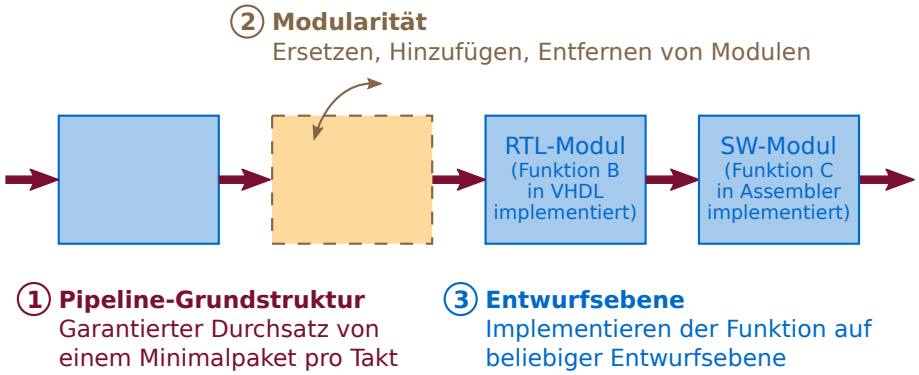

Abbildung 3.5: Die drei wesentlichen Eigenschaften des Architekturansatzes

3.4.1 Motivation

Die Ziele der Paketverarbeitungsarchitektur sind, Leistungsfähigkeit und Flexibilität zu maximieren. Die Architektur soll zum einen einen hohen Durchsatz garantieren, zum anderen soll sie die Flexibilität bieten, Verarbeitungsfunktionen modular auf einer beliebigen Entwurfsebene entsprechend der jeweiligen Anforderungen implementieren zu können.

Die zwei grundlegenden Arten der Parallelisierung, Pool und Pipeline, wurden im letzten Unterkapitel gegenübergestellt. Beide Konfigurationen haben ihre Vor- und Nachteile. Jedoch konnte festgestellt werden, dass eine Pipeline-Struktur vorzuziehen ist, wenn ein hoher Durchsatz *garantiert* werden soll. Der hier vorgestellte Architekturansatz soll daher auf einer Pipeline-Grundstruktur basieren. Um einen sehr hohen Durchsatz unterstützen zu können, sieht der Ansatz eine extreme Dimensionierung vor: Die zu Grunde liegende Pipeline-Struktur soll *ein Minimalpaket pro Takt* verarbeiten können.

Ein weiteres wichtiges Ziel ist Modularität. Modularität erlaubt, einzelne Funktionen als Module auf einfache Weise hinzuzufügen, zu ändern, auszutauschen oder zu entfernen. Eine Pipeline-Struktur ist prinzipiell gut für Modularität geeignet. Sie kann theoretisch beliebig um Stufen ergänzt werden, Stufen können geändert, ausgetauscht und entfernt werden, ohne den Durchsatz der Pipeline zu beeinflussen. Voraussetzung hierfür ist, dass die neuen oder geänderten Stufen den Durchsatzanforderungen der Pipeline genügen. Der hier vorgestellte Architekturansatz sieht daher vor, dass neue oder angepasste *Module beliebig* in die Pipeline *integriert* und ebenso Module beliebig entfernt werden können. Hierfür ist eine flexible Hardware-Infrastruktur erforderlich, wie sie z. B. in FPGAs zu finden ist – man kann sich jedoch auch andere Technologien vorstellen.

Um das Ziel der verschiedenen Entwurfsebenen verwirklichen zu können, müssen verschiedene Module zur Verfügung stehen, die Implementierungen auf jeweils unterschiedlichen Abstraktionsebenen ermöglichen. Der Architekturansatz sieht daher verschiedene Module mit jeweils *unterschiedlichen Verarbeitungsinfrastrukturen* vor. Die Module müssen, wie oben erwähnt, nach außen die zeitlichen Pipeline-Anforderungen erfüllen, sollen jedoch intern dem Entwickler unterschiedliche Abstraktions- bzw. Entwurfsebenen bieten. Beispielsweise soll es Module geben, mit denen man Funktionen in VHDL auf Register-Transfer-Ebene beschreiben kann, und Module, welche man in Assembler Software-basiert steuert. Solche unterschiedlichen Module können alle mittels herkömmlicher, aktueller FPGAs realisiert werden oder auch mittels

eines neuartigen, optimierten Bauelements, welches programmierbare Logikzellen und Verbindungsstrukturen aber auch größere Einheiten wie konfigurierbare Rechenwerke oder auch Prozessorkerne enthält (vgl. Cswitch-Chip in Abschnitt 2.4.2 und [122]).

Die neue Architektur soll somit auf einer Pipeline-Struktur basieren, welche ein Minimalpaket pro Takt verarbeiten kann. Die Grundstruktur der Pipeline soll aus funktionalen Modulen bestehen, welche beliebig variiert werden können. Zusätzlich sollen Funktionen auf unterschiedlichen Entwurfsebenen implementiert werden können, je nachdem, welche Art von Modul ausgewählt wird. Abbildung 3.5 illustriert diese drei wesentlichen Eigenschaften.

3.4.2 Pipeline-Grundstruktur

Die Grundstruktur der neuen Paketverarbeitungsarchitektur ist eine Pipeline. Unterkapitel 3.3 diskutierte die vielen positiven sowie auch einige negative Eigenschaften dieser Struktur. Eine der wichtigsten positiven Eigenschaften ist, dass eine Pipeline einfache Punkt-zu-Punkt-Verbindungen sowohl zwischen den einzelnen Verarbeitungseinheiten als auch häufig zwischen einzelnen Verarbeitungseinheiten und anderen Ressourcen besitzt. Dies ermöglicht eine ressourceneffiziente Realisierung, verhindert die Notwendigkeit von Arbitrierungs- oder Synchronisationsmechanismen und erlaubt somit einen deterministischen Durchsatz. Eine weitere wichtige Eigenschaft ist die inhärente Modularität einer Pipeline, welche das Hinzufügen von Verarbeitungseinheiten sowie die Spezialisierung von Verarbeitungseinheiten erlaubt, ohne den Durchsatz des Gesamtsystems zu beeinflussen.

Maximierung des Durchsatzes

Die Pipeline soll den maximal möglichen Durchsatz erzielen. Der Durchsatz ist definiert als das Verhältnis von Datenmenge und Zeit. In Unterkapitel 3.3 ist der Durchsatz einer Pipeline als der Kehrwert der maximalen Verarbeitungsdauer einer Pipeline-Stufe bestimmt worden – die Datenmenge wurde implizit als ein Paket angenommen (vgl. Gleichung (3.4)). Bei der Dimensionierung der Pipeline-Grundstruktur soll der Durchsatz maximiert werden, d. h. die Verarbeitungszeit pro Stufe wird minimiert und die Datenmenge, die in dieser Zeit verarbeitet wird, wird maximiert. Dabei muss allerdings darauf geachtet werden, dass die Architektur dennoch mit heutiger Technologie realisiert werden kann.

Zunächst wird die Verarbeitungsdauer pro Pipeline-Stufe betrachtet: Digitale Systeme arbeiten stets synchron zu einem Taktsignal. Somit ist die minimale Verarbeitungsdauer einer Pipeline-Stufe eine Taktperiode. RISC-Prozessoren basieren auf solchen Pipelines. Diese führen effektiv einen Befehl pro Takt aus, indem jede Stufe nur eine Taktperiode auf den Daten arbeitet und sie dann inklusive Steuersignalen in die nächste Stufe transportiert. FPGA-basierte Paketverarbeitungssysteme können ebenfalls als taktsynchrone Pipelines implementiert werden (vgl. [7, 11, 141]). Alle zwei Takte transportieren die Software-gesteuerten Xelerated Netzprozessoren die Daten von einer in die nächste Pipeline-Stufe (vgl. Abschnitt 2.4.1 und [91]). Die **Verarbeitungsdauer pro Stufe** in der Pipeline-Grundstruktur der neuen Paketverarbeitungsarchitektur wird daher auf **eine Taktperiode** festgelegt. Der Durchsatz des Systems ist somit proportional zur Taktfrequenz f.

Um den Durchsatz zu maximieren, müssen möglichst viele Daten parallel durch die Pipeline transportiert und verarbeitet werden. Die Daten, die pro Takt von einer Stufe in die nächste Stufe geschoben werden, werden im Folgenden als *(Daten-) Wort* bezeichnet, die Anzahl an Bytes dieses Worts, also das Datenvolumen pro Wort, entsprechend als *Wortbreite W*. Ziel ist es somit, die Wortbreite W zu maximieren. Bei der Dimensionierung wurden mehrere Aspekte berücksichtigt:

- Erstens, wenn die Wortbreite so groß ist, dass jede Pipeline-Stufe mehr als ein Paket gleichzeitig verarbeitet, entspricht dies einer Verarbeitung in einer Pool-Struktur. Somit treten auch die in Unterkapitel 3.3 diskutierten Nachteile auf – z. B. Konflikte beim zeitgleichen Zugriff auf gemeinsame Ressourcen. Um dies zu vermeiden, muss die maximale Datenmenge für die Verarbeitung pro Stufe somit auf ein Paket begrenzt sein.

- Zweitens ist die maximale Paketgröße i. d. R. in der Größenordnung von ein bis mehreren Kilobyte – z. B. bei Ethernet 1514 Byte, das entspricht gut 12 000 bit. Eine so große Menge an Daten pro Takt von einer Stufe zur nächsten parallel zu transportieren, ist jedoch mit dem heutigen Stand der Technik nicht effizient möglich und würde außerordentlich viele Chip-Ressourcen erfordern.

- Drittens weisen viele zu verarbeitende Pakete nicht die maximale Größe auf – im Gegenteil, etwa die Hälfte aller im Internet transportierten Pakete sind TCP-Bestätigungspakete und somit i. d. R. Pakete minimaler Größe – bei Ethernet 64 Byte, also 512 bit. Eine für Pakete maximaler Größe dimensionierte Wortbreite würde also vielfach kaum belegt, entsprechend würden die Pipeline-Stufen nicht effizient genutzt und somit würde wertvolle Chipfläche verschwendet.

- Viertens verarbeiten viele Hochgeschwindigkeitsnetzknoten ausschließlich die Header-Felder der Pakete, somit ist es hier nicht notwendig, dass das ganze Paket durch die Pipeline transportiert wird. Diese Tatsache wird in Zukunft allerdings mehr und mehr ungültig, da in zunehmendem Maße auch Funktionen auf den Nutzdaten der Pakete gefordert werden. Dies gilt insbesondere für Rand- und auch Zugangsnetzknoten. Beispiele für solche Funktionen sind Ver- und Entschlüsselung der Daten, Deep Packet Inspection oder Transkodierung von Multimedia-Daten.

Unter Einbeziehung dieser Aspekte wurde die **Wortbreite** der durch die Pipeline zu transportierenden Daten etwa auf die Größenordnung von **ein bis zwei Minimalpaketen** dimensioniert. Dies ist mit heutiger Technologie realisierbar, verschwendet bei kleinen Paketen kaum Ressourcen und alle Header bis einschließlich Schicht 4 sind i. d. R. im ersten Datenwort enthalten. Bei 100 Gbit/s-Ethernet-Netzen mit einer minimalen Paketgröße von 64 Byte, bietet sich daher eine Wortbreite zwischen 64 und 128 Byte, also zwischen 512 und 1024 bit, an. Abbildung 3.6 zeigt eine entsprechende Pipeline, bei der jeweils ein 1024 bit breites Datenwort in jedem Takt von einer Stufe zur nächsten transportiert wird.

Architekturvarianten

Aufgrund des vierten oben betrachteten Punktes erlaubt die Architektur zwei verschiedene Betriebsvarianten: Für Vermittlungsknoten, bei denen sich die Paketverarbeitung auf die Analyse

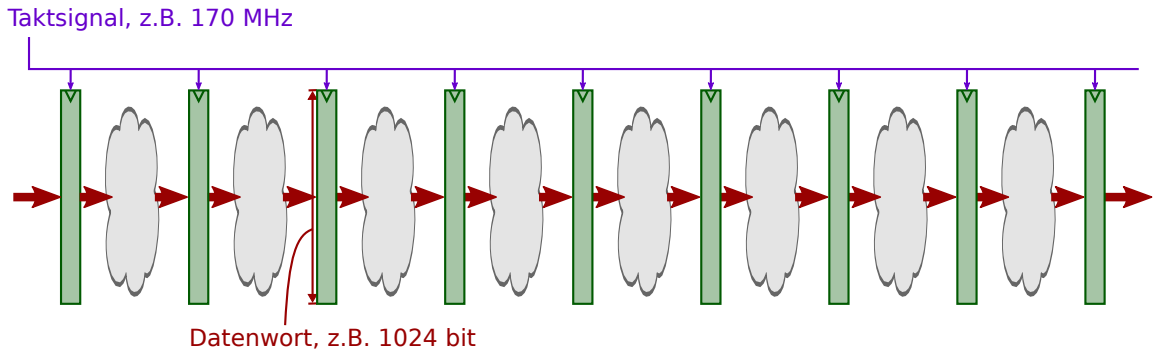

Abbildung 3.6: Zu Grunde liegende Pipeline-Struktur

und Modifikation der Header-Felder beschränkt, wie z. B. bei Hochgeschwindigkeitsnetzknoten in Kernnetzen, ist es möglich, jeweils nur das erste Wort jedes Pakets durch die Pipeline zu transportieren und die restlichen Daten für die Dauer der Verarbeitung in einem kleinen Puffer zwischenzuspeichern. Diese Konfiguration wird im Folgenden als *Nur-Header*-Variante (NH) bezeichnet. Für Vermittlungsknoten, welche in der Lage sein sollen, Verarbeitungsfunktionen auf allen Paketdaten auszuführen, kann die Architektur in der *Ganzes-Paket*-Variante (GP) realisiert werden. Bei dieser Variante, werden Pakete, die größer als die Wortbreite der Pipeline sind, vor der Pipeline in mehrere Datenworte unterteilt und nacheinander durch die Pipeline transportiert und dort verarbeitet.

Die Paketrate der Nur-Header-Variante ist, unabhängig von der Größe des Pakets, konstant und entspricht nominal der Taktfrequenz f.

$$P_{NH} = (1\ \text{PAKET}) \cdot f \tag{3.9}$$

Zur Realisierung eines Paketverarbeitungssystems für eine 100 Gbit/s-Ethernet Line Card muss eine Paketrate von 150 MP/s garantiert werden. Mit der vorgestellten Architektur wird dies somit bereits mit einer Taktfrequenz von 150 MHz erreicht.

Bei Minimalpaketen, bzw. allen Paketen die im ersten Datenwort Platz finden, weist die Ganzes-Paket-Variante dieselbe Paketrate wie die Nur-Header-Variante auf. Bei größeren Paketen (Paketgröße $L > W$) nimmt sie jeweils dann ab, wenn wieder ein neues Datenwort für das zu verarbeitende Paket begonnen wird. Die exakte Formel der Paketrate für die GP-Variante lautet:

$$P_{GP} = \frac{1}{\left\lceil \frac{L}{W} \right\rceil} \cdot (1\ \text{PAKET}) \cdot f \tag{3.10}$$

Dimensionierung von Wortbreite und Taktfrequenz

Um ein Paketverarbeitungssystem für einen geforderten Durchsatz mit der hier vorgestellten Architektur in der GP-Variante zu realisieren, muss also ein Kompromiss zwischen einer hohen Taktfrequenz und einer großen Wortbreite gefunden werden. Abbildung 3.7 zeigt beispielhaft die benötigte (unidirektionale) Paketrate über der Paketgröße für ein Paketverarbeitungssystem auf einer 100 Gbit/s-Ethernet Line Card sowie die tatsächliche garantierte Paketrate unter

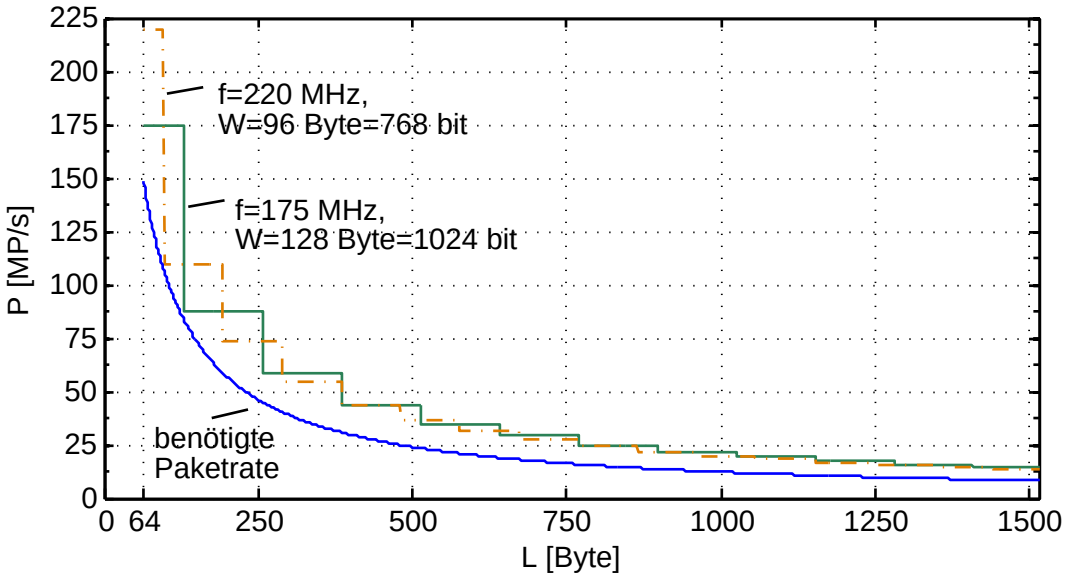

Abbildung 3.7: Benötigte (unidirektionale) Paketverarbeitungsrate auf einer 100 Gbit/s-Ethernet Line Card und tatsächliche maximale Paketrate für verschiedene Wortbreiten und Taktfrequenzen über der Paketgröße

Verwendung der GP-Variante der vorgestellten Architektur für zwei Kombinationen von Taktfrequenz und Wortbreite. Die mindestens benötigte Paketrate berechnet sich dabei äquivalent zu Gleichung (2.1) in Abhängigkeit der Datenrate D und der Paketgröße L nach folgender Formel (wobei G_{min} der minimale Paketabstand in Byte ist):

$$P = \frac{D}{L + G_{min}} \tag{3.11}$$

Bei der Dimensionierung sind zwei Punkte zu beachten:

- Erstens, um die Paketrate für Minimalpakete zu erreichen, muss die Taktfrequenz bei den gewählten großen Wortbreiten ($W \geq L_{min}$) mindestens dieser Paketrate entsprechen.

$$f \geq P_{max} = \frac{D}{L_{min} + G_{min}} \tag{3.12}$$

- Zweitens, aufgrund der Halbierung der Paketrate bei Paketen, welche ein Byte größer als die Wortbreite sind, ist es bei dieser Paketgröße am schwierigsten, dass die geforderte Paketrate erreicht werden kann. Durch Kombination von Gleichung (3.11) und Gleichung (3.10) für $L = W + 1$ Byte erhält man den erforderlichen Zusammenhang von Wortbreite W und Taktfrequenz f.

$$f \geq \frac{2D}{W + 1\,\text{Byte} + G_{min}} \qquad \Leftrightarrow \qquad W \geq \frac{2D}{f} - 1\,\text{Byte} - G_{min} \tag{3.13}$$

Abbildung 3.8 und Tabelle 3.2 illustrieren die beiden Bedingungen aus den Ungleichungen 3.12 und 3.13 für die Dimensionierung eines Paketverarbeitungssystems für eine 100 Gbit/s-Ethernet Line Card.

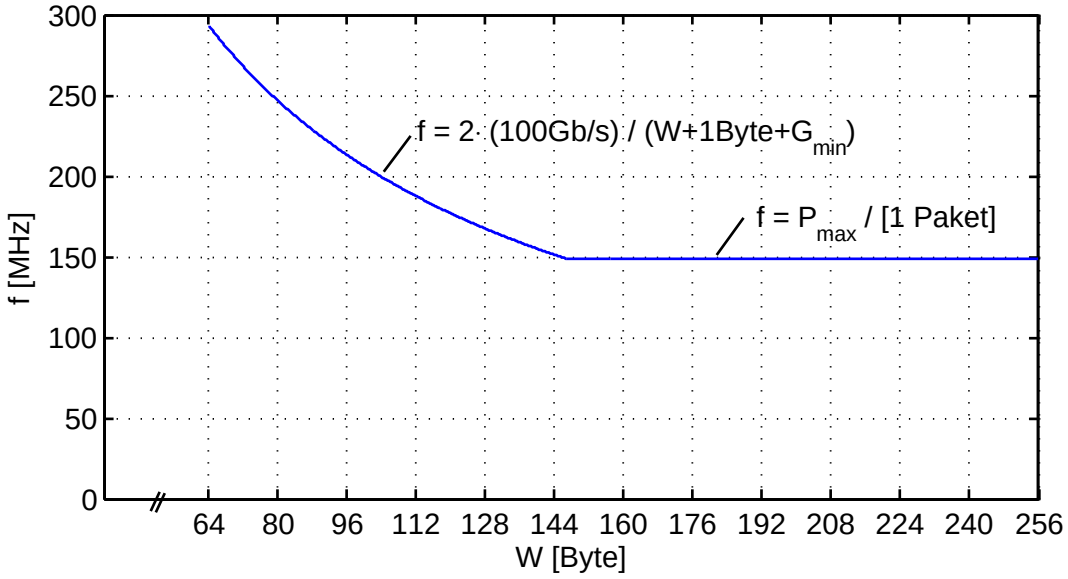

Abbildung 3.8: Benötigte Frequenz über der Wortbreite für ein 100 Gbit/s-Ethernet-System

Zur Dimensionierung eines Paketverarbeitungssystems in der NH-Variante der vorgeschlagenen Architektur muss lediglich Ungleichung 3.12 beachtet werden. Da der gesamte Paketkopf in einem Wort transportiert wird, muss hier nur die Taktfrequenz hoch genug gewählt sein.

Tabelle 3.2: Taktfrequenz und Wortbreite für eine Paketrate ausreichend für ein unidirektionales Paketverarbeitungssystem auf einer 100 Gbit/s-Ethernet-Line-Card

Taktfrequenz	**Wortbreite**
150 MHz	$\geq$ 146 Byte = 1168 bit
175 MHz	122 Byte = 976 bit
200 MHz	104 Byte = 832 bit
225 MHz	91 Byte = 728 bit
250 MHz	79 Byte = 632 bit
275 MHz	70 Byte = 560 bit
152 MHz	**144 Byte = 1152 bit**
168 MHz	**128 Byte = 1024 bit**
188 MHz	**112 Byte = 896 bit**
214 MHz	**96 Byte = 768 bit**
248 MHz	**80 Byte = 640 bit**
295 MHz	**64 Byte = 512 bit**

3.4.3 Verarbeitungsmodule

Das Ziel eines garantierten hohen Durchsatzes ist durch die Definition der der Architektur zu Grunde liegenden Pipeline-Struktur erreicht worden. Um die beiden anderen, wichtigen Ziele – Modularität und die Möglichkeit, Funktionen auf einer passenden Entwurfsebene implementieren zu können – zu erreichen, werden im Folgenden spezielle Verarbeitungsmodule eingeführt. Diese Verarbeitungsmodule sollen sich genauso wie die Stufen der oben beschriebenen Pipeline verhalten, d. h. sie sollen pro Takt ein Datenwort empfangen und ein Datenwort zur nächsten Stufe weiterschieben. Wenn die Module dieses Verhalten der zu Grunde liegenden Pipeline aufweisen, können sie beliebig in die Pipeline integriert werden, bzw. kann die Pipeline vollständig aus solchen Verarbeitungsmodulen aufgebaut sein und weiterhin den geforderten Durchsatz garantieren. Außerdem sollen die Module austauschbar sein und eine einfache Erweiterbarkeit und Anpassbarkeit von Systemen mit der hier vorgestellten Architektur sicherstellen. Hierbei können unterschiedliche Module jeweils den Entwurf auf einer anderen Abstraktions- bzw. Entwurfsebene ermöglichen.

Um die Austauschbarkeit und Erweiterbarkeit garantieren zu können, müssen alle Verarbeitungsmodule feste Schnittstellendefinitionen einhalten. Diese Schnittstellendefinitionen umfassen zum einen die Festlegung der Schnittstellensignale und -formate und zum anderen die Spezifikation der zeitlichen Anforderungen an diese Signale. Die Schnittstellensignale setzen sich zusammen aus dem im letzten Abschnitt dimensionierten Paket-Datenwort sowie Metadaten und einzelnen Steuersignalen. Die Metadaten enthalten zusätzliche Informationen zum aktuellen Paket, wie z. B. dessen Empfangsschnittstelle, Klassifizierungsergebnisse und Weiterleitungsinformationen. Die zeitlichen Randbedingungen an diese Signale diktiert, wie oben bereits erwähnt, die zu Grunde liegende Pipeline-Struktur der Architektur. Jedes Verarbeitungsmodul muss an seinen Eingangs- und Ausgangs-Schnittstellen den Pipeline-Durchsatz garantieren, d. h. es muss in jedem Taktzyklus ein Datenwort (inklusive Steuersignalen und Metadaten) empfangen und gleichzeitig ein verarbeitetes Datenwort ausgeben.

Der interne Aufbau der Verarbeitungsmodule ist nicht festgelegt, so dass sie intern beliebig realisiert sein können – auch eine interne Pool-Konfiguration von Verarbeitungseinheiten ist möglich. Somit ist auch die dem Entwickler bei der Implementierung von Funktionen zur Verfügung stehende Verarbeitungsinfrastruktur nicht festgelegt. Somit können z. B. Module bereitgestellt werden, welche intern aus einer Vielzahl paralleler Prozessorkerne aufgebaut sind und in Software programmiert werden. Ebenso können Module bestehend aus programmierbaren Logikelementen bzw. Lookup-Tables bereitgestellt werden, welche Funktionsbeschreibungen in einer Hardware-Beschreibungsprache wie VHDL realisieren können. Somit kann entsprechend der aktuell zu implementierenden Funktion ein Verarbeitungsmodul mit einer passenden Entwurfsebene zur Realisierung verwendet werden. Zusätzlich bietet dieser Architekturansatz auch die Möglichkeit, fremde Module (z. B. von einem anderen Hersteller oder von Universitäten und anderen Forschungseinrichtungen) in ein bestehendes System einzubinden, welches dann über Parameter oder andere Einstellmöglichkeiten angepasst werden kann.

Abbildung 3.9 zeigt die Einbindung eines Verarbeitungsmoduls in die Pipeline. Das nächste Unterkapitel 3.5 wird noch tiefer auf Verarbeitungsmodule eingehen. Insbesondere werden die Schnittstellen solcher Module werden diskutiert sowie mögliche unterschiedliche Arten von Verarbeitungsmodulen vorgestellt. Im Anschluss daran wird in Unterkapitel 3.6 dargestellt, wie

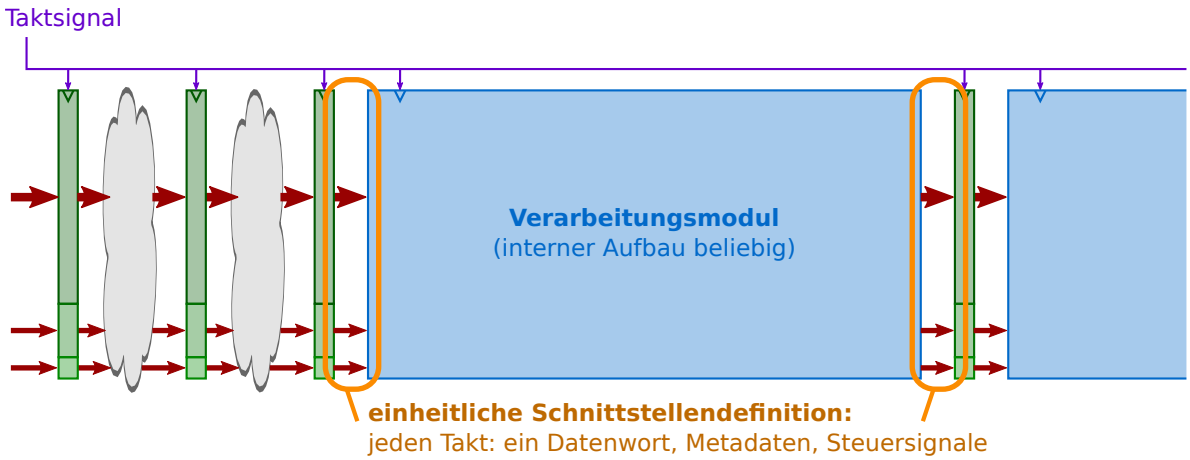

Abbildung 3.9: Verarbeitungsmodul und zu Grunde liegende Pipeline

der Ablauf für die Realisierung neuer Funktionen in einem System mit dieser Architektur aussieht, wobei auch auf mögliche Werkzeugunterstützung eingegangen wird.

3.5 Verarbeitungsmodule

Die im letzten Unterkapitel eingeführten Verarbeitungsmodule können auf viele unterschiedliche Arten realisiert sein. Sie bieten somit dem Entwickler von Verarbeitungsfunktionen verschiedene Entwurfsebenen für die Implementierung, aus denen er entsprechend der jeweils spezifischen Anforderungen die passende auswählen kann. Damit diese Module in das System integriert werden können, müssen sie einheitlichen Schnittstellendefinitionen gehorchen. Diese beschreibt nachfolgend Abschnitt 3.5.1. Im Anschluss daran gibt Abschnitt 3.5.2 einen Überblick über mögliche Arten von Verarbeitungsmodulen mit unterschiedlichen Abstraktionsebenen. Abschnitt 3.5.3, Abschnitt 3.5.4 und Abschnitt 3.5.5 beschreiben daraufhin beispielhaft drei Modularten tiefergehend.

3.5.1 Schnittstellen

Alle Verarbeitungsmodule müssen identische Schnittstellen aufweisen. Dadurch ist es möglich diese Module auf einfache Weise in ein bestehendes Paketverarbeitungssystem der hier beschriebenen Architektur zu integrieren.

Um Abhängigkeiten zwischen Verarbeitungsmodulen möglichst gering zu halten, sollten die Schnittstellen zwischen den Modulen möglichst schmal sein. Die Schnittstelle zwischen Modulen des hier vorgestellten Architekturansatzes beschränkt sich auf folgende Signale:

- Paketdaten
- Metadaten
- wenige Steuersignale

Die Paketdaten werden entsprechend der architekturell festgelegten Pipeline-Grundstruktur sequentiell über die in Abschnitt 3.4.2 definierten Datenworte durch das System transportiert. Wie dort beschrieben wurde, haben diese Datenworte etwa die Größe von ein bis zwei Paketen minimaler Größe, also im Falle von Ethernet von 64 bis 128 Byte. Jedes Modul muss über Schnittstellen zum Empfangen und Ausgeben dieser Paket-Datenworte verfügen.

Neben den Paket-Datenworten müssen noch zusätzliche Daten mit Metainformationen durch die Pipeline transportiert werden. Diese so genannten *Metadaten* können zusätzliche Informationen zum aktuellen Paket, wie dessen Empfangsschnittstelle, dessen Größe, sowie Ergebnisse von schon durchgeführten Verarbeitungsfunktionen enthalten, die für nachfolgende Funktionen benötigt werden.

Typische Elemente innerhalb der Metadaten sind:

Informationen des MAC-Empfangsmoduls, wie die Empfangsschnittstellennummer, die Größe des Pakets sowie dessen Schicht-2-Protokolltyp

Header-Analyse-Ergebnisse, wie der Typ des Schicht-3-Protokolls, die Quell- und Zieladresse auf Schicht-3, der Typ des Schicht-4-Protokolls sowie die Quell- und Ziel-Port-Nummer auf Schicht-4

Klassifizierungs- und Suchergebnisse, wie die gefundene Weiterleitungsadresse auf Schicht 3, die entsprechende Adresse auf Schicht 2 und die Sendeschnittstellennummer sowie eine ermittelte Dienstgüteklasse oder eine Identifikationsnummer für eventuelle Ausnahmebehandlungen.

Neben diesen Elementen sollte zusätzlich Platz für weitere Informationen sein, um spätere unvorhergesehene Anforderungen realisieren zu können. Je nach Umfang und Codierung werden so insgesamt acht bis 64 Byte Metadaten benötigt. Da ein System dieser Architektur auf Basis einer konfigurierbaren Chiptechnologie, wie z. B. eines FPGAs, realisiert wird, ist es bei einer generisch ausgelegten Systemimplementierung zusätzlich auch möglich, die Breite der Metadaten bei Bedarf nachträglich zu erhöhen.

Ein Beispiel für eine konkrete Definition, welche Informationen in Metadaten enthalten sein müssen, ist in der Dokumentation der Software-Architektur der Intel Netzprozessoren [165] zu finden. Ein weiteres Beispiel gibt der IETF Internet-Draft *ForCES LFB Library* [166], der eine Auflistung enthält, welche Metadaten zwischen den logischen Funktionsblöcken (LFB, *Logical Function Block*) eines Routers entsprechend der ForCES-Spezifikation (*Forwarding and Control Element Separation*, Trennung von Weiterleitungs- und Steuerelementen) übermittelt werden.

Zusätzlich zu den Paketdaten und den Metadaten umfasst die Schnittstelle zwischen Verarbeitungsmodulen noch einige Steuersignale. Diese zeigen beispielsweise den Beginn oder das Ende eines Pakets an oder markieren Pakete, welche aufgrund von Slow-Path-Verarbeitung nicht im Fast-Path modifiziert werden sollen.

Zeitliche Anforderungen

Die Verarbeitungsmodule müssen denselben Durchsatz aufweisen wie die zu Grunde liegende Pipeline-Struktur. Diese übergibt ein Datenwort pro Taktzyklus von einer Pipeline-Stufe an die nächste. Entsprechend müssen auch die Verarbeitungsmodule in jedem Taktzyklus ein solches Datenwort an ihrer Eingangsschnittstelle aufnehmen und ein verarbeitetes Datenwort über ihre Ausgangsschnittstelle wieder ausgeben. Parallel zu den Datenworten müssen auch die Metadaten in jedem Takt empfangen und nach der Verarbeitung weitergesendet werden.

Die Verarbeitungsdauer innerhalb eines Verarbeitungsmoduls, d. h. deren Latenz, ist nicht vorgegeben oder eingeschränkt. Auch die Art der internen Verarbeitung ist beliebig, solange die oben beschriebenen Durchsatzanforderungen an den Schnittstellen eingehalten werden.

3.5.2 Arten von Verarbeitungsmodulen

Wie oben beschrieben, ist die einzige Randbedingung, die Verarbeitungsmodule erfüllen müssen, dass sie die einheitlich spezifizierte Definition der Schnittstellensignale einschließlich deren zeitlichen Verhaltens einhalten. Der interne Aufbau kann beliebig sein. So können Module realisiert werden, die wie die zu Grunde liegende Pipeline die Verarbeitung als Pipeline durchführen. Genauso kann die Verarbeitung jedoch auch in einer Pool-Konfiguration oder einer Kombination der beiden prinzipiellen Parallelisierungen erfolgen.

Ebenso ist es möglich, dass die Taktfrequenz innerhalb eines Moduls anders als die der Grund-Pipeline ist. So kann die Verarbeitung innerhalb eines Moduls beispielsweise mit der doppelten oder vierfachen Taktfrequenz durchgeführt werden, was zur Folge hat, dass nur jeden zweiten bzw. vierten internen Takt mit der Verarbeitung eines neuen Datenwortes begonnen werden muss. Ebenso kann mit der halben Frequenz gearbeitet werden, wobei dann jeweils zwei Datenworte gleichzeitig verarbeitet werden müssen.

Außerdem können die Module intern auch unterschiedliche Verarbeitungsinfrastrukturen einsetzen. Verarbeitungsmodule können z. B. einerseits programmierbare Logikelemente und Flipflops wie FPGAs zur Implementierung von Verarbeitungsfunktionen zur Verfügung stellen. Andererseits können sie verschieden spezialisierte Prozessoren bereitstellen, auf denen Funktionen Software-gesteuert ausgeführt werden können. Auch mikroprogrammierbare Module sind denkbar, genauso wie über Parameter nur begrenzt anpassbare Verarbeitungsmodule. Prinzipiell sind Verarbeitungsmodule mit den unterschiedlichsten Verarbeitungsinfrastrukturen möglich, da lediglich deren Schnittstellen vorgegeben sind. Nachfolgend werden beispielhaft einige Arten von Verarbeitungsmodulen aufgelistet, welche eine Verarbeitung auf verschiedenen Entwurfsebenen ermöglichen. Im nachfolgenden Unterkapitel werden dann drei dieser Modularten noch tiefergehend beschrieben.

Programmierbare-Logik-Module – Module mit einer Verarbeitungsinfrastruktur wie in einem FPGA, d. h. aufgebaut aus einer Matrix von Logikblöcken bestehend aus Abbildungstabellen und Flipflops. Wie bei FPGAs erfolgt auch hier die Beschreibung der auszuführenden Funktionen in VHDL oder einer anderen Hardware-Beschreibungssprache. Alternativ kann in Zukunft auch eine Beschreibung auf Verhaltensebene erfolgen, welche dann für eine

Realisierung in einem solchen Modul mit konfigurierbaren Logikblöcken synthetisiert wird. Eine genauere Beschreibung dieser Modulart folgt in Abschnitt 3.5.3.

Mikrocode-Module – Module mit einer mikroprogrammierbaren Verarbeitungsinfrastruktur. Solche Module können universell aufgebaut sein aus verschiedenen Rechenwerken, Logikwerken, Speichern sowie verschiedenen Registersätzen oder auch anwendungsspezifisch, z. B. durch den Einsatz spezieller Klassifizierungseinheiten (vgl. [8]), TCAMs oder Datenanalysemodulen, wobei alle Elemente über Mikrocode taktgenau gesteuert werden können. Durch die Mikroprogrammierung ist die Steuerung hochoptimierter Verarbeitungen möglich, bei der auch viele Operationen parallel auf den zur Verfügung stehenden Elementen ausgeführt werden können. Die Verarbeitung in solch einem Modul kann sowohl in einem Pool von mehreren Verarbeitungseinheiten oder auch in Form einer Pipeline erfolgen.

Prozessor-Pool-Module – Module bestehend aus einem Pool von Prozessorkernen. Diese Prozessorkerne können sowohl einen allgemeinen Befehlssatz als auch einen auf ein bestimmtes Problem spezialisierten Befehlssatz aufweisen. Die Realisierung von Verarbeitungsfunktionen erfolgt hier mittels Software-Programmierung in Assembler oder einer höheren Programmiersprache, wobei die maximale Anzahl der Verarbeitungstakte durch die Anzahl und Taktfrequenz der Prozessorkerne beschränkt ist. Näheres zu Prozessor-Pool-Modulen folgt in Abschnitt 3.5.4.

Datenfluss-Pipeline-Module – Module mit einer Verarbeitungsarchitektur bestehend aus einer Pipeline von programmierbaren Verarbeitungseinheiten, durch die die Paket- und Metadaten zur Verarbeitung transportiert werden. Die Verarbeitungsdauer pro Einheit beträgt dabei nur einen oder wenige Takte. Wiederum kann die Verarbeitungsinfrastruktur universell oder spezialisiert sein. Abschnitt 3.5.5 beschreibt diese Modulart detaillierter.

Spezialfunktionsmodule – Module, die eine spezielle Verarbeitungsfunktion realisieren. Diese Module bieten nur eine begrenzte Flexibilität, da ihre Funktion bereits fest ist. Diese kann jedoch über Parameter oder Konfigurationsdateien noch leicht angepasst werden. Beispiele für solche Module sind IP-Adresssuchmodule, Paketklassifizierungsmodule und Prüfsummenberechnungsmodule.

Durch die einheitlichen Schnittstellen der Verarbeitungsmodule ist es ohne Weiteres möglich, Module von anderen Firmen oder auch Universitäten in ein Paketverarbeitungssystem der vorgestellten Architektur zu integrieren. Solche fremden Module können einerseits spezielle Verarbeitungsinfrastrukturen enthalten. Diese können verwendet werden, um eigene Funktionen darauf zu implementieren. Andererseits können Firmen oder Forschungseinrichtungen Module mit implementierten Funktionen verkaufen oder öffentlich bereitstellen. So könnten einerseits Firmen Geld mit der Entwicklung und dem Verkauf solcher Module verdienen, andererseits könnten Universitäten und Forschungseinrichtungen Verarbeitungsmodule für neue oder experimentelle Protokolle oder Dienste zur Verfügung stellen, um deren Untersuchung in Testnetzen sowie deren Verbreitung zu fördern. Diese Implementierung könnten sowohl als *Open-Source*-Module als auch bereits synthetisiert mit einigen Parametrisierungsmöglichkeiten bereitgestellt werden.

3.5.3 Programmierbare-Logik-Module

Mit einem Programmierbare-Logik-Modul soll man Verarbeitungsfunktionen wie auf einem FPGA realisieren können. Somit hat man die Möglichkeit, Funktionen bis auf Flipflop- und Gatterebene hinab zu entwerfen und zu implementieren. Jedoch kann man auch komplexere Elemente wie Addierer, Schieberegister und Speicher auf einfache Weise beschreiben und auch vorgefertigte, nur noch zu parametrisierende Elemente wie FIFO-Warteschlangen, Verschlüsselungsschaltungen oder Prozessorkerne aus Bibliotheken einsetzen. Dabei kann der Entwickler eine beliebige Verarbeitungsstruktur verwenden. So kann er je nach den spezifischen Anforderungen der Funktion zwischen Pool- und Pipeline-Anordnungen von Verarbeitungselementen sowie Kombinationen beider Anordnungen variieren.

Für ein solches Modul können – falls das Verarbeitungssystem auf einem FPGA basiert – die im FPGA vorhandenen Flipflops, Lookup-Tables und Verbindungsstrukturen direkt verwendet werden. Ein speziell für diese Architektur gefertigter Chip müsste für solche Module ebenfalls solche oder ähnliche Basiselemente zur Verfügung stellen. Der Entwickler beschreibt die Struktur und das Verhalten des Moduls in einer Hardware-Beschreibungssprache wie VHDL oder Verilog. Um korrekte Schnittstellen bezüglich des Formats zu garantieren, kann diese Art von Modulen dem Funktions-Entwickler bereits vollständig beschriebene Schnittstellendefinitionen in der jeweiligen Hardware-Beschreibungssprache zur Verfügung stellen. Außerdem können sie Zugriffsfunktionen auf Elemente der Metadaten und Paketdaten bereitstellen, um die Implementierung zu vereinfachen. Zur Validierung der Funktion kann er das Modul in einem Simulationswerkzeug, wie bei FPGA-Entwürfen üblich, Taktzyklus-genau simulieren.

Der Entwickler muss beim Entwurf und der Implementierung der Verarbeitungsarchitektur seines Moduls selbst dafür Sorge tragen, dass dieses die erforderlichen Durchsatzanforderungen erfüllt. Dies kann durch testweise Umsetzung des beschriebenen Moduls in auf der Zieltechnologie verfügbare Elemente (Synthese-Schritt) und deren Platzierung auf dem Chip (Place & Route-Schritt) überprüft werden. Wie das Modul abschließend in ein Gesamtsystem integriert und schließlich auf dem Chip realisiert wird, wird in Unterkapitel 3.6 gezeigt.

3.5.4 Prozessor-Pool-Module

Mit einem Prozessor-Pool-Modul kann man Verarbeitungsfunktionen wie auf einem Standard-Netzprozessor als Software programmieren. Somit können auf einfache und schnelle Weise neue Funktionen in das Paketverarbeitungssystem integriert werden, ohne dafür Hardware-Entwurfskenntnisse zu benötigen. Das Modul besteht hierfür aus einem Pool von Prozessorkernen. Jeder dieser voneinander unabhängigen Prozessorkerne führt die Software auf den ihm zugewiesenen Paket-Datenworten aus. Damit ist die Implementierung für einen Entwickler sehr einfach, da die Funktion vollständig auf einem Prozessor ausgeführt wird und er diese nicht wie bei einer Pipeline auf mehrere Verarbeitungseinheiten aufteilen muss (vgl. Unterkapitel 3.3).

