Linköping Studies in Science and Technology

Dissertation No. 865

Protocol Processing in Network Terminals

Ulf Nordqvist

INSTITUTE OF TECHNOLOGY
LINKÖPINGS UNIVERSITET

Department of Electrical Engineering
Linköpings universitet, SE-581 83 Linköping, Sweden
Linköping 2004

Protocol Processing in Network Terminals

Department of Electrical Engineering
Linköpings universitet, SE-581 83 Linköping, Sweden
Linköping 2002

ISBN: 91-7373-914-6 **ISSN: 0345-7524**

Abstract

The bandwidth and number of users in computer networks are rapidly growing today. The need for added functionality in the network nodes is also increasing. The requirements on the processing devices get harder and harder to meet using traditional hardware architectures. Hence, a lot of effort is currently focused on finding new improved hardware architectures dedicated for processing of packets and network protocols.

In the emerging research area of protocol processing, there exist many hardware platform proposals. Most of them aim for router applications, not so many for terminals. As a starting point for terminal research this thesis explores a number of different router design alternatives and some common computer architecture concepts. These concepts and architectures have been examined and evaluated to see if some ideas apply also to protocol processing in network terminals.

- Requirements on protocol processors for terminals can be summarized as:
- Low silicon area
- Low power consumption
- Low processing latency
- High processing throughput
- Flexible implementation

Fulfilling these requirements while supporting offloading of as much protocol processing as possible to the network interface is the key issue of this thesis. Offloading means that the protocol processing can be executed in a special unit that does not need to execute the host applications as well. The protocol processor unit basically acts as a smart network interface card.

A novel terminal platform solution is proposed in this thesis. The dual processor platform is accelerated using a programmable protocol processor. The processor uses a number of different dedicated hardware blocks, which operate in parallel, to accelerate the platform in a configurable way. These hardware blocks have been selected and specified to fulfill requirements set by a number of common network protocols. To find these requirements, the protocol processing procedure has been

investigated and divided into processing tasks. These different tasks have been explored to see which are suitable for hardware acceleration and which should be processed in the other part of the platform which is a general purpose micro controller.

The dedicated datapath, simplified control, and minimal usage of data buffers make the proposed processor attractive from a power perspective. Further it accelerates the platform so that high speed operation is enabled. Different implementation alternatives are provided in this thesis. Which one to select depends on what kind of terminal the platform is going to be used for. Further this thesis includes a discussion around how the ability to reassembly fragmented packets demands architectural modifications.

I hope You will enjoy reading this dissertation!

Linköping in December 2003

Ulf Nordqvist

Preface

This thesis is intended for a Doctoral degree in Technical Sciences. The thesis presents the results of my research during the period 1999-2003 at Linköping Universitet, Sweden.

The following publications are included in the thesis:

- Publication 1: Dake Liu, Ulf Nordqvist, Christer Svensson "Configuration based architecture for high speed and general purpose protocol processing", *in proceedings of SIPS 1999*, Taipei, Taiwan, p.p. 540-547
- Publication 2: Ulf Nordqvist, Tomas Henriksson, and Dake Liu, "CRC Generation for Protocol Processing", *in proceedings of NORCHIP 2000*, Turku, Finland, November 6-7, 2000, pp. 288-293
- Publication 3: Ulf Nordqvist, Dake Liu, "Configurable CRC Generator", *in proceedings of the IEEE Design and Diagnostics of Electronic Circuits and Systems, DDECS, Brno, April 17-19, 2002, pp. 192-199*
- Publication 4: Ulf Nordqvist, Dake Liu, "Packet Classification and Termination in a Protocol Processor", *in proceedings of the second Workshop on Network Processors (HPCA/NP2), Anaheim, USA, February 2003, pp 88-99*
- Publication 5: Ulf Nordqvist and Dake Liu, "Power Optimized Packet Buffering in a Protocol Processor", to appear *in the proceedings of the ICECS 2003, held in Sharjah, United Arab Emirates, Dec. 2003*
- Publication 6: Ulf Nordqvist and Dake Liu, "Control Path in a Protocol Processor", *to appear in the proceedings of the Midwest Symposium on Circuit and Systems, MWCAS 2003, Cairo, Egypt, Dec. 2003*
- Publication 7: Ulf Nordqvist and Dake Liu, "Chapter 8: Packet Classification and Termination in a Protocol Processor", *Book chapter in the Network Processor Design; Issues and Practices, Volume 2, Chapter 8, Morgan Kaufman Publishers, 2003, ISBN: 0-12-198157-6*

Other publications related to the scope of this thesis and to the author, but not included in the thesis are:

- Publication 8: Ulf Nordqvist, "On Protocol Processing", *in proceedings of CCSSE 2001, Norrkoping, Sweden, 2001, Mar 14-15, pp. 83-89*
- Publication 9: Ulf Nordqvist, "A Comparative Study of Protocol Processors", *in proceedings of CSSCE 2002, Norrkoping, Sweden, Oct 23-24 2002, pp. 107-113*
- Publication 10: Ulf Nordqvist, "Power Efficient Packet Buffering in a Protocol Processor", *Swedish System on a Chip Conference, Vol. SSOCC'03, Apr 2003*
- Publication 11: Ulf Nordqvist, "A Programmable Network Interface Accelerator", *Licentiate degree thesis, Linköpings Universitet, Thesis no. 998, Jan 2003*
- Publication 12: Tomas Henriksson, Ulf Nordqvist, and Dake Liu, "Embedded Protocol Processor for Fast and Efficient Packet Reception", *in proceedings of International Conference on Computer Design, Freiburg, Germany, pp. 414-419, Sep 2002*
- Publication 13: Tomas Henriksson, Ulf Nordqvist, and Dake Liu, "Specification of a configurable general-purpose protocol processor", *in the IEE proceedings of Circuits, Devices and Systems,. Vol. 149, No. No. 3, pp. 198-202, Juni 2002*
- Publication 14: Tomas Henriksson, Henrik Eriksson, Ulf Nordqvist, Per Larsson-Edefors, and Dake Liu, "VLSI Implementation of CRC-32 for 10 Gigabit Ethernet", *In proceedings of The 8th IEEE International Conference on Electronics, Circuits and Systems,* Malta, September 2-5, 2001, vol. III, pp. 1215-1218
- Publication 15: Tomas Henriksson, Ulf Nordqvist and Dake Liu, "Specification of a configurable General-Purpose Protocol Processor", *In proceedings of Second International Symposium on Communication systems, Networks and Digital Signal Processing*, Bournemouth, UK, July 19-20, 2000, pp. 284-289
- Publication 16: Tomas Henriksson, Ulf Nordqvist, and Dake Liu, "Configurable Port Processor Increases Flexibility in the Protocol Processing Area", *In proceedings of COOLChips III An International Symposium on Low-Power and High-Speed Chips,* Kikai-Shinko-Kaikan, Tokyo, Japan, April 24-25, 2000, pp. 275
- Publication 17: A Herzog, Nahid Shahmehri, A Bednarski, I Cisalita, Ulf Nordqvist, L Saldahni, D Szentivanyi och Måns Östring, "*Security Issues in E-Home Network and Software Infrastructures*", Proceedings of CCSSE, Norrköping, Sweden, Okt 2001

Acknowledgments

The credit for making this thesis possible to accomplish does not belong only to one single person. Instead it is the sincerely appreciated support of many different persons who have helped during my research work that deserves a hats-off from me.

First of all, I would like to thank my supervisor, Professor Dake Liu, for guidance, inspiring discussions, proofreading, and for giving me the opportunity to do this work.

I would like to acknowledgment my fellow Ph.D. student Dr Tomas Henriksson for valuable discussions, for coauthoring papers and for sharing information and ideas.

Lic. Eng. Kalle Folkesson and Lic. Eng Daniel Wiklund are acknowledged for proof reading this thesis.

I would also like to thank the professor at my former research group, Electronic Devices, Professor Christer Svensson and the former professor at that group, Professor Per Larsson-Edefors for inspiring and helping me. Especially during the beginning of my career as a Ph.D. student.

Sharing the working environment with an inner circle of Ph.D. students like Dr. Daniel Eckerbert, Dr. Henrik Eriksson, Mikael Olausson, Erik Tell, Stefan Andersson, Peter Caputa, Andreas Ehliar and Sumant Sathe has been a true pleasure.

All of the former and present members of the Electronic Devices and the Computer Engineering groups are acknowledged for the technical and administrative support.

I would like to thank the swedish strategic research foundation (SSF) for the funding of my research through ECSEL and Stringent projects.

Finally I would like to thank my relatives for supporting but not understanding my work.