Die Grundstruktur eines Prozessor-Pool-Moduls ist in Abbildung 3.10 dargestellt. Das Modul weist an seinem Ein- und Ausgang die einheitlich festgelegten Schnittstellensignale auf und hält damit auch die zeitlichen Anforderungen ein. Am Eingang übergibt ein Verteilmodul die ankommenden Datenworte und Metadaten reihum den Prozessorkernen. Die Prozessorkerne

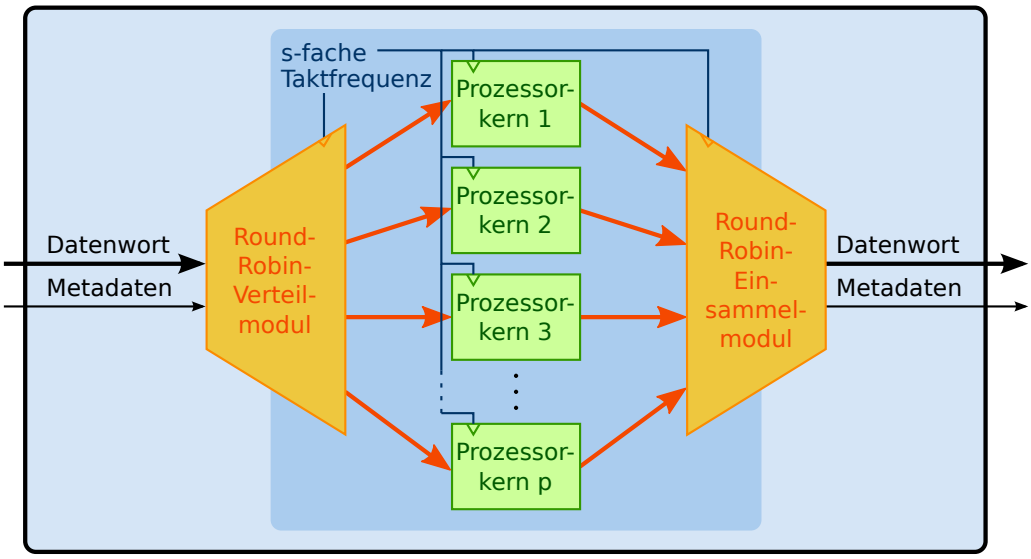

Abbildung 3.10: Prozessor-Pool-Modul bestehend aus p Prozessorkernen mit s-facher Taktfrequenz

verarbeiten diese Daten für eine festgelegte Zeitdauer. Danach kopiert ein Einsammel-Modul die Daten wieder zurück und gibt sie an der Ausgangsschnittstelle weiter zum nächsten Verarbeitungsmodul.

Für das Kopieren der Paket- und Metadaten gibt es verschiedene Optionen. Die ankommenden Daten kann das Verteilmodul direkt in einen Teil des Registersatzes des jeweiligen Prozessorkerns kopieren. In diesem Fall kann sich das auszuführende Programm das Laden der Paket- und Metadaten aus dem Speicher sowie das Speichern geänderter Werte sparen, wodurch man eine kürzere Ausführungsdauer erreichen kann. Alternativ kann das Verteilmodul die Daten in den lokalen Speicher des Prozessors schreiben. So werden keine Register mit nicht benötigten Paketdaten belegt. Falls alle Prozessorkerne nur über einen gemeinsam genutzten Paketspeicher verfügen, muss das Paket dorthin kopiert werden und dem jeweiligen Prozessorkern die entsprechende Startadresse übergeben werden. Bei dieser Variante benötigt das Laden und Speichern der Paketdaten vom Programm aus die meiste Zeit.

Unabhängig davon, wo die Paketdaten abgelegt werden, kann das Prozessor-Pool-Modul auch vorsehen, dass nur ein Teil des großen Datenworts für den Prozessor zugänglich gespeichert wird und der Rest während der Verarbeitung in einem globalen Paketpuffer verbleibt. Welcher Teil benötigt wird, kann dabei bei der Programmierung vom Entwickler festgelegt werden. Diese Variante nutzt die Tatsache aus, dass eine Verarbeitungsfunktion häufig nur auf einen Teil des Datenworts, z. B. eine bestimmte Protokollschicht, zugreifen muss. Dadurch wird u. U. weniger Zeit für die Kopieraktionen vor und nach der Verarbeitung in den Prozessorkernen benötigt.

Wie oben erwähnt, sorgt das Prozessor-Pool-Modul selbst dafür, dass die Durchsatzanforderungen an den Schnittstellen eingehalten werden. Bei p Prozessoren und einem Datenwort pro Prozessor muss das Verteilmodul jedem Prozessorkern alle p Takte ein Datenwort übergeben und das Einsammelmodul entsprechend wieder zurückkopieren. Somit dürfte die Verarbeitung pro Prozessorkern nicht mehr als p Takte benötigen. Die Prozessorkerne eines solchen Moduls können jedoch hoch optimiert für die Ausführung ihres begrenzten Befehlssatzes sein. Daher

können sie u. U. schneller getaktet werden als die dem Gesamtsystem zu Grunde liegende Pipeline-Struktur. Wenn die p Prozessorkerne des Pools mit der s-fachen Taktfrequenz betrieben werden, stehen somit die in Gleichung (3.14) angegebene Anzahl an Takten für die Ausführung der Verarbeitungsfunktion zur Verfügung.

$$\text{max. Ausführungsdauer} = p \cdot s \text{ Takte} \tag{3.14}$$

Beispielsweise kann eine Funktion bei einem Prozessor-Pool-Modul mit $p = 16$ Kernen und einer um den Faktor $s = 4$ höheren Taktfrequenz $4 \cdot 16 = 64$ Takte benötigen. Die Anzahl an Verarbeitungstakten ist somit beschränkt. Ein Prozessor-Pool-Modul ist daher nur für die Ausführung einzelner Verarbeitungsfunktionen mit einem Umfang in einer Größenordnung von 50 bis 150 Takten geeignet, was jedoch für viele Aufgaben ausreicht (vgl. [158]).

Um die Ausführungsdauer zu maximieren, sollte der Prozessor-Pool über möglichst viele ($p \uparrow$), möglichst schnelle ($s \uparrow$) Prozessorkerne verfügen. Aus diesem Grund sind für nicht näher spezifizierte Funktionsbereiche ein Pool von RISC-Prozessorkernen eine gute Wahl, da RISC-Kerne aufgrund ihrer Pipeline-Verarbeitung sehr schnell getaktet werden können und aufgrund ihres geringen Befehlssatzumfangs wenig Chipfläche benötigen. Eine Spezialisierung des Befehlssatzes auf den vorgesehenen Aufgabenbereich beschleunigt die Verarbeitung weiter. Für einen genau definierten Einsatzbereich können auch Spezialprozessoren eingesetzt werden. Diese Spezialisierungen bringen jedoch, wie bereits zuvor beschrieben, die Gefahr mit sich, dass die Prozessoren für unvorhergesehene Funktionen ungeeignet sind. Auch VLIW- oder superskalare Architekturen können verwendet werden, um mehrere Befehle pro Takt ausführen zu können und somit die Verarbeitung zu beschleunigen. Hierbei muss allerdings abgewogen werden, ob diese Befehlsparallelisierung effizient genutzt werden kann, oder ob der dafür benötigte zusätzliche Chipflächenbedarf nicht besser für zusätzliche Kerne genutzt werden sollte, was entsprechend die zur Verfügung stehende Taktanzahl erhöhen würde. Falls die Prozessoren Latenzen beim Zugriff auf Speichermodule oder Coprozessoren aufweisen, sind am besten Prozessorkerne mit Hardware-unterstütztem Multi-Threading zu verwenden. Damit können die Wartezeiten effizient für die Verarbeitung anderer Pakete genutzt werden.

Damit der Prozessor-Pool den vom Gesamtsystem geforderten Durchsatz garantieren kann, sollte jeder Prozessorkern völlig unabhängig von den anderen Kernen des Pools arbeiten. Somit können keine Unterbrechungen durch gegenseitige Abhängigkeiten beispielsweise beim Zugriff auf geteilte Ressourcen auftreten. Falls die Kerne gemeinsame Daten verwenden müssen, können diese Abhängigkeiten häufig durch lokale Kopien der geteilten Daten aufgelöst werden. Auf diese Weise können beispielsweise verteilte Zähler realisiert werden. Im Prinzip kann ein Prozessor-Pool-Modul auch über gemeinsam genutzte Ressourcen verfügen, allerdings muss die Zugriffsdauer darauf deterministisch bzw. nach oben begrenzt sein.

Bei einer Implementierung der vorgestellten Architektur auf einem FPGA, oder auch auf einem speziell für diese Architektur entwickelten Chip, können für die Realisierung eines Prozessor-Pool-Moduls entweder festverdrahtete, auf dem Chip integrierte Prozessorkerne oder Soft-Core-Prozessoren, welche die konfigurierbaren Ressourcen des Chips verwenden, eingesetzt werden. Erstere haben den Vorteil, dass sie in der Regel mit einer höheren Taktfrequenz betrieben werden können, letztere können jedoch besser angepasst werden. Beispielsweise können Soft-Core-Prozessoren so angepasst werden, dass das Verteil- und das Einsammelmodul direkt auf den Registersatz des Prozessors zugreifen können. Außerdem kann der Befehlssatz angepasst

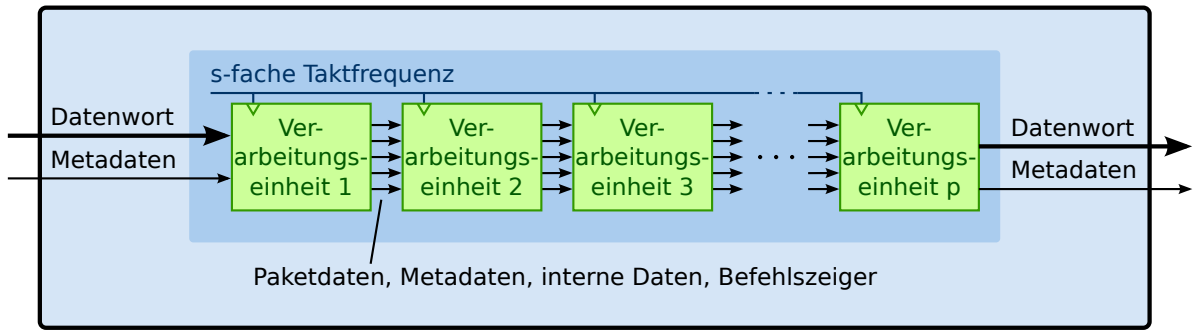

Abbildung 3.11: Datenfluss-Pipeline-Modul bestehend aus p Verarbeitungseinheiten mit s-facher Taktfrequenz ($s \geq 1$)

werden. Die FPGA-Technologie erlaubt weiterhin, dass ein Automatismus nach Programmieren der Software vor der Synthese des Gesamtsystems die Anzahl der Prozessorkerne im Pool entsprechend Gleichung (3.14) an die Ausführungsdauer des Programms anpasst. Außerdem können auch nicht verwendete Operationen oder Register – und somit Chipfläche – eingespart werden.

Prozessor-Pool-Module können also auf unterschiedliche Weisen aufgebaut sein. Abhängig von den Leistungsanforderungen und den Funktionsanforderungen sowie vom verwendeten Chip kann eine andere Variante zu bevorzugen sein. Aufgrund der einheitlichen Schnittstellen ist es natürlich auch möglich, in einem Paketverarbeitungssystem verschiedene Varianten von Prozessor-Pool-Modulen gleichzeitig einzusetzen. In Abschnitt 3.7.1 wird eine vom Autor selbst entworfene Implementierung vorgestellt und diskutiert. Dieses Modul besteht aus einfachen RISC-Prozessorkernen mit einem spezialisierten Befehlssatz und direktem Zugriff der Verteil- und Einsammelmodule auf den Registersatz.

3.5.5 Datenfluss-Pipeline-Module

Ein Datenfluss-Pipeline-Modul kann wie das zuvor beschriebene Prozessor-Pool-Modul ebenfalls als Software programmierte Verarbeitungsfunktionen ausführen. Dieses Modul basiert allerdings auf einer Pipeline von Verarbeitungseinheiten, welche die durchlaufenden Paketdaten schrittweise verarbeiten. Dieser Aufbau hat den Vorteil, dass der Transport der Paketdaten sowie auch der Zugriff auf externe Ressourcen jeweils über Punkt-zu-Punkt-Verbindungen erfolgen kann. Somit werden keine aufwändigen Verteil-, Einsammel- und Arbitrierungsmodule benötigt. Dies spart Chipfläche, ermöglicht eine hohe Taktfrequenz und verringert Wartezeiten bei der Verarbeitung. Der Aufbau von Modulen dieser Art ist an die Verarbeitungs-Pipeline der Netzprozessoren von Xelerated [91] angelehnt. Wie später gezeigt wird, weisen die hier vorgestellten Module allerdings noch weitergehende positive Eigenschaften bei der Integration in die hier beschriebene Verarbeitungsarchitektur auf. Abbildung 3.11 zeigt den prinzipiellen Aufbau eines Datenfluss-Pipeline-Moduls.

Die von der Architektur festgelegten Schnittstellenanforderungen werden vom Modul am Ein- und Ausgang stets eingehalten. Hierfür empfängt das Modul in jedem Takt der zu Grunde liegenden System-Pipeline ein Datenwort inklusive Metadaten und gibt eines am Ende seiner Pipeline aus. Wie beim Prozessor-Pool-Modul können intern die Verarbeitungseinheiten u. U. mit

einem höheren Takt betrieben werden. Somit wird die für die Verarbeitung verfügbare Taktanzahl wiederum durch das Produkt aus der Anzahl p an Verarbeitungseinheiten und dem Verhältnis s von interner Taktfrequenz zu System-Taktfrequenz beschränkt. Auch hier gilt also Gleichung (3.14). Bei gleichen Taktfrequenzen, d. h. bei $s = 1$, werden die Paket- und Metadaten in jedem Takt von einer Einheit zur nächsten weitertransportiert, bei s-facher interner Taktfrequenz alle s Takte.

Um die zur Verfügung stehende Anzahl an Verarbeitungstakten zu maximieren, sollten die Verarbeitungseinheiten möglichst klein ($p \uparrow$) und schnell ($s \uparrow$) sein. Superskalare oder VLIW-Einheiten mit mehreren parallelen Ausführungseinheiten können verwendet werden, um mehr Befehle pro Takt ausführen zu können. Auch hier muss jedoch abgewogen werden, ob eine größere Anzahl kleiner Verarbeitungseinheiten in der Pipeline nicht effizienter ist. Aufgrund der Datenfluss-orientierten Verarbeitung können die Verarbeitungseinheiten wesentlich effizienter realisiert werden als die Prozessorkerne in einem Prozessor-Pool-Modul. Hierauf wird in den folgenden Absätzen noch näher eingegangen.

Alle Verarbeitungseinheiten sind hintereinander als Pipeline angeordnet. Im Betrieb empfangen diese Einheiten alle s Takte die Paketdaten, die Metadaten sowie weitere Kontext-spezifische Daten. Damit führen sie ihren Teil der Gesamtverarbeitung aus und transportieren die Daten zur nächsten Einheit. Die mitgeführten Kontext-spezifischen Daten umfassen zuvor berechnete Zwischenergebnisse und einen Befehlsfolgezähler, der angibt, an welcher Stelle sich die Programmausführung momentan befindet. Rückkopplungen zwischen den Pipeline-Stufen des Moduls, wie man sie aus RISC-Prozessoren für das Weiterleiten (*Forwarding*) von bereits berechneten Werten kennt, werden nicht benötigt, weil sich alle zur Verarbeitung eines Pakets benötigten Daten stets in einer Pipeline-Stufe befinden.

Um Ressourcen zu sparen, kann vorgesehen werden, dass nicht das ganze (sehr große) Paketdatenwort durch die Verarbeitungseinheiten transportiert wird, sondern nur ein vom Programmierer festgelegter Bereich. Die restlichen Daten müssen dann in einem Ringspeicher außerhalb der Pipeline gepuffert werden. Zusätzlich können innerhalb der Datenfluss-Pipeline Austauschmodule vorgesehen werden, an denen die Daten aus der Pipeline mit einem Teil der Daten aus dem parallelen Puffer ausgetauscht werden können.

Intern verfügen die Einheiten über Paketdatenregister, Metadatenregister, Standardregister sowie lokale Register. Während die Standardregister zum Speichern von Zwischenergebnissen vorgesehen sind und mit den Paketdaten mitgeführt werden, sind die lokalen Register zum Speichern von Daten vorgesehen, welche in der jeweiligen Einheit verbleiben und somit von nachfolgenden Paketen weiter verwendet werden können. Mit Hilfe dieser Register können z. B. Paketzähler und andere Statistiken realisiert werden. Da die Verarbeitung sich über alle Einheiten der Datenfluss-Pipeline erstreckt, benötigt jede einzelne jeweils nur einen sehr kleinen Befehlsspeicher. Dies ist in Abbildung 3.12 für $s = 1$ dargestellt. Würde das Programm keine Verzweigungen aufweisen, müsste jeweils nur ein Befehl pro Befehlsspeicher gespeichert werden. Aufgrund von Verzweigungen spaltet sich der Programmfluss jedoch auf, so dass für alle möglichen Verzweigungen der entsprechende Befehl bereitgehalten werden muss.

Diese Tatsache, dass die Anzahl möglicher Befehle pro Verarbeitungseinheit stark beschränkt ist, gepaart mit der Tatsache, dass dieses System auf einem FPGA oder einem ähnlichen rekonfigurierbaren Chip realisiert wird, hat wichtige positive Auswirkungen auf die Leistungsfä-

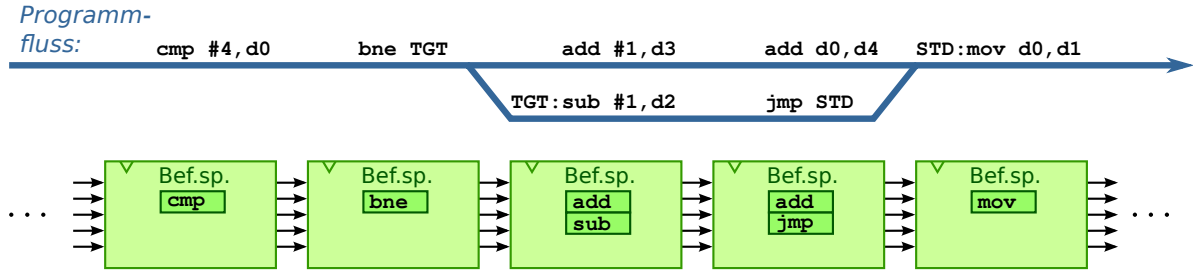

Abbildung 3.12: Verteilung eines Programms auf die Verarbeitungseinheiten der Datenfluss-Pipeline mit $s = 1$, d. h. ein Befehl pro Einheit

higkeit und den Ressourcenbedarf eines solchen Moduls. Nach Programmieren einer Funktion für dieses Modul, kann nämlich bestimmt werden, welche (wenigen) Operationen mit welchen Operanden jede einzelne Einheit jeweils lediglich ausführen können muss. Entsprechend kann jede Einheit darauf spezialisiert werden. Anstatt eines großen Multiplexers zur Auswahl eines der vielen Registerwerte zur Durchführung einer Operation wird nur noch ein kleiner Multiplexer benötigt, der aus wenigen möglichen Registern Daten auswählen können muss. Genauso muss jedes Rechenwerk nur noch einige wenige Operationen ausführen können. Daher sind mit einem solchen Modul im Vergleich zu einem Prozessor-Pool-Modul wesentlich höhere Durchsätze mit geringerem Ressourcenbedarf zu erwarten.

3.6 Nutzung der Flexibilität

Dieses Unterkapitel beschreibt, wie man ein Paketverarbeitungssystem mit der in den letzten Unterkapiteln beschriebenen modularen Architektur flexibel an neue Anforderungen anpassen kann. Zunächst sollen einige Szenarien für die Anpassung eines solchen Systems beschrieben werden, bevor in Abschnitt 3.6.2 auf die Auswahl eines passenden Verarbeitungsmoduls für die Implementierung neuer Funktionen eingegangen wird. In Abschnitt 3.6.3 wird das prinzipielle Vorgehen für die Anpassung bestehender bzw. die Implementierung neuer Funktionen sowie die Integration solcher geänderter bzw. neuer Module in die Verarbeitungs-Pipeline dargestellt. Daran anschließend beschreibt Abschnitt 3.6.4 Werkzeuge, die den Entwickler bei der Anpassung bzw. beim Entwurf und bei der Integration unterstützen können.

3.6.1 Szenarien

Ein Paketverarbeitungssystem mit der vorgestellten Architektur ist aus einer Pipeline verschiedener Verarbeitungsmodule aufgebaut. Diese führen nacheinander die auszuführenden Paketverarbeitungsfunktionen aus, wobei diese je nach Modulart als Software in Assembler oder einer höheren Sprache programmiert wurden, als programmierbarere Logikfunktion mittels einer Hardware-Beschreibungssprache beschrieben wurden oder mit nur wenigen Anpassungsmöglichkeiten in einem Spezialfunktionsmodul realisiert sind.

In der Grundkonfiguration wird ein solches System i. d. R. hauptsächlich aus Programmierbare-Logik-Modulen sowie aus Spezialfunktionsmodulen aufgebaut sein. Der Grund hierfür ist, dass

solche Module ressourceneffizienter sind und somit mehr Freiraum für mögliche zukünftige Erweiterungen auf dem Chip lassen. Falls die Chipfläche keinen begrenzenden Faktor darstellt, sind ebenso Grundkonfigurationen mit Software-basierten Modulen mit ihren entsprechenden Vorteilen denkbar.

Wenn ein System an neue Anforderungen angepasst werden soll, kann dies in Abhängigkeit von den Randbedingungen durch Anpassen der Funktion eines Moduls, Ersetzen eines Moduls oder Hinzufügen eines Moduls mit einer neuen Funktion erfolgen. Die Funktion eines neuen Moduls kann bereits fertig implementiert sein und muss nur noch über Parameter angepasst werden oder sie muss noch neu entworfen und implementiert werden.

Anpassung eines Moduls – In manchen Fällen erfordern neue Anforderungen nur geringfügige Änderungen an der Funktion eines bestehenden Moduls. Hier kann der Quellcode des betroffenen Moduls – in der entsprechenden Entwurfssprache (VHDL, C, Assembler, Mikrocode, etc.) – direkt modifiziert werden bzw. dessen Parameter angepasst werden. Bei größeren erforderlichen Anpassungen kann ein Entwickler die Funktion eines bestehenden Moduls auch komplett überarbeiten oder als Basis für eine neue Implementierung verwenden.

Modul mit fertiger Implementierung – Falls eine neue Funktion in das System integriert werden soll oder ein bestehendes Modul gegen ein anderes mit einer verbesserten oder angepassten Funktion oder Leistungsfähigkeit ersetzt werden soll, können hierfür u. U. Module mit bereits fertig implementierter Funktionalität erhältlich sein. Dies können sowohl Spezialfunktionsmodule mit Parametern zur Anpassung sein als auch Module, deren Funktion in Hardware (d. h. mittels programmierbarer Logik) oder Software implementiert wurde. Eine andere Abteilung, z. B. aus der Forschung und Vorausentwicklung, könnte dieses Modul bereits implementiert und getestet haben. Oder es könnten, wie bereits in Abschnitt 3.5.2 angeführt, andere Firmen ein solches Modul entwickelt haben und den Quellcode der Funktion oder die Netzliste des Moduls (u. U. in verschlüsselter Form) als *Intellectual Property* verkaufen. Ebenso könnten Forschungseinrichtungen, wie Universitäten, aber auch andere Unternehmen Module mit implementierten Funktionen kostenlos als *Open-Source*-Module bereitstellen.

Solche Module können einerseits Standardfunktionen realisieren andererseits aber auch neuartige Funktionen von z. B. noch in der Forschung oder der Standardisierung befindlichen neuen Protokollen, Protokollerweiterungen oder Diensten. Bei verfügbarem Quellcode können solche Funktionen noch zusätzlich an eigene Anforderungen angepasst oder in eigene Implementierungen integriert werden, ansonsten ist eine Anpassung über Parameter vorstellbar. Aufgrund der einheitlichen Schnittstellen zwischen den Modulen, können solche „fremden“ Module ohne Weiteres in das Paketverarbeitungssystem integriert werden.

Modul mit neuer Implementierung – Eine weitere wichtige Möglichkeit ist, eine neue oder angepasste Funktion selbst neu zu implementieren. Hierzu kann eines der zur Verfügung stehenden Verarbeitungsmodule verwendet werden. Bei der Auswahl, welche Art von Modul verwendet werden soll, können Aspekte wie die zur Verfügung stehenden Ressourcen, die geforderte Leistungsfähigkeit aber auch die erforderliche Entwicklungszeit in Betracht gezogen werden. Auf diese Auswahlkriterien wird im folgenden Abschnitt eingegangen.

Kombination verschiedener Module – Schließlich können für die Realisierung einer neuen Funktion auch verschiedene Module miteinander kombiniert werden. Beispielsweise kann ein bestehendes eigenes oder fremdes Modul mit einem eigenen neuen Modul kombiniert werden, das dessen Funktionalität ergänzt. Oder falls für die Realisierung spezieller Operationen eine Implementierung in programmierbarer Logik vorzuziehen ist, können diese Operationen isoliert in einem Hardware-Modul realisiert werden und alle anderen Funktionen in voran- und/oder nachgestellten Software-Modulen.

3.6.2 Modulauswahl

Wenn eine Funktion neu implementiert werden soll, kann zwischen verschiedenen Verarbeitungsmodulen mit unterschiedlichen Entwurfsebenen gewählt werden. Bei der Auswahl können mehrere Aspekte ausschlaggebend sein. Diese wurden im Wesentlichen bereits in Unterkapitel 3.1 im Zusammenhang mit den verschiedenen Hardware- und Software-Entwurfs- und Abstraktionsebenen diskutiert.

Ein Aspekt ist die verfügbare **Entwicklungszeit.** Soll die Funktion möglichst schnell und mit wenig Arbeitsaufwand zum Einsatz kommen, um z. B. einen bestimmten Dienst vor der Konkurrenz anbieten zu können, oder eine Funktion in kurzer Zeit in unterschiedlichen Varianten zu testen, dann ist eine Implementierung in Software vorzuziehen. Diese ist aufgrund der höheren Abstraktionsebene schneller zu realisieren als eine Implementierung auf Register-Transfer-Ebene. Ebenso wäre zukünftig eine Implementierung auf der Hardware-Verhaltensebene sinnvoll. Der Nachteil der hohen Abstraktion ist ein höherer Ressourcenverbrauch auf dem Chip und u. U. eine suboptimale Leistungsfähigkeit.

Falls die Funktion möglichst **platzsparend** integriert werden soll, um Platz für andere Erweiterungen zu lassen, ist eine ressourcenschonendere Implementierung z. B. mit einem Programmierbare-Logik-Modul oder ähnlichem das geeignetere Mittel. Eine solche Implementierung könnte eine Firma beispielsweise einsetzen, nachdem sich eine zunächst aus wirtschaftlichen Interessen eilig in Software realisierte Funktion schließlich etabliert hat, und nun die Chipfläche wieder effizienter genutzt werden soll, um damit Platz für neue Erweiterungen zu schaffen. Oder eine Firma implementiert die beste von mehreren zuvor in Software realisierten Funktionsvarianten schließlich für den tatsächlichen Betrieb (evtl. mit zusätzlichen Merkmalen) in Hardware. Auch der Einsatz von Spezialfunktionsmodulen ist sehr ressourceneffizient.

Für eine sehr **leistungsstarke** Realisierung einer Funktion kann sowohl die Entwurfsebene als auch die verfügbare Verarbeitungsinfrastruktur entscheidend sein. Der Entwurf auf einer niederen Software-Ebene und somit Hardware-näher ist meist effizienter als eine Hochsprachenbeschreibung. Außerdem ist entscheidend, wie gut die Verarbeitungsinfrastruktur der verschiedenen Module für ein Problem angepasst ist und somit wie effizient die gewünschte Funktion damit realisiert werden kann. Bei einer Realisierung auf Register-Transfer-Ebene können speziell an die benötigte Funktion angepasste Verarbeitungseinheiten realisiert werden. Auch mit Software-basierten Modulen können hohe Durchsätze erreicht werden, falls sie über einen passenden Befehlssatz verfügen und keine Operationen umständlich emuliert werden müssen. Falls vorhanden, können auch passende Operationen in einem Prozessor-Modul mit konfigurierbarem Befehlssatz integriert werden.

Auch die **Architektur** der Verarbeitungsinfrastrukturen kann mehr oder weniger geeignet sein. Falls z. B. mehrere Operationen parallel auf sehr vielen Paketdaten gleichzeitig ausgeführt werden müssen (z. B. bei Verschlüsselungs- oder anderen Codierungsfunktionen), ist eine Implementierung in einem Modul mit Prozessorkernen mit jeweils einer Ausführungseinheit ungeeignet. Hier ist eine Realisierung auf einer Hardware-Entwurfsebene vorzuziehen, so dass der Entwickler selbst eine hochparallele Verarbeitungsarchitektur implementieren kann. Ein weiterer wichtiger Punkt, der hinsichtlich der Architektur ausschlaggebend sein kann, ist, inwieweit bei der Verarbeitung mehrerer Pakete der Zugriff auf gemeinsame Daten möglich ist. Hierfür sind ein geteilter Speicher oder sonstige Kommunikationskanäle zwischen den Verarbeitungseinheiten notwendig und eine Pipeline-Konfiguration ist vorzuziehen. Bei Verwendung einer Hardware-Entwurfsebene kann die Verarbeitungsarchitektur wiederum speziell an die Funktion angepasst werden.

Zusammenfassend kann man festhalten, dass je nach geforderter Funktionalität sowie den Randbedingungen bezüglich Entwicklungszeit, Leistungsfähigkeit und Ressourcenbedarf ein anderes passendes Verarbeitungsmodul ausgewählt werden kann. Im nächsten Abschnitt wird dargestellt, wie das prinzipielle Vorgehen ist, um eine neue Verarbeitungsfunktion in einem solchen Modul zu implementieren oder anzupassen und es anschließend in die Pipeline des Paketverarbeitungssystems zu integrieren.

3.6.3 Vorgehen

Abbildung 3.13 zeigt einen Überblick über das Vorgehen für das Anpassen eines Paketverarbeitungssystems mit der in dieser Arbeit vorgestellten Architektur. Der erste Schritt wurde bereits im vorangehenden Abschnitt ausführlich besprochen: Die Überlegung, ob man ein bestehendes eigenes oder fremdes Modul lediglich entsprechend der Anforderungen ändert oder die Funktion neu implementiert. Im zweiten Fall muss man anschließend entscheiden, mit welcher Art von Verarbeitungsmodul man die neue Funktionalität realisiert.

Der zweite Schritt ist die Implementierung der neuen Funktion bzw. die Anpassung des Quellcodes oder der Parameter eines bereits implementierten Verarbeitungsmoduls. Bei der Implementierung bzw. Anpassung eines Moduls in programmierbarer Logik ist darauf zu achten, dass die strengen Schnittstellenanforderungen der Systemarchitektur eingehalten werden. Bei einem Software-basierten Modul sollte die Anzahl der Ausführungstakte möglichst klein sein.

Nach der Implementierung der Funktion in Software muss diese in für die Verarbeitungseinheiten verständlichen Maschinencode umgesetzt werden. In vielen Fällen kann erst jetzt die genaue Ausführungsdauer bestimmt werden. Diese Information kann dafür genutzt werden, für die Synthese des Moduls dessen Anzahl an Verarbeitungseinheiten so hoch festzulegen, dass es den erforderlichen Durchsatz einhalten kann. Vor oder während der Synthese des Moduls können die Verarbeitungseinheiten nun so angepasst werden, dass sie ausschließlich die erforderlichen Operationen ausführen können und keine unnötigen Register, Multiplexer und Rechenwerke beinhalten. Das Syntheseergebnis sagt aus, wie viele Ressourcen für die Umsetzung des Moduls in etwa erforderlich sind. Bei nur kleinen Änderungen der Software kann man u. U. auch das bisherige Verarbeitungsmodul wiederverwenden und nur den Programmcode ausgetauschen. Dies ist allerdings nur möglich, falls das Modul so dimensioniert ist, dass es auch

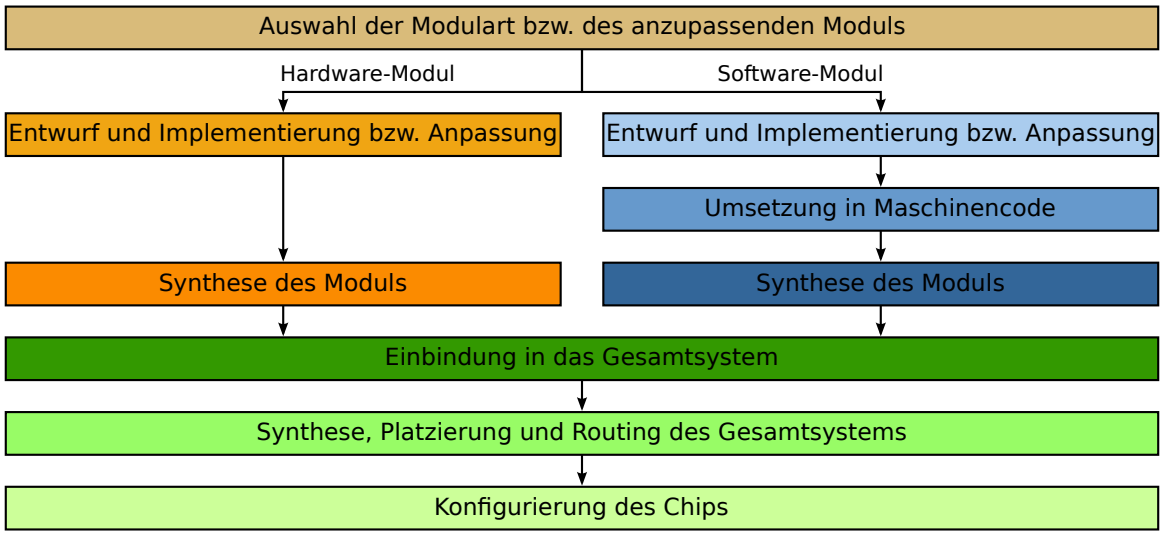

Abbildung 3.13: Schritte bei der Integration bzw. Anpassung einer Verarbeitungsfunktion

die neue Anzahl an Verarbeitungstakten damit ausführen kann, und ihm keine notwendigen Verarbeitungsressourcen aufgrund einer vorhergehenden Optimierung fehlen.

Der Synthese-Schritt bei der Realisierung eines auf programmierbarer Logik basierenden Verarbeitungsmoduls setzt die beschriebene Architektur in Logikelemente um. Aus der vorhergesagten maximal verwendbaren Taktfrequenz kann berechnet werden, ob das Modul die Durchsatzanforderungen einhalten kann. Falls nicht, muss die Implementierung nochmals optimiert werden.

Wenn nach der Synthese des neuen Moduls sichergestellt ist, dass das Modul auf dem Chip realisiert werden kann und auch die Schnittstellenanforderungen einhält, kann man das Modul in die Modul-Pipeline des Gesamtsystems einbinden. Hier muss darauf geachtet werden, dass das Modul an der richtigen Position eingebunden wird, so dass bereits alle erforderlichen Metainformationen vorhanden sind und die Modifikationen des Moduls keine negativen Auswirkungen auf nachfolgende Module haben.

Abschließend muss das gesamte System zusammen – wie bei allen FPGA-Realisierungen – synthetisiert, seine Elemente auf dem Chip platziert und die Verbindungen dazwischen gelegt werden. Dieser Schritt kann beschleunigt werden, indem nur die Elemente des neuen Moduls neu platziert und verbunden werden und für das restliche System die Platzierungsergebnisse von einer vorhergehenden Platzierung wiederverwendet werden (vgl. [167]). Danach kann der Chip mit den erzeugten Konfigurationsdaten eingerichtet werden.

Soll ein bereits fertig implementiertes Modul in die Pipeline integriert werden, können bzw. müssen einige der ersten Schritte übersprungen werden. Je nachdem, ob die neue Funktion in Form von Quellcode, in Form von kompiliertem Maschinencode oder bereits als synthetisierte Netzliste vorliegt, können noch Änderungen darin integriert werden oder kann das Modul direkt in das Gesamtsystem integriert werden.

3.6.4 Werkzeuge

Der im letzten Unterkapitel vorgestellte Ablauf der Integration einer neuen Funktion in das Paketverarbeitungssystem kann durch verschiedene Software-Werkzeuge erleichtert und automatisiert werden. Teilweise müssen diese speziell hierfür entwickelt werden, teilweise können jedoch auch bestehende Werkzeuge eingesetzt werden.

Die Implementierung von Funktionen für die Verarbeitungsmodule kann in bestehenden Entwicklungsumgebungen durchgeführt werden. Hierfür können bestehende Software-Entwicklungsumgebungen wie *Eclipse* [168] bzw. VHDL- und Verilog-Entwicklungsumgebungen wie *HDL Designer* [169] von Mentor Graphics verwendet werden. Erstere muss u. U. an die verwendete Programmiersprache angepasst werden.

Die Umsetzung des Software-Quelltexts in maschinenlesbaren Code kann automatisiert mittels entsprechender Compiler- und Assembler-Programme erfolgen. Diese Programme können die Implementierungen auch auf ihre Ausführbarkeit hin testen, z. B. dass keine unbegrenzt langen Schleifen durchlaufen werden. Nach der Umsetzung in Maschinencode ist auch die maximale Ausführungsdauer und somit die benötigte Anzahl von Verarbeitungseinheiten automatisch ermittelbar. Ebenso kann eine Software ermitteln, welche Module in einer Datenfluss-Pipeline tatsächlich benötigt werden und welche eingespart werden können.

Für das Testen der Implementierung können spezielle, an die jeweilige Modulart angepasste Simulationswerkzeuge entwickelt werden. Alternativ können auch bestehende Simulations-Werkzeuge erweitert werden. Auf einfache Weise kann das VHDL- und Verilog-Simulationswerkzeug *Modelsim* [170] von Mentor Graphics sowohl für den Test von Programmierbare-Logik-Modulen als auch für die taktgenaue Simulation der verschiedenen Software-Module verwendet werden. Hierfür simuliert man entsprechende Modelle der Verarbeitungsmodule zusammen mit dem übersetzten Software-Programmcode.

Die Einbindung eines Moduls in das Gesamtsystem kann ebenfalls durch ein Werkzeug erleichtert werden, welches automatisiert aus einer grafischen oder textuellen Darstellung der Pipeline, die entsprechende Verschaltung der Module durchführt. Für die Synthese- und Place- & Route-Schritte können die für FPGA-Entwicklungen üblichen Werkzeuge von Mentor Graphics [171], Synopsys [172] sowie den FPGA-Herstellern verwendet werden.

3.7 Implementierungsbeispiele

In diesem Unterkapitel sollen zum Abschluss dieses Kapitels zwei beispielhafte Implementierungen von Verarbeitungsmodulen vorgestellt werden: Ein Prozessor-Pool-Modul bestehend aus für die Paketverarbeitung spezialisierten, kleinen RISC-Prozessorkernen und ein Datenfluss-Pipeline-Modul bestehend aus einer Kette von Verarbeitungseinheiten.

3.7.1 Prozessor-Pool-Modul

Der prinzipielle Aufbau des Prozessor-Pool-Moduls entspricht der Darstellung in Abbildung 3.10 auf Seite 109. Nachfolgend wird zunächst auf Architektur, Befehlssatz und Implementierung der eingesetzten Prozessorkerne eingegangen. Anschließend werden die Architektur und Implementierungsdetails der Verteil- und Einsammel-Module beschrieben, die die Schnittstellen zwischen den umgebenden Verarbeitungsmodulen und dem Prozessor-Pool darstellen.

Prozessorkerne

Wichtigstes Entwurfsziel war, dass der Prozessor-Pool trotz seiner Flexibilität einen möglichst hohen Durchsatz unterstützen kann. Die Architektur der verwendeten Prozessorkerne ist von der RISC-Architektur DLX [173] von Hennessy und Patterson abgeleitet. Diese ist im Prinzip eine vereinfachte Variante der MIPS-Architektur und wird aufgrund seiner leicht verständlichen Struktur sehr häufig in Forschung und Lehre [174] eingesetzt. Des Weiteren wurden Teile des Befehlssatzes der Fast-Path-Prozessorkerne der Intel IXP2xxx-Netzprozessorfamilie [89] (vgl. Abschnitt 2.4.1) integriert, damit die Prozessorkerne Paketverarbeitungsfunktionen möglichst effizient durchführen können.

Die 32-bit-Architektur basiert auf einer klassischen fünfstufigen Pipeline, bestehend aus *Instruction-Fetch*-Stufe (Befehl Holen), *Decode*-Stufe (Dekodieren), *Execute*-Stufe (Ausführen), *Memory*-Stufe (Speicher) und *Write-Back*-Stufe (Zurückschreiben). Falls keine Abhängigkeiten zwischen Befehlen ein Anhalten der Pipeline oder das Löschen von Befehlen erforderlich macht, unterstützt die Architektur einen Befehlsdurchsatz von einem Befehl pro Takt (= 1 IPC, *Instructions Per Clock cycle*). Hierfür wurden verschiedene Maßnahmen getroffen, auf die im Folgenden näher eingegangen wird. Abbildung 3.14 zeigt den Aufbau der fünfstufigen Prozessor-Pipeline, wobei aus Gründen der Übersichtlichkeit nur die Datenpfade abgebildet sind.

Damit keine strukturellen Abhängigkeiten zwischen der Instruction-Fetch- und der Memory-Stufe möglich sind, verfügt die Prozessorarchitektur über getrennte Programm- und Datenspeicher (*Harvard*-Architektur). Um einen Zugriff garantiert innerhalb einer Taktperiode durchführen zu können, wurden beide Speicher als kleine, lokale *On-Chip*-Speicher realisiert. Somit muss die Prozessor-Pipeline nicht aufgrund von Speicherlatenzen anhalten und kann die begrenzte Ausführungszeit effizient nutzen. Eine kleine Speicherkapazität ist ausreichend zur Speicherung des Programms, dessen Ausführungsdauer sowieso aufgrund der begrenzten Chipfläche und des geforderten Durchsatzes beschränkt ist. Auch ein kleiner Datenspeicher ist für viele Verarbeitungsfunktionen in Hochgeschwindigkeitsnetzknoten ausreichend. Ausnahmen sind die IP-Adresssuche sowie Funktionen, die eine Zustandshaltung für mehrere Zehntausende von unterschiedlichen Flüssen oder Verkehrsklassen erfordern. Diese Funktionen sind jedoch aufgrund ihrer hohen Speicherzugriffsanforderungen prinzipiell schlecht für eine Software-Realisierung auf einem Hochgeschwindigkeits-Router geeignet. Hierfür bietet sich auf der vorgeschlagenen Architektur eine spezialisierte Implementierung in programmierbarer Logik oder durch eine Kombination von Hardware- und Software an. Die kleinen, schnellen Programm- und Datenspeicher ersetzen somit Cache-Speicher. Sie begrenzen die mögliche Speichermenge hart, garantieren dafür jedoch in jedem Fall eine feste Zugriffslatenz von einem Takt.