Abbreviations

ARP	Address resolution Protocol
ASIC	Application Specific Integrated Circuit
ASIP	Application Specific Instruction set Processor
ATM	Asynchronous Transfer Mode
CAM	Content Addressable Memory
CMAA	Control Memory Access Accelerator
CRC	Cyclic Redundancy Check
FP	Functional Page
FPGA	Field Programmable Gate Array
GMII	Gigabit Media Independent Interface
HBA	Host Bus Adaptor
ILP	Instruction Level Parallelism
IP	Internetwork Protocol
IPv4	Internetwork Protocol version 4
IPv6	Internetwork Protocol version 6
iSCSI	Internet Small Computer System Interface
LAN	Local Area Network
MAC	Media Access Control
MTU	Maximum Transmission Unit
NIC	Network Interface Card
NP	Network Processor
NT	Network Terminal
OS	Operating System
PaP	Packet Processor
PDU	Protocol Data Unit

PHY	Physical Layer
PLD	Programmable Logic Devices
PNI	Programmable Network Interface
PP	Protocol Processor
PPP	Programmable Protocol Processor
QoS	Quality of Service
SAN	Storage Area Network
SAR	Segmentation And Reassembly
SSL	Secure Socket Layer
STCAM	Simplified TCAM
TCAM	Ternary Content Addressable Memory
TCP	Transport Control Protocol
TOE	TCP Offload Engine
TTL	Time to Live
UDP	User Datagram Protocol
ULP	Upper Layer Protocol
XAC	eXtract And Compare
XDR	External Data Representation

Table of contents

Part I

Background

“There is nothing permanent except change.”
--Heraclitus

1

Introduction

Few research artifacts have made as big an impact on society as the Internet. Both the number of connected hosts and daily users now number in the millions. As more users get connected, more people are developing novel methods of using this global resource. A basic component of any advanced computing system today is a computer network. The computer network is used to exchange information of different types, for example data files, live audio, and video. A computer network consists of two types of entities: routers and terminals. The routers and the terminals are connected with links. A router is used to forward information on a link, which leads to the destination terminal. The terminals are the sources and sinks of information. The links today operate at very high data rates, thanks to the advances in optical transmission. The bottleneck in computer networks is since a few years the protocol processing entities in the networks. Network processors are becoming a predominant feature in the field of network hardware. As new network protocols emerge and data speeds increase, contemporary general-purpose network processors are entering their second generation and academic research is being actively conducted into new techniques for the design and implementation of these systems.

Much research has been focused on the routers and this has left the terminals lagging behind. This is one reason why this thesis focuses the efforts on terminals.

1.1 Motivation

This thesis addresses a number of problems that recently have become obvious to the network processing community. The need for new hardware platforms, i.e. network processors, is motivated by current network processing trends. These trends are described in more detail in “Protocol processing trends” on page 35. However, as an introduction and motivation for this work, the following subsections introduce the emerging need for increased intelligence and performance in network processing entities.

1.1.1 The bandwidth-processing power gap

The data rates of computer networks keep increasing beyond the Gbit/s limit. This presents a problem for the terminals, since the processing power does not increase as fast. This is sometimes referred to as the Bandwidth- Processing Power Gap. The data rate (bandwidth) of computer networks is typically increased by a factor of 4 (SONET/SDH) or 10 (Ethernet) for every new generation of standards. During the last year new standards have appeared almost every year. The processing power on the other hand grows almost with Moores law, which predicts a doubling every 18 months. The increase in processing power because of Moores law is not enough to keep up with the increase in bandwidth. It can be realized that the processing of computer network protocols must be more efficiently implemented.

Example:

Consider a general purpose RISC machine in a 10 Gbps network terminal. Assume min-sized packets (64 bytes), no gap between the packets arriving (worst case), and suppose that data arrives 32 bits in parallel. The data arrival then only takes 51 ns. A traditional 500 MHz RISC machine would then have to manage all the packet processing using 25 instructions per packet. The alternative is to buffer the data and fill the memory. Neither alternative is realistic if one consider the fact that the processor is supposed to process the application in parallel.

1.1.2 Need for increased network intelligence

The current growth rate of the Internet leads to congestion of major parts of the network since the infrastructure cannot be updated at the same speed. As a result, degraded connectivity and even starvation of transmissions are appearing. Moreover, certain flows may occupy more networking resources than others because nodes usually handle packets without considering any flow-specific information. Consequently, a flow may greedily use bandwidth by, for instance, transferring only relatively large packets. Therefore, the access to networks must be regulated according to reservations and the network must be protected against greedy flows. A network node has to apply more sophisticated methods than best effort to maintain the Quality of Service (QoS) for customers. Special QoS algorithms are

required for the packet processing of especially the upper protocol layers, starting at the network layer from which end-to-end transmissions are distinguishable. Enabling this increased functionality in network processing devices requires the hardware platforms to support a much larger flexibility, i.e. programmability and/or configurability.

1.2 Scope

While much focus, both from industry and academic research groups, has recently been on improving the network processing equipment in routers and switches, this thesis focuses on network terminals. A network terminal do not have the same requirements on throughput as a network core component. However, the problem will soon arise also for NTs. Wire speeds in local area networks (LANs) increase as fast as they do in backbone networks and common use of Gbit/s networks connected directly to network terminals is not very far away. In order to meet these requirements as well as the very high requirements on flexibility the protocols put on the terminal, it is essential to find a new protocol processing strategy and hardware platform dedicated for packet reception in terminals. This thesis provides both. The platform can be implemented in two different ways to meet the requirements of a few common protocols for two different application domains (i.e. type of terminals). This shows that the platform and strategy is generic enough to support protocol processing for a wide number of applications, networks, and terminal types.

The scope of this thesis does not include packet creation and transmission. Normally this task is much less complex for terminals. The requirements on processing latency is also lower during transmission of packets. Further, packet transmission is not as suitable for offloading as packet reception tasks. The proposed architecture was optimized for packet reception but it still support transmission by providing low latency protocol processing, which can be used to trigger low latency acknowledgments.

1.3 Contributions

As its main contributions this thesis:

- Proposes a dual processor architecture intended for offloading of protocol processing tasks from a network terminal host.
- Proposes a processing task allocation based on processing type instead of protocol and/or layer type.
- Provides an implementation methodology for this architecture dependent on the application domain.

- Provides an investigation of architectural implications of the data-flow based processing including memory and buffering design.
- Includes specification and implementation of several configurable hardware accelerators denoted Functional Pages.

1.4 Dissertation outline

This thesis consists of four parts. The first part includes this introductory chapter and a chapter on Computer Networks. The goal of chapter 2 is to provide an introduction to computer networks for readers unfamiliar with the area. It also defines and explains a number of concepts and acronyms that will be used throughout this thesis. Further, the two main types of network components, terminals and routers, are examined.

The second part including chapter 3-5 gives an overview of available technologies and methodology from earlier research on computer architecture and especially network processor design. Chapter 3 provides different parallelization schemes and implementation alternatives for processors. The chapter also lists common optimization methods and discusses usability for protocol processing. In chapter 4, protocol processing trends, tasks, and requirements are examined with focus on network terminals. The next chapter gives an overview of commercial and academic research in the area of network processing. Finally, chapter 5 includes a discussion on how this evaluation of different network processor projects aiming for router applications, can be used in our research which focus on terminal processing. I.e. what lessons can be learned from industry.

The third part deals with the proposed architecture in six chapters. In chapter 6, which is based on publication 1, an overview of the proposed dual processor platform is provided. It also lists tasks and explains their allocation to different parts of the processing platform. Based on publications 1 and 5, chapter 8 deals with memory and buffering issues. In chapter 9, a number of hardware accelerators are specified and implemented. Publication 2 and 3, deal with CRC acceleration which is a part of chapter 9. In chapter 10 control path implementations are presented based on publication 6. This chapter also defines the application domain the implementations are suitable for. In chapter 11, more general purpose network terminal applications are assumed. This chapter provides a discussion on how the architecture must be changed and discusses a suitable implementation method for terminals handling fragmented packets and more complex classification schemes. Chapter 11 is based on publication 1, 4, and 7.

Finally some conclusions are drawn in chapter 12 and chapter 13 discusses how the proposed architecture and other research result can be used when applied to gateway applications.

2

Computer Networks - Devices and Protocols

This chapter includes a brief introduction to the concept of packet based networks focusing on computer networks. Moreover, some ambiguous network terms used in this thesis are clarified. Readers seeking deeper understanding in this area, are recommend to use some of all available books on computer networks. A good first encounter of the area and an excellent starting point for further reading is provided in ([2.1] and [2.3]). Readers with a background knowledge in the area of computer networking including familiarity with the protocols and devices, may go directly to the next chapter for further reading.