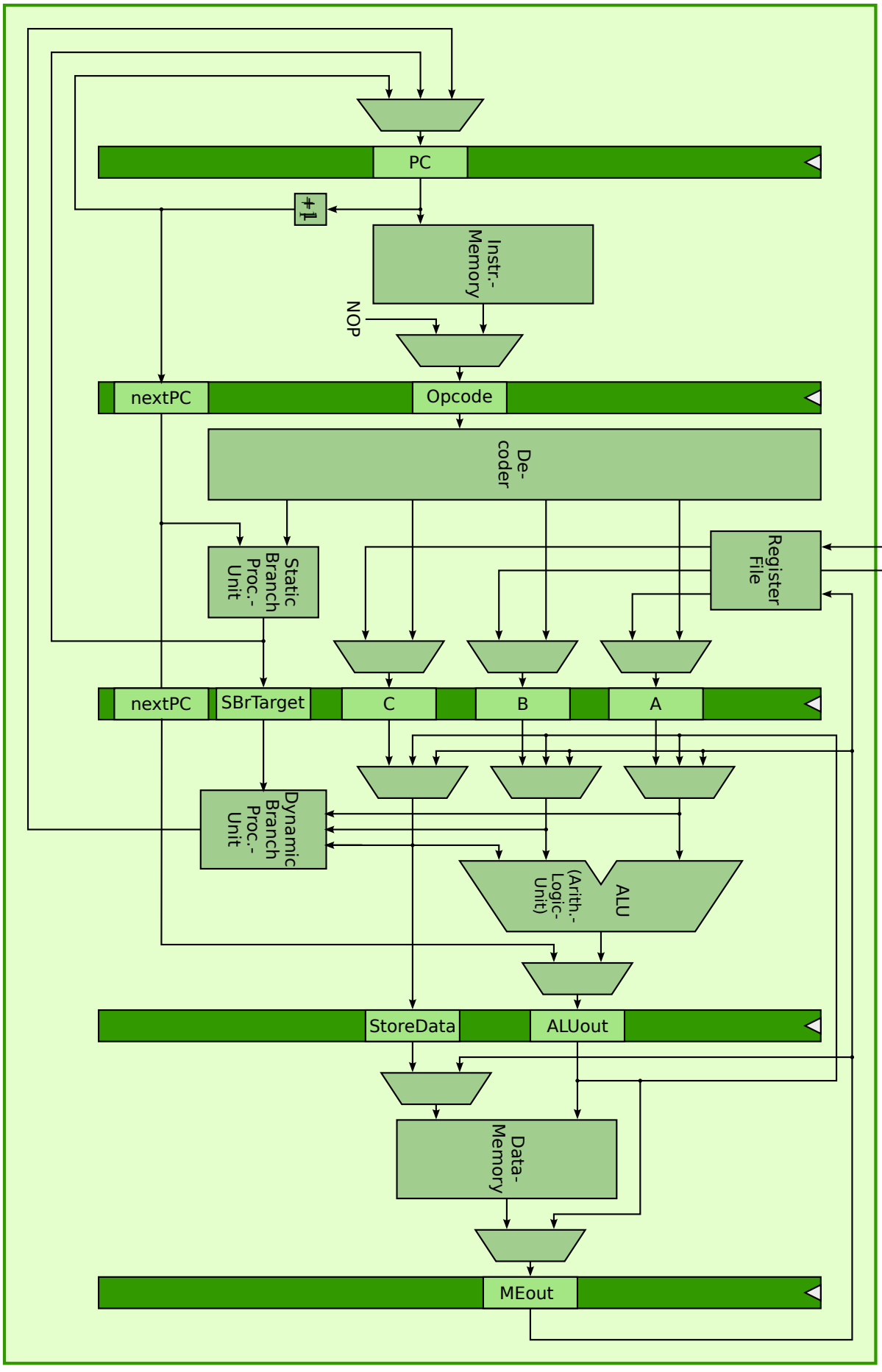

Abbildung 3.14: Fünfstufige Pipeline des RISC-Prozessors

Datenabhängigkeiten erkennt die Decode-Stufe automatisch und löst diese soweit möglich durch *Forwarding* auf. Bei Abhängigkeiten, die so nicht gelöst werden können, hält der Dekoder die Pipeline für maximal einen Takt an. Das Erkennen der Abhängigkeiten ist bei der verwendeten Architektur bereits vor der Ausführung möglich. Somit kann die Ausführungszeit eindeutig im Voraus berechnet werden. Außerdem kann versucht werden, nicht durch Forwarding auflösbare Abhängigkeiten durch Umsortieren von Befehlen aufzulösen und somit die Ausführungszeit zu optimieren. Vorhandene Datenabhängigkeiten könnte die Assembler-Software auch direkt in den Programmcode schreiben und somit die Dekodierstufe entlasten. Dies würde sich positiv auf die benötigte Chipfläche und u. U. den Durchsatz auswirken. Hierauf wurde jedoch verzichtet, da das Prozessor-Pool-Modul auch ohne aufwändige Compiler- und Assembler-Software schnell einsetzbar sein sollte.

Sprungabhängigkeiten werden durch Löschen der fälschlich geladenen Befehle von der Pipeline selbstständig aufgelöst (*cancelling branches*). Zusätzlich verfügt die Prozessorarchitektur über einen Mechanismus, dieses Löschen durch einen speziellen Befehl zu unterdrücken, um auch die Takte nach Sprungbefehlen sinnvoll nutzen zu können und so einen hohen Befehlsdurchsatz erzielen zu können. Hier können Befehle ausgeführt werden, die unabhängig vom Sprung ausgeführt werden müssen, oder Befehle, die, falls sie fälschlich ausgeführt werden, keine Auswirkungen auf die Programmausführung haben, z. B. das Beschreiben eines Registerwerts, der nicht mehr benötigt wird.

Der Befehlssatz der Prozessorkerne ist für die schnelle Paketverarbeitung spezialisiert. Um Bitfelder in Paket-Headern effizient verarbeiten zu können, unterstützt die Arithmetisch-Logische Einheit (*Arithmetic-Logic Unit*, ALU) der Ausführungsstufe Schiebeoperationen um eine beliebige Stellenzahl. Des Weiteren kann sie auch einen der Operanden vor einer logischen Operation beliebig verschieben. Außerdem kann die ALU außer auf ganzen Worten auch viele Operationen nur auf einzelnen Bytes oder Halbworten ausführen. Um kurze Programmausführungszeiten erreichen zu können, unterstützt die Sprungauswerteeinheit bedingte Verzweigungen in Abhängigkeit einzelner Bits oder Bytes. Tabelle 3.3 gibt einen Überblick über alle unterstützten Befehle, zusätzlich ist in Anhang A das Programmiermodell inklusive des vollständigen Befehlssatzes abgedruckt.

Ein Prozessorkern verfügt über einen Registersatz von 64 32-bit Registern. Damit bei der Programmausführung keine Zeit für das Laden und Speichern von Paket- und Metadaten benötigt wird, wurde der Registersatz so entworfen, dass das Verteilmodul diese Daten direkt in die Register schreiben und das Einsammelmodul diese nach Programmausführung wieder auslesen kann. Hierfür verfügt der Registersatz über eine zusätzlichen Schreib-/Leseschnittstelle. Damit das Schreiben und Lesen möglichst wenig Zeit in Anspruch nimmt, ist diese Schnittstelle so ausgelegt, dass sie den parallelen Zugriff auf mehrere Register erlaubt. Die genaue Breite der Schnittstelle lässt sich über einen Parameter vor der Synthese des Moduls entsprechend der Randbedingungen des verwendeten Chips sowie der zu realisierenden Funktion einstellen. Weiterhin lässt sich auch die Anzahl an Registern auf 32 statt 64 parametrisieren, um Ressourcen einsparen zu können.

Verteil- und Einsammellogik

Jeder Prozessorkern hat $p \cdot s$ Takte für die Verarbeitung eines Paket-Datenwortes Zeit, wenn das Prozessor-Pool-Modul über p Kerne und eine s-fache Taktfrequenz im Vergleich zum System-Pipeline-Takt verfügt (vgl. Gleichung (3.14) in Abschnitt 3.5.4). Diese $p \cdot s$ Takte werden allerdings außer für die Ausführung des Programmcodes auch für das Kopieren der Paket- und Metadaten in den Registersatz und am Ende wieder zurück an den Modulausgang benötigt. Daher muss auch dieses Kopieren möglichst schnell erfolgen. Abbildung 3.15 zeigt ein Blockschaltbild des entworfenen Prozessor-Pool-Moduls mit besonderem Fokus auf dem Aufbau der Verteil- und Einsammellogik, die für diese Kopieraktionen verantwortlich ist.

Tabelle 3.3: Überblick über den Befehlssatz der RISC-Kerne im Prozessor-Pool-Modul

Befehlsart	**Besonderheiten**	**Mnemonics**
Schiebe-/Rotier-operationen	um beliebige Stellenzahl, über Registergrenzen hinweg	`shf, shfi, dblshf`
Logische Operationen	vorangestelltes Schieben und Invertieren eines Operanden	`not, and, or, xor`
Arithmetische Operationen	Addition von Teilworten, 16-bit Multiplikation	`add, add8, add16, addc, addc8, addc16, addi, addci, sub, subc, subi, subci, mult16`
Bitoperationen	Suchen des ersten/letzten gesetzten Bits, Bits Zählen, vorzeichenricht. Erweitern	`fflone, ffrone, cntones, ext8, ext16`
Initialisierungs-operationen	Schreiben von Teilworten, Löschen/Setzen/Beibehalten der anderen Bereiche	`initclr, initclr8, initclr16, initfil, initfil8, initfil16, initw0, initw1, initb0, initb1, initb2, initb3`
Datenbewegeoperationen	Schreiben von Teilworten und Löschen/Beibehalten der anderen Bereiche	`movb, movbclr`
Speicherzugriffsbefehle	wortweise Adressierung mit Index oder Offset	`ldd, ldr, std, str`
Verzweigungs- und Sprungbefehle	bedingte Verzweigung prüft Flag-Kombinationen, einzelne Bits und beliebige Byte-Muster	`b<cc> (16 versch. <cc>), bbclr, bbset, brb0eq, brb1eq, brb2eq, brb3eq, brb0ne, brb1ne, brb2ne, brb3ne, br, bal, jmp, jal`

Die Daten sollen reihum in jedem Pipeline-Takt in einen anderen Prozessorkern übertragen werden. Allerdings ist der Durchsatz der Registersatzschnittstelle aufgrund ihrer begrenzten Datenbreite beschränkt. Daher ist es bei einem niedrigen Verhältnis s von Prozessortakt zu Pipeline-Takt nicht immer möglich, alle Daten innerhalb eines Pipeline-Taktes zu kopieren. Aus diesem Grund verfügt das Modul über einen Eingangs- und einen Ausgangspuffer, in denen die Paket- und Metadaten zwischengespeichert werden. So können die Daten über mehrere Takte hinweg kopiert werden, während überlappend dazu in jedem Pipeline-Takt mit dem Kopieren in und aus einem neuen Prozessorkern begonnen werden kann. Die Puffer sind im Bild links und rechts zu sehen. Sie sind durch hintereinander geschaltete Register realisiert, deren Größe davon abhängt, wie viele Daten pro Pipeline-Takt durch die Schreib-/Leseschnittstelle des Prozessor-Registersatzes transportiert werden können. Da der Kopierprozess deterministisch abläuft, können alle Register genau passend dimensioniert werden.

Ein Kopiermodul steuert den Kopierprozess für einen Prozessorkern. Die Daten werden hierfür jeweils in Fragmenten transportiert, die der Breite der Datenschnittstelle des Prozessors entsprechen. Hierfür verfügt das Modul über einen kleinen Zähler, der einen Multiplexer ansteuert, an den die Datenfragmente des Eingangspuffers angeschlossen sind. So legt das Modul nacheinander die einzelnen Fragmente an die Schnittstelle des Prozessors an. Gleichzeitig steuert es den Prozessorkern mit der Nummer *addr* des aktuell anliegenden Fragments und einem Freigabesignal *ena* an, damit dieser die Daten in seinem Registersatz übernimmt. Parallel zum Einlesen der neuen Daten erfolgt auch das Auslesen des zuletzt verarbeiteten Pakets. Damit dieses an die richtige Stelle im Ausgangspuffer gespeichert wird, steuert das Kopiermodul auch eine Schaltmatrix mit der aktuellen Fragmentnummer und dem Freigabesignal an. Durch das parallele Auslesen und Schreiben wirkt sich das Kopieren nur einfach auf die zur Verfügung stehende Verarbeitungszeit im Prozessor aus.

Da sich die Kopierprozesse mehrerer Prozessorkerne überlappen können, verfügt jeder Kern über ein eigenes Kopiermodul. Prinzipiell könnte auch ein Modul mehrere Kerne ansteuern, aufgrund des einfachen und ressourcenschonenden Aufbaus des oben beschriebenen Kopiermoduls wurde jedoch diese Lösung gewählt. Entsprechend ist jedes Kopiermodul wie im Bild angedeutet mit dem Eingangspuffer verbunden. Außerdem ist der Ausgang jedes Prozessorkerns zusammen mit den Steuerleitungen seines Kopiermoduls mit der Schaltmatrix vor dem Ausgangspuffer verbunden.

Ein Round-Robin-Scheduler steht über dieser Verteil- und Einsammellogik. Dieser aktiviert reihum in jedem Pipeline-Takt eines der Kopiermodule, damit dieses die neuen Daten in seinen Prozessorkern kopiert und dessen verarbeitete Daten ausliest.

Aufgrund des überlappenden Kopierens der Paket- und Metadaten in die Prozessorkerne ist dieser Vorgang sehr komplex und soll hier nicht näher dargestellt werden. Beispielhaft zeigt Abbildung 3.15 das parallele Auslesen der Pakete B, C und D aus den drei Prozessorkernen, wo sie zuvor verarbeitet wurden, und das parallele Kopieren der Pakete P, Q und R in diese Kerne. Jedes Paket wird in neun Fragmenten übertragen. Vorstellbar wäre somit eine Konfiguration mit 512 bit breiten Paketdatenworten und 64 bit breiten Metadaten bei einer Schnittstellenbreite von 64 bit (512 bit + 64 bit = $9 \cdot 64$ bit). Der Prozessortakt ist $s = 4$ mal so schnell wie der Pipeline-Takt, so dass bis zu vier Fragmente pro Pipeline-Takt übertragen werden können. Nach dem Start des Kopiermoduls durch den Scheduler kann das Kopiermodul im ersten Pipeline-Takt noch drei Fragmente in den Prozessor übertragen, im zweiten Takt überträgt es vier Frag-

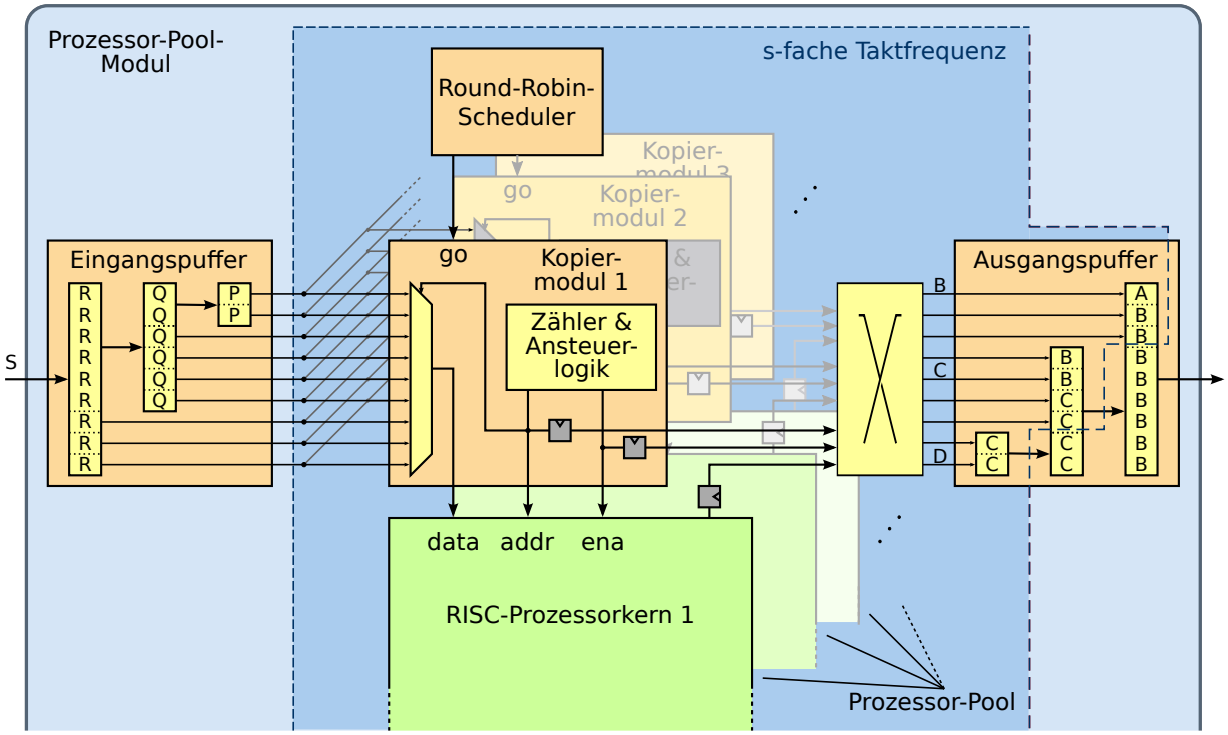

Abbildung 3.15: Verteil-Einsammel-Logik des Prozessor-Pool-Moduls

mente und im dritten Takt die letzten zwei. Die ausgelesenen Daten werden um eine Prozessor-Taktperiode verzögert, so dass hier im ersten Pipeline-Takt zwei, dann vier und schließlich die letzten drei Fragmente in den Ausgangspuffer kopiert werden. Im Beispiel ist die dritte Prozessor-Taktperiode dargestellt. Nach dieser wird das vollständige Paket B am Ausgang bereit stehen – rechtzeitig vor der nächsten Taktflanke des Pipeline-Taktes.

Die Verteil- und Einsammel-Logik wurde so entworfen und in VHDL implementiert, dass sie durch Einstellen von entsprechenden Parametern eine beliebige Anzahl von Prozessorkernen, ein beliebiges, ganzzahliges Verhältnis s von Modultakt zu Pipeline-Takt, sowie eine beliebige Schnittstellenbreite unterstützt. Ebenso kann auch die verwendete Datenwort- und Metadatenwortbreite eingestellt werden.

3.7.2 Datenfluss-Pipeline-Modul

Die Implementierung eines Datenfluss-Pipeline-Moduls hat prinzipiell denselben Aufbau und dieselbe Funktionsweise wie die in Abschnitt 3.5.5 vorgestellte Architektur. Der Aufbau entspricht somit im Wesentlichen dem in Abbildung 3.11 auf Seite 111, d. h. das Modul besteht aus einer Pipeline von Verarbeitungseinheiten, durch die die Datenworte der Pakete mit den Metadaten wandern. In Abschnitt 3.5.5 wandern die Daten alle s Takte in die nächste Einheit weiter. Dies bedeutet, dass die interne Taktfrequenz s-mal so hoch wie die Systemtaktfrequenz ist. Um die zeitlichen Anforderungen gering zu halten, wurde für diese Implementierung $s = 1$ gewählt, d. h. die interne Taktfrequenz entspricht der Frequenz des Gesamtsystems und die Daten wandern in jedem Takt weiter. Somit hat jede Einheit eine Taktperiode für die Verarbeitung eines Datenwortes Zeit.

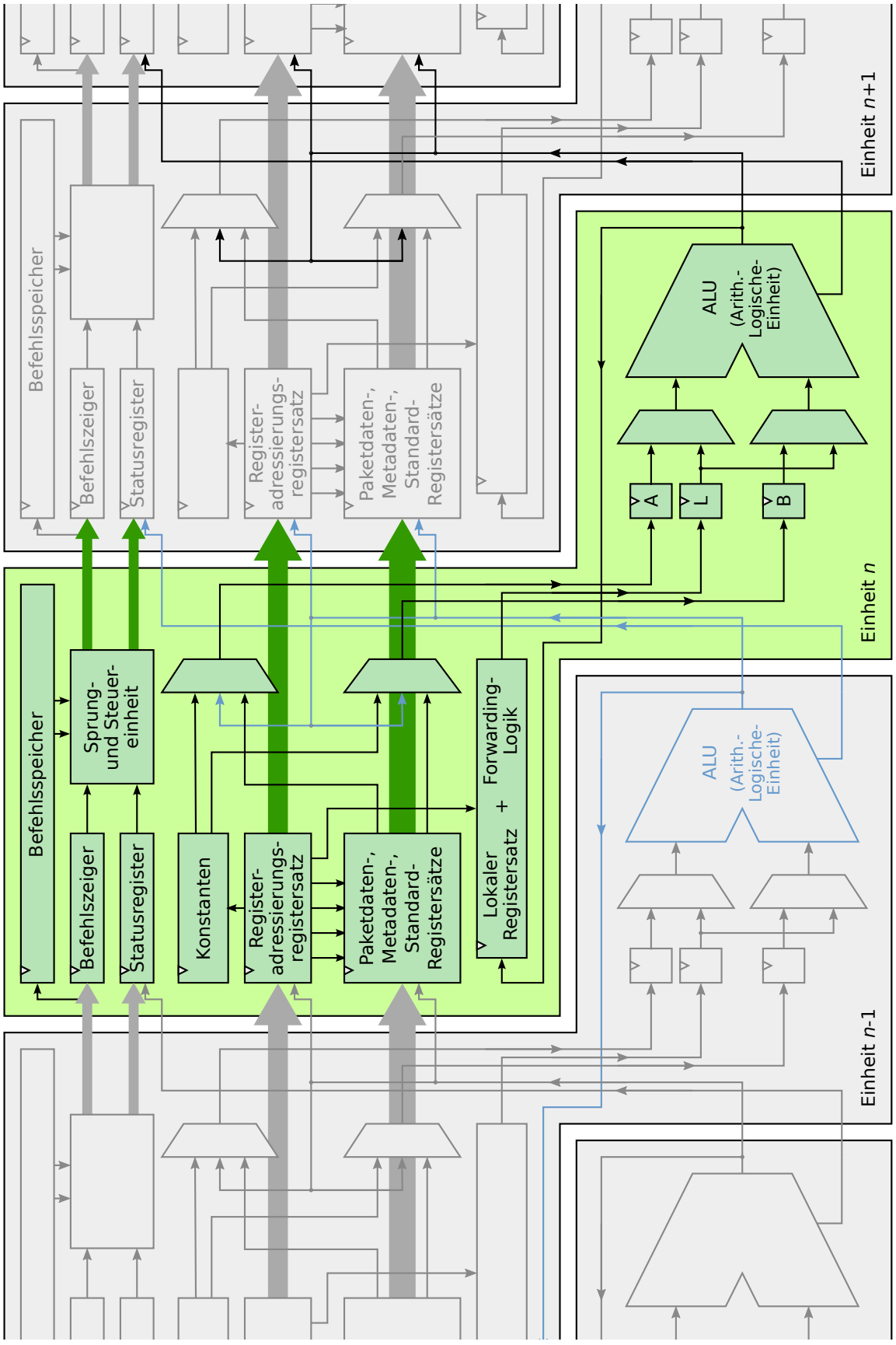

Abbildung 3.16: Verarbeitungseinheit des Datenfluss-Pipeline-Moduls (oben: Ladestufe, unten: Ausführungsstufe)

Abbildung 3.16 zeigt die implementierte Architektur einer Verarbeitungseinheit, sowie deren Anbindung an ihren Vorgänger und ihren Nachfolger. Eine solche Einheit hat effektiv eine Taktperiode für die Verarbeitung Zeit. Dennoch müssen die üblichen Ausführungsschritte eines Prozessors durchgeführt werden: Das Laden des Befehls, das Dekodieren desselben, die eigentliche Operationsausführung sowie das Schreiben des Ergebnisses. Damit die Einheit dennoch mit einer hohen Taktfrequenz arbeiten kann, ist die Verarbeitung innerhalb einer Einheit zweistufig als „Mini-Pipeline“ realisiert. Diese besteht aus der Ladestufe, die im Bild jeweils oben dargestellt ist, und der Ausführungsstufe, im Bild jeweils unten.

Die Ladestufe enthält als wesentliche Elemente einen Befehlsspeicher und diverse Registersätze. In Abschnitt 3.5.5 wurde gezeigt, dass ersterer i. d. R. nur wenige Einträge enthalten muss. Daher wurde dieser Speicher sehr breit ausgelegt, so dass die Befehlsworte direkt alle Steuerleitungen der Einheit enthalten. Eine solche Codierung entspricht der von horizontalem Mikrocode. Somit kann auf eine Dekodierschaltung verzichtet werden, was sich positiv auf die maximal mögliche Taktfrequenz auswirkt.

Der Paketdaten-, Metadaten- und Standard-Registersatz enthält die zum aktuellen Datenwort gehörenden Daten, wobei der Standard-Registersatz zur Ablage von Zwischenergebnissen gedacht ist. Alle in diesen Registersätzen enthaltenen Daten werden in jedem Takt von einer Verarbeitungseinheit zur nächsten transportiert. Innerhalb der Stufe können die Daten über Multiplexer als Operanden für die Ausführungsstufe ausgewählt werden.

Die Verarbeitungseinheiten enthalten keinen Datenspeicher, um Daten zu speichern, die lokal im Prozessor verbleiben und nicht weitergereicht werden. Der Grund hierfür ist, dass der Speicherzugriff allein bereits eine Taktperiode benötigen würde und die Mini-Pipeline somit um eine Stufe verlängert werden müsste. Alternativ dazu besitzt jede Einheit zwei weitere Registersätze: Den Konstanten-Registersatz, der nur gelesen werden kann, kann man für die Speicherung unmittelbarer Operanden verwenden. Den lokalen Registersatz mit Lese- und Schreibmöglichkeit kann man für Daten verwenden, die bei der Verarbeitung mehrerer Datenworte benötigt werden, wie z. B. für Zähler für Paketstatistiken.

Damit die Registersätze wie Datenspeicher verwendet werden können, muss auch eine indirekte Adressierung der darin gespeicherten Daten möglich sein. Um dies zu ermöglichen, verfügt jeder Registersatz über ein bis zwei so genannte Registeradressierungsregister. Darin kann man zur Laufzeit berechnete Indizes ablegen und diese in einem nachfolgenden Befehl zur Adressierung eines der Register verwenden.

Die Ausführungsstufe enthält im Wesentlichen eine ALU, die arithmetische und logische Operationen sowie Schiebeoperationen ausführen kann. Eine Übersicht der möglichen ALU-Operationen ist in Tabelle 3.4 angegeben. Anhang B beschreibt das Programmiermodell sowie alle möglichen Operationen im Detail.

Das ALU-Ergebnis der Einheit n wird je nach Zielregister zur nächsten positiven Taktflanke in der Verarbeitungseinheit $n+2$ in den entsprechenden Registersatz zurückgeschrieben (vgl. Abbildung 3.16). Damit kann das Ergebnis eines Befehls jedoch erst im übernächsten Takt wieder aus dem Registersatz gelesen werden. Um das Ergebnis auch direkt mit dem nächsten Befehl weiterverwenden zu können, wird es auch direkt zu den Operanden-Auswahlmultiplexern der

nachfolgenden Einheit $n+1$ geleitet. Dort kann der Programmierer es explizit mit dem nächsten Befehl auswählen.

Soll bei der Verarbeitung eines Datenwortes ein lokales Register verwendet werden, so kann der Programmierer nicht wissen, ob bei der Verarbeitung des vorhergehenden Datenworts dasselbe lokale Register Zielregister war. Daher wurde hier eine Forwarding-Logik implementiert, die bei solch einer Lese-Schreib-Abhängigkeit (*read-after-write dependency*) das zu schreibende Ergebnis zusätzlich am lokalen Register vorbei führt und direkt wieder als Operanden ausgibt. Die Registeradressierungsergebnisse sowie die Statusflags der ALU werden aus Ressourcen- und Durchsatzgründen nicht direkt zurückgeleitet und können somit erst mit dem zweiten Befehl nach dem erzeugenden Befehl gelesen bzw. ausgewertet werden.

Schließlich soll die Realisierung von Sprüngen betrachtet werden. Um die bei RISC-Pipelines üblichen Sprungabhängigkeiten und das damit notwendige Löschen von Befehlen zu vermeiden, werden Sprünge in der implementierten Datenfluss-Pipeline-Verarbeitungseinheit bereits in der Ladestufe ausgeführt. Bei bedingten Sprüngen wertet eine Sprung- und Steuereinheit die Flags aus dem Statusregister aus und leitet in Abhängigkeit davon entweder den standardmäßig verwendeten Befehlsfolgezähler oder einen Befehlsfolgezähler mit dem Sprungziel an die nächste Verarbeitungseinheit weiter. Beide sind absolut im Opcode angegeben. Unbedingte Sprünge werden durch entsprechendes Setzen des nächsten Befehlsfolgezählers realisiert. Die Auswertung und Ausführung von Sprüngen erfolgt völlig unabhängig von der Ausführung der ALU-Operationen. Daher können diese parallel ausgeführt werden. Bei gleichzeitiger Ausführung einer ALU-Operation haben Sprünge somit keinerlei Auswirkung auf die Programmausführungszeit.

Wie in Abschnitt 3.5.5 bereits beschrieben, ist es sinnvoll, nicht das gesamte Datenwort von 512 bit Breite oder mehr durch die Pipeline zu transportieren, da bei der Verarbeitung i. d. R. nicht immer alle Daten gleichzeitig verwendet werden. Daher wurde das implementierte Datenfluss-Pipeline-Modul so entworfen, dass jeweils nur ein Teil, typischerweise 128 oder 256 bit, durch die Register der Verarbeitungseinheiten wandern. Die restlichen Daten speichert ein Ringpuffer. Dennoch soll ein Programmierer die Möglichkeit haben, auf alle Daten eines Datenwor-

Tabelle 3.4: Überblick über die ALU-Operationen der Verarbeitungseinheiten im Datenfluss-Pipeline-Modul

Operationsart	**Ausprägung**	**Mnemonics**
Schiebe-/Rotier-operationen	links/rechts um jeweils $n = 1, 2, 4, 8$ bit	`alu_lsl<n>, alu_lsr<n>, alu_rol<n>, alu_ror<n>`
Logische Operationen	Nicht, Und, Oder, Exklusiv-Oder	`alu_not, alu_and, alu_or, alu_xor`
Arithmetische Operationen	Addition/Subtraktion mit und ohne Übertragsbit	`alu_axc, alu_axyc, alu_axy, alu_sxy, alu_sxyc, alu_sxy`

tes zugreifen zu können. Hierfür wurden Datenaustauschmodule entworfen und implementiert, die an beliebigen Stellen zwischen den Verarbeitungseinheiten platziert werden können. Welche Teil-Datenworte jeweils in einem Segment von Verarbeitungseinheiten verwendet wird, kann vom Programmierer bestimmt werden. Es ist auch möglich, dass verschiedene Ausführungszweige im gleichen Segment unterschiedliche Teil-Datenworte verwenden.

Das Datenfluss-Pipeline-Modul wurde vollständig in VHDL implementiert. Dabei kann über Parameter die Größe der verschiedenen Registersätze sowie die Größe des Befehlsspeichers eingestellt werden. Auch die Positionen der Datenaustauschmodule können beliebig bestimmt werden, ebenso wie die Breite der auszutauschenden Teil-Datenworte. Zusätzlich besteht die Möglichkeit, den Befehlsspeicher und den Konstanten-Registersatz vor der Synthese des Moduls fest vorzugeben, so dass das Synthese-Werkzeug jedes Modul entsprechend seiner wenigen Befehle optimieren kann. Somit kann das Modul, wie in Unterkapitel 3.6 beschrieben, an die individuellen Randbedingungen einer Verarbeitungsfunktion angepasst werden.

3.8 Zusammenfassung

Dieses Kapitel führte eine neue Architektur für schnelle und flexible Paketverarbeitungssysteme ein. Es wurden die unterschiedlichen Abstraktionsebenen beim Hardware- und Software-Entwurf für solche Systeme diskutiert und aufgezeigt, dass abhängig von den Anforderungen an Funktion, Flexibilität und Leistungsfähigkeit jeweils unterschiedliche Entwurfsebenen die geeignetsten sind. Die neue Architektur wurde daher so entworfen, dass sie Module unterstützt, die unterschiedliche Verarbeitungsinfrastrukturen aufweisen und deren Funktion auf unterschiedlichen Entwurfsebenen implementiert werden. Somit kann für die Realisierung einer neuen Funktion ein an die jeweiligen Randbedingungen angepasstes Verarbeitungsmodul verwendet werden. Dies ist die erste veröffentlichte Architektur, die diese höchste Stufe der Flexibilität unterstützt.

Das zweite wichtige Ziel beim Entwurf der neuen Architektur war, einen hohen Durchsatz garantieren zu können. Daher wurden die beiden prinzipiellen Arten der Parallelisierung auf Ebene der Verarbeitungseinheiten, Pool und Pipeline, analysiert und diskutiert. Beide Konfigurationen weisen Vor- und Nachteile bezüglich Durchsatz, Ressourcenaufwand und Implementierungsaufwand auf, wobei eine Pipeline eine Durchsatz-Garantie ressourceneffizienter realisieren kann und bei konstantem Durchsatz einfach um neue Funktionen erweitert werden kann. Die neue Architektur basiert daher auf einer Pipeline-Struktur. Um einen maximalen Durchsatz zu erreichen, verarbeitet diese ein sehr breites Datenwort von etwa 500 bis 1000 bit pro Taktperiode. Die Verarbeitungsmodule realisieren diese Pipeline-Struktur, indem ihre einheitlichen Schnittstellen denselben Durchsatz garantieren.

Die interne Realisierung der Verarbeitungsmodule ist beliebig. Es wurden verschiedene Arten möglicher Verarbeitungsmodule vorgestellt und insbesondere auf die Architektur von Programmierbare-Logik-Modulen, Prozessor-Pool-Modulen und Datenfluss-Pipeline-Modulen eingegangen. Am Ende dieses Kapitels wurden zusätzlich konkrete Implementierungsbeispiele der beiden letztgenannten Module beschrieben.

Schließlich wurde in diesem Kapitel dargestellt, wie man die Flexibilität eines Paketverarbeitungssystem nach der neuen Architektur nutzen kann. Jedes der Verarbeitungsmodule realisiert eine der auszuführenden Paketverarbeitungsfunktionen. Soll das System an neue Funktionen und Randbedingungen angepasst werden, so kann man die Funktion eines der Module modifizieren, ein Modul durch ein oder mehrere andere Module mit einer anderen Implementierung ersetzen oder ein Modul mit einer neuen Funktion hinzufügen. Hierbei können sowohl Module mit Implementierungen von anderen Firmen oder Universitäten in Form von Intellectual-Property- oder Open-Source-Implementierungen integriert werden als auch mit eigenen, neu entwickelten Implementierungen. Das Vorgehen für die Auswahl eines passenden Verarbeitungsmoduls, die Implementierung bzw. Anpassung von Funktionen und die Integration bzw. das Ersetzen eines Moduls in das Paketverarbeitungssystem wurde beschrieben ebenso wie mögliche Unterstützungen dabei durch Software-Werkzeuge.

4 Realisierung und Anwendung

In diesem Kapitel wird eine prototypische Realisierung eines Hochgeschwindigkeitspaketverarbeitungssystems mit der im letzten Kapitel präsentierten neuen Architektur vorgestellt (Unterkapitel 4.2) sowie die Umsetzung verschiedener Verarbeitungsfunktionen für ein solches System (Unterkapitel 4.3, Unterkapitel 4.4). Erste Ergebnisse dieser Arbeiten wurden vom Autor bereits vor einem internationalen Fachpublikum präsentiert und diskutiert [2].

Unterkapitel 4.1 diskutiert zunächst, warum Realisierbarkeit und Anwendbarkeit ebenso wie die Leistungsfähigkeit wichtige zu untersuchende Kriterien für die Bewertung der neuen Architektur darstellen.

4.1 Bewertungskriterien

Wichtige Kriterien für die Bewertung der in dieser Arbeit vorgestellten neuen Paketverarbeitungsarchitektur sind:

- Leistungsfähigkeit
- Realisierbarkeit
- Anwendbarkeit

Was unter diesen Bewertungskriterien zu verstehen ist, warum diese wichtig sind, um die vorgestellte Architektur bewerten zu können, und wie man diese überprüfen kann, wird im Folgenden beschrieben. Energieverbrauch, Sicherheit und ökonomische Aspekte sind weitere Bewertungskriterien, die in dieser Arbeit jedoch nicht betrachtet werden.

Leistungsfähigkeit – Eines der beiden Hauptziele beim Entwurf der Architektur war die Leistungsfähigkeit. Die Architektur soll einen deterministisch hohen Durchsatz garantieren können, so dass sie die Paketverarbeitung auf Line Cards mit einem 100 Gbit/s-Anschluss in sowie am Rand von Kernnetzen durchführen kann.

Wie bereits beim Entwurf der Pipeline-Struktur, die der Architektur zu Grunde liegt, dargestellt wurde (Abschnitt 3.4.2), unterstützt die vorgestellte Architektur einen **deterministischen Durchsatz.** Dies bedeutet, dass der Durchsatz des Systems stets eindeutig angegeben werden kann und hier nur von Systemparametern, wie der Wortbreite oder der Taktfrequenz, sowie von den Paketlänge und den Paketabständen abhängt. Insbesondere kann somit eine unter ungünstigsten Bedingungen (Volllast, minimale Paketlängen, minimale Paketabstände) garantierte Daten- und Paketrate berechnet werden.

In der Nur-Header-Variante (NH-Variante), bei der nur das erste Wort jedes Pakets durch die Verarbeitungs-Pipeline wandert, entspricht die Paketrate P_{NH} der Pipeline-Taktfrequenz f (Gleichung (3.9)). Die externe Datenrate $D_{NH,ext}$ – d. h. die Datenrate, die an den externen Schnittstellen des Systems unterstützt wird – ist somit mindestens gleich dem Produkt aus garantierter Paketrate und der Summe aus minimaler Paketlänge L_{min} und dem minimalen Paketabstand in Byte G_{min}:

$$D_{NH,ext} \;=\; (L+G)\cdot f \;\geq\; (L_{min}+G_{min})\cdot f \tag{4.1}$$

Die Paketrate der Ganzes-Paket-Variante ist ebenfalls deterministisch bestimmbar und ist nur abhängig von der Paketgröße, der Breite der Datenworte der Pipeline und der Taktfrequenz (vgl. Gleichung (3.10)). Wiederum ergibt sich die extern unterstützte Datenrate durch Multiplikation der Paketrate mit der Summe aus Paketlänge und Paketabstand, wobei sich die minimale Datenrate für eine Paketlänge L ergibt, die 1 Byte größer ist als die Wortbreite W, d. h. für $L = W + 1\,Byte$:

$$D_{GP,ext} \;=\; \frac{1}{\left\lceil \frac{L}{W} \right\rceil}\cdot(L+G)\cdot f \;\geq\; \frac{1}{2}\cdot(W+1\,Byte+G_{min})\cdot f \tag{4.2}$$

Die Implementierung der vorgeschlagenen Architektur gibt somit vor, wie leistungsfähig das System ist. Die Datenrate und die Paketrate sind stets direkt berechenbar. Die Gleichungen hierfür sind nochmals in Tabelle 4.1 gemeinsam dargestellt. Der garantierte Durchsatz eines Systems ist nur abhängig von der verwendeten Taktfrequenz und der Wortbreite der Pipeline (sowie vom minimalen Paketabstand).

Realisierbarkeit – Die vorgestellte Architektur muss mit der heute verfügbaren Technologie und für die heute in Kernnetzen erforderlichen Datenraten realisierbar sein. Um dies überprüfen zu können, muss ein System mit dieser Architektur prototypisch z. B. in VHDL implementiert und für ein oder mehrere FPGA-Varianten synthetisiert und das Place & Route durchgeführt werden. Somit erhält man Informationen darüber, wie viele Ressourcen auf dem Chip für die Realisierung benötigt werden und mit welcher maximalen Taktfrequenz das System betrieben werden kann. Dies gibt Aufschluss darüber, ob die vorgestellte Architektur so realisiert werden kann, dass sie auch die hohen Datenraten in Kern- und Metro-Routern unterstützt. Zusätzlich kann durch den Aufbau des Systems auf einer FPGA-

Tabelle 4.1: Gleichungen für die Paket- und Datenrate der Architektur

	Paketrate	**Datenrate**
Nur-Header	$P_{NH} = (1\ \text{PAKET})\cdot f \quad (3.9)$	$D_{NH,ext} = (L+G)\cdot f \geq (L_{min}+G_{min})\cdot f \quad (4.1)$
Ganzes-Paket	$P_{GP} = \frac{1}{\left\lceil \frac{L}{W} \right\rceil}\cdot(1\ \text{PAKET})\cdot f \quad (3.10)$	$D_{GP,ext} = \frac{1}{\left\lceil \frac{L}{W} \right\rceil}\cdot(L+G)\cdot f \geq \frac{1}{2}\cdot(W+1\,Byte+G_{min})\cdot f \quad (4.2)$

basierten Testumgebung die korrekte Funktionalität, z. B. im Zusammenspiel mit anderen Routern oder Endsystemen, demonstriert werden.

In Unterkapitel 4.2 wird eine solche prototypische Implementierung beschrieben. Das System wurde ausführlich auf einer FPGA-basierten Testumgebung auf dessen Funktion und Leistungsfähigkeit überprüft sowie in kleineren Netzszenarien mit anderen Systemen getestet. Dies sowie verschiedene Synthese- und Place & Route-Ergebnisse zeigen die Realisierbarkeit der Architektur.

Anwendbarkeit – Das zweite Hauptziel beim Entwurf der neuen Architektur war, höchste Flexibilität bereitzustellen. Um dies zu erreichen, bietet die Architektur die Möglichkeit, auf einfache Weise Verarbeitungsmodule hinzuzufügen, auszutauschen und anzupassen. Insbesondere ist es möglich, Implementierungen auf einer beliebigen, an das Problem angepassten Entwurfsebene zu realisieren. Daher muss überprüft werden, ob diese von der Architektur bereitgestellten Möglichkeiten effizient angewendet werden können. Hierfür muss man verschiedene Verarbeitungsfunktionen prototypisch mit Hilfe der gegebenen Module implementieren und die Module in unterschiedlichen Kombinationen miteinander testen. Diese Implementierungen können Probleme oder Einschränkungen der Architektur aufdecken, die man beim Entwurf sowie auch bei einer prototypischen Implementierung des Systems übersehen haben könnte.

Um die Anwendbarkeit der Architektur auf heutige oder zukünftige Verarbeitungsfunktionen zu überprüfen, wurden deshalb die Fast-Path-Funktionen eines IP-Routers sowie Funktionen neuartiger Protokolle zur Überlastregelung unter Verwendung verschiedener Verarbeitungsmodule auf unterschiedlichen Entwurfsebenen implementiert. Diese Implementierungen beschreiben und diskutieren Unterkapitel 4.3 und Unterkapitel 4.4.

4.2 Prototyp

Um die Realisierbarkeit der Paketverarbeitungsarchitektur zu überprüfen, wurde diese prototypisch implementiert. Diese Realisierung erfolgte zu großen Teilen in vom Autor angeleiteten Studien- und Diplomarbeiten [13–18]. Nach einer groben Beschreibung des Gesamtsystems im folgenden Abschnitt geben die darauf folgenden vier Abschnitte Details zur verwendeten Plattform sowie zu den verschiedenen Teilen der Implementierung. Die letzten zwei Abschnitte dieses Unterkapitels beschreiben und diskutieren durchgeführte Funktions- und Leistungstests sowie Synthese- und Place & Route-Ergebnisse.