2.1 History

While inventing and building the Internet the main goal was to set up a decentrally organized interconnection of computers with redundancy so that a breakdown of one part of the network would not affect the connectivity and efficiency of the overall Internet. Consequently, a node of the network only knows its neighbors, which are identified by unique addresses. Since routes through the network are not determined statically but dynamically depending on the current state of the network, every single data entity must be processed by every intermediate network node from the source to the destination of a transmission, including the terminal.

Further, the network communication was considered as the primary bottleneck while terminal hosts where seen as a source of infinite processing power. Today

this picture is totally the opposite but since the protocol standards used today are so well established we have to deal with this inherited problem instead of re-designing the protocol and entire network structure. This motivates why the research focus should also be on terminals, not only routers.

2.2 Protocol layers

The tasks involved with the end-to-end communication of two network nodes can be divided into a set of abstract functionalities, called layers, that form a hierarchy. Each layer can only pass information to the next higher or lower layer through defined interfaces (services). At each layer, protocols define the operations and responses necessary to exchange information between peer layers at different network nodes. This information is held by layer-specific header fields that are added to traffic entities. Lower layers only consider the transmission of traffic between neighboring network nodes whereas the higher layers affect the end-to-end transmission through several intermediate nodes.

2.2.1 ISO/OSI reference model

The Open Systems Interconnection (OSI) reference model by ISO is composed of seven abstract layers illustrated by figure 2.1.

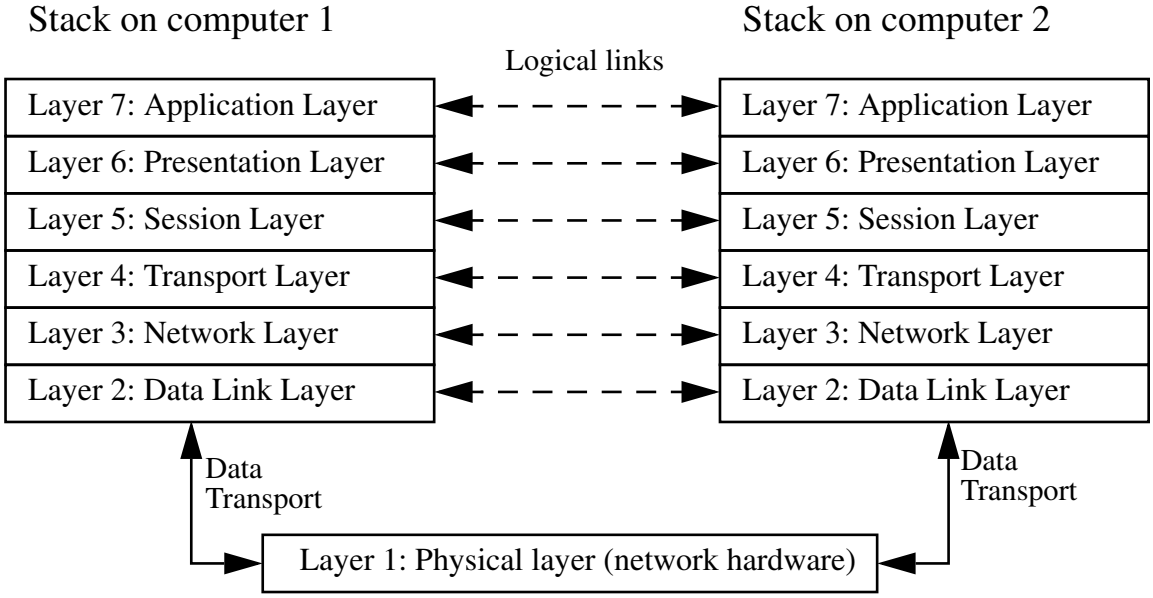

Figure 2.1: The 7 layer ISO/OSI reference model

- **Layer 1 - Physical**

The Physical layer defines the cable or physical medium itself, e.g. unshielded twisted pairs. All media are functionally equivalent. The main difference is in bandwidth, convenience, and cost of installation and maintenance. Converters from one media to another operate at this level.

- **Layer 2 - Data Link**

The Data Link layer defines the format of data on the network. A network data frame, a.k.a. packet, includes a checksum, source and destination address, and data. The largest packet that can be sent through a data link layer defines the Maximum Transmission Unit (MTU). The data link layer handles the physical and logical connections to the packet's destination, using a network interface. For example, a host connected to an Ethernet would have an Ethernet interface to handle connections to the outside world, and a loopback interface to send packets to itself.