4.2.1 Überblick über das Gesamtsystem

Ziel der Implementierung des Prototyps ist zum einen, zu sehen, ob die vorgestellte Architektur auch implementierbar und ein hoher Durchsatz von z. B. 100 Gbit/s mit der heutigen Technologie realisierbar ist, und zum anderen, diesen Prototypen für funktionale Tests sowie für beispielhafte Implementierungen von Paketverarbeitungsfunktionen einzusetzen. Hierfür benötigt der Prototyp außer der vorgestellten Paketverarbeitungs-Pipeline verschiedene weitere Blöcke:

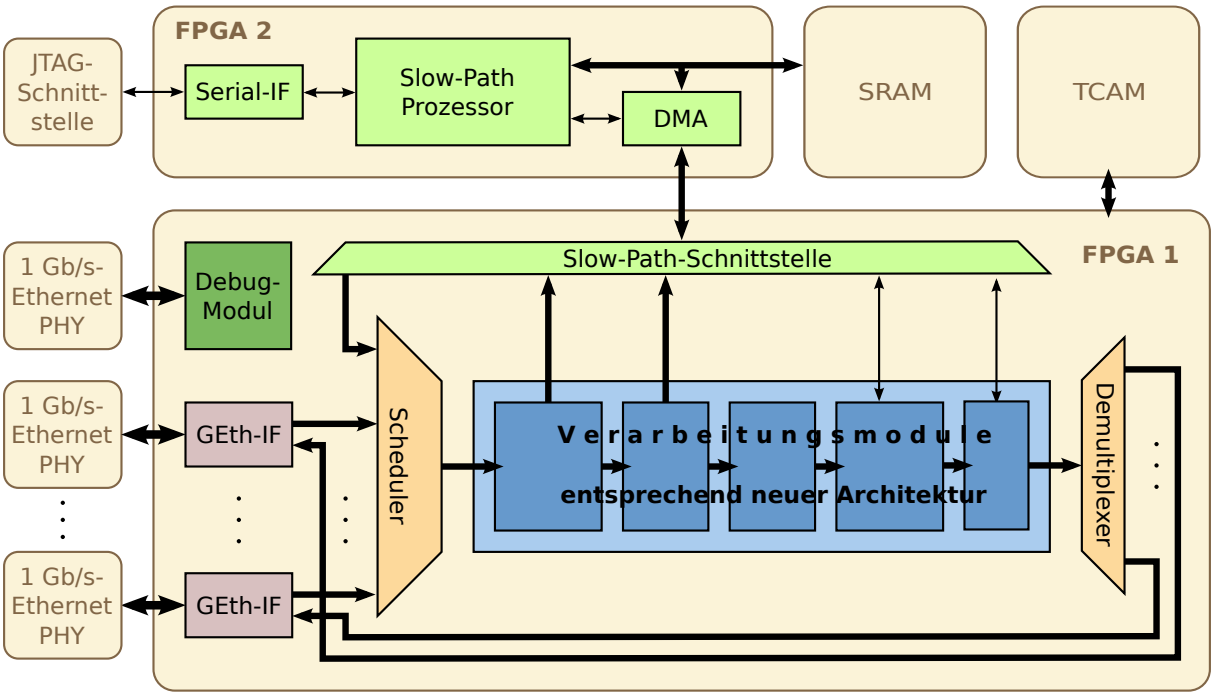

Abbildung 4.1: Überblick über den gesamten Prototyp (inklusive Slow-Path)

Um Pakete empfangen und nach der Verarbeitung wieder versenden zu können, ist eine Anbindung des Systems an eine Netzschnittstelle notwendig. Damit die Pakete, wie in einem Vermittlungsknoten, abhängig von ihrer Zieladresse über unterschiedliche Schnittstellen weiter versendet werden können, werden mehrere solcher Schnittstellen benötigt. Außerdem ist ein Scheduler notwendig, der gleichzeitig ankommende Pakete sequentiell in die Pipeline weiterleitet. Am Ende der Verarbeitungs-Pipeline benötigt man einen Demultiplexer, der die verarbeiteten Pakete entsprechend ihrer ermittelten Ausgangsschnittstelle zu dieser vermittelt.

Um zu testen, ob und wie ein System nach der vorgestellten neuen Architektur mit einem Slow-Path-Modul zusammenarbeiten kann, muss außerdem eine Slow-Path-Schnittstelle vorgesehen werden. Über diese soll dann ein Slow-Path-Prozessor mit der Fast-Path-Verarbeitungs-Pipeline kommunizieren können.

Abbildung 4.1 zeigt ein Blockschaltbild des realisierten Prototyps. Er ist auf zwei FPGAs realisiert. FPGA 1 enthält die zu testende Fast-Path-Verarbeitungsarchitektur und FPGA 2 einen daran gekoppelten Prozessor für Slow-Path-Funktionen. Die gesamte Hardware ist Teil der so genannten Universellen Hardware-Plattform, die Gegenstand des nächsten Abschnitts (Abschnitt 4.2.2) ist.

Die Fast-Path-Implementierung auf FPGA 1 verfügt über mehrere 1 Gbit/s-Ethernet-Anschlüsse. Die entsprechenden Schnittstellenmodule (GEth-IF) empfangen ankommende Ethernet-Rahmen von externen PHY-Chips und parallelisieren korrekt empfangene Daten in die von der Verarbeitungsarchitektur benötigten Worte der Breite W. Anschließend puffert jedes Schnittstellenmodul die Datenworte, bis das nachfolgende Scheduler-Modul die Daten anfordert, um sie der Pipeline von Verarbeitungsmodulen zu übergeben. Das Modul realisiert einen einfachen *Round-Robin*-Scheduler, der reihum von jedem belegten Schnittstellenpuffer ein Paket auswählt.

Im Anschluss an den Scheduler ist das Paketverarbeitungssystem mit der in dieser Arbeit vorgestellten neuen Architektur realisiert. Das System implementiert die GP-Variante der Architektur, d. h. es wird stets das ganze Paket, aufgeteilt in Datenworte, durch die Pipeline transportiert. Die Pipeline kann aus verschiedenen Verarbeitungsmodulen aufgebaut sein. Diese werden in Abschnitt 4.2.4 genauer beschrieben. Nach der Verarbeitung leitet ein Demultiplexer die Pakete zu ihrem jeweiligen – in den Metadaten angegebenen – Egress-GEth-IF-Modul.

Verschiedene Module sind über die Slow-Path-Schnittstelle mit FPGA 2 verbunden. Auf diesem befindet sich ein Slow-Path-Prozessor. Der Datenaustausch des Prozessors mit dem Fast-Path erfolgt über ein DMA-Modul (*Direct Memory Access*), das die Daten selbstständig vom Fast-Path in den SRAM und umgekehrt kopieren kann. Des Weiteren ist der Prozessor über eine serielle Schnittstelle an eine JTAG-Schnittstelle (*Joint Test Action Group*) angebunden, über die Ein- und Ausgaben für Steuer- und Managementaufgaben erfolgen können. Ein externer SRAM realisiert den Befehls- und Datenspeicher des Prozessors.

Zusätzlich verfügt FPGA 1 über ein Debug-Modul mit 1 Gbit/s-Ethernet-Anschluss, über das man zur Laufzeit zu Debug- sowie zu Steuerungs- und Managementzwecken direkt mit dem Fast-Path-Modul kommunizieren kann. Somit kann der Fast-Path auch ohne Slow-Path getestet werden. Das Slow-Path-Modul sowie dieses Debug-Modul wird in Abschnitt 4.2.5 genauer beschrieben.

Außerdem verfügt das System über einen TCAM, welcher an FPGA 1 angeschlossen ist. Dieser wird in manchen Konfigurationen zur IP-Adresssuche verwendet (vgl. Abschnitt 4.3.2). Nicht eingezeichnet in Abbildung 4.1 sind Verkehrsgeneratoren, die optional anstatt der Ethernet-Schnittstellen in das System integriert werden können. Hiermit kann das System mit zufällig erzeugten Paketen unter Volllast getestet werden, was aufgrund des begrenzten Durchsatzes der 1 Gbit/s-Ethernet-Schnittstellen des Prototyps sonst nicht möglich wäre. Diese Generatoren werden zusammen mit den Ethernet-Schnittstellenmodulen in Abschnitt 4.2.3 beschrieben.

Der Aufbau dieses Prototyps entspricht im Prinzip dem einer Line Card mit mehreren Netzschnittstellen eines aktuellen Hochleistungs-Routers, wie er in Abschnitt 2.1.4 vorgestellt wurde. Allerdings kann der Prototyp Pakete nur lokal vermitteln und nicht über ein Schaltnetzwerk zu einer anderen Line Card weiterleiten oder Pakete von anderen Karten zum Versenden erhalten. Somit kann es auch nicht zur Überlastung einzelner Ausgangsschnittstellen kommen, weshalb auf Paketpuffer und ein Verkehrsverwaltungsmodul verzichtet werden konnte. Diese Einschränkungen beeinflussen jedoch nicht die Funktionsweise der Verarbeitungs-Pipeline.

4.2.2 Hardware-Plattform

Für den Aufbau des Prototyps wurde die Universelle Hardware-Plattform [144] – kurz UHP – des Instituts für Kommunikationsnetze und Rechnersysteme verwendet. Die UHP ist eine vielseitig einsetzbare FPGA-basierte Plattform für den einfachen und schnellen Aufbau experimenteller und prototypischer digitaler Systeme. Neben FPGAs zur Realisierung digitaler Logikmodule stellt sie SRAM- und DRAM-Speicher sowie eine Vielzahl verschiedener Schnittstellen zu externen Geräten zur Verfügung.

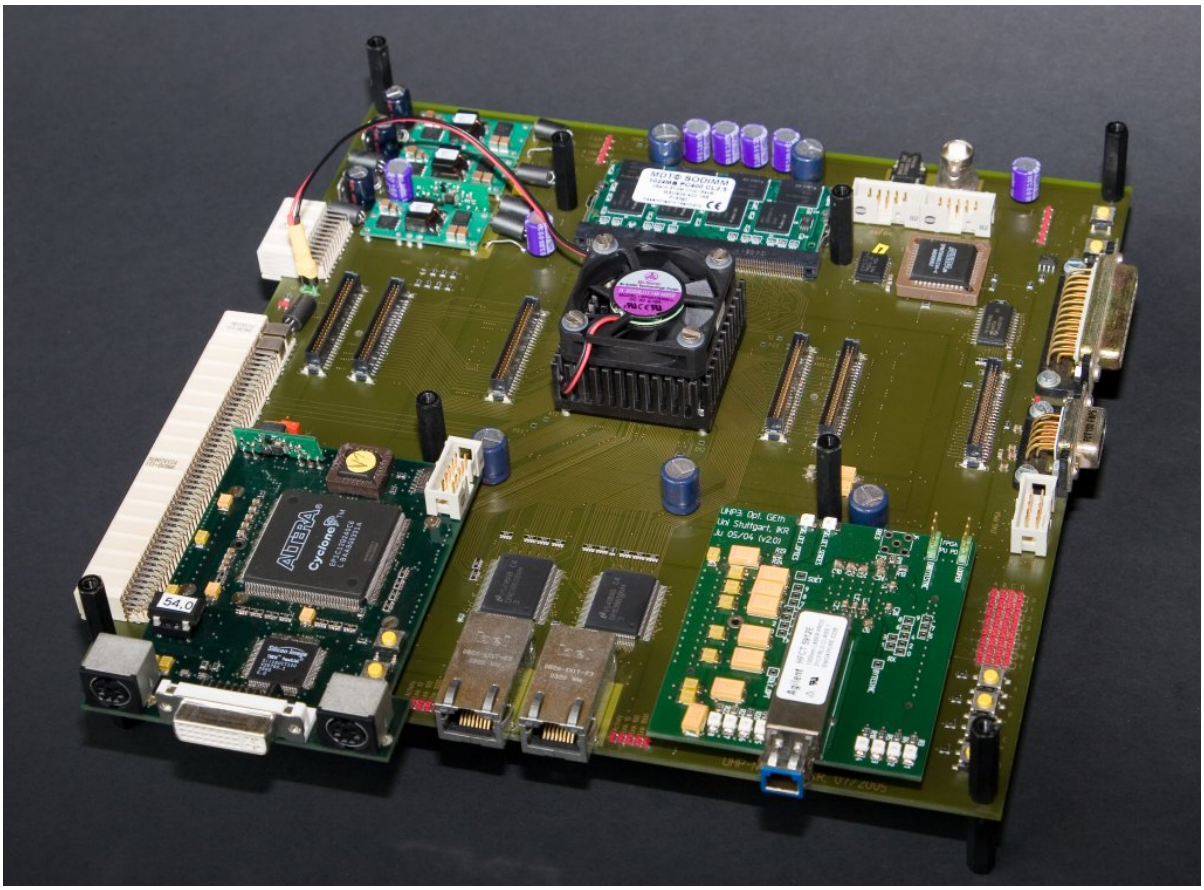

Abbildung 4.2: UHP-2-Platine

Beschreibung der Plattform

Die UHP ist hierarchisch aufgebaut. Die Grundplatine, UHP-1 genannt, enthält einen Altera APEX20KC FPGA, einen Steckplatz für SDRAM-Riegel sowie acht Steckplätze für weitere FPGA-Platinen, die so genannten UHP-2 Platinen. Je vier Steckplätze sind mittels eines Busses miteinander sowie mit dem FPGA der Grundplatine verbunden.

Die UHP-2-Platine gibt es seit 2005 in der zweiten Generation. Diese Platine enthält einen Altera Stratix II FPGA mit der Bezeichnung EP2S 60 F1020 C3 ES [175]. Zusätzlich sind auf der Platine ein Steckplatz für einen DDR2-SDRAM-Riegel, zwei elektrische 1 Gbit/s-Ethernet-Anschlüsse inklusive PHY-Chip sowie eine parallele (Centronics) und eine serielle (*Universal Asynchronous Receiver/Transmitter*, UART) Schnittstelle vorhanden und mit dem FPGA verbunden. Um die Platine um zusätzliche Schnittstellen oder sonstige Peripheriemodule erweitern zu können, verfügt sie zudem über vier Aufsteckplätze für Mezzanin-Erweiterungskarten, auch UHP-3-Karten genannt. Abbildung 4.2 zeigt ein Foto einer UHP-2 der zweiten Generation mit zwei aufgesteckten UHP-3-Karten.

Die UHP-3-Mezzanin-Karten gibt es in vielerlei Varianten. Es gibt Aufsteckkarten mit zusätzlichen SRAM-, DRAM- und TCAM-Speichermodulen, Aufsteckkarten zur Datenein- und -ausgabe über DVI- sowie PS/2-Schnittstellen sowie diverse Aufsteckkarten zur Datenkommunikation über elektrische und optische Ethernet-Schnittstellen mit 10 Mbit/s, 100 Mbit/s und

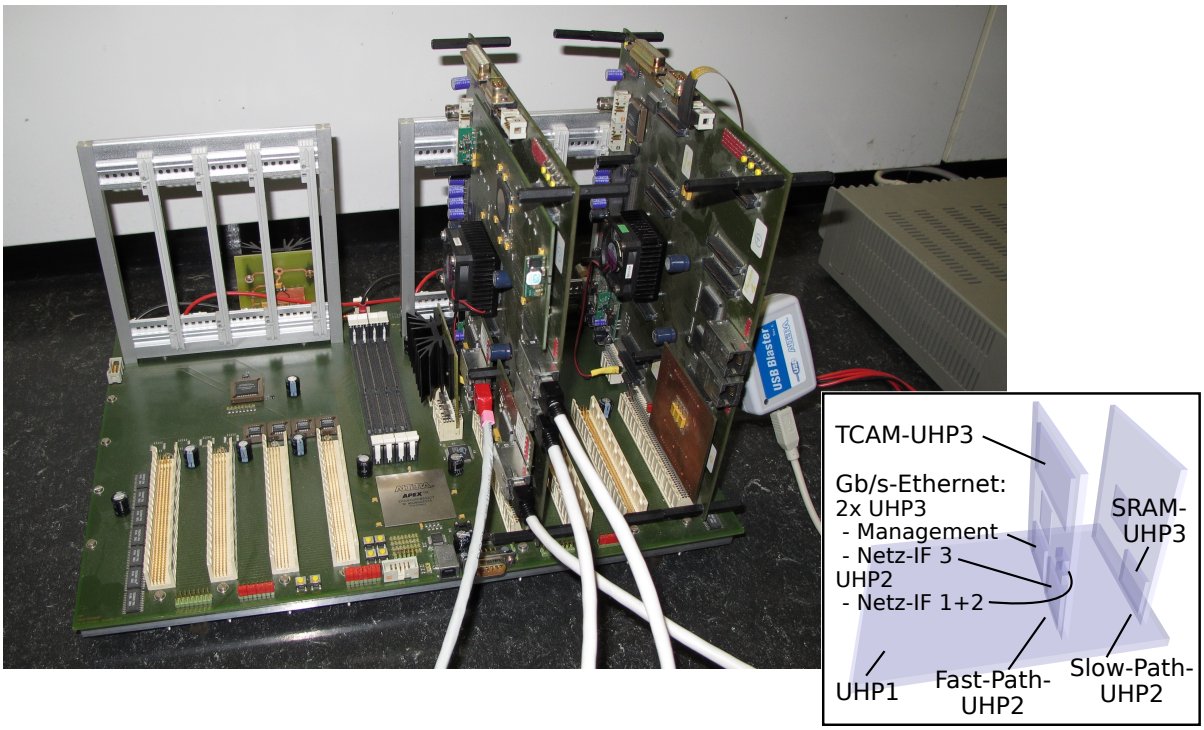

Abbildung 4.3: Aufbau des Prototyps auf der UHP

1 Gbit/s Datenrate. Diese Karten enthalten i. d. R. keinen eigenen FPGA, sondern werden vom FPGA der UHP-2 angesteuert.

Verwendete Konfiguration

Die UHP ist durch ihre Modularität und Flexibilität gut für eine prototypische Realisierung der vorgestellten Architektur geeignet. Lediglich die begrenzte mögliche Datenrate, die über die 1 Gbit/s-Ethernet-Anschlüsse empfangen und versendet werden kann, stellt eine gewisse Einschränkung dieser Plattform dar.

Mit den verfügbaren 1 Gbit/s-Ethernet-Anschlüssen auf den UHP-2- und UHP-3-Platinen kann Paketverkehr mit maximal 6 Gbit/s empfangen werden: $2 \cdot 1\,Gbit/s$ auf der UHP-2-Karte und $4 \cdot 1\,Gbit/s$ auf vier UHP-3-Karten. Jedoch gibt es nur sehr wenige FPGA-Plattformen mit 10 Gbit/s-Ethernet-Anschlüssen und diese sind aufgrund ihrer hohen technologischen Anforderungen sehr teuer und schwer erhältlich. Und selbst mit einer Plattform mit einigen wenigen solcher Anschlüsse ist es nicht möglich, das System unter Volllast bei 100 Gbit/s oder mehr zu testen. Karten mit 40 Gbit/s- oder 100 Gbit/s-Kommunikationsschnittstellen sind aufgrund der erst kürzlich abgeschlossenen Standardisierungen bisher nicht erhältlich. Wie das prototypisch realisierte System mittels interner Verkehrsgeneratoren dennoch unter Volllast getestet werden konnte, wird im folgenden Abschnitt 4.2.3 dargestellt.

Der Aufbau des im letzten Abschnitt bereits kurz vorgestellten Gesamtsystems ist in Abbildung 4.3 als Foto sowie schematisch abgebildet. Eine UHP-1-Grundplatine verbindet über einen

ihrer Busse zwei UHP-2-Platinen, die den Fast Path und den Slow Path des Systems realisieren. Der FPGA der UHP-1 wird nicht verwendet.

Die auf dem Foto linke UHP-2-Platine realisiert den Fast Path. Hierfür verwendet sie zwei aufgesteckte 1 Gbit/s-Ethernet-Aufsteckkarten, so dass insgesamt vier 1 Gbit/s-Ethernet-Schnittstellen zur Verfügung stehen. Wie in Abbildung 4.1 auf Seite 134 dargestellt, verwendet das System eine davon zu Debug-Zwecken und die anderen drei als Netzschnittstellen. Somit kann das System mit einem Durchsatz von bis zu 3 Gbit/s bzw. $3 \cdot 1,5\,MP/s = 4,5\,MP/s$ betrieben werden. Mittels der restlichen zwei Aufsteckplätze kann das System in manchen Konfigurationen eine zusätzliche TCAM-Aufsteckplatine verwenden.

Ein TCAM (*Ternary Content Addressable Memory*) ist ein *dreiwertiger* Assoziativspeicher. Assoziativspeicher werden auch inhaltsadressierte Speicher (*Content Addressable Memory*, CAM) genannt. Ein CAM-Speicher vergleicht ein angelegtes Datenwort mit allen gespeicherten Datenworten und gibt die Adresse des ersten übereinstimmenden Datenwortes aus. Die Besonderheit eines TCAM ist, dass zu jedem gespeicherten Datenwort eine Maske gespeichert wird, mit der man beim Vergleich des angelegten Wortes einzelne Bits nicht berücksichtigen kann. Nicht zu berücksichtigende Bits werden als „don't care"-Bits bezeichnet und stellen somit neben der logischen Null und der logischen Eins einen dritten möglichen Bitwert da. Daher die Bezeichnung *Ternary* (dreiwertiger) CAM. TCAMs werden häufig für die schnelle IP-Adresssuche in Hochgeschwindigkeits-Routern verwendet.

Die TCAM-UHP-3 ist mit dem TCAM-Chip NL5128RD [176] der Firma NetLogic bestückt. Dieser kann bei einer maximalen Taktfrequenz von 133 MHz effektiv jeden zweiten Takt eine Suche über alle seine 128 000 Einträge durchführen. Aufgrund der vielen benötigten Verbindungsleitungen zur UHP-2 belegt diese Karte zwei Steckplätze. Diese Karte sowie das dazugehörige VHDL-Logikmodul wurden in der Studienarbeit von Masini [18] realisiert, die der Autor dieser Arbeit betreute.

Die zweite UHP-2-Platine (auf dem Foto rechts) realisiert den Slow-Path des Systems und enthält hierfür eine UHP-3-Aufsteckkarte mit einem 8 Mbit SRAM-Modul. Die Ansteuerung und Programmierung des Slow-Path-Prozessors von außen erfolgt über eine JTAG-Schnittstelle, über die auch die Konfiguration des FPGAs erfolgt.

4.2.3 Eingangs- und Ausgangsschnittstellen

Der prinzipielle Aufbau der Eingangs- und Ausgangs-Netzschnittstellen des Prototyps sowie deren wichtigste Signale sind in Abbildung 4.4 dargestellt. Der implementierte Logikentwurf unterstützt bis zu $n = 15$ 1 Gbit/s-Ethernet-Anschlüsse sowie eine Schnittstelle für den Empfang von Paketen vom Slow Path. Der Empfang und der Versand der Pakete erfolgt über n Rx-/Tx-Einheiten, jeweils bestehend aus 1 Gbit/s-Ethernet-Schnittstellenmodul und Wortbreiten-Anpassungsmodul. Der Scheduler wählt empfangene Pakete reihum aus und übergibt sie der Pipeline von Verarbeitungsmodulen. Nach der Verarbeitung vermittelt ein Demultiplexer die Pakete wieder an eine der Rx-/Tx-Einheiten. Erste Versionen dieser Module wurden unter Anleitung des Autors im Rahmen der Studienarbeit von Mück [17] realisiert.

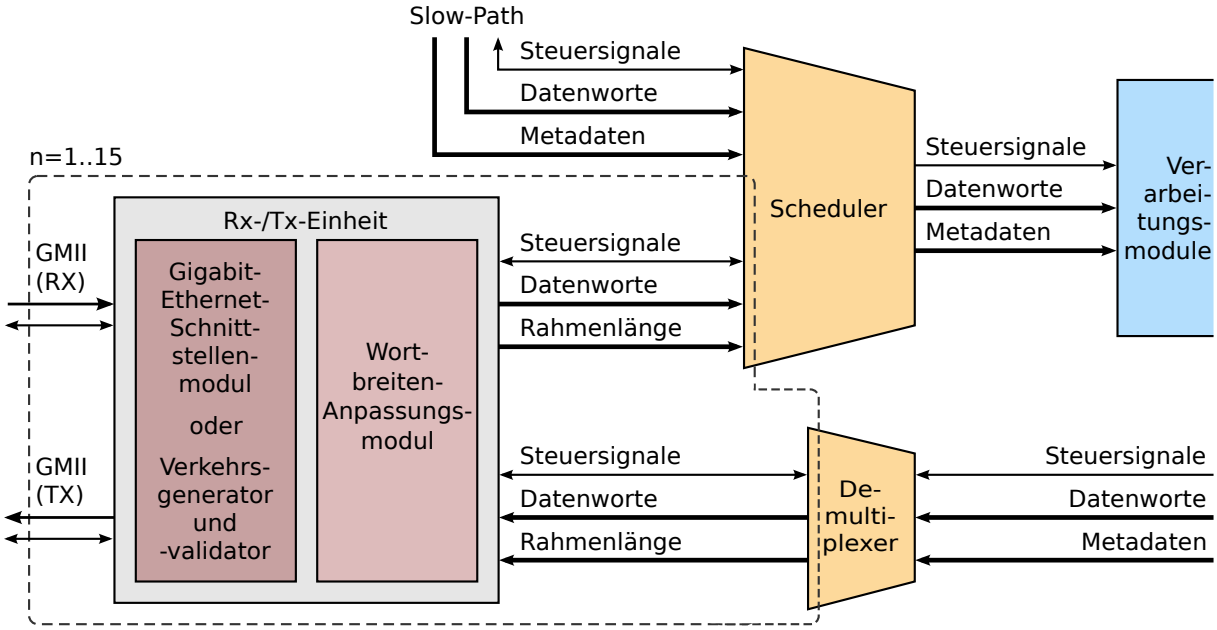

Abbildung 4.4: Schnittstellenmodule

Für die 1 Gbit/s-Ethernet-Schnittstellenmodule wurde das generische Ethernet-Schnittstellenmodul [177] des IKR verwendet. Dieses kommuniziert mit dem PHY-Chip über eine GMII-Schnittstelle (*Gigabit Media Independent Interface*). Nach dem Empfang von Rahmen überprüft es die Rahmenprüfzeichenfolge (*Frame Check Sequence*, FCS) und gibt korrekt empfangene Rahmen an das nachfolgende Wortbreiten-Anpassungsmodul weiter. Beim Versand von Rahmen berechnet das Modul entsprechend die Prüfzeichenfolge, schreibt diese ans Ende des Rahmens und übermittelt diesen an den PHY-Chip.

Das zweite Modul in einer Rx-/Tx-Einheit ist das Wortbreiten-Anpassungsmodul. Dieses wurde entworfen, um die über GMII übermittelten Worte mit einer Breite von 8 bit beim Empfang von Rahmen auf die von der Architektur geforderten sehr breiten Datenworte von mindestens 512 bit zu parallelisieren. Beim Versand von Paketen serialisiert das Modul die Datenworte entsprechend wieder auf die von GMII geforderten Bytes. Die Byteanzahl der Wortbreite auf Systemseite ist auf eine beliebige Zweierpotenz parametrisierbar. Typische Wortbreiten sind 64 Byte = 512 bit, 128 Byte = 1024 bit und 256 Byte = 2048 bit. Da alle Datenworte eines Pakets direkt hintereinander zur Verarbeitungs-Pipeline übermittelt werden sollen, enthält das Modul einen Puffer, in dem mindestens ein Paket maximaler Größe Platz findet. Dieser Puffer ermöglicht außerdem, dass das Anpassungsmodul unterschiedliche Eingangs- und Ausgangstaktfrequenzen unterstützt. Somit kann die Systemtaktfrequenz unabhängig von der Taktfrequenz der Ethernet-Schnittstellen gewählt werden.

Die Hauptaufgabe des Scheduler-Moduls ist, die empfangenen und auf System-Wortbreite parallelisierten Pakete reihum von den Rx-/Tx-Einheiten sowie von der Slow-Path-Schnittstelle anzufordern und zur Pipeline der Verarbeitungsmodule weiterzuleiten. Die Auswahl der nächsten Schnittstelle sowie das anschließende Auslesen des Pakets realisieren zwei kleine, gekoppelte Automaten. Diese wurden für optimalen Durchsatz entworfen. So ist der Scheduler in der Lage, zwei Pakete – selbst bei minimaler Größe – ohne zeitliche Lücke von unterschiedlichen Schnittstellen zu übertragen.

Die zweite Aufgabe des Scheduler-Moduls ist, die initialen Metadaten für jedes Paket zu erzeugen. Die bereits hier bekannten Metainformationen sind die Eingangs-Schnittstelle sowie die Paketlänge. Pakete vom Slow Path können bereits mehr Einträge in den Metadaten enthalten. Außerdem generiert das Modul die Steuersignale zur Signalisierung gültiger Paketdaten.

Der Demultiplexer auf der Ausgangsseite ist zustandslos. Er steuert entsprechend des in den Metadaten angegebenen Egress-Ports kombinatorisch eine der Rx-/Tx-Einheiten an. Zusätzlich extrahiert er die Paketlänge aus den Metadaten. Diese sowie die Datenworte werden ebenfalls an die entsprechende Rx-/Tx-Einheit übergeben. Die Rx-/Tx-Einheit serialisiert die Datenworte, wie oben beschrieben, auf einzelne Bytes und übermittelt diese zum Versenden an den PHY-Chip.

Wie bereits in Abschnitt 4.2.2 aufgezeigt, kann über die Netzschnittstellen der verwendeten Plattform der maximale Durchsatz des Verarbeitungssystems nicht erreicht werden. Um den Prototypen dennoch unter Volllast, d. h. ohne zeitliche Abstände zwischen den Paketen, testen zu können, sind die 1 Gbit/s-Ethernet-Schnittstellenmodule in den Rx-/Tx-Einheiten gegen Paketgeneratoren und -validatoren austauschbar. Die Paketgeneratoren erzeugen pseudo-zufälligen Paketverkehr. Die erzeugten Pakete werden wie beschrieben vom Scheduler-Modul zu den Paketverarbeitungsmodulen transportiert und dort verarbeitet.

Die Validatoren erzeugen zeitversetzt denselben Paketverkehr wie die Generatoren. Damit sie überprüfen können, ob alle Pakete korrekt verarbeitet und weitertransportiert wurden, müssen die durchgeführten Änderungen für die Validatoren einfach berechenbar sein. Darauf muss bei der Erzeugung relevanter Paketfelder sowie bei der Konfiguration der Verarbeitungsmodule geachtet werden. Beispielsweise dürfen zwar komplexe Algorithmen zur IP-Adresssuche verwendet werden, jedoch muss das Ergebnis, d. h. Ausgangs-Port und Ziel-MAC-Adresse, für das Validatormodul auch auf einfache Weise ermittelbar sein.

Für die Realisierung der Paketgeneratoren und -validatoren wurden die Module von Mutter [140] verwendet und angepasst. Diese verwenden rückgekoppelte Schieberegister zur Erzeugung der pseudo-zufälligen Pakete. Die Generatoren können zur Laufzeit konfiguriert werden, ob sie Pakete zufälliger Länge oder nur Minimalpakete erzeugen sollen, sowie welche Abstände zwischen den Paketen sein sollen. Die MAC- und IP-Adressen werden abhängig vom erzeugenden Generator ganz oder zum Teil deterministisch gesetzt. Die IP-Prüfsumme wird stets aktuell berechnet.

4.2.4 Verarbeitungsmodule

Es wurden drei verschiedene Arten von Verarbeitungsmodulen entsprechend der vorgestellten Architektur realisiert: Programmierbare-Logik-Module, Prozessor-Pool-Module und Datenfluss-Pipeline-Module. Alle diese Module verhalten sich nach außen entsprechend der von der Architektur festgelegten grundlegenden Pipeline, in der sehr breite Datenworte jeden Taktzyklus um eine Stufe weitergeschoben werden.

Die Breite der Datenworte ist einstellbar als beliebige Zweierpotenz. Sinnvolle Breiten sind somit 512, 1024 oder auch 2048 bit. Die Metadaten wurden mit einer Breite von 64 bit festgelegt und enthalten folgende Felder: Ingress-Port (4 bit), Egress-Port (4 bit), Paketlänge (11 bit) und

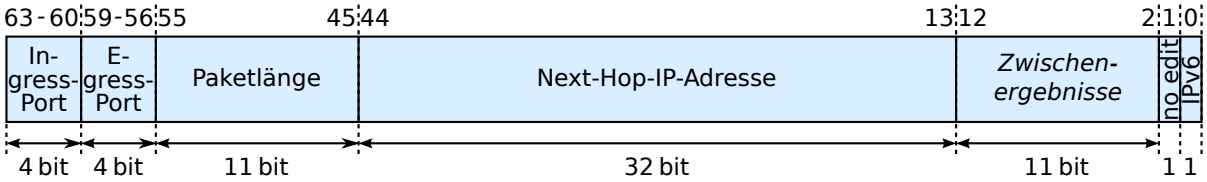

Abbildung 4.5: Format der Metadaten

Next-Hop-IP-Adresse (32 bit). Außerdem ist ein Bit zur Markierung von IPv6-Paketen reserviert, sowie ein Steuerbit zur Markierung von Paketen vom Slow Path, die nicht mehr im Fast Path verarbeitet werden sollen (vgl. Abschnitt 4.2.5). Die restlichen 11 bit sind nicht belegt und können für den lokalen Austausch von Ergebnissen zwischen zwei Modulen verwendet werden. Abbildung 4.5 zeigt die entsprechende Aufteilung der Metadaten. Drei Steuersignale regeln den Transport zwischen den Modulen: Das Signal *valid* zeigt an, dass gültige Daten und Metadaten anliegen. Das Signal *start* markiert das erste und das Signal *end* das letzte Wort eines Pakets.

Nachfolgend werden kurz die drei realisierten Verarbeitungsmodularten beschrieben:

Programmierbare-Logik-Modul – Diese Art von Modulen basiert, wie in Abschnitt 3.5.3 beschrieben, direkt auf der verfügbaren FPGA-Technologie, d. h. auf den konfigurierbaren Lookup-Tabellen, Flipflops, Speichern und Verbindungsstrukturen des Chips. Die Funktionalität wird wie bei anderen FPGA-Implementierungen mittels Verhaltens- und Struktur-Beschreibungen in VHDL beschrieben oder mittels entsprechender Software-Werkzeuge, wie dem HDL-Designer [169] (vgl. Abschnitt 3.6.4), entwickelt. Bei der Entwicklung musste lediglich die einheitliche Schnittstellendefinition für die Verarbeitungsmodule eingehalten werden. Für eine komfortable Realisierung von Zugriffen auf Protokoll- und Metadatenfelder wurde ein allgemein nutzbares VHDL-*Package* definiert mit Definitionen von Konstanten mit Bitpositionen und Feldlängen sowie mit Typdefinitionen für Bitbereiche von Header-Feldern.

Prozessor-Pool-Modul – Die Implementierung dieses Moduls entspricht der Beschreibung in Abschnitt 3.7.1. Für die Realsierung der Befehls- und Datenspeicher wurden die SRAM-Blöcke des FPGAs verwendet. Da die Realisierung der Registersätze mit 32 oder 64 Worten zu je 32 bit bei einer großen Anzahl p von Prozessorkernen zu einem erheblichen Ressourcenverbrauch an Flipflops führt, wurden auch die Registersätze mit SRAM-Blöcken realisiert. Damit in der Decode-Stufe bis zu drei Register gleichzeitig ausgelesen werden können, ist jeder Registersatz durch drei parallele SRAM-Blöcke mit identischem Inhalt realisiert. Für die Realisierung des Multiplikationsbefehls konnten die für Signalverarbeitungsanwendungen optimierten Multiplizierer verwendet werden. Die anderen arithmetischen Operationen werden effizient von den in den Logikelementen enthaltenen Volladdierern und schnellen Carry-Ketten realisiert. Wie bereits in Abschnitt 3.7.1 beschrieben, ist die Anzahl der zu verwendenden Prozessorkerne, die Anzahl der Register sowie das Taktverhältnis s von Prozessortakt zu Systemtakt unter gewissen Randbedingungen beliebig einstellbar. Eine erste grundlegende Version dieses Moduls wurde im Rahmen der Studienarbeit von Klingler [16] realisiert, die der Autor dieser Arbeit betreute.

Für die Umsetzung des Assembler-Codes in binären Maschinencode wurde eine Assembler-Software entwickelt. Diese erkennt Syntax-Fehler, zu große unmittelbare Operanden sowie zu weite Sprungdistanzen. Die Software kann den assemblierten Programmcode in ver-

schiedenen Formaten für Simulations- und Fehlersuchzwecke sowie zur eigentlichen Verwendung auf dem Chip generieren. Programmiermodell und Befehlssatz der Prozessorkerne sind in Anhang A abgedruckt.

Datenfluss-Pipeline-Modul – Dieses Modul entspricht der in Abschnitt 3.7.2 beschriebenen Implementierung. Wie beim Prozessor-Pool-Modul wurde die ALU unter Verwendung der optimierten Volladdierer und Carry-Ketten realisiert. Für die Befehlsspeicher werden die FPGA-internen SRAM-Blöcke verwendet. Alle Registersätze werden durch Flipflops realisiert. Vor der Synthese dieses Moduls kann ausgewählt werden, ob der Befehlsspeicher und der Konstanten-Registersatz als Nur-Lese-Speicher (*Read-Only-Memory*, ROM) ausgeführt werden sollen oder auch nach der Konfiguration des Chips von außen über das Managementsystem (vgl. Abschnitt 4.2.5) beschreibbar sein sollen. Im ersten Fall kann die Synthese-Software in den meisten Fällen einen erheblichen Teil der Ressourcen aufgrund von nicht benötigten Registern, Multiplexern und Rechenwerken einsparen. Im zweiten Fall macht es Sinn, vor der Synthese über weitere Parameter die genaue Größe der verschiedenen Registersätze und Speicher anzugeben. Dies ist global für alle Verarbeitungseinheiten einstellbar.

Zur Programmierung des Datenfluss-Pipeline-Moduls wurde eine Art „Assembler“ in VHDL implementiert. Dieser setzt die angegebenen Operationen in den benötigten horizontalen Mikrocode um. Außerdem werden Syntaxfehler erkannt. Allerdings muss der Programmierer bisher selbst die Initialisierungsdaten für den Konstantenspeicher erzeugen. Auch Sprungdistanzen, Wartetakte und notwendige Segmentwechsel müssen bisher manuell ermittelt und umgesetzt werden. Programmiermodell und Befehlssatz der Verarbeitungseinheiten sind in Anhang B abgedruckt. Das Datenfluss-Pipeline-Modul und der Assembler wurden unter Anleitung des Autors in der Studienarbeit von Buck [13] implementiert.

Um eine einheitliche Trennung der Module zu gewährleisten, wurde ein Spezialregister für die Daten, Metadaten und Steuerbits implementiert, das jeweils zwischen die Module gesetzt wird. Somit kann jedes Modul stets von denselben zeitlichen Randbedingungen bezüglich seiner Eingangs- und Ausgangssignale ausgehen. Eine weitere Funktion des Spezialregisters ist das Verwerfen von Paketen sowie das Umleiten von Paketen in den Slow Path. Hierfür verfügt es über zwei entsprechende Eingänge *drop* und *divert_frame* sowie eine Paketschnittstelle zum Slow Path. Den realisierten Slow Path sowie Möglichkeiten zur Steuerung und Überwachung des implementierten Prototyps beschreibt der folgende Abschnitt.

4.2.5 Steuerung und Management

In diesem Abschnitt wird gezeigt, wie das realisierte Fast-Path-Paketverarbeitungssystem, welches in den letzten zwei Abschnitten beschrieben wurde, an einen Slow-Path-Prozessor angebunden wurde. Außerdem wird die so genannte Management-Schnittstelle vorgestellt. Mit dieser ist es unabhängig vom Slow Path möglich, Einstellungen, Registerwerte und Speicherinhalte des Prototyps effizient zur Laufzeit auszulesen und zu ändern. Diese Schnittstelle war wesentlich für die Inbetriebnahme und die durchgeführten Tests.

Slow-Path-Schnittstelle

Der Slow-Path-Prozessor ist auf einem zweiten FPGA realisiert, der auf einer zweiten UHP 2 platziert ist und über den Bus einer UHP 1 mit dem Fast-Path-FPGA auf der anderen UHP 2 kommunizieren kann (vgl. Abschnitt 4.2.2).

Der prinzipielle Aufbau des Slow-Path-Moduls und dessen Anbindung an den Fast Path wurde bereits in Abbildung 4.1 auf Seite 134 dargestellt. Die wesentlichen Komponenten sind der Prozessor, ein externer SRAM-Speicher sowie ein DMA-Modul. Der SRAM-Speicher wird vom Prozessor als Programm- und Datenspeicher verwendet. Hier werden auch die zu verarbeitenden Pakete abgelegt. Um den Prozessor vom zeitaufwändigen Kopieren von Paketen und anderen Daten, wie aufgezeichnete Statistiken oder Routing-Informationen, vom und zum Fast Path zu entlasten, wurde das DMA-Modul in das System integriert. Dieses erhält vom Prozessor Aufträge, welche Daten wohin kopiert werden sollen, und führt diese daraufhin eigenständig aus. Um die beiden Kopierrichtungen zu entkoppeln und u. U. eine Richtung priorisieren zu können, besteht das DMA-Modul intern aus zwei getrennten Modulen. Diese arbeiten unabhängig voneinander, müssen sich jedoch den Bus zum Fast Path sowie zum Speicher teilen. Die Zugriffskontrolle erfolgt hierbei mittels entsprechender Arbitrierungslogik.

Auf Seiten des Fast Paths können beliebige Module mit dem Slow Path kommunizieren. Hierfür stehen zwei verschiedene Arten von Schnittstellen zur Verfügung: Solche, die für eine unidirektionale Übertragung von Paketen inklusive Metadaten zum Slow Path optimiert sind und solche, über die der Slow-Path-Prozessor auf Speicher und Konfigurationsregister einzelner Module lesend und schreibend zugreifen kann. Alle Datenbewegungen zum Slow Path müssen dabei vom Prozessor angestoßen werden, jedoch können die Paketschnittstellen über Interrupt-Leitungen die Ankunft von zu übertragenden Paketen signalisieren. Die Übertragung von Paketen vom Slow Path zum Fast Path erfolgt über das vorne eingeführte Scheduler-Modul (vgl. Abschnitt 4.2.3), das die Slow-Path-Schnittstelle in dieser Realisierung wie einen gleichberechtigten Eingangs-Port behandelt. Eine nochmalige Verarbeitung des Pakets im Fast Path kann durch Setzen eines entsprechenden Bits in den Metadaten verhindert werden (vgl. Abschnitt 4.2.4).

Der Slow Path und die Anbindung an den Fast Path wurden im Rahmen der Diplomarbeit von Witowski [14] realisiert, die vom Autor betreut wurde. Für den Entwurf und die Implementierung verwendete Witowski den *SOPC-Builder* (*System On a Programmable Chip*, SOPC) des FPGA-Herstellers Altera. Dieser stellt viele der benötigten Komponenten und Standard-Kommunikationsschnittstellen zur Verfügung. Als Prozessor wurde der Nios II verwendet, den Altera inklusive einer umfangreicher Entwicklungsumgebung bereitstellt.

Management-Schnittstelle

Zur Inbetriebnahme und zum Testen des Prototyps ist es hilfreich, Registerwerte innerhalb des Systems während des Betriebs auslesen und auch ändern zu können. Somit können z. B. Zähler oder Registerwerte an unterschiedlichen Stellen des Systems ausgelesen werden, um Paketverluste bzw. Fehlfunktionen erkennen, nachvollziehen und eingrenzen zu können. Des Weiteren können Einstellungen von Verarbeitungsmodulen für unterschiedliche Tests geändert werden,

ohne das System mehrmals durch eine zeitaufwändige Synthese neu generieren zu müssen. Auch das Konfigurieren, Starten, Stoppen und Auswerten der Lasttests mit den internen Verkehrsgeneratoren erforderte diese Möglichkeit, effizient in den laufenden Prototypen eingreifen zu können.

Um diese Anforderungen erfüllen zu können, wurde das am Institut für Kommunikationsnetze und Rechnersysteme in den letzten Jahren entwickelte Management-System [178] in den prototypischen Aufbau integriert. Dieses System besteht aus Logikmodulen, die in das zu testende System integriert werden müssen, und einer Software, die kontinuierlich die Registerwerte ausliest und zu ändernde Werte in das System überträgt. Die Kommunikation erfolgt über eine Ethernet-Verbindung.

4.2.6 Funktionstests

Das in den letzten Abschnitten beschriebene prototypische Paketverarbeitungssystem wurde in verschiedenen Konfigurationen und mit unterschiedlichen Verarbeitungsfunktionen auf korrekte Funktionalität, auch unter Volllast, überprüft. Zum einen wurden die einzelnen Module beim Entwurf ebenso wie das Gesamtsystem nach Fertigstellung aller Module taktgenau simuliert. Zum anderen wurde der Prototyp im Betrieb auf der Universellen Hardware-Plattform in verschiedenen Szenarien getestet.

Für die **taktgenauen Simulationen** wurde die Simulations-Software Modelsim [170] von Mentor Graphics verwendet (vgl. auch Abschnitt 3.6.4). Mit dieser Software können beliebige Signale des Systems bei der Simulation aufgezeichnet und anschließend dargestellt werden. Somit kann man sowohl Details einzelner Signale auf Taktzyklus-Zeitskala als auch längerfristiges Verhalten von Modulen sowie des Gesamtsystems prüfen. Beispielsweise konnte somit überprüft werden, ob das Scheduler-Modul tatsächlich in jedem Taktzyklus ein neues Datenwort, auch beim Wechsel zwischen Minimalpaketen, zur Paketverarbeitungs-Pipeline übermittelt. Ebenso konnte das interne Verhalten der Verarbeitungsmodule sowie deren Schnittstellen eingehend untersucht werden. Außerdem konnten auf Modulebene zwischen den Verarbeitungsfunktionen jeweils die verarbeiteten Datenworte und Metadaten überprüft werden, ob die Verarbeitung korrekt war.

Die taktgenaue Simulation mit Modelsim ist gut geeignet, die Funktionalität der Module und des Systems unter idealen Randbedingungen und für kurze Zeitspannen zu überprüfen. Allerdings muss für die Simulation weniger Millisekunden realer Zeit die Software mehrere Stunden ausgeführt werden. Somit kann das Verhalten des Systems nicht über einen längeren Zeitraum getestet werden. Außerdem kann das System in der Simulation nicht zusammen mit anderen Geräten, wie PCs, Ethernet-Switches oder Routern, getestet werden. Aus diesen Gründen wurde die prototypische Implementierung auch im Betrieb auf der Universellen Hardware-Plattform getestet.

Der **Test auf der Universellen Hardware-Plattform** umfasst zwei unterschiedliche Arten der Untersuchung: Zum einen wurde der Prototyp verbunden mit anderen PCs getestet. Hierbei wurde das System, wie bereits in Abschnitt 4.2.2 aufgezeigt, aufgrund der beschränkten Datenraten der verfügbaren Netzschnittstellen nur zu einem Bruchteil ausgelastet. Dafür konnte seine Funktionalität in echten Netzszenarien und im Zusammenspiel mit anderen Systemen getestet

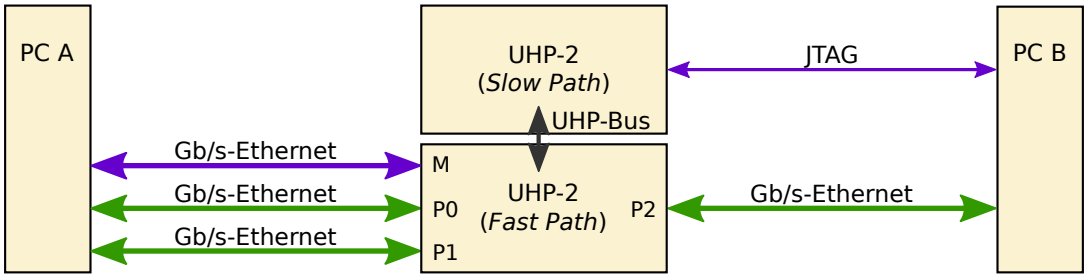

Abbildung 4.6: Schematischer Testaufbau bestehend aus zwei UHP-2 und zwei PCs

werden. Zum anderen wurde der Prototyp mit den in Abschnitt 4.2.3 beschriebenen On-Chip-Paketgeneratoren und -validatoren getestet. Somit konnte auch der Volllastfall im Betrieb auf der UHP überprüft werden.

Abbildung 4.6 zeigt schematisch den Aufbau der verwendeten Testumgebung. Sie besteht aus zwei PCs und zwei über den UHP-Bus miteinander verbundenen UHP-2-Platinen. Die beiden UHP-2-Platinen realisieren den Fast Path und den Slow Path des prototypischen Paketverarbeitungssystems. Ihre Konfiguration wurde in Abschnitt 4.2.2 beschrieben. Die drei Netzschnittstellen des Fast Path sind über 1 Gbit/s-Ethernet mit den beiden PCs verbunden, wobei die Ports P0 und P1 mit PC A verbunden sind und Port P2 mit PC B. Die Management-Schnittstelle M des Fast Path ist ebenfalls über 1 Gbit/s-Ethernet an PC A angebunden. Die Slow-Path-UHP-2 ist über ihre JTAG-Schnittstelle mit PC B verbunden.Über diese Schnittstelle kann der Slow-Path-Prozessor vom PC aus gesteuert werden.

Für die **funktionalen Tests** wurden drei IP-Subnetze an den drei Netzschnittstellen des Paketverarbeitungssystems eingerichtet und die Routing-Informationen auf der UHP-2 und den beiden PCs entsprechend konfiguriert. Zur Erzeugung von Paketen auf den PCs wurden sowohl ein vom Autor programmierter Paketgenerator, das Netzdiagnose-Werkzeug *Ping*, der Internet-Browser *Firefox* sowie das Programm *Secure Shell* (SSH) verwendet.

Der Paketgenerator kann UDP- und RCP-Minimalpakete sowie TCP-Quick-Start-Bandbreiteanforderungspakete erzeugen. Hierbei ist sowohl der Abstand zwischen den Paketen einstellbar als auch bei den RCP- und Quick-Start-Paketen die angeforderte Bandbreite sowie andere Header-Felder. Bei den durchgeführten Tests erzeugte der Software-Paketgenerator auf PC A Pakete. Diese sendete das Routing-Modul des Linux-Kernels mittels der zuvor entsprechend konfigurierten Routing- und ARP-Einstellungen (*Address Resolution Protocol*) zum Paketverarbeitungssystem. Dieses verarbeitet die eingehenden Pakete und sendet die Pakete weiter – falls kein Routing-Modul in das System integriert war, wurde das Paket über Port P0 weitergesendet, ansonsten entsprechend der vorkonfigurierten Routing-Tabelle. Zur Überprüfung, ob die Pakete korrekt verarbeitet wurden, wurden sowohl Zähler- und Registerinhalte innerhalb des Systems ausgewertet, als auch Paketaufzeichnungen des Software-Paketanalyse-Werkzeugs *Wireshark* auf den beiden PCs.

Mit den Programmen Ping, SSH und Firefox konnte das prototypische Paketverarbeitungssystem im Zusammenspiel mit allgemein verwendeten Anwendungen getestet werden. Mit Ping konnten ICMP (*Internet Control Message Protocol*) Echo-Requests erzeugt werden, welche das Paketverarbeitungssystem je nach Konfiguration entweder zum anderen PC weiterleiten,

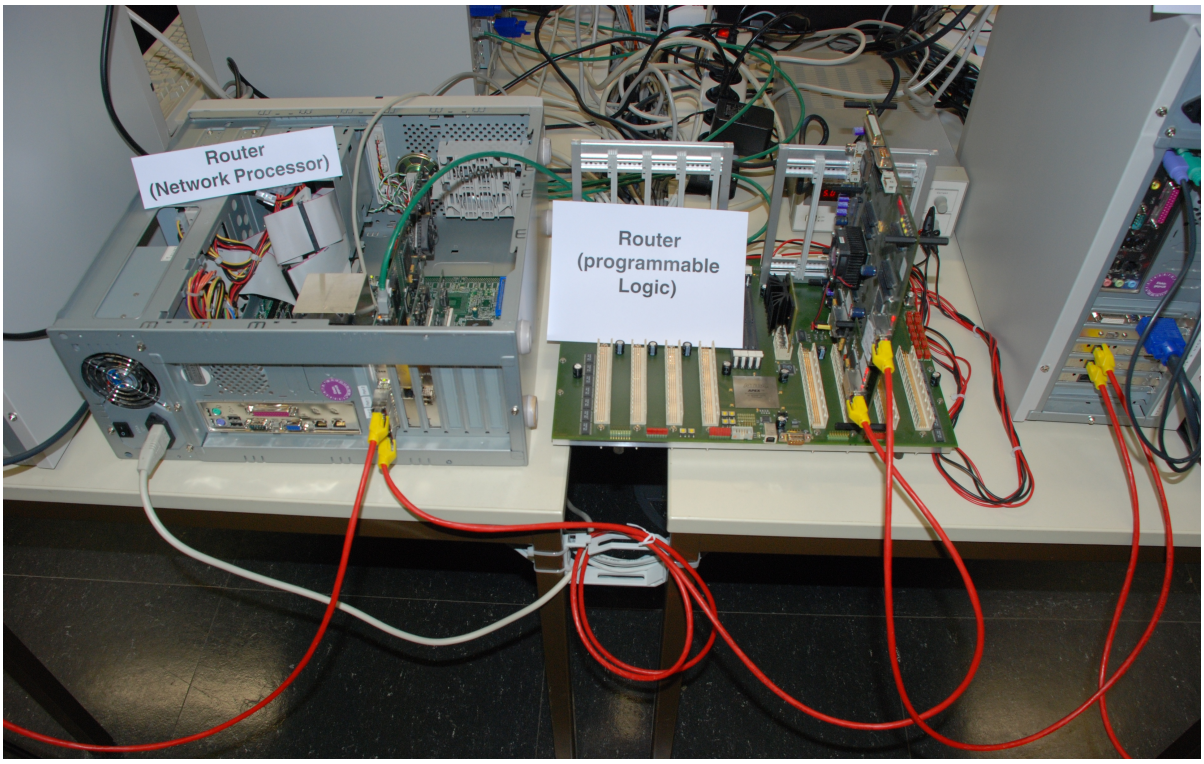

Abbildung 4.7: Quick-Start-Demonstrationsaufbau: PC-, Netzprozessor- und UHP-Router

verwerfen, zum Slow Path weiterleiten und dort beantworten oder direkt im Fast Path beantworten sollte. Das Programm SSH wurde zur Erzeugung größerer TCP/IP-Datenübertragungen verwendet, um die korrekte Funktionalität in einer weiteren Anwendung zu überprüfen.

Die verwendete Firefox-Software, sowie eine entsprechende Server-Software, erweiterte Scharf im Rahmen seiner Doktorarbeit [179] so, dass diese sowohl mittels Standard-TCP als auch mittels Quick-Start TCP Bilder herunterladen konnte. Somit konnte dieses neuartige Protokoll mittels bekannter Browser-Anwendungen getestet und demonstriert werden. Abbildung 4.7 zeigt ein Foto eines Teils des Demonstrationsaufbaus bestehend aus zwei PC-basierten Software-Routern, einem Netzprozessor-basierten Router mit dem Intel IXP 2400 (vgl. Abschnitt 2.4.1) sowie dem hier betrachteten prototypischen Router auf der UHP.

Für die **Tests unter Volllast** wurden die drei Netzschnittstellen durch drei Paketgeneratoren und -validatoren ersetzt. So konnte das System in unterschiedlichen Konfigurationen sowohl mit nur Minimalpaketen als auch mit Paketen mit gleichverteilten Längen unter Volllast betrieben werden. Hier wurden die Netzszenarien so gewählt, dass die durchgeführten Veränderungen an den Paketen vom Validator effizient nachvollzogen und überprüft werden konnten.

Die Generator-Module auf dem verwendeten Stratix II FPGA konnten mit einer Taktfrequenz von etwa 125 MHz betrieben werden, wobei sie jeweils ein 16 bit-Wort pro Takt erzeugten. Da aufgrund der beschränkten Ressourcen des Chips nicht mehr als drei Generatoren realisiert werden konnten, war so eine Maximallast von nur 48 Gbit/s erzeugbar. Aus diesem Grund wurde das Paketverarbeitungssystem für Volllasttests mit einem entsprechend gedrosselten Takt betrieben, so dass die Pipeline dennoch voll ausgelastet werden konnte.

Die durchgeführten Tests deckten ansonsten schwierig zu findende Fehler im System auf, die daraufhin behoben werden konnten. Somit konnte mit Hilfe dieser Tests die korrekte Funktionalität des Prototyps in unterschiedlichen Volllastsituationen gezeigt werden.

4.2.7 Ressourcenverbrauch und Durchsatz

Die Umsetzung der VHDL-Beschreibung des Prototyps in eine Netzliste von auf dem FPGA verfügbaren Elementen erfolgte mit Hilfe der Synthese-Software *Precision RTL Synthesis* [180] von Mentor Graphics. Das Place & Route führte die *Quartus II* Software [181] von Altera durch. Diese liefert Informationen über den Ressourcenverbrauch auf dem verwendeten FPGA und die maximale Taktfrequenz, mit der das System fehlerfrei betrieben werden kann.

Randbedingungen

Es wurden Untersuchungen für zwei unterschiedliche Altera FPGAs durchgeführt: Zum einen für den Altera Stratix II mit der Bezeichnung EP2S 60 F1020 C3 ES [175], der auf der UHP-2 eingesetzt wird, und zum anderen für den Altera Stratix IV GT mit der Bezeichnung EP4S 100G 5 F45 I1 [182]. Letzterer ist zurzeit der größte und schnellste verfügbare FPGA von Altera, der einen Datendurchsatz von 100 Gbit/s unterstützt (daher die 100G in der Bezeichnung). Er verfügt über 32 Transceiver mit 11,3 Gbit/s Datenrate und 16 Transceiver mit 6,5 Gbit/s Datenrate.

Die Synthesen wurden für eine Wortbreite in der Verarbeitungs-Pipeline von 512 bit und 1024 bit durchgeführt. Erstere ermöglichen einen geringeren Ressourcenbedarf, während letztere eine geringere Taktfrequenz benötigen, um einen gewünschten hohen Durchsatz erreichen zu können.

Die Synthese- und die Place & Route-Software wurde im Wesentlichen mit ihren Standardeinstellungen verwendet. Precision RTL Synthesis wurde als Randbedingung lediglich die Eingangstaktfrequenz übergeben, woraus es die benötigten internen Frequenzen berechnen konnte. Quartus II wurden Randbedingungen bezüglich der zu verwendenden Pins des Stratix II Chips übergeben sowie einzuhaltende Setup- und Hold-Zeiten bei der Kommunikation der Ethernet-Logikmodule mit den externen PHY-Chips. Das Optimierungsziel von Quartus war auf *„balanced“* gesetzt, so dass einerseits möglichst wenig Logikressourcen verwendet werden und andererseits gleichzeitig eine möglichst hohe Taktfrequenz eingesetzt werden kann.

Das Basissystem, auf dem alle synthetisierten Systeme aufbauen, besteht aus folgenden Modulen: Den in Abschnitt 4.2.3 beschriebenen Eingangs- und Ausgangsschnittstellenmodulen, bestehend aus drei 1 Gbit/s-Ethernet-Schnittstellenmodulen mit jeweils anschließendem Wortbreiten-Anpassungsmodul, dem Scheduler und dem Demultiplexer, sowie der in Abschnitt 4.2.5 eingeführten Management-Schnittstelle mit einigen Zählern und Kontrollregistern. Um nur die Leistungsfähigkeit und den Ressourcenbedarf des Fast Path untersuchen zu können, wurde die Slow-Path-Schnittstelle für diese Synthesen nicht in die Systeme integriert.

Metriken

Altera FPGAs der Familien Stratix II bis IV bestehen aus einer Matrix von so genannten *Adaptive Lookup Modules* (ALM). Dies sind konfigurierbare Logikblöcke (entsprechend Abschnitt 2.3.4) mit jeweils zwei 4-Input-LUTs und vier 3-Input-LUTs, zwei Volladdierern und zwei Flipflops. Die sechs LUTs können auf unterschiedliche Weisen (und mit gewissen Abhängigkeiten) als zwei Abbildungstabellen mit bis zu sechs Eingängen oder als eine Tabelle mit sieben Eingängen verwendet werden. Eine solche Abbildungstabelle wird als ALUT (*Adaptive LUT*) bezeichnet.

Ein ALM realisiert somit i. d. R. zwei ALUT-Flipflop-Paare, wobei jedes solche Paar eine Abbildungstabelle mit nachgeschaltetem Flipflop, eine Abbildungstabelle ohne Flipflop oder nur ein Flipflop (ohne Abbildungstabelle) realisieren kann. Es gibt jedoch auch Fälle, in denen ein ALM nur ein ALUT-Flipflop-Paar realisieren kann. Dies ist z. B. der Fall, wenn eine Abbildungstabelle mit sieben Eingängen realisiert werden soll oder eine Abbildungstabelle mit fünf oder sechs Eingängen nicht mit einer anderen Tabelle aufgrund fehlender gemeinsamer Eingänge kombiniert werden kann. Der eingesetzte Stratix II verfügt über 24 176 ALMs und somit über 48 352 ALUT-Flipflop-Paare. Der verwendete Stratix IV enthält 212 480 ALMs, das sind 424 960 ALUT-Flipflop-Paare.[1]

Für die Angabe des Gesamtressourcenbedarfs gibt Quartus die Anzahl belegter ALUT-Flipflop-Paare an. Hierbei zählt die Software zusätzlich auch solche Paare hinzu, die aufgrund der oben erwähnten Fälle vermutlich oder sicher nicht mehr genutzt werden können. Außerdem zieht sie eine geschätzte Anzahl von Paaren wieder ab, die bei einer späteren Realisierung bei Ressourcenknappheit durch Kombination von einzelnen ALUTs und Flipflops vermutlich eingespart werden können. Bei Teilmodulen gibt Quartus lediglich die Anzahl der verwendeten ALUT-Flipflop-Paare ohne diese Korrekturen an. Im Rahmen dieser Dissertation werden Ressourcenbedarfe im Folgenden ebenso angegeben – bei Gesamtsystemen in ALUT-Flipflop-Paare mit Korrektur, bei Teilmodulen in ALUT-Flipflop-Paaren ohne Korrektur.

Der Speicher auf FPGAs ist in Form von verteilten Speicherblöcken verschiedener Kapazität realisiert. Der verwendete Altera Stratix II verfügt über 2 544 kbit Speicher in Form von 329 *M512*-Blöcken mit einer Kapazität von je 576 bit, 255 *M4K*-Blöcken mit einer Kapazität von je 4 608 bit und zwei *M-RAM*-Blöcken mit jeweils 589,8 kbit. Der untersuchte Altera Stratix IV verfügt über 21 234 kbit Speicher als 1 280 *M9K*-Blöcken mit jeweils 9 216 bit und 64 *M144K*-Blöcken mit jeweils 147 kbit. Zusätzlich können jeweils zehn ALMs zu zusätzlichen 640 bit-Speichern umfunktioniert werden. Der im Folgenden angegebene Speicherbedarf gibt stets die vom jeweiligen System geforderte Anzahl an Speicherbits an. Bei der Realisierung dieses Speichers auf dem Chip werden aufgrund der Fragmentierung in die oben eingeführten Blöcke i. d. R. mehr Speicherbits belegt.

Die angegebenen Taktfrequenzen beziehen sich auf die maximal unterstützte Taktfrequenz innerhalb des Paketverarbeitungssystems. Quartus berechnet diese Frequenz anhand des längsten kombinatorischen Pfades (kritischer Pfad) zwischen zwei Flipflops zuzüglich *Setup*- und

[1] Für den Vergleich mit anderen FPGAs entspricht, laut Altera, ein ALM etwa 2,5 *Logic Elements*, welche als konfigurierbare Logikblöcke mit einem 4-Input-LUT, einem Volladdierer und einem Flipflop definiert sind.
Stratix II: 24 176 ALMs entsprechen etwa 60 000 Logic Elements – daher die 60 in der Chip-Bezeichnung.
Stratix IV: 212 480 ALMs entsprechen etwa 530 000 Logic Elements – daher die 5 in der Chip-Bezeichnung.

Hold-Zeiten. Die Taktfrequenzen der Ein- und Ausgangsschnittstellen sowie der Management-Komponenten werden nicht betrachtet. Lediglich bei den Frequenzangaben bezüglich der Prozessor-Pool-Module wird die maximale Taktfrequenz der Prozessorkerne angegeben. Wie in Abschnitt 3.5.4 motiviert, sollte diese stets ein Vielfaches der Pipeline-Taktfrequenz sein, um eine hohe Anzahl an Verarbeitungstakten zu ermöglichen.

Zusätzlich zu den Ressourcen- und Taktfrequenzangaben wird zusätzlich auch die erreichbare Datenrate für die verschiedenen untersuchten Systeme angegeben. Diese bezieht sich stets auf die extern unterstützte Datenrate eines Ethernet-basierten Systems, d. h. mit einem minimalen Paketabstand von 20 Byte. Die in Tabelle 4.3 angegebenen Datenraten beziehen sich auf Systeme in der Ganzes-Paket-Variante der vorgeschlagenen Architektur (vgl. Abschnitt 3.4.2) und berechnen sich nach Gleichung (4.2). Die fett gedruckten Datenraten unterstützen eine Datenrate von mehr als 100 Gbit/s. Zusätzlich sind in der Tabelle die Taktfrequenzen, die größer als 150 MHz sind, fett abgedruckt. Diese heben somit Systeme hervor, welche in einer NH-Konfiguration mehr als 150 MP/s unterstützen und somit schnell genug für eine 100 Gbit/s-Line-Card sind.

Ergebnisse für das Basissystem

Zunächst werden der Ressourcenbedarf und die Leistungsfähigkeit des oben eingeführten Basissystems betrachtet. Der erste Abschnitt von Tabelle 4.2 listet die für die Eingangs- und Ausgangs-Schnittstellenmodule benötigten Ressourcen auf. Die 1 Gbit/s-Ethernet-Schnittstellenmodule sind unabhängig von der in der Verarbeitungs-Pipeline verwendeten Wortbreite, da diese noch auf kleineren Worten operieren. Erst der Ressourcenbedarf der anschließenden Wortbreitenanpassungsmodule und des Scheduler-Moduls hängt teilweise von der Wortbreite ab. Der gemeinsame Ressourcenverbrauch dieser Module liegt bei etwa 13 000 bzw. 16 000 ALUT-Flipflop-Paaren und somit bereits bei etwa 27 % bzw. 33 % der Ressourcen des verwendeten Stratix II FPGAs (jeweils für 512 bit bzw. 1024 bit Wortbreite). Beim Stratix IV belegen diese Module weniger als 4 % der verfügbaren ALUT-Flipflop-Paare. Die für Lasttests eingesetzten Verkehrsgeneratoren und -validatoren benötigen im Vergleich zu den Ethernet-Schnittstellen beinahe die vierfache Anzahl an ALUT-Flipflop-Paaren.

Tabelle 4.3 auf Seite 153 gibt den Ressourcenbedarf und die Leistungsfähigkeit verschiedener Gesamtsysteme an. Ein „leeres“ System, das nur aus dem Basissystem und zwei Pipeline-Registern besteht, benötigt 17 800 bzw. 22 400 ALUT-Flipflop-Paare. Die Differenz zu der im letzten Abschnitt angegebenen Anzahl an ALUT-Flipflop-Paaren ergibt sich aus dem Ressourcenbedarf für die Management-Komponenten, die zwei Pipeline-Register sowie den von Quartus abgeschätzten Korrekturbetrag für unnutzbare ALUT-Flipflop-Paare (s. o.). Der angegebene Speicher wird zur kurzzeitigen Pufferung der Paketdaten in den Ein- und Ausgangsschnittstellenmodulen sowie von Daten in der Management-Schnittstelle verwendet.

Mit einer Taktfrequenz von etwa 200 MHz bestimmt sich die Datenrate des Systems zu 69,7 Gbit/s bei der GP-Variante mit 512 bit breiten Worten und zu 119,8 Gbit/s bei der GP-Variante mit 1024 bit breiten Worten. In der NH-Variante ist (mit beiden Wortbreiten) ein Durchsatz von bis zu 200 MP/s und mindestens 134 Gbit/s erreichbar.

Ergebnisse für Systeme mit (ausschließlich) Programmierbare-Logik-Modulen

Die Leistungsfähigkeit und der Ressourcenbedarf von Programmierbare-Logik-Modulen hängen einzig von der jeweiligen Implementierung ab. In Tabelle 4.3 sind daher Synthese-Ergebnisse von Systemen mit Programmierbare-Logik-Modulen angegeben, die die wesentlichen Funktionen eines IPv4-Routers implementieren – die Umsetzung dieser Funktionen mit Hilfe der verschiedenen Verarbeitungsmodule werden im folgenden Unterkapitel noch ausführlich beschrieben.

Die Ergebnisse dreier solcher Systeme sind in den Zeilen 2, 3 und 7 von Tabelle 4.3 angegeben. In Zeile 2 ist ein System angegeben, das die IP-Adresssuche mit Hilfe eines externen TCAM-Bausteins durchführt. Entsprechend benötigt dieses System vergleichsweise wenig Ressourcen (7 500 bzw. 13 000 ALUT-Flipflop-Paare) für die zusätzlichen Verarbeitungsmodule gegenüber dem oben betrachteten leeren System. Die Variante mit 1024 bit-Worten benötigt aufgrund der breiteren Pipeline-Register mehr zusätzliche Ressourcen.

Die Systeme mit der algorithmischen Tree-Bitmap-IP-Adresssuche (Zeilen 3 und 7) benötigen insgesamt rund 40 000 ALUT-Flipflop-Paare, was im Falle der Stratix-II-Realisierung 85 % der verfügbaren Ressourcen darstellt. Noch kritischer ist jedoch der Speicherbedarf, so dass die Variante mit 1024 bit breiten Worten nicht mehr auf dem Stratix II FPGA realisiert werden kann. Auf diese Problematik wird in Abschnitt 4.3.3 nochmals tiefer eingegangen.

Die Systeme unterstützen Pipeline-Taktfrequenzen zwischen 177 MHz und 240 MHz, wobei diese Grenzen bei fast allen Systemen durch einen kritischen Pfad am Ausgang eines Pufferspeichers im Wortbreitenanpassungsmodul und nicht innerhalb der Programmierbare-Logik-Module begründet sind. Lediglich bei der 512 bit-Variante des Stratix IV Systems liegt der kritische Pfad im Update-Modul des IP-Adresssuchmoduls. Wie in der Tabelle zu sehen ist, sind

Tabelle 4.2: Ressourcenbedarf für verschiedene Teilmodule

	W = 512 bit	**W = 1024 bit**
Testumgebung		
drei 1 Gbit/s-Ethernet-Schnittstellenmodule	5,4 k ALUT-FFs	5,4 k ALUT-FFs
drei Wortbreiten-Anpassungsmodule	4,3 k ALUT-FFs	6,7 k ALUT-FFs
Scheduler-Modul	3,4 k ALUT-FFs	4,0 k ALUT-FFs
drei Verkehrsgeneratoren und -validatoren	19,5 k ALUT-FFs	19,5 k ALUT-FFs
Prozessor-Pool-Modul		
Prozessorkern	2,3 k ALUT-FFs	2,3 k ALUT-FFs
Datenfluss-Pipeline-Modul		
Verarbeitungseinheit (RAM)	1,2 k ALUT-FFs	1,2 k ALUT-FFs
Verarbeitungseinheit (ROM)	0,3 k–0,5 k ALUT-FFs	0,3 k–0,5 k ALUT-FFs

die 1024 bit-Systeme als GP-Variante somit in der Lage, externe Datenraten von über 100 Gbit/s zu unterstützen. In der NH-Variante sind alle betrachteten Systeme 100 Gbit/s-Ethernet-fähig, da sie alle mit einer Taktfrequenz von über 150 MHz die geforderten 150 MP/s unterstützen.

Ergebnisse für Systeme mit Prozessor-Pool-Modul

Der mittlere Teil von Tabelle 4.2 auf Seite 150 gibt den Ressourcenbedarf eines einzelnen Prozessorkerns des prototypisch realisierten Prozessor-Pool-Moduls (vgl. Abschnitt 3.7.1 und Abschnitt 4.2.4) an. Mit über 2 000 ALUT-Flipflop-Paaren belegt jeder Prozessorkern bereits etwa 5 % des Stratix II FPGAs. Daher ist die Anzahl der Kerne auf einem solchen (kleinen und schon mehrere Jahre alten) Chip zusammen mit den stets erforderlichen Eingangs- und Ausgangsschnittstellenmodulen auf etwa zehn begrenzt. Beim betrachteten Stratix IV belegt ein Kern nur etwa 0,5 % der Chip-Ressourcen.

Der Aufbau und somit auch der Ressourcenbedarf eines Prozessorkerns ist unabhängig davon, ob die Verarbeitungs-Pipeline eine Datenwortbreite von 512 bit oder 1024 bit aufweist. Zusätzlich zu den angegebenen ALUT-Flipflop-Paaren benötigt jeder Prozessorkern RAM-Bits für seinen Befehls- und Datenspeicher sowie für seinen Registersatz, der wie in Abschnitt 4.2.4 beschrieben durch drei parallele Speicher realisiert wurde, um ALUT-Flipflop-Paare zu sparen. Ob 32 oder 64 Register in den Prozessorkernen realisiert werden, ändert zwar die Anzahl an benötigten Speicherbits, macht jedoch bei der Umsetzung auf dem FPGA aufgrund der Realisierung mit M512- bzw. M9K-Blöcken keinen Unterschied.

Tabelle 4.3 gibt den Ressourcenbedarf von Paketverarbeitungssystemen an, die verschieden dimensionierte Prozessor-Pool-Module sowie die im letzten Unterabschnitt beschriebenen Programmierbare-Logik-Module mit den wesentlichen IP-Router-Funktionen enthalten. Der Stratix II ist bei einer Wortbreite von 512 bit mit den IPv4-Modulen und einem Prozessor-Pool mit sechs Prozessorkernen bereits vollständig belegt. Die maximale Taktfrequenz der Prozessorkerne beträgt 97 MHz. Der kritische Pfad verläuft hierbei durch die Forwarding-Logik und das Rechenwerk in der Execute-Stufe der RISC-Pipeline. Für eine höhere Taktfrequenz wäre eine Aufteilung der entsprechenden Operationen auf zwei Stufen oder eine Optimierung des Rechenwerks notwendig. Um sinnvolle Programme mit 20 bis 40 Taktperioden innerhalb dieses Prozessor-Pool-Moduls ausführen zu können, sollte der Pool mit einer etwa $s = 4$–8-fachen Taktfrequenz als die Basis-Paketverarbeitungs-Pipeline betrieben werden. Mit $s = 4$ wäre somit auf dem Stratix II eine Datenrate von 8,2 Gbit/s in der GP-Variante bzw. 16,3 Gbit/s in der NH-Variante unterstützbar.

Auf dem betrachteten Stratix IV belegt ein ebenso aufgebautes System mit 20 Prozessorkernen 94 200 bzw. 117 500 ALUT-Flipflop-Paare, das entspricht 22 % bzw. 28 % der verfügbaren Ressourcen. Der Prozessortakt ist hier auf maximal 140 MHz beschränkt. Der kritische Pfad verläuft bei der 512 bit Variante wiederum in der Execute-Stufe. Bei der Variante mit 1024 bit breiten Worten wird die Taktfrequenz durch die Signalverzögerung beim Initialisieren und Auslesen des Registersatzes vor und nach der Verarbeitung begrenzt. Dasselbe gilt für das System mit 102 Prozessorkernen, welches den Stratix IV nahezu vollständig belegt. Hier zeigt sich somit der Nachteil einer Pool-Anordnung von vielen parallelen Einheiten. Zur Erhöhung der Taktfrequenz müsste dieser Pfad von und zur Verteil- und Einsammellogik ebenfalls als Pipeline realisiert werden. Der unterstützte Durchsatz dieser Systeme ist für $s = 2$ bei den ersten beiden

Systemen und $s = 1$ beim System mit 102 Kernen in der GP-Variante auf 23 Gbit/s, 39 Gbit/s bzw. 21 Gbit/s begrenzt. In der NH-Variante liegt der Durchsatz des ersten 20-Kern-Systems bei 70 MP/s oder 47 Gbit/s. Um höhere Durchsätze erreichen zu können, wären kleinere oder schnellere – z. B. auch festverdrahtete – Prozessorkerne notwendig.

Ergebnisse für Systeme mit Datenfluss-Pipeline-Modul

Schließlich werden der Ressourcenbedarf und die Leistungsfähigkeit der Datenfluss-Pipeline-Module betrachtet. Tabelle 4.2 auf Seite 150 gibt die Anzahl der von einer einzelnen Verarbeitungseinheit benötigten ALUT-Flipflop-Paare an. Hierbei wird zwischen Realisierungen mit RAM- und ROM-Befehls- und Konstantenspeichern unterschieden. Bei ersteren kann man das auszuführende Programm noch nachträglich im Betrieb ändern, bei letzteren hat die Synthese-Software die Möglichkeit, nicht benötigte Register, Multiplexer und Rechenwerke einzusparen und so die Verarbeitungseinheit kompakter und schneller zu realisieren (vgl. Abschnitt 3.5.5).

Eine RAM-Verarbeitungseinheit benötigt 1 200 ALUT-Flipflop-Paare und somit 2,5 % der verfügbaren Ressourcen des verwendeten Stratix II, bzw. 0,3 % des untersuchten Stratix IV FPGA. Das ist etwa halb so viel wie der Ressourcenbedarf eines RISC-Kerns des implementierten Prozessor-Pool-Moduls. Gründe für diesen geringen Ressourcenbedarf sind die Realisierung als 16 bit-Architektur, die Einsparung eines Dekoders aufgrund horizontal-mikrocodierter Befehlswörter, ein einfacheres Rechenwerk aufgrund weniger Befehle, die Einsparung von Forwarding-Multiplexern sowie die kürzere interne Pipeline-Länge. Andererseits werden alle Register mit Flipflops realisiert und nicht als SRAM wie bei den RISC-Prozessorkernen.

Wenn der Befehlsspeicher und der Konstantenregistersatz unveränderlich ist, variiert der Ressourcenbedarf bei den durchgeführten Untersuchungen zwischen etwa 300 und 500 ALUT-Flipflop-Paaren. Hierbei wurden die ROMs mit dem Programmcode und den Konstanten für die RCP-Paketverarbeitung initialisiert (die entsprechende Umsetzung wird in Unterkapitel 4.4 beschrieben). Der Ressourcenbedarf pro Verarbeitungseinheit konnte somit in diesem Fall um bis zu 75 % reduziert werden.

Tabelle 4.3 dokumentiert den Ressourcenbedarf und die Leistungsfähigkeit verschiedener Paketverarbeitungssysteme mit unterschiedlich dimensionierten Datenfluss-Pipeline-Modulen. Ein System mit 17 Verarbeitungseinheiten wurde auf beiden FPGAs jeweils einmal mit RAMs und einmal mit ROMs (wieder mit dem RCP-Code initialisiert) realisiert. Wie zu erwarten, ist die Anzahl an benötigten ALUT-Flipflop-Paaren bei den Systemen mit ROM-Verarbeitungsmodulen wesentlich kleiner als bei den Systemen mit RAM-Verarbeitungsmodulen.

Die maximal unterstützte Pipeline-Taktfrequenz unterscheidet sich ebenfalls stark: Bei den Stratix II Systemen ist in der RAM-Variante eine Taktfrequenz von etwa 130 MHz möglich, während in der ROM-Variante eine Taktfrequenz von etwa 200 MHz möglich ist. Somit sind letztere Systeme in der GP-Variante und mit 1024 bit-Worten in der Lage, eine externe Datenrate von bis zu 116,8 Gbit/s zu unterstützen. Bei den Stratix IV Systemen sind mit RAMs als Befehlsspeicher knapp 190 MHz möglich und mit ROMs 233 MHz bzw. 282 MHz. In der GP-Variante sind somit bei 1024 bit Wortbreite sowohl mit der RAM-Variante als auch mit der ROM-Variante mehr als 100 Gbit/s unterstützbar. In der NH-Variante unterstützen alle diese Stratix-IV Systeme die benötigten 150 MP/s. Die Taktfrequenzen sind bei den RAM-Varianten stets durch die

Tabelle 4.3: Ressourcenbedarf und maximale Taktfrequenz für verschiedene Verarbeitungssysteme

FPGA	Systemkonfiguration	Wortbreite	ALUT-FFs	Speicher	Taktfrequenz	Datenrate (GP)
Stratix II	**Basissystem („Leeres“ System)**	512 bit	17,8 k (37 %)	590 kbit	**205** MHz	69,7 Gbit/s
	(nur 2 Pipeline-Register)	1024 bit	22,4 k (46 %)	885 kbit	**201** MHz	**119,8** Gbit/s
Stratix II	**Programmierbare Logik (TCAM)**	512 bit	25,3 k (52 %)	621 kbit	**197** MHz	67,0 Gbit/s
	(3 IPv4-Basis-Module, TCAM-Modul)	1024 bit	35,4 k (73 %)	936 kbit	**196** MHz	**116,8** Gbit/s
Stratix II	**Progr. Logik (Tree Bitmap)**	512 bit	41,0 k (85 %)	651 kbit	**177** MHz	60,2 Gbit/s
	(3 IPv4-Basis-Module, Tree-Bitmap-Modul)	1024 bit	– zu wenig Speicherblöcke auf FPGA –			
Stratix II	**Prozessor-Pool + Prog. Logik**	512 bit	47,9 k (99 %)	838 kbit	97 MHz[1]	8,2 Gbit/s[1]
	(**6 Proz.-kerne**, 3 IPv4-Basis-, TCAM-Modul)	1024 bit	– zu wenig ALUT-FF-Paare auf FPGA –			
Stratix II	**Datenfluss-Pipeline + Prog. Logik**	512 bit	45,5 k (94 %)	617 kbit	136 MHz	46,2 Gbit/s
	(**17 Verarb.-einh. (RAM)**, 3 IPv4-Basis-Module)	1024 bit	46,1 k (95 %)	900 kbit	125 MHz	74,5 Gbit/s
Stratix II	**Datenfluss-Pipeline + Prog. Logik**	512 bit	27,4 k (57 %)	605 kbit	**202** MHz	68,7 Gbit/s
	(**17 Verarb.-einh. (ROM)**, 3 IPv4-Basis-Module)	1024 bit	34,7 k (72 %)	888 kbit	**196** MHz	**116,8** Gbit/s
Stratix IV	**Progr. Logik (Tree Bitmap)**	512 bit	36,7 k (9 %)	646 kbit	**240** MHz	81,6 Gbit/s
	(3 IPv4-Basis-Module, Tree-Bitmap-Modul)	1024 bit	45,5 k (11 %)	982 kbit	**228** MHz	**135,9** Gbit/s
Stratix IV	**Prozessor-Pool + Prog. Logik**	512 bit	94,2 k (22 %)	1 400 kbit	140 MHz[2]	23,8 Gbit/s[2]
	(**20 Proz.-kerne**, 3 IPv4-Basis-, TCAM-Modul)	1024 bit	117,5 k (28 %)	1 716 kbit	132 MHz[3]	39,3 Gbit/s[3]
Stratix IV	**Prozessor-Pool + Prog. Logik**	512 bit	388,9 k (92 %)	4 229 kbit	63 MHz[4]	21,4 Gbit/s[4]
	(**102 Proz.-kerne**, 3 IPv4-Basis-, TCAM-Modul)	1024 bit	– zu wenig ALUT-FF-Paare auf FPGA –			
Stratix IV	**Datenfluss-Pipeline + Prog. Logik**	512 bit	44,8 k (11 %)	616 kbit	**188** MHz	63,9 Gbit/s
	(**17 Verarb.-einh. (RAM)**, 3 IPv4-Basis-Module)	1024 bit	50,3 k (12 %)	900 kbit	**189** MHz	**112,6** Gbit/s
Stratix IV	**Datenfluss-Pipeline + Prog. Logik**	512 bit	236,4 k (56 %)	782 kbit	**159** MHz	54,1 Gbit/s
	(**140 Verarb.-einh. (RAM)**, 3 IPv4-Basis-Module)	1024 bit	249,6 k (59 %)	1 136 kbit	138 MHz	82,2 Gbit/s
Stratix IV	**Datenfluss-Pipeline + Prog. Logik**	512 bit	23,8 k (6 %)	605 kbit	**282** MHz	95,9 Gbit/s
	(**17 Verarb.-einh. (ROM)**, 3 IPv4-Basis-Module)	1024 bit	29 k (7 %)	888 kbit	**233** MHz	**138,9** Gbit/s

[1,2,3,4]Proz.-Taktfreq. angegeben. [1]$s = 4$: 24 Takte, Pipeline 24 MHz. [2,3]$s = 2$: 40 Takte, Pipeline 70 MHz / 66 MHz. [4]$s = 1$: 102 Takte, Pipeline 63 MHz.

Lese- oder Schreibverzögerung beim Zugriff auf Speicherblöcke begrenzt – entweder von den Befehlsspeichern oder von den parallel zu den Verarbeitungseinheiten geschalteten Paketpuffern. Bei den ROM-Varianten liegt der kritische Pfad weiterhin im Wortbreitenanpassungsmodul.

Die maximale Anzahl an Datenfluss-Pipeline-Verarbeitungseinheiten mit RAM in einem Stratix IV liegt bei etwa 140. Allerdings ist die Anzahl hierbei nicht durch die ALUT-Flipflop-Paare begrenzt, sondern durch die SRAM-Blöcke auf dem FPGA. Der Grund hierfür ist, dass jede der Einheiten für ihren Befehlsspeicher trotz dessen geringer Größe jedoch aufgrund dessen großer Breite von etwa 100 bit jeweils drei eigene M9K-Blöcke benötigt.

Zusammenfassung und Schlussfolgerungen

In diesem Abschnitt wurde der Ressourcenbedarf und die maximale Taktfrequenz des prototypischen Paketverarbeitungssystems der neuen Architektur präsentiert und diskutiert. Dabei wurden verschiedene Konfigurationen betrachtet. Zusammenfassend lässt sich hierbei feststellen, dass viele der untersuchten Systeme einen Durchsatz von über 150 MP/s unterstützen können und somit für 100 Gbit/s-Ethernet Line Cards geeignet sind.

Insbesondere in der NH-Architekturvariante, d. h. falls die Systeme nur das erste Datenwort eines jeden Pakets durch die Pipeline transportieren und dort verarbeiten, können die meisten Systeme mehr als 150 MP/s unterstützen, da sie mit mehr als 150 MHz betrieben werden können. In der GP-Variante, d. h. wenn das ganze Paket verarbeitet werden kann, ist eine Datenwortbreite von 1024 bit notwendig, um einen Durchsatz von 100 Gbit/s garantieren zu können.

Beim Vergleich der unterschiedlichen Verarbeitungsmodule fällt auf, dass die Programmierbare-Logik-Module i. d. R. die höchsten Taktfrequenzen und somit die höchsten Durchsätze unterstützen können. Der Grund hierfür ist, dass diese häufig nur einfache Operationen in einer Taktperiode durchführen müssen und komplexere Operationen bei der Implementierung einzeln optimiert werden können.

Die Prozessor-Pool-Module können nur mit den geringsten Frequenzen betrieben werden. Ursache für die geringen Taktfrequenzen sind die komplexen Forwarding-Mechanismen, welche eine lange kombinatorische Verzögerung mit sich bringen. Der Prototyp zeigt die prinzipielle Realisierbarkeit und Funktionsweise eines solchen Systems – für einen Einsatz in 100 Gbit/s-Netzen sind jedoch noch Optimierungen hinsichtlich des Ressourcenbedarfs und der Taktfrequenz notwendig. Alternativ können auch festverdrahtete Prozessorkerne eingesetzt werden.

Die Datenfluss-Pipeline-Module eignen sich sehr gut für die Paketverarbeitung mit 100 Gbit/s, da sie wenige Ressourcen benötigen und mit einer hohen Taktfrequenz betrieben werden können. Insbesondere wenn der Befehlsspeicher als ROM realisiert wird, sind damit sehr hohe Durchsätze realisierbar.

Schließlich kann festgestellt werden, dass der auf der eingesetzten Universellen Hardware-Plattform verwendete Stratix II aus dem Jahre 2005 gut zum Testen für kleinere Systeme geeignet ist. Für die Realisierung größerer und erweiterbarer Paketverarbeitungssysteme in Kern- und Metronetzen ist ein größerer und leistungsfähigerer FPGA notwendig. Hierfür ist der zweite in

den Untersuchungen verwendete FPGA, der Stratix IV von Altera, gut geeignet – und vermutlich auch ein aktueller Xilinx Virtex 5 FPGA (vgl. Abschnitt 2.4.2).

Um in etwa fünf bis zehn Jahren auch Systeme mit einem Durchsatz von 400 Gbit/s oder 1 Tbit/s realisieren zu können, muss bei gleichbleibender minimaler Paketgröße die Taktfrequenz entsprechend erhöht werden. Jedoch ist zu erwarten, dass dies zukünftige FPGA-Generationen sowie auch speziell für diese Architektur gefertigte Chips unterstützen werden. Schließlich sind die aktuellen FPGA-Taktfrequenzen noch um etwa einen Faktor zehn kleiner als die Frequenzen aktueller Standardprozessoren und bereits heute gibt es Ankündigungen von FPGAs mit innovativen Architekturen und Taktfrequenzen mit bis zu 1,5 GHz [107, 110, 112] (vgl. Abschnitt 2.4.2).

Ob ein System mit der vorgestellten Architektur auch einfach und effizient verwendbar ist, um verschiedene Paketverarbeitungsfunktionen mit Hilfe seiner verschiedenen Verarbeitungsmodule zu realisieren, wird in den nächsten beiden Unterkapiteln überprüft.