Ethernet addresses a host using a unique, 48-bit address called its Ethernet address or Media Access Control (MAC) address. MAC addresses are usually represented as six colon-separated pairs of hex digits, e.g. 8:0:20:11:ac:85. This number is unique and is associated with a particular Ethernet device. The protocol-specific header specifies the MAC address of the packets source and destination. When a packet is sent to all hosts (broadcast), a special MAC address (ff:ff:ff:ff:ff:ff) is used.

- **Layer 3 - Network**

Almost all computer networking applications use Internetwork Protocol (IP) as its network layer interface. IP is responsible for routing, i.e. directing datagrams from one network to another. The network layer may have to break large datagrams, larger than the MTU, into smaller packets and the host receiving the packets will have to reassemble the fragmented datagram. The Internetwork Protocol identifies each host with a 32-bit IP address. IP addresses are written as four dot-separated decimal numbers between 0 and 255, e.g. 129.79.16.40. The leading 1-3 bytes of the IP identify the network and the remaining bytes identifies the host on that network. For large sites, the first two bytes represent the network portion of the IP, and the third and fourth bytes identify the subnet and host respectively.

Even though IP packets are addressed using IP addresses, hardware addresses must be used to actually transport data from one host to another. The Address Resolution Protocol (ARP) is used to map the IP address to a hardware address.

- **Layer 4 - Transport**

The transport layer subdivides user-buffer into network-buffer sized datagrams and enforces desired transmission control. Two transport protocols, Transmission Control Protocol (TCP) and User Datagram Protocol (UDP), sits at the transport layer. Reliability and speed are the primary differences between these two protocols. TCP establishes connections between two hosts on the network through sockets, which are determined by the IP address and port number. TCP keeps track of the packet delivery order and the packets that must be resent. Maintaining this information for each connection makes TCP a connection oriented protocol. UDP

on the other hand provides a low overhead transmission service, but with less error checking.

- **Layer 5 - Session**

The session protocol defines the format of the data sent over the connections.

- **Layer 6 - Presentation**

External Data Representation (XDR) sits at the presentation level. It converts local representation of data to its canonical form and vice versa. The canonical uses a standard byte ordering and structure packing convention, independent of the host

- **Layer 7 - Application**

Provides network services to the end-users. Mail, file transfer protocol (ftp), telnet, and Domain Name System (DNS) are examples of network applications.

2.2.2 TCP/IP reference model

Although the OSI model is widely used and often cited as the standard, TCP/IP protocol has become the totally dominant protocol stack description. TCP/IP reference model is designed around a simple four-layer scheme. It does omit some features found under the OSI model. Also it combines the features of some adjacent OSI layers and splits other layers apart. The TCP/IP protocol stack distinguishes the following layers.

- **Layer 1 - Link**

This layer defines the network hardware and device drivers.

- **Layer 2 - Network**

This layer is used for basic communication, addressing, and routing. TCP/IP uses IP and ICMP protocols at the network layer.

- **Layer 3 - Transport**

Handles communication among programs on a network. TCP and UDP falls within this layer.

- **Layer 4 - Application**

End-user applications reside at this layer. Commonly used applications include DNS, rlogin, talk, and ftp.

2.2.3 Traditional layered processing

A traditional way of describing a protocol layer is illustrated by figure 2.2. The figure is a very general description of protocol layers but it shows the layered structure that causes many of the problems emerging today. The layers are today very well specified and it provides us with an interface between the different ser-