4.3 Umsetzung von IP-Router-Funktionen

Um die Anwendbarkeit von Systemen mit der vorgestellten Architektur zu überprüfen, wurden die Fast-Path-Funktionen eines IP-Routers sowie Funktionen möglicherweise zukünftig eingesetzter Protokolle (Unterkapitel 4.4) implementiert. In diesem Unterkapitel wird auf die Umsetzung der IPv4-Fast-Path-Funktionen eingegangen. Abschnitt 4.3.1 beschreibt die Implementierung von Basisfunktionen der IP-Verarbeitung auf Register-Transfer-Ebene und auf Software-Ebene. Die aufwändige IP-Adresssuche kann nicht effizient für die geforderten Paketraten in Software implementiert werden. Daher wurde diese prototypisch in Hardware realisiert. Abschnitt 4.3.2 stellt ein IP-Adresssuchmodul unter Verwendung eines TCAMs vor und Abschnitt 4.3.3 ein Modul, das dafür den Tree-Bitmap-Algorithmus verwendet.

4.3.1 Basisfunktionen

Basisfunktionen der Fast-Path-Verarbeitung eines IPv4-Routers wurden prototypisch sowohl mit Hilfe von Programmierbare-Logik-Modulen als auch mit Hilfe eines Prozessor-Pool-Moduls realisiert. Zu den wichtigsten Basisfunktionen eines IPv4-Routers gehört zum einen die Analyse und Überprüfung diverser Header-Felder auf Schicht 2 und 3, in Folge dessen fehlerhafte Pakete verworfen und besonders zu behandelnde Pakete zur Verarbeitung im Slow Path markiert werden. Zum anderen gehört hierzu auch das Dekrementieren des Rest-Lebenszeit-Felds (*Time To Live, TTL*) und die entsprechende Anpassung der Prüfsumme. Die Fragmentierung von Paketen sowie die Broadcast- und Multicast-Behandlung wurden nicht betrachtet.

Funktionen in programmierbarer Logik

Die Basisfunktionen eines IPv4-Routers wurden in drei Module untergliedert, welche jeweils mit einem Programmierbare-Logik-Modul realisiert wurden. Da die Module nur die Header-Felder verarbeiten müssen, operieren sie jeweils nur auf dem ersten Datenwort eines Pakets und

leiten die anderen Pakete lediglich weiter ins nächste Modul. Eine erste Version dieser IPv4-Verarbeitungsmodule wurde in der Studienarbeit von Mück [17] unter der Anleitung des Autors realisiert.

Schicht-2-Überprüfung: Dieses Modul führt Überprüfungen hinsichtlich des Schicht-2-Protokolls durch, wobei der Prototyp ausschließlich Ethernet als Schicht-2-Technologie unterstützt. Das Modul überprüft

- ob die Rahmenlänge (angegeben in den Metadaten) zwischen 64 Byte und 1518 Byte ist
- ob die Ethernet-Zieladresse mit der Adresse des Ingress-Ports übereinstimmt
- ob das im Ethernet-Typfeld angegebene nächste Protokoll IPv4 ist.

Wenn eine der beiden ersten Überprüfungen negativ ausfällt, verwirft das Modul das Paket. Ist das folgende Protokoll nicht IPv4, so leitet es das Paket in den Slow Path um. Alle anderen Pakete leitet das Modul weiter zum nächsten Verarbeitungsmodul.

Alle drei Überprüfungen können durch wenige einfache Vergleichsoperationen realisiert werden. Daher kann die Verarbeitung kombinatorisch innerhalb einer Taktperiode erfolgen.

Schicht-3-Überprüfung: Dieses Modul überprüft verschiedene Felder im IPv4-Header, dabei prüft es

- ob die IP-Version vier ist
- ob die angegebene Header-Länge fünf ist (d. h. keine IP-Optionen vorhanden sind)
- ob die Zieladresse eine Adresse des Routers ist oder innerhalb des lokalen Subnetzes des Ingress-Ports liegt
- ob die IP-Prüfsumme stimmt.

In Abhängigkeit der Prüfergebnisse verwirft das Modul das Paket, leitet es in den Slow Path um oder übergibt es dem nächsten Verarbeitungsmodul.

Die Berechnung der Prüfsumme erfordert die Einerkomplement-Addition aller 16 bit-Worte im IP-Header. Um einen hohen Durchsatz unterstützen zu können, wird dies in einer kurzen Pipeline über mehrere Takte hinweg durchgeführt. Alle anderen Überprüfungen können wiederum durch einfache kombinatorische Komparatoren realisiert werden.

TTL- und Prüfsummen-Modifikation: Dieses Modul

- überprüft das Rest-Lebensdauer-Feld (TTL) im IP-Header
- dekrementiert dieses Feld um eins
- aktualisiert daraufhin die Prüfsumme über den IP-Header.

Falls die Rest-Lebensdauer null ist, verwirft das Modul das Paket. Für die Berechnung der neuen Prüfsumme wird das in [183] beschriebene inkrementelle Verfahren angewendet. Dieses Modul ist als dreistufige Pipeline realisiert.

Zusammenfassend kann festgestellt werden, dass die Implementierung dieser Funktionen in VHDL meist nur wenige Zeilen Code erforderte. Lediglich die Berechnung der Prüfsumme in einer Pipeline-Struktur ist etwas aufwändiger.

Tabelle 4.4 zeigt den Ressourcenbedarf der drei implementierten Verarbeitungsmodule. Der sehr kleine Ressourcenbedarf des Schicht-2-Überprüfungsmoduls liegt daran, dass dieses nur

die Kombinatorik der eigentlichen Funktionalität enthält und keine Register. Das Schicht-3-Überprüfungsmodul benötigt aufgrund der aufwändigen Prüfsummenberechnung etwa sechs mal so viele ALUT/Flipflop-Paare, die fünf Registerstufen zur Verzögerung der Paket- und Metadaten sind durch Speicherzellen realisiert. Das dritte Modul realisiert seine zwei Registerstufen durch Flipflops. Dies erklärt den vergleichsweise hohen Bedarf an ALUT/Flipflop-Paaren. Relativ zur Gesamtkapazität des Stratix-II-FPGAs (bzw. des Stratix IV) benötigen alle drei Module zusammen etwa 3 % (0,4 %) der Logikzellen bei einer Datenwortbreite W von 512 bit bzw. 5 % (0,6 %) für $W = 1024$ bit.

Funktionen in Assembler

Einige IP-Basisfunktionen wurden zusätzlich in Assembler zur Ausführung in einem Prozessor-Pool-Modul implementiert. Hiermit sollte überprüft werden, ob dies prinzipiell möglich ist und wie viele Befehle bzw. Takte zur Ausführung solcher Funktionen erforderlich sind. Daher wurde ein Teil des Assembler-Codes aus der Beispiel-Implementierung eines IPv4-Weiterleitungs-Software-Moduls von Intel für die IXP2xxx-Netzprozessoren [184] in die Assembler-Sprache des Prozessor-Pool-Moduls übersetzt. Der übersetzte Teil prüft das IP-Paket vor der IP-Adresssuche auf Korrektheit. Da der Befehlssatz des verwendeten Prozessor-Pool-Moduls große Ähnlichkeit mit dem Befehlssatz der Intel Netzprozessoren aufweist, war diese Umsetzung ohne Weiteres möglich.

Tabelle 4.5 listet die implementierten IP-Basisfunktionen auf und gibt die Anzahl der pro Funktion benötigten Befehle an. Bei einer negativ ausfallenden Prüfung verzweigt das Programm aus der Prüfsequenz, was zu zwei zusätzlichen Ausführungstakten führt. Da die Ausführung danach jedoch abbricht – da das Paket entweder verworfen oder zum Slow-Path umgeleitet wird – beeinflussen diese zwei Takte nur bei der letzten Prüfung die Gesamtausführungszeit aller Funktionen. Die Extraktion der einzelnen Header-Felder aus den verschiedenen Registern sowie das Vergleichen benötigt – aufgrund des spezialisierten Befehlssatzes – pro Funktion meist nur wenige Takte. Lediglich die Berechnung der IP-Prüfsumme benötigt 15 Takte. Die Gesamtausführungsdauer von 44 Takten könnte beispielsweise mit einem Prozessor-Pool-Modul mit 16 Prozessorkernen mit der dreifachen Taktfrequenz des Pipeline-Taktes realisiert werden ($16 \cdot 3 = 48 > 44$).

Tabelle 4.4: Ressourcenbedarf für Verarbeitungsmodule mit IPv4-Basisfunktionen

Funktion	**W = 512 bit**	**W = 1024 bit**
Schicht-2-Überprüfung (1-stufig)	0,05 k ALUT-FFs	0,05 k ALUT-FFs
Schicht-3-Überprüfung (6-stufig)	0,3 k ALUT-FFs + 4 kbit Speicher	0,3 k ALUT-FFs + 8 kbit Speicher
TTL-/Prüfsummen-Modifikation (3-stufig)	1,2 k ALUT-FFs	2,2 k ALUT-FFs
Alle realisierten Funktionen	1,5 k ALUT-FFs	2,5 k ALUT-FFs

Schlussfolgerungen

Die Umsetzung der IP-Basisfunktionen war sowohl für die Programmierbare-Logik-Module als auch für die Prozessor-Pool-Module einfach möglich. Bei der Realisierung in programmierbarer Logik mussten die Prüfsummen-Berechnungen als Pipeline realisiert werden, um einen hohen Durchsatz gewährleisten zu können. Bei der Realisierung in Software zeigten sich die Vorteile des spezialisierten Befehlssatzes, so dass zur Überprüfung einzelner Header-Felder jeweils nur wenige Takte benötigt wurden.

4.3.2 Hardware-unterstützte IP-Adresssuche

Die Suche nach dem längsten übereinstimmenden Präfix bei der IP-Adresssuche ist für die hohen Paketraten in Metro- und Kernnetzen nicht schnell genug in Software realisierbar. In diesem Abschnitt wird die Realisierung eines IP-Adresssuchmoduls beschrieben, das für die Suche einen externen TCAM-Baustein verwendet. Dieses Modul wurde in Form eines Programmierbare-Logik-Moduls realisiert.

TCAM-Bausteine sind dreiwertige inhaltsadressierte Speicherbausteine (vgl. Abschnitt 4.2.2), mit denen sehr schnell eine Suche nach dem längsten passenden Präfix durchgeführt werden kann, wie es für die IP-Adresssuche notwendig ist. Der TCAM muss dafür alle Subnetzadressen der Weiterleitungstabelle enthalten, wobei jeweils die Endgerät-Adressbits *don't care*-Bits sind. Hierbei muss sichergestellt sein, dass Einträge mit längerem Netzpräfix vor Einträgen mit jeweils kürzerem Präfix stehen. Nach Anlegen der zu untersuchenden IP-Zieladresse an den Eingang des TCAMs gibt dieser den Index der ersten übereinstimmenden Subnetzadresse zurück. Dieser kann nun zum Auslesen der Weiterleitungsinformationen aus einer Weiterleitungstabelle verwendet werden.

Abbildung 4.8 zeigt den prinzipiellen Aufbau des IP-Adresssuchmoduls. Die IP-Zieladresse wird aus jedem ersten Wort der Paketdaten abgezweigt und an ein TCAM-Schnittstellen-Mo-

Tabelle 4.5: Anzahl der Befehle bzw. Ausführungstakte der IP-Basisfunktionen

IPv4-Basisfunktion	**Befehle**	**Takte (max.)**
Zählen der Pakete pro Ingress-Port (im Datenspeicher)	4	5
Prüfen, ob Paketlänge aus Metadaten größer 20	4	6
Prüfen auf Version 4 und Header-Länge 5	1	3
Prüfen, ob IP-Gesamtlänge und Paketlänge (Metadaten) konsistent	4	6
Prüfen, ob IP-Gesamtlänge größer 20	2	4
Berechnen und Prüfen der IP-Prüfsumme	15	17
Prüfen d. Quelladresse auf 0, Loopback, Multi-/Broadcast, Class E	12	14
Alle Funktionen (vor IP-Adresssuche)	42	44

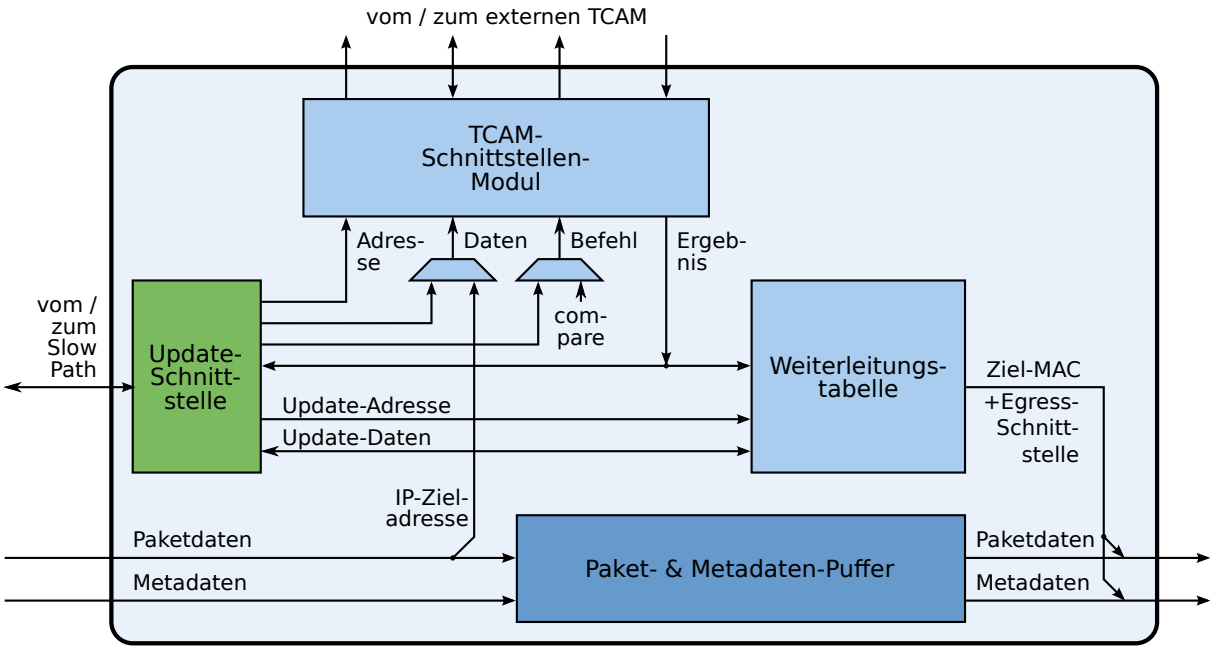

Abbildung 4.8: IP-Adresssuchmodul für externen TCAM

dul angelegt. Dieses steuert die Kommunikation mit dem externen TCAM-Baustein. Nachdem der TCAM – nach einer festen Latenz – den gefundenen Index zurückgegeben hat, wird damit die Weiterleitungstabelle adressiert. Da der Prototyp zunächst keine IP-Adressauflösung über ARP unterstützt, gibt diese Tabelle direkt die MAC-Adresse des nächsten Routers und die entsprechende Egress-Schnittstelle aus, anstatt zunächst dessen IP-Adresse. In einer fortgeschrittenen Implementierung dieses Moduls kann dies entsprechend erweitert werden. Während dieser Suchschritte speichert ein Puffer die Paket- und Metadaten. Nach einer festen Latenz gibt der Puffer diese Daten wieder aus, so dass die MAC-Adresse und der Ausgangs-Port in den Paket- bzw. Metadaten überschrieben werden können.

Die Weiterleitungstabelle wurde durch FPGA-internen SRAM realisiert, was dessen Kapazität einschränkt. Zu Testzwecken wurde lediglich eine Speichertiefe von 64 Einträgen realisiert. Für den Einsatz in Metro- und Kernnetzen kann dieser Speicher auch durch einen externen SRAM-Baustein ersetzt werden. Der verwendete TCAM der Firma Netlogic [176] unterstützt alle zwei Takte eine Suche bei einer Taktfrequenz von 133 MHz. Aufgrund von Einschränkungen der PLLs auf dem FPGA wurde der Chip jedoch bei Tests stets mit einer Frequenz von 125 MHz betrieben, so dass ein Maximaldurchsatz von 62,5 MP/s erreicht werden konnte. Wie in Abschnitt 4.2.7 angegeben wurde, könnte das Programmierbare-Logik-Modul an sich auch mit einer höheren Taktfrequenz betrieben werden. Tabelle 4.6 zeigt den vergleichsweise geringen Ressourcenverbrauch für dieses Modul für eine Wortbreite von 512 bit und 1024 bit.

4.3.3 Algorithmische IP-Adresssuche

Anstatt für die schnelle IP-Adresssuche einen energieintensiven TCAM-Baustein zu verwenden, ist es auch möglich, diese Suche mittels mehrerer Speicherzugriffe gemäß spezieller Algorithmen durchzuführen. Eine effiziente und häufig genutzte Klasse solcher IP-Adresssuchal-

gorithmen basiert auf einer Suche über eine Baum-Datenstruktur, welche den gesamten IP-Adressraum abdeckt. Die Verfahren dieser Klasse werden als *Trie*-basierte Suchalgorithmen bezeichnet.[2] Um bei der Suche übermäßig viele Speicherzugriffe zu vermeiden, repräsentieren die Knoten auf jeder Ebene des Baums i. d. R. mehrere Bits der IP-Adresse (Multi-Bit-Suchbaum). Der Algorithmus traversiert in Abhängigkeit der jeweiligen Adressbits – beginnend bei den höchstwertigen – einen Pfad von Knoten zu Knoten durch den Baum, bis kein passender Eintrag mehr vorhanden ist. Die zuletzt gefundene Routing-Information ist die mit dem längsten übereinstimmenden IP-Subnetzpräfix.

Ruiz-Sánchez [58] gibt einen Überblick über die wichtigsten solchen Verfahren, welche sich hinsichtlich der Anzahl und Wortbreite der Speicherzugriffe, des benötigten Speicherbedarfs und des Aufwands, die Routing-Informationen anzupassen, unterscheiden. Bezüglich dieser Kriterien stellt der *Tree-Bitmap*-Algorithmus von Eatherton [59] einen der besten Algorithmen dar. Cisco verwendet diesen Algorithmus in seinen CRS-1 Kernnetz-Routern [97] (vgl. Abschnitt 2.4.1). In diesem Abschnitt wird ein Programmierbare-Logik-Modul mit einer Implementierung des Tree-Bitmap-Algorithmus vorgestellt und dessen Ressourcen- und Leistungsdaten diskutiert. Mit dieser Implementierung sollte untersucht werden, ob und wie solche Trie-basierten Algorithmen innerhalb der vorgestellten neuen Paketverarbeitungsarchitektur mit einem Durchsatz von einem Minimalpaket pro Takt realisiert werden können. Die Implementierung führte Teuchert im Rahmen seiner Studienarbeit [3, 15] unter Anleitung des Autors durch.

Der Tree-Bitmap-Algorithmus speichert die Informationen über abgelegte IP-Routing-Informationen in den Knoten des Baums mittels zweier *Bitmaps:* Einer so genannten Inneren Bitmap mit Informationen bezüglich der Bitkombinationen der aktuell betrachteten Adressbits und einer Äußeren Bitmap mit Informationen über nachfolgende Kind-Knoten. Die Bitmaps realisieren eine äußerst kompakte Darstellung, wobei ein gesetztes Bit das Vorhandensein weiterer Informationen bedeutet und über die Anzahl der zuvor gesetzten Bits ein Versatz (*Offset*) bezüglich einer Startadresse im Speicher ermittelt werden kann, wo die weiterführenden Informationen abgelegt sind. Die genaue Beschreibung des Algorithmus und seiner Datenstrukturen ist in [59] abgedruckt.

[2] Der Begriff *Trie* soll einerseits an *Tree* (Baum) erinnern, da diese Algorithmen auf baumartigen Datenstrukturen operieren, andererseits hat der Begriff seinen Ursprung in dem Wort *re**trie**ve* (wieder auffinden), da diese Algorithmen das Auffinden zuvor abgelegter (Routing-)Informationen zum Ziel haben.

Tabelle 4.6: Ressourcenbedarf für TCAM-unterstützte IP-Adresssuche

IP-Adresssuche mit externem TCAM	**W = 512 bit**	**W = 1024 bit**
Logik und Flipflops	3,1 k ALUT-FFs	4,1 k ALUT-FFs
Paket- & Metadatenpuffer	17 kbit Speicher	33 kbit Speicher
kleine interne Weiterleitungstabelle	3 kbit Speicher	3 kbit Speicher
Puffer in TCAM-Schnittstellenmodul	6 kbit Speicher	6 kbit Speicher

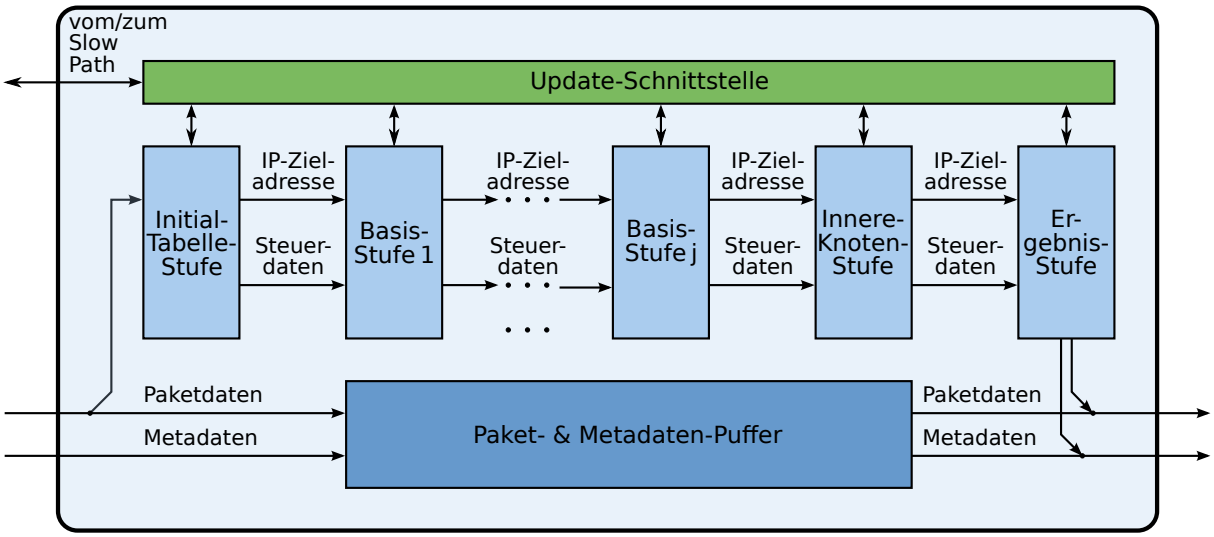

Abbildung 4.9: IP-Adresssuchmodul mit Tree-Bitmap-Algorithmus-Implementierung

Abbildung 4.9 zeigt ein Blockschaltbild des implementierten Moduls. Das erste Modul, die Initial-Tabellen-Stufe, verwendet die ersten (i. d. R. 6–12) Bits der aus den Paketdaten abgezweigten IP-Adresse als Index, um die Adresse des ersten Tree-Bitmap-Knoten in Basis-Stufe 1 zu laden. Diese lädt den entsprechenden Knoten und wertet die Äußere Bitmap aus, um daraus die Adresse des nächsten Knotens zu ermitteln. Nacheinander führen alle Basis-Stufen diese Verarbeitung durch, bis kein weiterer passender Knoten gefunden wird. Die Anzahl der Basis-Stufen ist so dimensioniert, dass alle 32 bit bzw. 128 bit der IP-Adresse verarbeitet werden können. Nach der letzten Basis-Stufe verarbeitet die Innere-Knoten-Stufe bis zu zwei Innere Bitmaps der zuletzt besuchten Knoten. Mit den Ergebnissen der letzten Stufe kann die Ergebnis-Stufe schließlich die Informationen bezüglich des nächsten Routers aus dem Speicher laden. Die Ergebnisse werden in die Paket- und Metadaten geschrieben, welche während der Suche im Paket- & Metadatenpuffer lagen.

Die Verarbeitung innerhalb der beschriebenen Stufen ist ebenfalls jeweils als Pipeline, bestehend aus ein bis acht Registerstufen, aufgebaut, um einen hohen Durchsatz erreichen zu können. Das Modul wurde so implementiert, dass es in hohem Maße anpassbar ist. So kann man z. B. die Anzahl der pro Stufe zu verarbeitenden IP-Adressbits beliebig einstellen und somit die Anzahl der zu verwendenden Stufen. Des Weiteren lassen sich diverse Optimierungen ein- und ausschalten. Beispielsweise stellt auch die nachträgliche Verarbeitung der Inneren Bitmap in einer speziellen Stufe am Ende der Pipeline eine Optimierung des Tree-Bitmap-Algorithmus dar und das Modul kann auch ohne diese Optimierung betrieben werden, so dass bereits die Basis-Stufen die Inneren Bitmaps auswerten.

Tabelle 4.7 zeigt den Ressourcenbedarf für eine beispielhafte Realisierung des Moduls in der in Abbildung 4.9 dargestellten Konfiguration, wobei zunächst 8 bit und anschließend viermal 6 bit der IPv4-Adresse verarbeitet werden. Dieses Modul benötigt etwa 10 000 ALUT/Flipflop-Paare und somit etwa 20 % der Ressourcen des auf der UHP verwendeten Stratix II und unter 2,5 % des untersuchten Stratix IV FPGA. Auch die benötigten Speicherbits für den Paket- & Metadatenpuffer stellt kein Problem für die betrachteten Chips dar.

Nicht angegeben in Tabelle 4.7 ist jedoch der benötigte Speicher der Tree-Bitmap-Datenstrukturen. Die beiden betrachteten FPGAs weisen eine Gesamtspeicherkapazität von 2,5 Mbit bzw. 21,2 Mbit auf. Allerdings benötigt das vorgestellte Modul durchschnittlich etwa 70 bit pro Präfix (vgl. hierzu [15]). Diese Speicherblöcke können somit maximal 36 000 bzw. 303 000 Routing-Einträge unterstützen. Dies ist allerdings nur bei optimaler Ausnutzung der Blöcke möglich. Zudem wird dieser Speicher auch für andere Module auf dem FPGA benötigt. Aktuelle Kernnetzrouter enthalten bereits über 300 000 Präfixe [185]. Somit ist eine Realisierung auf einem Stratix II nur mit Hilfe externer Speicher-Chips möglich und auch bei den neusten FPGAs, wie dem betrachteten Stratix IV, ist der Speicher der wesentliche begrenzende Faktor.

Dieses Modul wurde, wie bereits in Abschnitt 4.2.7 beschrieben, in verschiedenen Kombinationen mit anderen Verarbeitungsmodulen in das Gesamtsystem integriert. In solchen Systemen unterstützt das Modul eine Taktfrequenz von 177 MHz auf dem Stratix II und bis zu 240 MHz auf dem Stratix IV. Somit ist es gut für einen Einsatz auf 100 Gbit/s-Line-Cards geeignet. Weitere Ergebnisse zu Durchsatz und Ressourcenverbrauch dieses Moduls in seinen verschiedenen Konfigurationen kann in [3, 15] gefunden werden.

4.4 Umsetzung von neuen Protokollen und Diensten

Das Rate Control Protocol (RCP) und die TCP Quick-Start Erweiterung wurden bereits in Abschnitt 2.1.3 als viel versprechende Ansätze für eine zukünftige Router-unterstützte Überlastregelung vorgestellt. Um die Anwendbarkeit der neuen Paketverarbeitungsarchitektur zu überprüfen, wurden die für diese Protokolle benötigten Fast-Path-Router-Funktionen auf verschiedenen Verarbeitungsmodulen implementiert und innerhalb des vorgestellten Prototyps getestet. In Abschnitt 4.4.3 wird schließlich ausgehend von den in diesem und im letzten Unterkapitel beschriebenen Implementierungen ein Ausblick auf die Umsetzbarkeit anderer Protokolle und Dienste auf Systemen dieser Architektur gegeben.

4.4.1 Rate Control Protocol

Das Rate Control Protocol (RCP) sieht vor, dass die Endgeräte den Routern in jedem Paket in einem speziellen RCP-Header-Feld ihre gewünschte Senderate mitteilen. Die Router akzeptieren diesen Wert oder verringern ihn. Nach der Übertragung des Pakets kopiert der Empfänger den von den Routern akzeptierten Wert in einem Antwortpaket in ein anderes Feld des RCP-

Tabelle 4.7: Ressourcenbedarf für die IP-Adresssuche nach dem Tree-Bitmap-Algorithmus

IP-Adresssuche (Tree-Bitmap-Algorithmus)	**W = 512 bit**	**W = 1024 bit**
Logik und Flipflops	9,7 k ALUT-FFs	10,7 k ALUT-FFs
Paket- & Metadatenpuffer	36 kbit Speicher	69 kbit Speicher

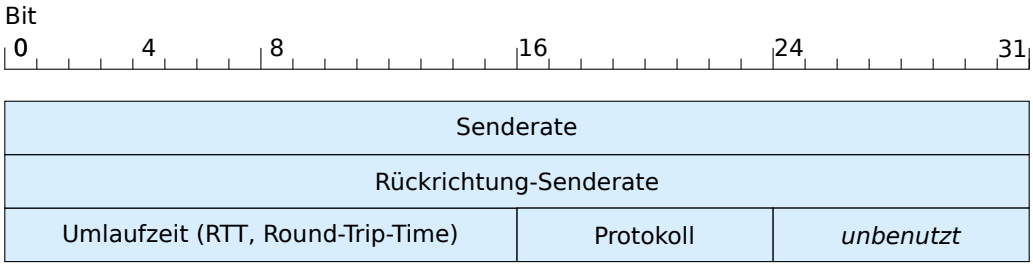

Abbildung 4.10: Aufbau des RCP-Headers

Headers (Rückrichtung-Senderate), welches von den Routern auf dem Rückweg nicht mehr verarbeitet wird. Nach Empfang dieses Antwortpakets kann der Sender mit der angegebenen Rate senden. Abbildung 4.10 zeigt den Aufbau des RCP-Headers.

Die RCP-spezifische Verarbeitung in den Routern umfasst folgende Schritte:

1. Identifizierung ankommender RCP-Pakete anhand des Protokoll-Felds im IP-Header
2. Addition der Paketlänge zur Summe aller Paketlängen des entsprechenden Ausgangs-Ports
3. Überprüfen des Werts im Umlaufzeit-Feld (*Round Trip Time*, RTT). Falls dieser kleiner als ein Maximalwert ist, Addition des Werts zur Summe der Umlaufzeiten des entsprechenden Ausgangs-Ports und Inkrementierung eines dazu gehörigen Paketzählers
4. Überprüfen des Werts im Senderate-Feld. Falls dieser größer als die aktuell erlaubte Rate ist, Überschreiben dieses Felds mit dem Wert der erlaubten Rate.

Diese Funktionalität wurde sowohl für das prototypische Prozessor-Pool-Modul als auch für das prototypische Datenfluss-Pipeline-Modul in Assembler implementiert und auf dem Prototyp getestet. Tabelle 4.8 zeigt die hierfür jeweils benötigte Anzahl an Befehlen und Takten. Sowohl das Prozessor-Pool-Modul als auch das Datenfluss-Pipeline-Modul benötigen etwa 20 Takte. Beide Module sind somit mit relativ wenigen Ressourcen realisierbar.

Die etwas kleinere Befehlsanzahl beim Datenfluss-Pipeline-Modul liegt in erster Linie daran, dass bei diesem Modul alle Zähler und Konstanten in (den lokalen) Registern abgelegt sind und somit nicht wie beim Prozessor-Pool-Modul extra Lade- und Speicherbefehle zum Lesen und Zurückschreiben benötigt werden. Des Weiteren können die Sprungbefehle bei diesem Modul parallel zu einer Operation – d. h. in null Takten – ausgeführt werden. Die Extraktion des Paketlängenfelds aus den Metadaten ist jedoch beim Prozessor-Pool-Modul aufgrund seines geeigne-

Tabelle 4.8: Anzahl der Befehle bzw. Ausführungstakte der RCP-Funktionalität

RCP-Funktionalität	**Befehle**	**Takte (max.)**
auf Prozessor-Pool-Modul	24	24
auf Datenfluss-Pipeline-Modul	17	17

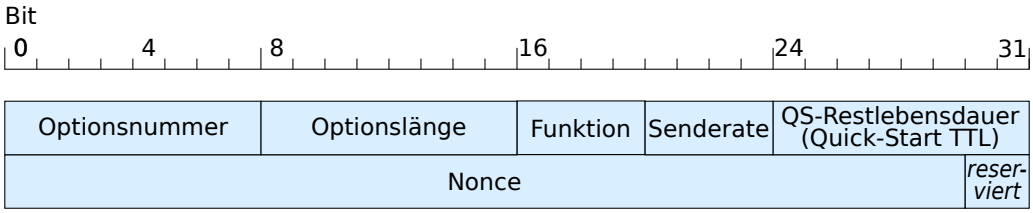

Abbildung 4.11: Aufbau der IP-Option für Quick-Start

teren Befehlssatzes effizienter realisierbar. Wie aus der Tabelle ableitbar ist, können die Befehle bei der Prozessor-Pool-Implementierung so angeordnet werden, dass auch hier ein Befehl pro Takt ausgeführt werden kann.

4.4.2 TCP Quick-Start Erweiterung

Die TCP Quick-Start Erweiterung sieht vor, dass die Endgeräte beim Aufbau einer TCP-Verbindung sowie nach längeren Sendepausen anstatt des TCP Slow-Starts über eine IP-Option eine Senderate anfordern, welche die Router entsprechend ihrer Auslastung verringern können. Abbildung 4.11 zeigt den Aufbau der für die Hinrichtung verwendeten IP-Option. Das Ergebnis der Anfrage wird vom Empfänger zum Sender in einer TCP-Option wiederum ohne Einfluss der Router zurückgesendet.

Die Verarbeitung der Quick-Start IP-Option umfasst folgende Schritte:

1. Identifizierung ankommender Quick-Start Senderate-Anforderungen anhand verschiedener Felder im IP-Header (Header-Länge, Optionsnummer und -länge, Quick-Start-Funktion)

2. Addition der Paketlänge zur Summe aller Paketlängen des entsprechenden Ausgangs-Ports

3. Überprüfen des Werts im Senderate-Feld. Falls mehr angefordert wird als am entsprechenden Ausgangs-Port erlaubt, entsprechende Verringerung des Werts

4. Verringern der noch erlaubten Datenrate des entsprechenden Ausgangs-Ports um den gerade zugesagten Wert

5. Falls verringerte Senderate gleich Null, Nullsetzen aller anderen Quick-Start-Header-Felder, sonst Dekrementieren des Quick-Start-TTL-Felds und Anpassen des Nonce-Feldes[3] entsprechend der Verringerung des Senderate-Werts

6. Aktualisierung der IP-Prüfsumme.

IP-Optionen werden üblicherweise im Slow Path verarbeitet, da sie nur selten vorkommen. Um Paketumsortierungen beim TCP-Verbindungsaufbau zu vermeiden, ist bei Quick-Start jedoch eine Verarbeitung im Fast Path vorzuziehen (vgl. auch [179]). Um die Anwendbarkeit der in

[3] Das Nonce-Feld enthält eine dem Sender bekannte zufällige Bitfolge, von der bestimmte Bits bei Dekrementierung der Senderate verändert werden müssen. Der Sender kann damit – mit einer bestimmten Wahrscheinlichkeit – eine unerlaubt wieder hochgesetzte Senderate erkennen.

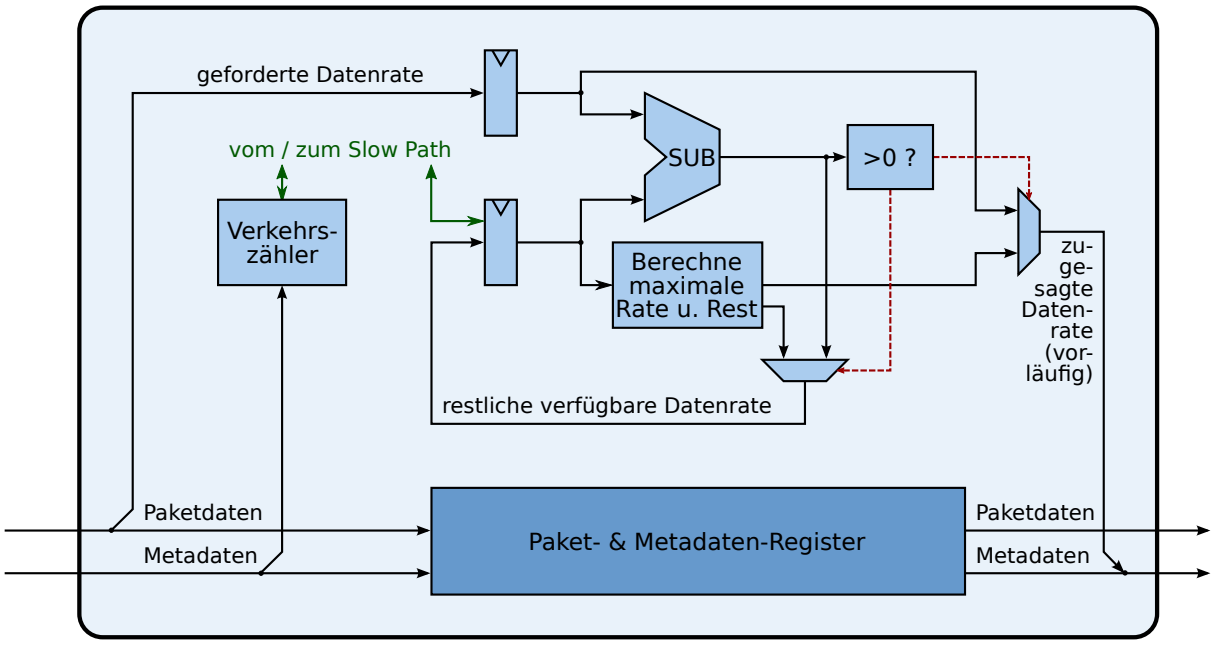

Abbildung 4.12: Prinzipieller Aufbau des Quick-Start Programmierbare-Logik-Moduls

dieser Arbeit vorgestellten Architektur zu untersuchen, wurde daher versucht, diese Funktionalität mit Hilfe der vorgestellten prototypischen Verarbeitungsmodule zu realisieren.

Schritte 3 und 4 erfordern das Ändern einer globalen Variablen – der maximalen noch erlaubten Datenrate – in Abhängigkeit von deren gegenwärtigem Wert und dem Wert der aktuellen Ratenanforderung. Der Wert dieser Variablen darf daher während der hierfür benötigten Lese-Schreib-Abfolge nicht von einer parallel arbeitenden Einheit gelesen oder geändert werden, um die Konsistenz dieser Variablen zu erhalten. Um dies sicherzustellen, wurden die entsprechenden Verarbeitungsschritte (sowie Schritt 2) *nicht parallelisiert* mit einem schnellen Programmierbare-Logik-Modul realisiert. Die anderen Funktionen wurden in Software für ein Prozessor-Pool-Modul implementiert.

Abbildung 4.12 zeigt den prinzipiellen Aufbau des Programmierbare-Logik-Moduls. Der Verkehrszähler addiert die in den Metadaten angegebenen Paketlängen pro Ausgangsschnittstelle und misst somit den dort auftretenden Verkehr. Diese Messdaten verwendet der Slow-Path, um in regelmäßigen Abständen die zusagbare verfügbare Datenrate im zweiten Teil des Programmierbare-Logik-Moduls zu aktualisieren. In diesem wird bei jedem Quick-Start-Paket die angeforderte Rate mit der noch verfügbaren Datenrate verglichen und abhängig vom Ergebnis wird entweder die geforderte Rate oder die maximal noch mögliche Rate zugesagt. Entsprechend wird die noch verfügbare Datenrate angepasst.

Das Modul wurde so aufgebaut, dass der kritische Verarbeitungsabschnitt – das Vergleichen der Raten und das Anpassen der noch verfügbaren Rate – in einer Taktperiode durchgeführt wird. Somit kann in jedem Takt ein Quick-Start-Paket verarbeitet werden (vgl. hierzu auch [5]). Die maximale Taktfrequenz in einem System auf dem Stratix II beträgt 178 MHz bzw. 255 MHz auf dem Stratix IV, wobei das oben beschriebene Vergleichen und Anpassen der noch verfügbaren Rate den kritischen Pfad realisiert. Bei den Systemen mit 1024 bit breiten Worten wird die maximale Taktfrequenz jedoch weiterhin durch das Wortbreitenanpassungsmodul beschränkt.

Der in Tabelle 4.9 angegebene Ressourcenbedarf inklusive Puffer-Flipflops von etwa 2 800 bzw. 3 800 ALUT-Flipflop-Paaren ist vergleichsweise gering.

Tabelle 4.10 gibt die Anzahl der Befehle bzw. die Verarbeitungstakte der in Assembler implementierten Quick-Start-Funktionen an: Die Identifizierung, das Überschreiben der erlaubten Senderate im IP-Options-Header sowie die Anpassung der Nonce, des TTL-Felds und der IP-Prüfsumme. Mit maximal 53 Takten zur Ausführung ist eine Realisierung mit mehreren Prozessorkernen und einer im Vergleich zur Pipeline höheren Taktung möglich. Die im Vergleich zur RCP-Verarbeitung hohe Zahl von über 50 Verarbeitungstakten liegt in erster Linie in der aufwändigen Veränderung der Nonce (16 Takte) und der Anpassung der Prüfsumme (15 Takte). Diese Funktionen waren auch bei einer Implementierung dieser Funktionen auf dem Intel Netzprozessor IXP2400 im Rahmen der Studienarbeit von Suriyajan [186] vergleichsweise aufwändig. Hier würde sich somit auch eine Implementierung auf Register-Transfer-Ebene lohnen. Eine andere Möglichkeit wäre die Verwendung eines Prozessor-Moduls mit einem geeigneteren Befehlssatz.

4.4.3 Weitere Protokolle und Dienste

Aufgrund der Erfahrungen bei der Umsetzung der in den letzten beiden Unterkapiteln beschriebenen Paketverarbeitungsfunktionen lassen sich weitergehende Aussagen zur Anwendbarkeit der vorgestellten Architektur auch bei anderen Protokollen und Diensten machen.

Verarbeitung von Header-Feldern

Die vorgestellten Umsetzungen der IP-, RCP- und Quick-Start-Router-Funktionen erforderten verschiedene Operationen, welche auch bei vielen anderen aktuellen und vermutlich auch zukünftigen Paketverarbeitungsfunktionen benötigt werden: Die Analyse von Header-Feldern, die Verarbeitung einzelner extrahierter Werte in umfangreicheren Algorithmen oder externen Co-Prozessoren, die Änderung Router-lokaler Variablen in Abhängigkeit der verarbeiteten Pakete

Tabelle 4.9: Ressourcenbedarf für Quick-Start-Verarbeitungsmodul (Ratenverarbeitung)

Quick-Start-Raten-Verarbeitung	**W = 512 bit**	**W = 1024 bit**
Logik und Flipflops inklusive Puffer	2,8 k ALUT-FFs	3,8 k ALUT-FFs

Tabelle 4.10: Anzahl der Befehle bzw. Ausführungstakte der Quick-Start-Funktionalität

Quick-Start-Funktionalität Nachverarbeitung	**Befehle**	**Takte (max.)**
auf Prozessor-Pool-Modul	53	53

sowie die Modifikation einzelner Header-Felder. Nachfolgend werden beispielhaft einige der in Abschnitt 2.1.3 eingeführten zukünftigen Protokolle und Mechanismen betrachtet.

Die neuartigen Schicht-2-Netztechnologien für Weitverkehrsnetze PBB-TE und MPLS-TP müssten, ähnlich wie die IPv4-Basisfunktionen, ohne Probleme als Programmierbare-Logik-Modul oder als Assembler-Code für ein Prozessor-basiertes Modul realisiert werden können. Die Suche der Weiterleitungsinformationen bei diesen Protokollen ist im Vergleich zu IP wesentlich einfacher und kann durch einfaches Indizieren einer Tabelle im Speicher realisiert werden. Die Verarbeitung von gestapelten MPLS-Kennzeichnern (MPLS Label Stacks) sollte bis zu einer festen Anzahl (z. B. bis zu drei) vom Fast-Path-Modul verarbeitet werden können. Pakete mit höheren zu verarbeitenden Stapeln können in den Slow Path umgeleitet werden.

Umfangreiche Klassifizierungen von Paketen anhand ihrer Header-Felder zur Unterscheidung unterschiedlicher Kunden, Dienste oder Anwendungen, wie sie auch in einem Open-Flow-Switch erforderlich sind, sollten ebenfalls mit Hilfe der Verarbeitungsmodule der neuen Architektur einfach und schnell realisierbar sein. Für solche Klassifizierungen werden wie bei der IP-Adresssuche ebenfalls häufig Trie-basierte Algorithmen, wie beim Tree-Bitmap-Adresssuchmodul, verwendet oder es werden TCAMs eingesetzt. Eine entsprechende Realisierung bzw. Einbindung wurde in Unterkapitel 4.3 beschrieben. Zähler für Statistiken bezüglich verschiedener Flüsse, Kunden oder Dienste können sehr effizient sowohl in Software als auch in programmierbarer Logik durch parallel verwendete Zählmodule realisiert werden, die von einer Slow-Path-Funktion regelmäßig ausgelesen werden.

Verarbeitung des ganzen Pakets

Die Realisierung von Verarbeitungsfunktionen, welche sich über mehrere Datenworte eines Pakets oder das ganze Paket erstrecken, wie z. B. die Verschlüsselung, Komprimierung, Umcodierung oder Analyse von Nutzdaten wurde bisher nicht betrachtet. Ob und wie solche Funktionen mit der vorgestellten Architektur realisiert werden, soll hier betrachtet werden.

Die Datenworte eines Pakets wandern direkt hintereinander durch die Pipeline. Die Verarbeitung mehrerer solcher hintereinander eintreffender Worte in einem Programmierbare-Logik-Modul ist ohne großen Aufwand machbar. Der Zustand kann nach der Verarbeitung eines Wortes einfach bei der Verarbeitung des nächsten Wortes weiterverwendet werden oder die Worte eines Pakets können gepuffert und anschließend parallel verarbeitet werden. Falls notwendig, können außerdem jeder Zeit mehrere Pakete parallel verarbeitet werden. Einzige Einschränkung bei der Realisierung einer Funktion in Programmierbarer Logik ist lediglich, den geforderten Pipeline-Durchsatz an den Modulschnittstellen zu garantieren.

Bei der betrachteten Implementierung des Prozessor-Pool-Moduls ist nicht vorgesehen, dass sich die Verarbeitung über mehrere Worte erstreckt. Jedes Wort wird isoliert in einem eigenen Prozessorkern verarbeitet. Somit ist eine Verarbeitung eines ganzen, sich über mehrere Worte ersteckenden Wortes mit diesem Modul nur möglich, falls diese Verarbeitungen unabhängig voneinander ablaufen können.

Falls jedoch bei der Verarbeitung eines Wortes jeweils der Zustand des vorhergehenden Wortes benötigt wird, muss das zu verwendende Prozessor-Pool-Modul anders aufgebaut sein: Entwe-

der muss ein Mechanismus vorhanden sein, Daten zwischen den Prozessorkernen weiterzugeben, oder die Verarbeitung eines Paketes muss komplett in einem Prozessorkern erfolgen. Falls die Verarbeitung eines Wortes mehr als einen Pipeline-Takt, d. h. mehr als s Prozessortakte erfordert[4], kann die Verarbeitung von nachfolgenden Worten entsprechend erst später beginnen. Im ersten Fall, wenn die Verarbeitung in verschiedenen Kernen stattfindet, können die wartenden Kerne in der Zwischenzeit u. U. andere, unabhängige Operationen auf ihren Daten ausführen. Ansonsten müssen sie warten. Im zweiten Fall müssen die nachfolgenden Worte so lange im Speicher gepuffert werden. Bei beiden Konfigurationen ist die Verarbeitung kürzerer Pakete schneller beendet als die großer Pakete. Damit die Pakete nicht umsortiert werden, müssen die fertig verarbeiteten Pakete dennoch so lange in den Prozessorkernen verweilen, bis die maximale Verarbeitungszeit verstrichen ist. Entsprechend muss ein solches System ausreichend groß dimensioniert sein.

Beim implementierten Datenfluss-Pipeline-Modul ist bereits ein Mechanismus vorhanden, mit dem Ergebnisse eines Wortes beim nächsten Wort weiterverwendet werden können: Die Übergabe über den lokalen Registersatz (vgl. Abschnitt 3.7.2). Die Übergabe des Zustandes ist hier völlig problemlos, da die zu übergebenden Daten einfach im Register verbleiben, während die restlichen Daten weiterwandern und das nächste Wort in die anderen Register der Einheit geschrieben wird.[5] (Falls die Verarbeitung der einzelnen Worte unabhängig voneinander erfolgen kann, wird dieser Mechanismus natürlich, wie oben, nicht benötigt.) Wie beim Prozessor-Pool-Modul ist eine Verarbeitungsdauer von mehr als s Modultakten kein Hindernis, da sich die Verarbeitung eines Wortes auch über mehrere Verarbeitungseinheiten erstrecken darf. Wiederum müssen nachfolgende Worte lediglich u. U. mit ihrer Verarbeitung warten, bis der Zustand vom vorherigen Wort übergeben wird. Außerdem muss die Pipeline so lange ausgelegt sein, dass auch Pakete maximaler Länge vollständig verarbeitet werden können.

Somit kann festgehalten werden, dass auch Paketverarbeitungsfunktionen, die sich über mehrere Worte oder das ganze Paket erstrecken, mit dieser Architektur möglich sind. Lediglich das vorgestellte Prozessor-Pool-Modul müsste hierfür noch angepasst werden.

4.5 Zusammenfassung

Dieses Kapitel bewertete die neue Architektur. Die dabei untersuchten Kriterien sind die Leistungsfähigkeit, die Realisierbarkeit und die Anwendbarkeit eines Paketverarbeitungssystems mit dieser Architektur.

Die **Leistungsfähigkeit** eines solchen Systems ist deterministisch. Ihr unterstützter Durchsatz ist unabhängig vom zu verarbeitenden Verkehr und hängt nur von der Wortbreite und der Taktfrequenz der Pipeline-Struktur ab, welche der Architektur zu Grunde liegt. Um einen hohen Durchsatz von beispielsweise 100 Gbit/s oder 150 MP/s garantieren zu können, muss das System lediglich entsprechend dimensioniert sein.

[4] s = Verhältnis Prozessortakt zu Pipeline-Takt

[5] Allerdings ist die Übergabe von Ergebnissen an *vorhergehende* Worte, z. B. an das erste Wort mit den Header-Daten, über diesen Mechanismus nicht möglich. Hierfür wäre zusätzliche Hardware notwendig.

Die **Realisierbarkeit** eines so dimensionierten Systems wurde durch den Entwurf und die Implementierung eines Prototyps auf FPGA-Basis gezeigt. Das System realisiert die zu Grunde liegende Pipeline-Struktur mit einer vor der Synthese einstellbaren Datenwortbreite von 512 bit oder 1024 bit durch verschiedene Verarbeitungsmodule: Programmierbare-Logik-Module, Prozessor-Pool-Module und Datenfluss-Pipeline-Module. Die korrekte Funktion des Systems wurde durch taktgenaue Simulationen sowie Tests auf einer FPGA-basierten Plattform in einfachen Netzszenarien überprüft. Ein Test mit On-Chip-Paketgeneratoren validierte die Funktionsfähigkeit in Volllastsituationen.

Die Synthese- und Place & Route-Ergebnisse beweisen, dass ein System mit der vorgestellten Architektur auf aktuellen FPGAs für einen Einsatz auf 100 Gbit/s-Ethernet Line Cards realisiert werden kann. Hierbei ist sowohl eine ausschließliche Verarbeitung der Header-Felder möglich als auch eine Verarbeitung aller Paketdaten in den Verarbeitungsmodulen. Beispielsweise kann die Fast-Path-Verarbeitung eines IPv4-Routers bei Verwendung von Programmierbare-Logik-Modulen einen Durchsatz von 135 Gbit/s in der GP-Variante und 153 Gbit/s in der NH-Variante unterstützen. Dabei stehen noch 89 % des FPGAs für Erweiterungen mittels Prozessor-Pool- oder Datenfluss-Pipeline-Modulen zur Verfügung. Datenfluss-Pipeline-Module können mit über 140 Verarbeitungseinheiten und einem Durchsatz von beinahe 140 Gbit/s in der GP-Variante und 156 Gbit/s in der NH-Variante auf einem aktuellen Hochleistungs-FPGA realisiert werden. Das untersuchte Prozessor-Pool-Modul kann mit über 100 Prozessorkernen auf dem verwendeten FPGA realisiert werden. Allerdings erfordern diese Kerne weitere Optimierungen, da damit bisher keine 100 Gbit/s erreichbar sind. Dennoch zeigt die prototypische Realisierung die prinzipielle Funktionsweise eines solchen Moduls. Die Realisierung von Systemen mit 400 Gbit/s und mehr sollte in kommenden Jahren ebenfalls aufgrund der fortschreitenden Entwicklung der FPGA-Technologie möglich sein.

Die **Anwendbarkeit** eines so realisierten Systems zur flexiblen Umsetzung von aktuellen und zukünftigen Paketverarbeitungsfunktionen wurde durch beispielhafte Implementierungen solcher Funktionen gezeigt. So wurden die wichtigsten IP-Router-Funktionen sowie die Router-Funktionen der beiden noch experimentellen Protokolle RCP und TCP Quick-Start teilweise auf Software- und teilweise auf Register-Transfer-Ebene realisiert. Hierfür wurden die drei prototypisch implementierten Verarbeitungsmodularten eingesetzt.

Mit Hilfe der Programmierbare-Logik-Module sind sehr effiziente und ressourcenschonende Implementierungen möglich. Die Implementierungen auf den beiden Software-gesteuerten Modulen war ebenfalls gut möglich. Aufgrund des spezialisierten Befehlssatzes einerseits bzw. der effizienten Datenfluss-Architektur andererseits benötigten die umgesetzten Funktionen zwischen 20 und 50 Takten zur Ausführung, was einen Einsatz mit 100 Gbit/s innerhalb der entwickelten Architektur auf heutiger Technologie erlaubt. Wie zu erwarten, erfordert eine Realisierung mittels eines Prozessor-basierten Moduls wesentlich mehr Ressourcen als eine Implementierung in programmierbarer Logik. Einen Kompromiss stellt die Realisierung in einem Datenfluss-Pipeline-Modul mit unveränderlichem Befehlsspeicher dar, da die Synthese-Werkzeuge hier viele Ressourcen aufgrund nicht benötigter Einheiten einsparen können.

Die Module wurden in unterschiedlichen Kombinationen in dem prototypisch realisierten Paketverarbeitungssystem integriert und dort, wie oben beschrieben, getestet. Das einfache Austauschen von Modulen innerhalb des Prototyps zeigt die Flexibilität und Modularität der entwickelten Architektur. Abschließend erlaubten die beispielhaft implementierten Funktionen zu-

sätzlich die Prognose, dass auch andere zukünftige Verarbeitungsfunktionen – auch solche bei denen das ganze Paket verarbeitet werden muss – erfolgreich in einem System mit der vorgestellten Architektur realisiert werden können.

5 Zusammenfassung und Ausblick

Diese Dissertation stellte eine neue Architektur für zukünftige Paketverarbeitungssysteme in Kern- und Metronetzen vor. Die Architektur unterstützt hohe deterministische Paket- und Datenraten und bietet eine große Flexibilität zur Anpassung der Verarbeitung an neue Anforderungen. Sie erfüllt somit die wichtigsten Anforderungen an die Paketverarbeitung in zukünftigen Vermittlungsknoten.

Zur Motivation dieser Architektur wurden die Anforderungen an zukünftige Paketverarbeitungssysteme untersucht sowie Ansätze und Lösungen aus Industrie und Forschung diskutiert:

Anforderungen – Paketverarbeitungssysteme müssen eintreffende Pakete analysieren und klassifizieren, Berechnungen auf Basis der extrahierten Werte vornehmen und Modifizierungen an den Paketen durchführen. Diese Funktionen müssen im Wesentlichen auf den Header-Daten der Protokolle auf Schicht 2 bis 4 vorgenommen werden. Zukünftig ist zunehmend auch eine tiefergehende Verarbeitung bis einschließlich der Nutzdaten erforderlich. Die Verarbeitung muss in Kern- und Metronetzen in den nächsten Jahren mit einer Datenrate von 100 Gbit/s bzw. einer Paketrate von 150 MP/s erfolgen, wobei diese Durchsätze nach Möglichkeit garantiert werden sollen.

Außerdem muss die Verarbeitung flexibel an beliebige neue Anforderungen bezüglich Leistungsfähigkeit und Funktionalität angepasst werden können, um neue und geänderte Dienste und Protokolle umsetzen zu können. Hierbei ist sowohl die Anpassbarkeit eines solchen Systems wichtig als auch, dass die entsprechenden Änderungen einfach und schnell durchführbar sind.

Ansätze und Lösungen – FPGAs und Netzprozessoren sind am besten für die Realisierung solcher Systeme geeignet. Die Analyse aktueller Produkte und Lösungen aus Industrie und Forschung ergab, dass beide Bauelemente prinzipiell in der Lage sind, die geforderten hohen Durchsätze zu erreichen. Außerdem können beide flexibel an neue Verarbeitungsanforderungen angepasst werden – erstere auf Register-Transfer-Ebene, letztere auf Software-Ebene. Allerdings sind auf beiden Entwurfsebenen auch Einschränkungen der Flexibilität vorhanden. In den betrachteten Lösungen werden teilweise Pool- und teilweise Pipeline-Anordnungen der Verarbeitungseinheiten zum Erreichen eines hohen Durchsatzes verwendet, wobei mit letzterer ein deterministischer Durchsatz garantiert werden kann.

Aufbauend auf diesen Erkenntnissen wurden die verschiedenen Hardware- und Software-Entwurfsebenen sowie die beiden gegensätzlichen Parallelisierungsarten genauer analysiert:

Entwurfsebenen – Die Eigenschaften der verschiedenen Entwurfsebenen von FPGAs und Netzprozessoren wurden eingehend bezüglich ihrer Flexibilität und Leistungsfähigkeit diskutiert. Die höhere Abstraktionsebene von Netzprozessoren sowie deren optimierte Hardware erleichtern den Entwurf und die Realisierung Durchsatz-starker Verarbeitungsfunktionen. Andererseits können spezielle Funktionen auf den niedrigeren Hardware-Entwurfsebenen von FPGAs effizienter realisiert werden. Außerdem ermöglichen FPGAs auch eine architekturelle Anpassung der realisierten Verarbeitungsinfrastruktur.

Beide Entwurfsebenen haben somit ihre Vor- und Nachteile und sind je nach den jeweiligen Anforderungen besser für die Realisierung einer Paketverarbeitungsfunktion geeignet. Daher soll die neue Architektur die Möglichkeit bieten, eine Funktion auf der jeweils am besten passenden Entwurfsebene zu implementieren.

Pool und Pipeline – Die beiden prinzipiellen Parallelisierungsarten von Verarbeitungseinheiten wurden hinsichtlich Durchsatz, Implementierungsaufwand und Ressourcenaufwand untersucht. Hierbei zeigte sich, dass Systeme in Pool-Anordnungen i. d. R. im Mittel einen höheren Durchsatz erreichen und Verarbeitungsfunktionen einfach implementiert werden können.

Jedoch sind insbesondere für Kern- und Metronetze Verarbeitungssysteme mit Pipeline-Anordnung vorzuziehen, da diese unabhängig von der Verkehrsklassenverteilung einen deterministischen Durchsatz garantieren können. Punkt-zu-Punkt-Verbindungen zwischen den Einheiten ermöglichen einen hohen Durchsatz, verhindern Umsortierungen, machen Synchronisationen unnötig und benötigen vergleichsweise wenige Ressourcen. Außerdem erlaubt die Modularität einer Pipeline spezialisierte Module sowie das einfache Hinzufügen und Austauschen von neuen oder geänderten Funktionen.

Die neue Architektur für schnelle und flexible Paketverarbeitung gründet auf den Schlussfolgerungen dieser Analysen und Diskussionen. Sie hat folgende wesentlichen Eigenschaften:

- **Pipeline-Basisstruktur**
  - Wortbreite in der Größenordnung von ein bis zwei minimalen Paketlängen
  - ein Datenwort plus Metadaten pro Taktperiode, somit **deterministischer Durchsatz**
- **verschiedene Verarbeitungsmodule**
  - garantierter Pipeline-Durchsatz
  - einheitliche Schnittstellen, bestehend aus Datenwort, Metadaten, Steuersignalen
  - interne Verarbeitungsinfrastruktur beliebig, somit **beliebige Entwurfsebene** möglich

Die Architektur kann in zwei Varianten realisiert werden: In der Nur-Header-Variante (NH) wird nur das erste Datenwort jedes Pakets durch die Pipeline transportiert. Hier entspricht die Paketrate exakt der Taktfrequenz des Systems. In der Ganzes-Paket-Variante (GP) können alle Daten des Pakets verarbeitet werden, dafür ist eine anspruchsvollere Dimensionierung des Systems für einen gewünschten Durchsatz erforderlich. Aber auch hier ist der Durchsatz des Systems deterministisch und *der garantierte Durchsatz ist nur von der Dimensionierung von Wortbreite und Taktfrequenz sowie der minimalen Paketlänge und dem minimalen Paketabstand*

der verwendeten Netztechnologie abhängig. Die entsprechenden Formeln zur Berechnung von Datenrate und Paketrate sind in Tabelle 4.1 auf Seite 132 abgedruckt.

Die unterschiedlichen Verarbeitungsmodule können intern beliebig aufgebaut sein, solange die Daten und Metadaten an ihren Schnittstellen den vorgegebenen Durchsatz einhalten. *Somit können Module mit verschiedenen Verarbeitungsinfrastrukturen eingesetzt werden, mit denen Paketverarbeitungsfunktionen auf einer jeweils dem Problem angepassten Entwurfsebene implementiert werden können.* Ebenso können Module mit Funktionen von anderen Firmen oder Forschungseinrichtungen in Form von *Open-Source-* oder *Intellectual-Property-Modulen* in die Verarbeitungs-Pipeline integriert werden.

Nach der Darstellung dieses prinzipiellen Ansatzes wurden die Architekturen unterschiedlicher Verarbeitungsmodule vorgestellt und diskutiert und es wurden Szenarien gezeigt, wie die große Flexibilität eines entsprechenden Paketverarbeitungssystems genutzt werden kann:

Verarbeitungsmodule – Ein Programmierbare-Logik-Modul kann, wie ein FPGA, in einer Hardware-Beschreibungssprache entwickelte Verarbeitungsfunktionen mit Hilfe von kleinen Abbildungstabellen und Flipflops realisieren.

Ein Prozessor-Pool-Modul basiert auf einem Pool von Prozessorkernen, die die Verarbeitung der Datenworte durchführen. Um den Durchsatz einzuhalten, ist die Verarbeitungsdauer in Abhängigkeit der Pool-Größe begrenzt. Daher sollten die Prozessorkerne möglichst klein sein und, falls möglich, mit einem höheren Takt als die System-Pipeline betrieben werden. Die beispielhafte Implementierung verwendet 32-bit-RISC-Kerne basierend auf dem DLX-Prozessor mit einem für die Paketverarbeitung spezialisierten Befehlssatz, der an den Befehlssatz der Intel IXP2xxx Netzprozessoren angelehnt ist.

Ein Datenfluss-Pipeline-Modul besteht aus einer Pipeline von Verarbeitungseinheiten, die jeweils für einen oder – bei einem entsprechend höheren Modultakt – für einige wenige Takte ein Datenwort verarbeiten und es dann zur nächsten Einheit übergeben. Aufgrund der Datenfluss-gesteuerten Ausführung ist kein Forwarding notwendig. Zudem können die Einheiten nach Schreiben des Programmcodes stark optimiert werden, da jede Einheit nur wenige mögliche Befehle ausführen können muss. Die beispielhafte Implementierung verwendet Verarbeitungseinheiten, die effektiv jeweils eine Taktperiode pro Datenwort zur Verfügung haben und ineinander verschränkt jeweils eine zweitstufige Mini-Pipeline realisieren.

Nutzung der Flexibilität – Es wurden unterschiedliche Szenarien aufgezeigt, wie ein Paketverarbeitungssystem nach dieser Architektur durch Änderung der Funktion eines bestehenden Moduls oder Integration eines Moduls mit neuer Funktionalität flexibel an neue Anforderungen angepasst werden kann. Abhängig von den Anforderungen und Randbedingungen können bei der Auswahl eines Verarbeitungsmoduls Kriterien wie die Entwicklungszeit, der Ressourcenbedarf, der erforderliche Durchsatz und eine geeignete Infrastruktur entscheidend sein. Die einzelnen Schritte bei der Integration bzw. Anpassung eines Moduls wurden beschrieben, ebenso wie mögliche Unterstützung hierbei durch Software-Werkzeuge.

Die Realisierbarkeit und Anwendbarkeit der neuen Architektur wurden durch einen Prototyp und die Umsetzung verschiedener Verarbeitungsfunktionen gezeigt:

Prototyp – Ein Paketverarbeitungssystem in der GP-Variante der Architektur mit den beschriebenen drei Verarbeitungsmodularten wurde prototypisch in VHDL implementiert. Funktionstests basierend auf taktgenauen Simulationen und Tests auf einer FPGA-Plattform in einfachen Netzszenarien mit anderen Routern, Switches und PCs sowie unter Volllast mit Chip-internen Paketgeneratoren validierten die korrekte Funktionalität.

Syntheseergebnisse mit Wortbreiten von 512 bit und 1024 bit auf einem Stratix II und einem Stratix IV FPGA von Altera zeigen die Realisierbarkeit eines solchen Systems mit einem Durchsatz von mehr als 100 Gbit/s bzw. 150 MP/s auf aktuellen FPGAs. In einer NH-Konfiguration ist hierfür eine Pipeline-Frequenz von 150 MHz erforderlich, dies unterstützen fast alle untersuchten Stratix-IV-Systeme. In der GP-Variante unterstützt jeweils nur die Version mit 1024 bit breiten Worten einen Durchsatz von mehr als 100 Gbit/s. Lediglich die realisierten Prozessor-Pool-Module bedürfen noch Optimierungen. Mit kommenden FPGA-Generationen sowie mit auf diese Architektur spezialisierten Chips sollten auch zukünftige höhere Datenraten von 400 Gbit/s oder 1 Tbit/s erzielbar sein.

Umsetzung von Verarbeitungsfunktionen – Die Implementierung diverser aktueller sowie möglicher zukünftiger Paketverarbeitungsfunktionen von IP-Routern auf den verschiedenen prototypischen Verarbeitungsmodulen demonstriert die Anwendbarkeit der neuen Architektur. Dabei wurde gezeigt, wie durch den Einsatz passender Verarbeitungsinfrastrukturen Funktionen einfach und schnell realisiert werden können. Module wurden flexibel in das System integriert, ausgetauscht und verändert. Ausgehend von diesen Implementierungen wurde auch die Umsetzbarkeit anderer Verarbeitungsfunktionen diskutiert und skizziert.

Resümierend kann festgestellt werden: Die in dieser Dissertation vorgestellte Architektur erreicht die gesetzten Ziele eines hohen garantierten Durchsatzes und größter Flexibilität. Sie ist die erste veröffentlichte Architektur, die erlaubt, die Paketverarbeitungsfunktionen auf jeweils unterschiedlichen Entwurfsebenen zu implementieren. Zugleich garantiert ihre Pipeline-Struktur einen deterministischen Durchsatz. Die Architektur ist mit heutiger Technologie realisierbar und ist gut für die Umsetzung verschiedenster Paketverarbeitungsfunktionen geeignet.

Aufbauend auf diesen Ergebnissen können sich weiterführende Arbeiten bezüglich der Architektur, der Nutzbarkeit sowie der zu verwendenden Plattform anschließen:

Ausblick – Neben den drei vorgestellten sind weitere Verarbeitungsmodularten denkbar, z. B. Module mit mikroprogrammierbaren Einheiten oder mit Kombinationen von Software-gesteuerten Einheiten und programmierbarer Logik oder auch spezielle parametrisierbare Module für anspruchsvolle Funktionen wie Deep Packet Inspection. Des Weiteren kann die Gesamtarchitektur weiterentwickelt werden, z. B. mit dem Ziel den Energieverbrauch gering zu halten, Virtualisierungstechniken zu unterstützen oder Ausfallsicherheit zu bieten. Die Unterstützung durch Software-Werkzeuge wurde im Rahmen dieser Arbeit diskutiert. In diesem Umfeld sind Arbeiten zur Realisierung entsprechender Compiler und Optimierungswerkzeuge sowie einer integrierten Entwicklungsumgebung denkbar.

Bisher wurde im Wesentlichen die Implementierung der neuen Architektur auf einem FPGA betrachtet. Weiterführende Arbeiten könnten Architekturen für spezielle Paketverarbeitungs-Chips entwerfen, die für die Realisierung der vorgestellten Architektur optimiert sind. Diese könnten Bereiche fein-granularer Rekonfigurierbarkeit wie FPGAs aufweisen jedoch z. B. auch Bereiche mit festverdrahteten kleinen Verarbeitungseinheiten für Prozessor-Pool-Module.

Um die Verbreitung dieses Architekturansatzes zu fördern, ist eine Implementierung auf dem NetFPGA-System [146] und eine Offenlegung des Quellcodes der implementierten Module und Funktionen empfehlenswert. Zudem kann die Architektur somit weltweit zur Untersuchung von Verarbeitungsfunktionen neuartiger Protokolle, Dienste und Netztechnologien dienen.

A Programmiermodell und Befehlssatz des Prozessor-Pool-Moduls

Nachfolgend ist das Programmiermodell und der komplette Befehlssatz der in Abschnitt 3.7.1 beschriebenen Implementierung eines Prozessor-Pool-Moduls abgedruckt.

A.1 Programmiermodell

Alle Prozessorkerne führen dasselbe Programm aus: *Single-Processor*-Modell.

Jeder Prozessorkern verfügt über 64 Register `r0,...,r63` zu je 32 bit:

- Register `r0` ist konstant 0, Schreibbefehle auf `r0` sind wirkungslos.
- Die Metadaten und Paketdaten sind beginnend von oben im Registersatz abgelegt, z. B. sind bei 64 bit Metadaten und 512 bit Datenworten die Register folgendermaßen belegt: `r63, r62:` Metadaten; `r61,...,r46:` Paketdaten.

Arithmetische Operationen können das Statusregister beschreiben (Suffix `.sf`):

- Das Statusregister enhält folgende Flags: Vorzeichenflag N (Negative), Nullflag Z (Zero), Überlaufflag V (oVerflow), Übertragsflag C (Carry).
- Die Flags werden mit Hilfe bedingter Verzweigungsbefehle ausgewertet.

Der Speicher kann nur wortweise adressiert werden.

A.2 Befehlssatz

Hinweise

- Ausdrücke in eckigen Klammern sind optional.
- Bei vielen Befehlen wird durch Anhängen des optionalen Suffixes `.sf` nach Ausführung der angegebenen Operation die Flags entsprechend des Ergebnisses gesetzt.
- Die verwendeten Abkürzungen werden im Anschluss an den Befehlssatz erklärt.

Tabelle A.1: Schiebe- und Rotierbefehle der Prozessorkerne des Prozessor-Pool-Moduls

Befehl	**Syntax**
schiebe/rotiere um *ra* Stellen	`shf[.sf]  rc := rb <shf>ra`
schiebe/rotiere um *n*5 Stellen	`shfi[.sf]  rc := rb <shf>n5`
schiebe mit Registerübertrag ...	
... um *n*5 Stellen logisch nach rechts	`dblshf[.sf] rc := ra,rb >>n5`

Tabelle A.2: Logische Befehle der Prozessorkerne des Prozessor-Pool-Moduls

Befehl	**Syntax**
logisches Nicht ($\overline{rb}$)	`not[.sf] rc := rb [<shf>n5]`
logisches Und ($ra \cdot rb$)	`and[.sf] rc := ra,rb [<shf>n5]`
logisches Und (erster Operand invertiert, $\overline{ra} \cdot rb$)	`and[.sf] rc := !ra,rb [<shf>n5]`
logisches Und (zweiter Operand invertiert, $ra \cdot \overline{rb}$)	`and[.sf] rc := ra,!rb [<shf>n5]`
logisches Oder ($ra + rb$)	`or[.sf]  rc := ra,rb [<shf>n5]`
logisches Exklusiv-Oder ($ra \oplus rb$)	`xor[.sf] rc := ra,rb [<shf>n5]`

Tabelle A.3: Arithmetische Befehle der Prozessorkerne des Prozessor-Pool-Moduls

Befehl	**Syntax**
addiere *ra* und *rb*	`add[.sf]  rc := ra,rb`
addiere *ra*, *rb* und Übertragsbit	`addc[.sf]  rc := ra,rb`
addiere *ra* und niederwertige 8 bit von *rb*	`add8[.sf]  rc := ra,rb`
addiere *ra*, nied'wert. 8 bit von *rb* u. Übertragsbit	`addc8[.sf]  rc := ra,rb`
addiere *ra* und niederwertige 16 bit von *rb*	`add16[.sf]  rc := ra,rb`
addiere *ra*, nied'wert. 16 bit von *rb* u. Übertragsbit	`addc16[.sf] rc := ra,rb`
addiere *ra* und unmittelbares *imm*11	`addi[.sf]  rc := ra,#imm11`
addiere *ra*, unmittelbares *imm*11 und Übertragsbit	`addci[.sf]  rc := ra,#imm11`
subtrahiere *rb* von *ra*	`sub[.sf]  rc := ra,rb`
subtrahiere *rb* und Übertragsbit von *ra*	`subc[.sf]  rc := ra,rb`
subtrahiere unmittelbares *imm*11 von *ra*	`subi[.sf]  rc := ra,#imm11`
subtrahiere unmittelb. *imm*11 u. Übertr'bit von *ra*	`subci[.sf]  rc := ra,#imm11`
multipliziere niederwertige 16 bit von *ra* und *rb*	`mult16[.sf] rc := ra,rb`

Tabelle A.4: Initialisierungs-/Transport-Befehle d. Prozessorkerne d. Prozessor-Pool-Moduls

Befehl	**Syntax**
schreibe ...	
... unmittelbares *imm*16 nach Bit 15 bis 0, ...	`initclr rc := #imm16`
... unmittelbares *imm*16 nach Bit 23 bis 8, ...	`initclr8 rc := #imm16`
... unmittelbares *imm*16 nach Bit 31 bis 16, ...	`initclr16 rc := #imm16`
lösche andere Bits	
schreibe ...	
... unmittelbares *imm*16 nach Bit 15 bis 0, ...	`initfil rc := #imm16`
... unmittelbares *imm*16 nach Bit 23 bis 8, ...	`initfil8 rc := #imm16`
... unmittelbares *imm*16 nach Bit 31 bis 16, ...	`initfil16 rc := #imm16`
setze andere Bits zu Eins	
schreibe ...	
... unmittelbares *imm*16 nach Bit 15 bis 0, ...	`initw0 rc := #imm16`
... unmittelbares *imm*16 nach Bit 31 bis 16, ...	`initw1 rc := #imm16`
... unmittelbares *imm*8 nach Bit 7 bis 0, ...	`initb0 rc := #imm8`
... unmittelbares *imm*8 nach Bit 15 bis 8, ...	`initb1 rc := #imm8`
... unmittelbares *imm*8 nach Bit 23 bis 16, ...	`initb2 rc := #imm8`
... unmittelbares *imm*8 nach Bit 31 bis 24, ...	`initb3 rc := #imm8`
lasse andere Bits unverändert	
schreibe (verschobenes / rotiertes) *rb* entsprechend Byte-Maske *bm*4 ...	
... und lasse andere Bits unverändert	`movb[.sf] rc := rb,bm4 [<shf>n5]`
... und lösche andere Bits	`movbclr[.sf] rc:=rb,bm4 [<shf>n5]`

Tabelle A.5: Spezialbefehle der Prozessorkerne des Prozessor-Pool-Moduls

Befehl	**Syntax**
ermittle die Position des ersten gesetzten Bits ...	
... von links	`fflone[.sf] rc := ra`
... von rechts	`ffrone[.sf] rc := ra`
erweitere vorzeichenrichtig ...	
... die niederwertigen 8 bit ...	`ext8[.sf] rc := rb [<shf>n5]`
... die niederwertigen 16 bit ...	`ext16[.sf] rc := rb [<shf>n5]`
des (geschobenen / rotierten) *rb*	
ermittle die Anzahl der gesetzten Bits	`cntones[.sf] rc := ra`

Tabelle A.6: Speicherzugriffsbefehle der Prozessorkerne des Prozessor-Pool-Moduls

Befehl	**Syntax**
lade von Adresse $ra + disp14$	`ldd rc := (ra,disp14)`
lade von Adresse $ra + rb$	`ldr rc := (ra,rb)`
speichere an Adresse $ra + disp14$	`std rc := (ra,disp14)`
speichere an Adresse $ra + rb$	`str rc := (ra,rb)`

Tabelle A.7: Verzweigungs- und Sprungbefehle der Prozessorkerne d. Prozessor-Pool-Moduls

Befehl	**Syntax**
verzweige, falls Bedingung *cc* erfüllt	`b<cc> disp13 [defer n2]`
verzweige, falls Bitpositon *n*5 gleich Null	`bbclr ra,n5,disp13 [defer n2]`
verzweige, falls Bitpositon *n*5 gleich Eins	`bbset ra,n5,disp13 [defer n2]`
verzweige, falls Bits 7 bis 0 gleich *imm*8	`brb0eq ra,#imm8,disp8 [defer n2]`
verzweige, falls Bits 15 bis 8 gleich *imm*8	`brb1eq ra,#imm8,disp8 [defer n2]`
verzweige, falls Bits 23 bis 16 gleich *imm*8	`brb2eq ra,#imm8,disp8 [defer n2]`
verzweige, falls Bits 31 bis 24 gleich *imm*8	`brb3eq ra,#imm8,disp8 [defer n2]`
verzweige, falls Bits 7 bis 0 ungleich *imm*8	`brb0ne ra,#imm8,disp8 [defer n2]`
verzweige, falls Bits 15 bis 8 ungleich *imm*8	`brb1ne ra,#imm8,disp8 [defer n2]`
verzweige, falls Bits 23 bis 16 ungleich *imm*8	`brb2ne ra,#imm8,disp8 [defer n2]`
verzweige, falls Bits 31 bis 24 ungleich *imm*8	`brb3ne ra,#imm8,disp8 [defer n2]`
verzweige	`br disp13 [defer n2]`
verzweige u. sichere Rücksprungadresse nach *ra*	`bal ra,disp13 [defer n2]`
springe	`jmp rb [defer n2]`
springe und sichere Rücksprungadresse nach *ra*	`jal ra,rb [defer n2]`

Mit dem optionalen Postfix `defer n2` können bis zu zwei (abhängig von `n2`) nachfolgende Befehle nach dem Verzweigungs- oder Sprungbefehl unbedingt ausgeführt werden, anstatt dass sie automatisch im Falle eines Sprungs gelöscht werden.

Der bedingte Verzweigungsbefehl `b<cc>` verzweigt in Abhängigkeit der zuvor gesetzten Flags. Auf welche Bedingungen geprüft und welches Kürzel hierfür anstatt `<cc>` verwendet werden muss, kann nachfolgend angegebener Abkürzungsübersicht entnommen werden.

Tabelle A.8: Verwendete Abkürzungen in den Befehlssatztabellen des Prozessor-Pool-Moduls

Abkürzung	Symbole	Beschreibung
ra rb rc	r0 – r31 bzw. r0 – r63	Register
<shf>	<< >> a a> r< r>	schiebe logisch links schiebe logisch rechts schiebe arithmetisch links schiebe arithmetisch rechts rotiere links rotiere rechts
n2 n5	0–2 0 – 31	zu verwendende *delay slots* Schiebeweite bzw. Bitposition
bm4	%0001 %0010 %0100 %1000	Maske für Byte 0 (Bits 7 bis 0) Maske für Byte 1 (Bits 15 bis 8) Maske für Byte 2 (Bits 23 bis 16) Maske für Byte 3 (Bits 31 bis 24)
imm8 imm11 imm16	0 – 255 0 – 2047 0 – 65535	8 bit breiter unmittelbarer Operand 11 bit breiter unmittelbarer Operand 16 bit breiter unmittelbarer Operand
disp8 disp13 disp14	0 – 255 0 – 8192 0 – 16383	8 bit breite Verzweigungsdistanz 13 bit breite Verzweigungsdistanz 14 bit breiter Offset
<cc>	 eq ne cs cc ls hi lt ge le gt mi pl vs vc	Verzweige, falls ... ... gleich, d. h. Z=1 (**eq**ual) ... nicht gleich, d. h. Z=0 (**n**ot **e**qual) ... kleiner (vorz.-los), d. h. C=1 (**c**arry **s**et) ... größer oder gleich (vorz.-los), d. h. C=0 (**c**arry **c**lear) ... kleiner oder gleich (vorz.-los), d. h. C+Z=1 (**l**ower or **s**ame) ... größer (vorz.-los), d. h. C+Z=0 (**hi**gher) ... kleiner (vorz.-beh.), d. h. N≠V (**l**ess **t**han) ... größer oder gleich (vorz.-beh.), d. h. N=V (**g**reater or **e**qual) ... kleiner oder gleich (vz.-b.), d.h. Z=1 & N=V (**l**ess or **e**qual) ... größer (vorz.-beh.), d. h. Z=0 & N≠V (**g**reater **t**han) ... negativ (vorz.-beh.), d. h. N=1 (**mi**nus) ... positiv (vorz.-beh.), d. h. N=0 (**pl**us) ... Überlauf (vorz.-beh.), d. h. V=1 (o**v**erflow **s**et) ... kein Überlauf (vorz.-beh.), d. h. V=0 (o**v**erflow **c**lear)

B Programmiermodell und Befehlssatz des Datenfluss-Pipeline-Moduls

Nachfolgend ist das Programmiermodell und der komplette Befehlssatz der in Abschnitt 3.7.2 beschriebenen Implementierung eines Datenfluss-Pipeline-Moduls abgedruckt.

B.1 Programmiermodell

Die Programmausführung erfolgt verteilt über die Datenfluss-Pipeline, wobei jede Verarbeitungseinheit einen Befehl ausführt. Die interne Wortbreite der Einheiten beträgt 16 bit.

Jede Einheit verfügt über folgende Registersätze, die in jedem Takt zur nächsten Einheit weiter transportiert werden:

- Paketdatenregistersatz – enthält ein Segment des aktuellen Paket-Datenworts
- Metadatenregistersatz – enthält die aktuellen Metadaten
- Standardregistersatz – kann beliebig verwendet werden
- Konstanten – können während der Programmausführung nicht verändert werden

Die Anzahl der jeweiligen Register ist konfigurierbar. Die Standardkonfiguration enthält acht Paketdatenregister (d. h. Segmentgröße ist 128 bit), vier Metadatenregister (64 bit), acht Standardregister und acht Konstanten pro Einheit.

Zusätzlich gibt es einen lokalen Registersatz, der Daten enthält, die in der jeweiligen Einheit verbleiben. Die Standardkonfiguration sieht vier lokale Register vor.

Des Weiteren enthält jede Einheit für jeden der Registersätze einen Registeradressierungsregistersatz, über den die Register indirekt angesprochen werden können. Der Standardregistersatz enthält zwei Registeradressierungsregister.

Ein Statusregister enthält ein Vorzeichenflag N (Negative), ein Nullflag Z (Zero), ein Überlaufflag V (oVerflow) und ein Übertragsflag C (Carry). Dieses kann indirekt über die im Befehlssatz angegebenen ALU-Operationen beschrieben werden. Die Flags können durch bedingte Sprünge ausgewertet werden.

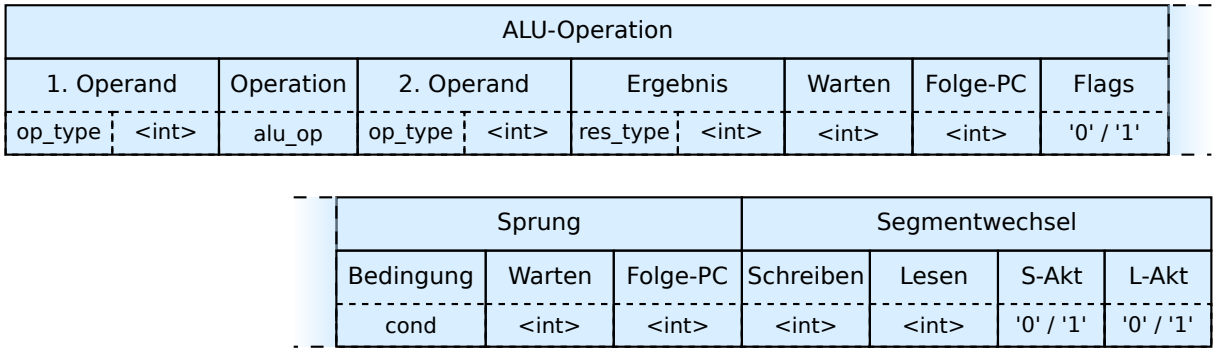

Abbildung B.1: Aufbau eines Befehls des Datenfluss-Pipeline-Moduls

Ergebnisse einer ALU-Operation können erst im übernächsten Takt aus dem Ergebnisregister gelesen werden. Falls das Ergebnis im folgenden Befehl als Operand verwendet werden soll, so muss explizit der Operand `pr` (*previous result*) angegeben werden. Eine Ausnahme ist der lokale Registersatz: Hier kann das Ergebnis direkt im nächsten Takt (und somit beim nächsten Datenwort) wieder gelesen werden.

Ein Befehlsfolgezähler enthält die Adresse des aktuellen Befehls. Dieser wird nach jedem Befehl mit einem neuen Wert überschrieben. Über bedingte Sprungbefehle kann in Abhängigkeit der Statusflags einer von zwei nächsten Befehlsadressen ausgewählt werden. Die Statusflags können jeweils erst im zweiten Takt nach einer ALU-Operation ausgewertet werden. Die Größe des Befehlsspeichers ist konfigurierbar, die Standardeinstellung sind acht Befehle.

Jeweils nach einer konfigurierbaren Anzahl von Verarbeitungseinheiten können die Daten in den Paketdatenregistern (i. d. R. 128 bit) gegen ein anderes Segment des Paket-Datenwortes (i. d. R. 512 bit oder 1024 bit) ausgetauscht werden. In welches Segment zurückgeschrieben wird und welches geladen wird, kann frei ausgewählt werden.

B.2 Befehlssatz

Jeder Befehl kann eine ALU-Operation und einen Sprungbefehl enthalten sowie Angaben bezüglich des nächsten Segmentwechsels. Der Aufbau eines solchen Befehlswortes ist in Abbildung B.1 dargestellt.

Die beiden Operanden sowie das Ergebnis der ALU-Operation werden jeweils durch einen Operandentyp *op_type* (z. B. Paketdatenregister) und eine Zahl *op_num* spezifiziert. Wobei *op_num* die Registernummer des jeweiligen Registersatzes bezeichnet. Alternativ kann ein Register auch indirekt über ein Registeraddressierungsregister ausgewählt werden, in diesem Fall ist die in *op_num* angegebene Zahl irrelevant, bzw. gibt beim Standardregistersatz das zu verwendende Registeradressierungsregister an. Der Operand `pr` bezeichnet das Ergebnis der vorhergehenden ALU-Operation. *alu_op* gibt die auszuführende Operation an. Zusätzlich kann spezifiziert werden, ob die Flags bei der Operation beeinflusst werden sollen.

Das Feld *Folge-PC* gibt die Adresse des nächsten Befehls in der nächsten Verarbeitungseinheit an. Außerdem kann spezifiziert werden, wie viele Takte bzw. Einheiten zunächst gewartet wer-

den soll, bis der nächste Befehl ausgeführt wird – dies kann zur Realisierung von ineinander verschränkten Befehlsabfolgen eingesetzt werden.

Bei der Spezifizierung eines Sprunges muss die Sprungbedingung und zusätzlich der *Folge-PC* sowie der *Warten*-Wert bei einem Sprung (d. h. falls die Bedingung wahr ist) angegeben werden.

Über die letzten vier Felder kann angegeben werden, in welches Segment die Daten aus den Paketdatenregistern beim nächsten Segmentwechsel zurückgeschrieben werden sollen und welches Segment anstatt dessen geladen werden soll. Die entsprechend im *Schreiben*- und *Lesen*-Feld angegebenen Werte werden nur übernommen, falls die entsprechenden Register über *S-Akt* bzw. *L-Akt* freigegeben werden.