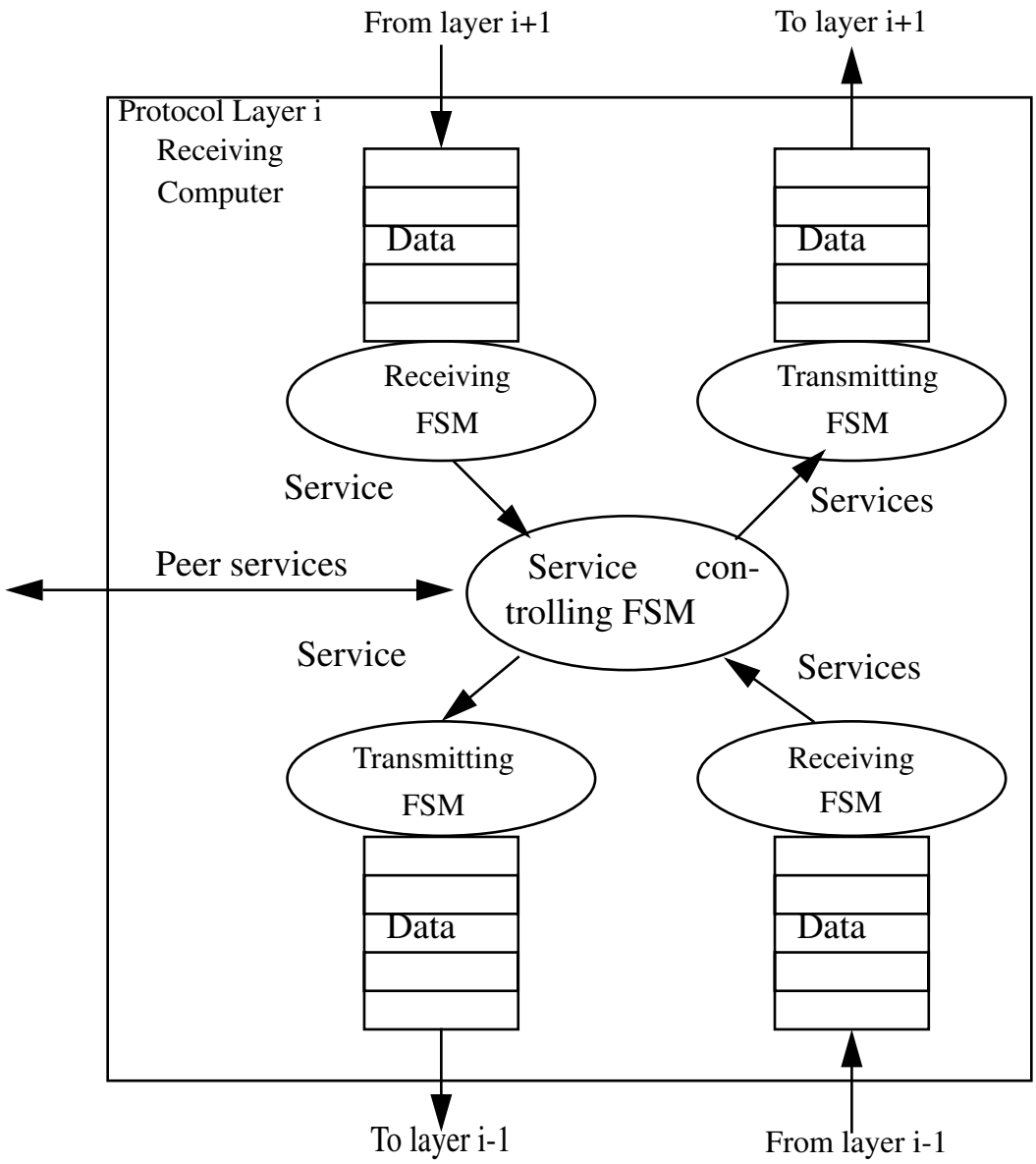

Figure 2.2: Traditionally layered protocol processing concept. During reception each protocol layer receives data and other services from the layer below. The data is processed in order to provide the peer services to the transmitting computer. In the same way all protocol layers provide services to the layer above. Each layer's service provider is called an entity. The entities can be implemented in software or hardware or in a combination of both.

vice entities, e.g. devices or pieces of software. The problem is the wasted processing this architecture gives when providing services on all layers while it is only the top layer services that is going to be used by the host while it performs the actual application processing. The layered processing scheme is depicted by figure 2.2.

2.3 Classification of networks

Computer networks can be categorized according to different criterias:

2.3.1 Classification according to geographic coverage

- **LAN**

A Local Area Network (LAN) interconnects end-devices within a relatively small area, e.g. bounded by a room, a building, or some buildings belonging to the same institution. In the latter case, LANs are also called campus area or enterprise networks. LAN protocols function at the lowest two layers of the OSI reference model, between the physical layer and the data link layer. A LAN is a high-speed data network. It typically connects workstations, personal computers, printers, servers, and other devices. Devices commonly used in LANs include repeaters, hubs, bridges, LAN switches, and routers.

- **Wireless LAN**

Today there exist a number of different protocols for wireless LAN applications. They differ a lot in terms of performance, cost-figures, coding schemes and connection orientation. The main alternatives are IEEE standard 802.11 and HiperLAN ([2.5] and [2.6]).