Die nachfolgenden Tabellen spezifizieren die möglichen Einträge für *op_type*, *alu_op*, *res_type* und *cond*.

Tabelle B.1: Mögliche ALU-Operanden der Datenfluss-Verarbeitungseinheiten

Operand (op_type)	**Syntax direkte Adressierung**	**Syntax indirekte Adressierung**
Paketdatenregister (*original data*)	`od`	`od_with_rar`
Metadatenregister (*meta data*)	`md`	`md_with_rar`
Standardregister (*utility register*)	`ur`	`ur_with_rar`
Lokales Register (*fixed register*)	`fr`	`fr_with_rar`
Konstante (*constant memory*)	`cm`	`cm_with_rar`
letztes Ergebnis (*previous result*)	`pr`	—

Tabelle B.2: Mögliche Ergebnisregister der Datenfluss-Verarbeitungseinheiten

Ergebnis (res_type)	**Syntax direkte Adressierung**	**Syntax indirekte Adressierung**
Paketdatenregister (*original data*)	`od`	`od_with_rar`
Metadatenregister (*meta data*)	`md`	`md_with_rar`
Standardregister (*utility register*)	`ur`	`ur_with_rar`
Lokales Register (*fixed register*)	`fr`	`fr_with_rar`
Paketdatenregister-Adressierungsregister	`rar_od`	—
Metadatenregister-Adressierungsregister	`rar_md`	—
Standardregister-Adressierungsregister	`rar_ur`	—
Lokales-Register-Adressierungsregister	`rar_fr`	—

Tabelle B.3: ALU-Operationen der Datenfluss-Verarbeitungseinheiten

ALU-Operation (alu_op)	**Syntax**
addiere 1. Operanden und Übertragsbit	`alu_axc`
addiere 1. Operanden, 2. Operanden und Übertragsbit	`alu_axyc`
addiere 1. Operanden und 2. Operanden	`alu_axy`
subtrahiere Übertragsbit vom 1. Operanden	`alu_sxc`
subtrahiere 2. Operanden und Übertragsbit vom 1. Operanden	`alu_sxyc`
subtrahiere 2. Operanden vom 1. Operanden	`alu_sxy`
schiebe 1. Operanden um <*n*> bit (<*n*> = 1, 2, 4, 8) nach links	`alu_lsl<n>`
schiebe 1. Operanden um <*n*> bit (<*n*> = 1, 2, 4, 8) nach rechts	`alu_lsr<n>`
rotiere 1. Operanden um <*n*> bit (<*n*> = 1, 2, 4, 8) nach links	`alu_rol<n>`
rotiere 1. Operanden um <*n*> bit (<*n*> = 1, 2, 4, 8) nach links	`alu_ror<n>`
logisches Nicht von 1. Operanden	`alu_not`
logisches Und von 1. Operanden und 2. Operanden	`alu_and`
logisches Oder von 1. Operanden und 2. Operanden	`alu_or`
logisches Exklusiv-Oder von 1. Operanden und 2. Operanden	`alu_xor`

Tabelle B.4: Sprungbedingungen der Datenfluss-Verarbeitungseinheiten

Sprungbedingung (cond)	**Syntax**
Springe, falls ...	
... gleich, d. h. Z=1 (**eq**ual)	`eq`
... nicht gleich, d. h. Z=0 (**n**ot **e**qual)	`ne`
... kleiner (vorz.-los), d. h. C=1 (**c**arry **s**et)	`cs`
... größer oder gleich (vorz.-los), d. h. C=0 (**c**arry **c**lear)	`cc`
... kleiner oder gleich (vorz.-los), d. h. C+Z=1 (**l**ower or **s**ame)	`ls`
... größer (vorz.-los), d. h. C+Z=0 (**hi**gher)	`hi`
... kleiner (vorz.-beh.), d. h. N≠V (**l**ess **t**han)	`lt`
... größer oder gleich (vorz.-beh.), d. h. N=V (**g**reater or **e**qual)	`ge`
... kleiner oder gleich (vz.-b.), d.h. Z=1 & N=V (**l**ess or **e**qual)	`le`
... größer (vorz.-beh.), d. h. Z=0 & N≠V (**g**reater **t**han)	`gt`
... negativ (vorz.-beh.), d. h. N=1 (**mi**nus)	`mi`
... positiv (vorz.-beh.), d. h. N=0 (**pl**us)	`pl`
... Überlauf (vorz.-beh.), d. h. V=1 (**o**verflow **s**et)	`os`
... kein Überlauf (vorz.-beh.), d. h. V=0 (**o**verflow **c**lear)	`oc`

Literaturverzeichnis

[1] Simon Hauger. „A Novel Architecture for a High-Performance Network Processing Unit: Flexibility at Multiple Levels of Abstraction“. In: *Proceedings of the IEEE International Conference on High Performance Switching and Routing (HPSR 2009)*. Juni 2009.

[2] Simon Hauger. „Designing High-Speed Packet Processing Tasks at Arbitrary Levels of Abstraction — Implementation and Evaluation of a MIXMAP System“. In: *Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS 2009)*. Oktober 2009.

[3] Domenic Teuchert und Simon Hauger. „A Pipelined IP Address Lookup Module for 100 Gbps Line Rates and beyond“. In: *Proceedings of the 15th Open European Summer School (EUNICE 2009)*. September 2009.

[4] Simon Hauger, Thomas Wild, Arthur Mutter, Andreas Kirstädter, Kimon Karras, Rainer Ohlendorf, Frank Feller und Joachim Scharf. „Packet Processing at 100 Gbps and Beyond—Challenges and Perspectives“. In: *Beiträge zur 10. ITG-Fachtagung Photonische Netze*. Mai 2009, S. 43–52.

[5] Simon Hauger, Michael Scharf, Jochen Kögel und Chawapong Suriyajan. „Evaluation of Router Implementations for Explicit Congestion Control Schemes“. In: *Journal of Communications* 5.3 (März 2010), S. 197–204.

[6] Simon Hauger, Michael Scharf, Jochen Kögel und Chawapong Suriyajan. „Quick-Start and XCP on a Network Processor: Implementation Issues and Performance Evaluation“. In: *Proceedings of the IEEE International Conference on High Performance Switching and Routing (HPSR 2008)*. Mai 2008.

[7] Simon Hauger, Sascha Junghans, Martin Köhn und Detlef Sass. *A Scalable Architecture for Flexible High-Speed Packet Classification*. Interner Bericht. Institut für Kommunikationsnetze und Rechnersysteme (IKR), Universität Stuttgart, Dezember 2006.

[8] Simon Hauger, Sascha Junghans, Arthur Mutter und Detlef Sass. „A Flexible Microprogrammed Packet Classifier for Edge Nodes of Transport Networks“. In: *Beiträge zur 7. ITG-Fachtagung Photonische Netze*. April 2006.

[9] Magnus Proebster, Michael Scharf und Simon Hauger. „Performance Comparison of Router Assisted Congestion Control Protocols: XCP vs. RCP“. In: *Proceedings of the 2nd International Workshop on the Evaluation of Quality of Service through Simulation in the Future Internet - Held in conjunction with SIMUTools 2009*. März 2009.

[10] Michael Scharf, Simon Hauger und Jochen Kögel. „Quick-Start TCP: From Theory to Practice“. In: *Proceedings of the 6th International Workshop on Protocols for FAST Long-Distance Networks*. März 2008.

[11] Detlef Sass, Simon Hauger und Martin Köhn. „Architecture and Scalability of a High-Speed Traffic Measurement Platform with a Highly Flexible Packet Classification“. In: *Computer Networks* 53.6 (April 2009), S. 810–820.

[12] Jochen Kögel, Simon Hauger, Sascha Junghans, Martin Köhn, Marc C. Necker und Sylvain Stanchina. „Design and Evaluation of a Burst Assembly Unit for Optical Burst Switching on a Network Processor“. In: *EUNICE 2005: Networks and Applications Towards a Ubiquitously Connected World*. Hrsg. von Carlos D. Kloos, Andres Marin und David Larrabeiti. Bd. 196. IFIP. Springer Boston, Juli 2006, S. 3–16.

[13] Matthias Buck. „Entwurf und Realisierung eines FPGA-basierten Netzprozessormoduls mit einer synchronen datenfluss-orientierten Paketverarbeitungs-Pipeline“. Studienarbeit. Institut für Kommunikationsnetze und Rechnersysteme, Universität Stuttgart, Februar 2010.

[14] Florian Witowski. „Hardware-/Software-Integration eines Soft-Core-Prozessors für Steuerungsaufgaben in einen prototypischen, FPGA-basierten Router“. Diplomarbeit. Institut für Kommunikationsnetze und Rechnersysteme, Universität Stuttgart, Februar 2010.

[15] Domenic Teuchert. „Entwurf und Realisierung eines Moduls für eine effiziente IP Adresssuche für einen FPGA-basierten Hochgeschwindigkeits-Router in VHDL“. Studienarbeit. Institut für Kommunikationsnetze und Rechnersysteme, Universität Stuttgart, August 2009.

[16] Micha Klingler. „Erweiterung des IKR-RISC Prozessors für den Einsatz in einem FPGA-basierten Netzprozessor“. Studienarbeit. Institut für Kommunikationsnetze und Rechnersysteme, Universität Stuttgart, Januar 2009.

[17] Florian Mück. „Entwurf und Realisierung eines einfachen, erweiterbaren Internet-Routers in VHDL“. Studienarbeit. Institut für Kommunikationsnetze und Rechnersysteme, Universität Stuttgart, September 2008.

[18] Michelangelo Masini. „Entwurf und Realisierung einer CAM-Erweiterungskarte mit zugehörigem VHDL-Logikmodul fur die Universelle Hardware-Plattform“. Studienarbeit. Institut für Kommunikationsnetze und Rechnersysteme, Universität Stuttgart, Dezember 2007.

[19] International Organization for Standardization. *ISO/IEC 7498–1:1994 – Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*. ISO/IEC, 1994.

[20] Andrew S. Tanenbaum. *Computernetzwerke*. 4. Aufl. Pearson Education Deutschland GmbH, 2003.

[21] Robert Braden. *Requirements for Internet Hosts – Communication Layers*. RFC 1122, IETF. Oktober 1989.

[22] Robert Braden. *Requirements for Internet Hosts – Application and Support*. RFC 1123, IETF. Oktober 1989.

[23] David E. Taylor. „Survey and Taxonomy of Packet Classification Techniques“. In: *ACM Computing Surveys* 37.3 (2005), S. 238–275.

[24] Sarang Dharmapurikar, Praveen Krishnamurthy, Todd S. Sproull und John W. Lockwood. „Deep packet inspection using parallel bloom filters“. In: *IEEE Micro* 24.1 (Januar 2004), S. 52–61.

[25] Jörg Sommer, Sebastian Gunreben, Ahlem Mifdaoui, Frank Feller, Martin Köhn, Detlef Saß und Joachim Scharf. „Ethernet – A Survey on its Fields of Application“. In: *IEEE Communications Surveys & Tutorials* 12.2 (April 2010), S. 263–284.

[26] IEEE Computer Society. *802.3: IEEE Standard for Local and Metropolitan Area Networks–Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*. Dezember 2008.

[27] Vinton G. Cerf und Robert E. Kahn. „A Protocol for Packet Network Interconnection“. In: *IEEE Transactions on Communications* 22.5 (Mai 1974), S. 637–648.

[28] Jon Postel. *Internet Protocol*. RFC 791, IETF. September 1981.

[29] Jon Postel. *Transmission Control Protocol*. RFC 793, IETF. September 1981.

[30] Barry M. Leiner, Vinton G. Cerf, David D. Clark, Robert E. Kahn, Leonard Kleinrock, Daniel C. Lynch, Jon Postel, Larry G. Roberts und Stephen Wolff. „A brief history of the Internet“. In: *ACM SIGCOMM Computer Communication Review* 39.5 (Oktober 2009), S. 22–31.

[31] Fred Baker. *Requirements for IP Version 4 Routers*. RFC 1812, IETF. Juni 1995.

[32] Eric C. Rosen, Arun Viswanathan und Ross Callon. *Multiprotocol Label Switching Architecture*. RFC 3031, IETF. Januar 2001.

[33] Eric C. Rosen, Dan Tappan, Guy Fedorkow, Yakov Rekhter, Dino Farinacci, Tony Li und Alex Conta. *MPLS Label Stack Encoding*. RFC 3032, IETF. Januar 2001.

[34] Jon Postel. *User Datagram Protocol*. RFC 768, IETF. August 1980.

[35] K. K. Ramakrishnan, Sally Floyd und David L. Black. *The Addition of Explicit Congestion Notification (ECN) to IP*. RFC 3168, IETF. September 2001.

[36] IEEE Computer Society. *802.1Qay: IEEE Standard for Local and Metropolitan Area Networks – Virtual Bridged Local Area Networks - Amendment: Provider Backbone Bridge Traffic Engineering*. Mai 2007.

[37] Ben Niven-Jenkins (Ed.), Deborah Brungard (Ed.), Malcolm Betts (Ed.), Nurit Sprecher und Satoshi Ueno. *Requirements of an MPLS Transport Profile*. RFC 5654, IETF. September 2009.

[38] IEEE Computer Society. *802.1Q: IEEE Standard for Local and Metropolitan Area Networks–Virtual Bridged Local Area Networks*. Mai 2006.

[39] IEEE Computer Society. *802.1ad: IEEE Standard for Local and Metropolitan Area Networks–Virtual Bridged Local Area Networks, Amendment 4: Provider Bridges*. Mai 2006.

[40] IEEE Computer Society. *802.1ah: IEEE Standard for Local and Metropolitan Area Networks–Virtual Bridged Local Area Networks, Amendment 6: Provider Backbone Bridges*. August 2008.

[41] IEEE Computer Society. *802.1ag: IEEE Standard for Local and Metropolitan Area Networks – Virtual Bridged Local Area Networks - Amendment 5: Connectivity Fault Management*. Dezember 2007.

[42] Cisco Systems, Inc. *Understanding MPLS-TP and Its Benefits*. Cisco Systems, Inc. 2009. URL: http://www.cisco.com/en/US/technologies/tk436/tk428/white_paper_c11-562013.pdf.

[43] Stephen E. Deering und Robert M. Hinden. *Internet Protocol, Version 6 (IPv6), Specification*. RFC 2460, IETF. Dezember 1998.

[44] Sally Floyd, Mark Allman, Amit Jain und Pasi Sarolahti. *Quick-Start for TCP and IP*. RFC 4782 (experimental), IETF. Januar 2007.

[45] Aaron Falk, Yuri Pryadkin und Dina Katabi. *Specification for the Explicit Control Protocol (XCP)*. IETF Internet Draft, work in progress. Januar 2007.

[46] Nandita Dukkipati. „Rate Control Protocol (RCP): Congestion Control to Make Flows Complete Quickly“. Diss. Stanford University, Dept. of Electrical Engineering, Oktober 2007.

[47] Jennifer Rexford und Constantine Dovrolis. „Future Internet architecture: clean-slate versus evolutionary research“. In: *Communications of the ACM* 53.9 (September 2010), S. 36–40.

[48] National Science Foundation. *GENI – Exploring Networks of the Future*. National Science Foundation. Januar 2010. URL: http://www.geni.net/wp-content/uploads/2010/02/GENI_at_a_Glance_January_2010_Final-1.pdf.

[49] Mauro Campanella. *FEDERICA – Federated E-infrastructure Dedicated to European Researchers Innovating in Computing network Architectures*. European Commission. Februar 2009. URL: http://www.fp7-federica.eu/documents/web_leaflet_FEDERICA_ENG_feb09.pdf.

[50] Phuoc Tran-Gia. *G-Lab Phase 1 – Studien und Experimentalplattform für das Internet der Zukunft*. Bundesministerium für Bildung und Forschung. Dezember 2008. URL: http://www.german-lab.de/fileadmin/Press/G-Lab_White_Paper_Phase1.pdf.

[51] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker und Jonathan Turner. „OpenFlow: Enabling Innovation in Campus Networks“. In: *ACM SIGCOMM Computer Communication Review* 38.2 (2008), S. 69–74.

[52] Deepankar Medhi und Karthikeyan Ramasamy. *Network Routing: Algorithms, Protocols, and Architectures*. 1. Aufl. Morgan Kaufmann, März 2007.

[53] H. Jonathan Chao und Bin Liu. *High Performance Switches and Routers*. Wiley-IEEE Press, April 2007.

[54] Hormuzd Khosravi und Todd A. Anderson. *Requirements for Separation of IP Control and Forwarding*. RFC 3654, IETF. November 2003.

[55] Douglas Comer. *Network Systems Design Using Network Processors*. Pearson Education, Inc., Upper Saddle River, NJ, USA, 2006.

[56] John Freeman. „An Industry Analysist's Perspective on Network Processors“. In: *Network Processor Design – Issues and Practices*. Hrsg. von Patrick Crowley, Mark A. Franklin, Haldun Hadimioglu und Peter Z. Onufryk. Bd. 1. Morgan Kaufmann Publishers, 2003. Kap. 9, S. 191–218.

[57] Amit Singhal und Raj Jain. „Terabit switching: a survey of techniques and current products“. In: *Computer Communications* 25.6 (2002), S. 547 –556.

[58] Miguel Á. Ruiz-Sánchez, Ernst W. Biersack und Walid Dabbous. „Survey and Taxonomy of IP Address Lookup Algorithms“. In: *IEEE Network* 15.2 (2001), S. 8–23.

[59] Will Eatherton, George Varghese und Zubin Dittia. „Tree Bitmap: Hardware/Software IP Lookups with Incremental Updates“. In: *ACM SIGCOMM Computer Communication Review* 34.2 (2004), S. 97–122.

[60] Huan Liu. „Routing table compaction in ternary CAM“. In: *IEEE Micro* 22.1 (Januar 2002), S. 58–64.

[61] Texas Instruments Incorporated. *The Chip that Jack Built*. 2009. URL: http://www.ti.com/corp/docs/kilbyctr/jackbuilt.shtml.

[62] IEEE Global History Network. *Robert Noyce*. 2009. URL: http://www.ieeeghn.org/wiki/index.php/Robert_Noyce.

[63] Gordon E. Moore. „Cramming More Components onto Integrated Circuits“. In: *Electronics* 38.8 (April 1965), S. 114–117.

[64] Gordon E. Moore. *Progress In Digital Integrated Electronics*. 1975. URL: http://download.intel.com/museum/Moores_Law/Articles-Press_Releases/Gordon_Moore_1975_Speech.pdf.

[65] Wolfgang Arden u. a. *International Technology Roadmap for Semiconductors – 2007 Edition – Executive Summary*. http://www.itrs.net/Links/2007ITRS/ExecSum2007.pdf. 2007.

[66] Ed Clarke. „FPGAs and Structured ASICs: Low-Risk SoC for the Masses“. In: *Design & Reuse – Industry Articles* (Januar 2006).

[67] Jordan Plofsky. „The Changing Economics of FPGAs, ASICs and ASSPs“. In: *RTC Magazine* (April 2003).

[68] John F. Wakerly. *Digital Design: Principles and Practices*. 4. Aufl. Upper Saddle River, NJ 07458, USA: Pearson Prentice Hall, August 2005.

[69] Ian Kuon und Jonathan Rose. „Measuring the Gap Between FPGAs and ASICs“. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26.2 (Februar 2007), S. 203–215.

[70] Altera Corporation. *Power-Optimized Solutions for Telecom Applications*. Altera Corporation. Januar 2009. URL: http://www.altera.com/literature/wp/wp-01089-power-optimized-telecom.pdf.

[71] Susan Chen, Xin Wu und Prabhuram Gopalan. *Xilinx Next Generation 28 nm FPGA Technology Overview*. Xilinx, Inc. Februar 2010. URL: http://www.xilinx.com/support/documentation/white_papers/wp312_Next_Gen_28_nm_Overview.pdf.

[72] Zhi Guo, Walid Najjar, Frank Vahid und Kees Vissers. „A quantitative analysis of the speedup factors of FPGAs over processors“. In: *Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays (FPGA '04)*. 2004, S. 162–170.

[73] In-Stat. *Zero to Hero: The Emerging Network Processor Market*. Januar 2000. URL: http://www.marketresearch.com/product/display.asp?productid=1156216.

[74] Niraj Shah. „Understanding Network Processors“. Master Thesis. University of California, Berkeley, September 2001.

[75] Linley Gwennap. *Processors for Comms and Networking*. Presentation at Linley Tech Processor Conference 2009. September 2009.

[76] Wikipedia. *List of defunct network processor companies — Wikipedia, The Free Encyclopedia*. November 2009. URL: http://en.wikipedia.org/w/index.php?title=List_of_defunct_network_processor_companies&oldid=328246592.

[77] Optical Internetworking Forum. *System Packet Interface Level 4 (SPI-4) Phase 2 Revision 1: OC-192 System Interface for Physical and Link Layer Devices*. Implementation Agreement. Oktober 2003.

[78] Cortina Systems Inc. und Cisco Systems Inc. *Interlaken Protocol Definition*. Proprietary Material. Oktober 2008.

[79] Agere, Inc. *The Case for a Classification Language*. White Paper. September 1999.

[80] George Lawton. „Will Network Processor Units Live Up to Their Promise?“ In: *IEEE Computer* 37.4 (April 2004), S. 13–15.

[81] Xianghui Hu, Xinan Tang und Bei Hua. „High-performance IPv6 forwarding algorithm for multi-core and multithreaded network processor“. In: *Proceedings of the 11th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '06)*. 2006, S. 168–177.

[82] Jochen Kögel. „Design and Implementation of a Burst Assembly Unit based on a Network Processor“. Master Thesis. Institut für Kommunikationsnetze und Rechnersysteme, Universität Stuttgart, Januar 2005.

[83] Gene M. Amdahl. „Validity of the single processor approach to achieving large scale computing capabilities“. In: *Conference Proceedings of AFIPS Spring Joint Computer Conference*. 1967, S. 483–485.

[84] Intel Corporation. *Packet Processing with Intel Multi-Core Processors*. Intel Corporation. 2008. URL: http://download.intel.com/netcomms/solutions/ipservices-wireless/Intel_Holland_Tunnel_Whitepaper_Final2.pdf.

[85] Intel Corporation. *Integrating Services at the Edge*. Intel Corporation. 2009. URL: http://edc.intel.com/Download.aspx?id=2977&returnurl=/default.aspx.

[86] Simon Stanley. *TEMs Rate Multicore & Network Processor Suppliers*. Insider Research Services. April 2010. URL: http://img.lightreading.com/cci/pdf/cci0410_toc.pdf.

[87] Patrick Crowley und Jon Turner. *On the Use of General-Purpose Multi-Core Processors in Networking Devices*. PRESTO: Workshop on Programmable Routers for the Extensible Services of TOmorrow. Mai 2007.

[88] Netronome Systems. *NFP-3200 Network Flow Processor*. Netronome Systems. 2010. URL: http://netronome.com/files/file/Netronome%20NFP%20Product%20Brief%20(2-10).pdf.

[89] Erik J. Johnson und Aaron R. Kunze. *IXP2400/2800 Programming – The Complete Microengine Coding Guide*. Intel Press, April 2003.

[90] Xelerated AB. *Xelerated HX300 Family – 100 Gbps NPU with Integrated Traffic Manager, Switch, Programmable Pipeline and Ethernet MACs*. Xelerated AB. April 2010. URL: http://www.xelerated.com/Uploads/Files/64.pdf.

[91] Jakob Carlström und Thomas Boden. „Synchronous dataflow architecture for network processors“. In: *IEEE Micro* 24.5 (September 2004), S. 10–18.

[92] Jakob Carlström, Gunnar Nordmark, Joachim Roos, Thomas Boden, Lars-Olof Svensson und Pär Westlund. „A 40 Gb/s network processor with PISC dataflow architecture“. In: *Digest of Technical Papers of 2004 IEEE International Solid-State Circuits Conference (ISSCC 2004)*. Bd. 1. Februar 2004, S. 60–61.

[93] EZchip Technologies Ltd. *NP-4: 100-Gigabit Network Processor for Carrier Ethernet Applications*. EZchip Technologies Ltd. 2010. URL: http://www.ezchip.com/Images/pdf/NP-4_Short_Brief_online.pdf.

[94] Ran Giladi. *Network Processors: Architecture, Programming, and Implementation*. Morgan Kaufmann Publishers, 2008.

[95] Bay Microsystems, Inc. *Chesapeake: 40G Programmable Network Processor with MTM (Multimedia Traffic Management)*. Bay Microsystems, Inc. 2008. URL: http://www.baymicrosystems.com/assets/files/Chesapeake_Product_Brief_RevB.pdf.

[96] Chuck Gershman. *Sustained performance key to next-gen network processors*. Januar 2001. URL: http://www.eetimes.com/story/OEG20011001S0064.

[97] Will Eatherton. *The Push of Network Processing to the Top of the Pyramid*. Keynote Slides at ACM/IEEE Symposium on Architectures for Networking and Communications Systems 2005 (ANCS '05). 2005.

[98] Cisco Systems, Inc. *The Cisco QuantumFlow Processor: Cisco's Next Generation Network Processor*. Cisco Systems, Inc. 2008. URL: http://www.cisco.com/en/US/prod/collateral/routers/ps9343/solution_overview_c22-448936.pdf.

[99] Alcatel-Lucent. *Game-Changing Speed and Services – 100 Gigabit Ethernet interfaces for metro, edge and core IP networks*. Alcatel-Lucent. URL: http://www.alcatel-lucent.com/features/100GE/game_changing_ss.html.

[100] Roy Rubenstein. „Pushing packet performance“. In: *New Electronics* 43.2 (Januar 2010), S. 27–28.

[101] Juniper Networks, Inc. *JUNOS TRIO – Programmable Silicon Optimized for the Universal Edge*. Juniper Networks, Inc. 2009. URL: www.juniper.net/us/en/local/pdf/whitepapers/2000331-en.pdf.

[102] Juniper Networks, Inc. *T SERIES CORE ROUTERS ARCHITECTURE OVERVIEW – High-End Architecture for Packet Forwarding and Switching*. Juniper Networks, Inc. Februar 2010. URL: www.juniper.net/us/en/local/pdf/whitepapers/2000302-en.pdf.

[103] Anthony Torza. *Using FPGA Technologiy to Solve the Challenges of Implementing High-End Networking Equipment: Adding a 100 GbE MAC to Existing Telecom Equipment*. Xilinx, Inc. September 2008. URL: www.xilinx.com/support/documentation/white_papers/wp280.pdf.

[104] Altera Corporation. *Using 10-Gbps Transceivers in 40G/100G Applications*. Altera Corporation. Februar 2010. URL: www.altera.com/literature/wp/wp-01080-stratix-iv-gt-40g-100g.pdf.

[105] Altera Corporation. *Stratix V Device Handbook*. Altera Corporation. April 2010. URL: http://www.altera.com/literature/hb/stratix-v/stratix5_handbook.pdf.

[106] Altera Corporation. *Stratix V FPGAs: Built for Bandwidth*. Altera Corporation. 2010. URL: http://www.altera.com/literature/br/br-stratix-v-hardcopy-v.pdf.

[107] Achronix Semiconductor Corporation. *Introduction to Achronix FPGAs*. Achronix Semiconductor Corporation. August 2008. URL: http://www.achronix.com/docs/Introduction_to_picoPIPE_WP001.pdf.

[108] EN-Genius. *programmablelogicZONE Products for the week of December 15, 2008 – Achronix Semiconductor Says... Patented picoPIPE Acceleration Delivers World's Fastest FPGA*. Dezember 2008. URL: http://www.en-genius.net/site/zones/programmablelogicZONE/product_reviews/plp_121508.

[109] Achronix Semiconductor Corporation. *Speedster 100 Gbps Processing Platform — Bridge100*. Achronix Semiconductor Corporation. 2009. URL: http://achronix.com/docs/Bridge100_Product_Brief_PB007.pdf.

[110] Tabula, Inc. *SpaceTime Architecture – The best path from ideas to production silicon*. Tabula, Inc. 2010. URL: http://www.tabula.com/technology/TabulaSpacetime_WhitePaper.pdf.

[111] Tabula, Inc. *A Cost Effective Way to Design Next Generation Communications Infrastructure Equipment*. Tabula, Inc. 2010. URL: www.tabula.com/products/Tabula_Communications_WhitePaper.pdf.

[112] Tier Logic Incorporated. *Tier Logic 3D Technology – Unique New 3D Technology Delivers Low-Cost FPGAs and No-Risk, Timing-Exact ASICs*. Tier Logic Incorporated. März 2010. URL: http://www.tierlogic.com/uploads/press-room-files/Tier-Logic-3D-TierFPGA-and-TierASIC-Technology-Brief.pdf.

[113] TPACK A/S. *TPX4004 40–80 Gbps Carrier Packet Engine*. TPACK A/S. 2010. URL: http://www.tpack.com/index.php?eID=tx_bee3nawsecuredl&u=0&file=fileadmin/user_upload/PDF/Product_Briefs/TPX4004_v1_web.pdf&t=1272982516&hash=8668c0f939ec32ee0510759dd9bf7bba.

[114] Lars A. Pedersen. *Carrier Ethernet technologies in optical transport*. Presentation at Linley Group Carrier Ethernet Design Seminar 2010. Januar 2010.

[115] TPACK A/S. *SOFTSILICON for Flexible Packet Transport*. TPACK A/S. Januar 2008. URL: http://www.tpack.com/resources/tpack-white-papers/softsilicon.html.

[116] Ethernity Networks, Ltd. *Ethernity Networks – The Programmable Network*. Ethernity Networks, Ltd. 2010. URL: http://www.ethernitynet.com/.

[117] Ethernity Networks, Ltd. *Next Generation Multi Service Access Node Universal Line Card Processor*. Ethernity Networks, Ltd. September 2009. URL: http://www.ethernitynet.com/documents/NextGenerationMultiServiceAccessNode.pdf.

[118] Altera Corporation. *Intellectual property*. Altera Corporation. Dezember 2009. URL: http://www.altera.com/literature/sg/product-catalog.pdf.

[119] Xilinx, Inc. *ISE Design Suite: Intellectual Property*. Xilinx, Inc. Mai 2010. URL: http://www.xilinx.com/ipcenter/.

[120] MorethanIP GmbH. *More than IP – Products/Solutions – 100/40 Gigabit Ethernet*. MorethanIP GmbH. April 2010. URL: http://www.morethanip.com.

[121] Sarance Technologies Inc. *Sarance Technologies – Products – IP Cores*. Sarance Technologies. April 2010. URL: http://www.sarance.com/products/.

[122] Cswitch Inc. *Cswitch CS90: The Configurable Solution for Next-Gen Networks*. Cswitch Inc. 2008. URL: http://www.cswitch.com/www/Products/devices.htm.

[123] Umar Saif, James W. Anderson, Anthony Degangi und Anant Agarwal. „Gigabit routing on a software-exposed tiled-microprocessor“. In: *Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS '05)*. 2005, S. 51–60.

[124] Jörg-Christian Niemann. „Ressourceneffiziente Schaltungstechnik eingebetteter Parallelrechner – GigaNetIC“. Diss. Universität Paderborn, Dezember 2008.

[125] Jathin S. Rai, Yu-Kuen Lai und Gregory T. Byrd. „Packet Processing on a SIMD Stream Processor“. In: *Network Processor Design – Issues and Practices*. Hrsg. von Patrick Crowley, Mark A. Franklin, Haldun Hadimioglu und Peter Z. Onufryk. Bd. 3. Morgan Kaufmann Publishers, 2005. Kap. 7, S. 119–144.

[126] Michitaka Okuno, Shinji Nishimura, Shinichi Ishida und Hiroaki Nishi. „Cache-Based Network Processor Architecture: Evaluation with Real Network Traffic“. In: *IEICE Transactions on Electronics* E89-C.11 (November 2006), S. 1620–1628.

[127] Shinichi Ishida, Michitaka Okuno und Hiroaki Nishi. „Evaluation of cache base network processor by using real backbone network trace“. In: *Proceedings of the IEEE International Conference on High Performance Switching and Routing (HPSR 2006)*. 2006, 6 pp.

[128] Michael Meitinger, Rainer Ohlendorf, Thomas Wild und Andreas Herkersdorf. „FlexPath NP - A network processor architecture with flexible processing paths“. In: *Proceedings of International Symposium on System-on-Chip (SOC 2008)*. November 2008, S. 1–6.

[129] R. Ohlendorf, A. Herkersdorf und T. Wild. „FlexPath NP: a network processor concept with application-driven flexible processing paths“. In: *Proceedings of CODES+ISSS'05*. September 2005, S. 279–284.

[130] Carsten Albrecht, Roman Koch und Erik Maehle. „DynaCORE — A Dynamically Reconfigurable Coprocessor Architecture for Network Processors“. In: *Proceedings of the 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP '06)*. 2006, S. 101–108.

[131] Christoforos Kachris. „Reconfigurable Network Processing Platforms“. Diss. Technische Universiteit Delft, Dezember 2007.

[132] Kimon Karras, Thomas Wild und Andreas Herkersdorf. *100 Gbit/s Netzwerkprozessorarchitekturen: (NPU100) – Part of 100GET-E3*. Präsentation beim IKR-LIS Workshop 2007. Oktober 2007.

[133] Kimon Karras, Thomas Wild und Andreas Herkersdorf. „A Folded Pipeline Network Processor Architecture for 100 Gbit/s Networks“. In: *Proceedings of the 6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS '10)*. 2010.

[134] Wangyang Lai und Chin-Tau Lea. „A programmable state machine architecture for packet processing“. In: *IEEE Micro* 23.4 (Juli 2003), S. 32–42.

[135] Francisco Barat und Rudy Lauwereins. „Reconfigurable Instruction Set Processors: A Survey“. In: *Proceedings of the 11th IEEE International Workshop on Rapid System Prototyping (RSP 2000)*. Juni 2000, S. 168–173.

[136] Tilman Wolf, Ning Weng und Chia-Hui Tai. „Design considerations for network processor operating systems“. In: *Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS '05)*. 2005, S. 71–80.

[137] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti und M. Frans Kaashoek. „The Click modular router“. In: *ACM Transactions on Computer Systems* 18.3 (2000), S. 263–297.

[138] Niraj Shah, William Plishker, Kaushik Ravindran und Kurt Keutzer. „NP-Click: a productive software development approach for network processors“. In: *IEEE Micro* 24.5 (September 2004), S. 45 –54.

[139] Michael K. Chen, Xiao Feng Li, Ruiqi Lian, Jason H. Lin, Lixia Liu, Tao Liu und Roy Ju. „Shangri-La: achieving high performance from compiled network applications while enabling ease of programming“. In: *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '05)*. 2005, S. 224–236.

[140] Arthur Mutter, Martin Köhn und Matthias Sund. „A Generic 10 Gbps Assembly Edge Node and Testbed for Frame Switching Networks“. In: *Proceedings of the 5th International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM 2009)*. April 2009.

[141] Yuhua Chen, Jonathan S. Turner und Zhi Zhai. „Design of an ultra fast pipelined wavelength scheduler for optical burst switching“. In: *Springer Photonic Network Communications* 14 (2007), S. 317–326.

[142] John W. Lockwood, Naji Naufel, Jonathan S. Turner und David E. Taylor. „Reprogrammable network packet processing on the field programmable port extender (FPX)“. In: *Proceedings of the 2001 ACM/SIGDA 9th International Symposium on Field Programmable Gate Arrays (FPGA '01)*. 2001, S. 87–93.

[143] David E. Taylor, Jonathan S. Turner, John W. Lockwood, Todd S. Sproull und David B. Parlour. „Scalable IP lookup for Internet routers“. In: *IEEE Journal on Selected Areas in Communications* 21.4 (Mai 2003), S. 522–534.

[144] Institute of Communication Networks and Computer Engineering, Universität Stuttgart. *The Universal Hardware Platform (UHP)*. 2005. URL: http://www.ikr.uni-stuttgart.de/Content/UHP/.

[145] Sascha Junghans. „Realisierbarkeit von Scheduling-Modulen in Optical Burst Switching-Kernknoten“. Communication Networks and Computer Engineering Report No. 93 (Dissertation). Universität Stuttgart, Institut für Kommunikationsnetze und Rechnersysteme, 2007.

[146] Greg Watson, Nick McKeown und Martin Casado. „NetFPGA: A Tool For Network Research and Education“. In: *2nd Workshop on Architecture Research using FPGA Platforms (WARFP)*. Februar 2006.

[147] Glen Gibb, John W. Lockwood, Jad Naous, Paul Hartke und Nick McKeown. „NetFPGA – An Open Platform for Teaching How to Build Gigabit-Rate Network Switches and Routers“. In: *IEEE Transactions on Education* 51.3 (August 2008), S. 364–369.

[148] Nandita Dukkipati, Glen Gibb, Nick McKeown und Jiang Zhu. „Building a RCP (Rate Control Protocol) Test Network“. In: *Proceedings of the 15th Annual IEEE Symposium on High-Performance Interconnects (HOTI '07)*. 2007, S. 91–98.

[149] Jad Naous, David Erickson, G. Adam Covington, Guido Appenzeller und Nick McKeown. „Implementing an OpenFlow switch on the NetFPGA platform“. In: *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS '08)*. 2008, S. 1–9.

[150] Gordon Brebner, Phil James-Roxby, Eric Keller und Chidamber Kulkarni. „Hyper-Programmable Architectures for Adaptable Networked Systems“. In: *Proceedings of the 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP '04)*. 2004, S. 328–338.

[151] Eric Keller und Gordon Brebner. „Programming a Hyper-Programmable Architecture for Networked Systems“. In: *Proceedings of International Conference on Field-Programmable Technology (FPT)*. Dezember 2004.

[152] Michael Attig und Gordon Brebner. „Systematic Characterization of Programmable Packet Processing Pipelines“. In: *Proceedings of 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '06)*. April 2006, S. 195–204.

[153] Chidamber Kulkarni, Gordon Brebner und Graham Schelle. „Mapping a domain specific language to a platform FPGA“. In: *Proceedings of the 41st Annual Design Automation Conference (DAC '04)*. 2004, S. 924–927.

[154] Gordon Brebner. „Packets everywhere: The great opportunity for field programmable technology“. In: *Proceedings of International Conference on Field-Programmable Technology (FPT 2009)*. Dezember 2009, S. 1–10.

[155] Graham Schelle und Dirk Grunwald. „CUSP: a modular framework for high speed network applications on FPGAs“. In: *Proceedings of the 2005 ACM/SIGDA 13th International Symposium on Field-Programmable Gate Arrays (FPGA '05)*. 2005, S. 246–257.

[156] Cristian Soviani, Ilija Hadzic und Stephen A. Edwards. „Synthesis of high-performance packet processing pipelines“. In: *Proceedings of the 43rd ACM/IEEE Design Automation Conference*. Juli 2006, S. 679–682.

[157] Cristian Soviani, Ilija Hadzic und Stephen A. Edwards. „Synthesis and Optimization of Pipelined Packet Processors“. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 28.2 (Februar 2009), S. 231–244.

[158] Kaushik Ravindran, Nadathur Rajagopalan Satish, Yujia Jin und Kurt Keutzer. „An FPGA-based soft multiprocessor system for IPv4 packet forwarding“. In: *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '05)*. 2005, S. 487–492.

[159] Michael Meredith. *A look inside behavioral synthesis*. Forte Design Systems. August 2004. URL: http://www.eetimes.com/showArticle.jhtml;?articleID=18900783.

[160] Philippe Coussy und Adam Moraviec, Hrsg. *High-Level Synthesis – from Algorithm to Digital Circuit*. Springer Science + Business Media B.V., 2008.

[161] Stamatis Vassiliadis, Stephan Wong und Sorin Cotofana. „Microcode Processing: Positioning and Directions“. In: *IEEE Micro* 23.4 (2003), S. 21–31.

[162] Matthias Meyer. „An On-Chip Garbage Collection Coprocessor for Embedded Real-Time Systems“. In: *Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*. August 2005, S. 517–524.

[163] Chuck Semeria. *Implementing a Flexible Hardware-based Router for the New IP Infrastructure*. JuniperNetworks Inc., White Paper. 2001.

[164] Rainer Hagenau, Carsten Albrecht, Erik Maehle und Andreas C. Döring. „Parallel Processing in Network Processor Architectures (Parallelverarbeitung in Netzwerkprozessorarchitekturen)“. In: *it - Information Technology* 47.3 (März 2005), S. 123–131.

[165] Intel. *Intel Internet Exchange Architecture Portability Framework Reference Manual*. Intel Corporation. November 2003.

[166] Weiming Wang, Evangelos Haleplidis, Kentaro Ogawa, Fenggen Jia und Joel Halpern. *ForCES LFB Library*. IETF Internet-Draft (Informational). März 2010.

[167] Altera Corporation. *Increasing Productivity With Quartus II Incremental Compilation*. Altera Corporation. Mai 2008. URL: http://www.altera.com/literature/wp/wp-01062-quartus-ii-increasing-productivity-incremental-compilation.pdf.

[168] The Eclipse Foundation. *Eclipse.org home*. The Eclipse Foundation. URL: http://www.eclipse.org/.

[169] Mentor Graphics. *Mentor Graphics – HDL Designer*. Mentor Graphics. Juli 2010. URL: http://www.mentor.com/products/fpga/hdl_design/hdl_designer_series/.

[170] Mentor Graphics. *Mentor Graphics – Modelsim*. Mentor Graphics. Juli 2010. URL: http://www.mentor.com/products/fpga/simulation/modelsim.

[171] Mentor Graphics. *Mentor Graphics – Advanced FPGA Synthesis*. Mentor Graphics. Juli 2010. URL: http://www.mentor.com/products/fpga/synthesis/.

[172] Synopsys, Inc. *Synopsys – FPGA Implementation*. Synopsys, Inc. Juli 2010. URL: http://www.synopsys.com/Tools/Implementation/FPGAImplementation.

[173] John L. Hennessy und David A. Patterson. *Computer Architecture: A quantitative approach*. 2. Aufl. San Francisco, CA, USA: Morgan Kaufmann Publishers, Inc., 1996.

[174] Paul J. Kühn und Matthias Meyer. *Technische Informatik I*. Unterlagen zur Lehrveranstaltung. Oktober 2008.

[175] Altera Corporation. *Stratix II Device Handbook*. Altera Corporation. Mai 2007. URL: http://www.altera.com/literature/hb/stx2/stratix2_handbook.pdf.

[176] NetLogic Microsystems, Inc. *NL5000 RD Knowledge-based Processors 128K, and 64K Records*. Revision 2.0. NetLogic Microsystems, Inc. Januar 2006.

[177] Philipp Schrem. *Generisches Ethernet-Schnittstellenmodul*. Interne Dokumentation. Juni 2006.

[178] Arthur Mutter und Martin Köhn. *Management System for Hardware Designs*. Vortrag. November 2006.

[179] Michael Scharf. „Fast Startup Internet Congestion Control Mechanisms for Broadband Interactive Applications“. to be published as Communication Networks and Computer Engineering Report (Dissertation). Universität Stuttgart, Institut für Kommunikationsnetze und Rechnersysteme, 2009.

[180] Mentor Graphics Corporation. *Precision Synthesis User's Manual*. Mentor Graphics Corporation. April 2010.

[181] Altera Corporation. *Quartus II Handbook Version 10.0*. Altera Corporation. Juli 2010. URL: http://www.altera.com/literature/hb/qts/quartusii_handbook_10.0.pdf.

[182] Altera Corporation. *Stratix IV Device Handbook*. Altera Corporation. März 2010. URL: http://www.altera.com/literature/hb/stratix-iv/stratix4_handbook.pdf.

[183] Anil Rijsinghani. *Computation of the Internet Checksum via Incremental Update*. RFC 1624, IETF. Mai 1994.

[184] Intel. *Intel Internet Exchange Architecture Software Building Blocks Developer's Manual*. Intel Corporation. März 2004.

[185] *BGP Analysis Report: APNIC R&D and Route-Views Oregon IPv4 BGP Reports*. Oktober 2010. URL: http://bgp.potaroo.net/index-bgp.html.

[186] Chawapong Suriyajan. „Design and Implementation of Quick-Start and XCP Router Functions on a Network Processor“. Master Thesis. Institut für Kommunikationsnetze und Rechnersysteme, Universität Stuttgart, Oktober 2007.