- **WAN**

A Wide Area Network (WAN) interconnects LANs that are possibly based on different technologies and may belong to different organizational units spread over a large geographic area. Explicit routing is needed to find a path between LANs. A WAN network with intermediate extension limited to a town or a city is also called a Metropolitan Area Network (MAN).

- **SAN/NAS**

Storage Area Networks (SAN) are normally used for connections to and from file servers. They provide a very high bandwidth and the dominating protocol is Internet SCSI (iSCSI). The usage of SAN is currently growing very fast. Normally the SAN protocols are used on top of TCP/IP. Network Attached Storage (NAS) is a storage device that is connected directly to the network. Some examples on host bus adaptors (HBA) and SAN accelerators will be discussed in chapter 5. More information regarding SAN protocols, devices and applications can be found at [2.7].

2.3.2 Classification according to connectivity

Especially when talking about the Internet, a hierarchy of networks is introduced. Routers under a single technical administration and a single routing policy are interconnected to an Autonomous System (AS) which aggregates a group of

address prefixes. The interconnection of autonomous systems forms the Internet. An autonomous system will belong to the highest (outermost) level of a network, the so-called access network, if customer links enter the Internet via this autonomous system. The first router seen by the customers traffic is the edge router of an Internet Service Provider (ISP) running the autonomous system. The remaining interconnection of autonomous systems without the network edge constitutes the lowest network level, the core or backbone network.

2.3.3 Classification according to data rates

Some common (PHY/MAC-layer) network standards and their performance figures are listed below.

Table 1: Common networks and their performance figures.

Network	Speed
Fast Ethernet	100 Mb/s
GEthernet - GMII interface	1 Gb/s
10 GEthernet - XGMII interface	10 Gb/s
OC-1	52 Mb/s
OC-48	2.5 Gb/s
OC-192	10 Gb/s

The networks listed in table 1 are and will continue to be some of the most common for a number of years. They all have such high throughput that host processors benefit from efficient offloading.

2.4 Network services

When the Internet was relatively small and primarily used by researchers in an open setting, the primary service was a point-to-point connection between two specific hosts allowing users to transfer files and interactively connect to remote hosts. Packets were treated equally and independently forwarded without modification from one specific host to another. While this is still the default behavior for IP packets, the shift in focus from an open research network to a vehicle for commerce, banking, telecommuting, and personal communication along with a tremendous increase in size has led to the development and deployment of new services that require routers to forward and process packets in a variety of ways.

2.4.1 Quality of Service

To classify traffic as real-time, non-real-time, or best effort, the idea of Quality of Service (QoS) was invented. Fundamentally, QoS enables the possibility to provide better service (e.g. bandwidth or latency) to certain customers (flows). It is easy to imagine instances where some packets are more important than others and require priority processing. A common way of generating additional revenue from a service is through service differentiation. This is done either by raising the priority of a flow or by limiting the priority of another flow. Using congestion-management, it is possible to raise the priority of a flow by servicing queues in different ways. The queue management used for congestion avoidance raises priority by dropping lower-priority flows before higher-priority flows. Policing and shaping provide priority to a flow by limiting the throughput of other flows.

By varying the quality of service, Internet service providers (ISPs) can charge customers more to have their packets be given priority potentially increasing the ISPs revenue. This is the reason why the QoS concept has been a huge research area for several years now. However QoS has not been used in many networks so far. The reason for this is the complex administration required for billing which makes it costly.

Regardless, future routers which can

- **perform more complex decisions than simply forwarding packets based only on their destination address**
- **perform more complex packet processing than IPmin, and**
- **perform service differentiation rather than use best-effort packet delivery**

will be able to distinguish themselves in the marketplace.

QoS may be viewed as a reason to give terminals multiple ports and have them perform more router-like operations, e.g. flow-based classification.

2.4.2 Firewalls

These devices are used to block the flow of packets between networks. For example, when placed at the boundary between a company and the rest of the Internet, its job might be to block external access to internal services and to block packets containing sensitive information from reaching destinations outside the company. This may be as simple as rejecting packets originating from a particular host to as complex as rejecting all the packets comprising an electronic mail message containing executable code matching the characteristics of a known computer virus. One of the benefits of quickly filtering out undesired packets is that it helps to reduce the effects of a denial-of-service (DoS) attack where devices under attack waste so much time handling spurious packets that legitimate packets are not handled in a timely manner. Since routers must examine each packet to deter-

mine its disposition and are already required to discard malformed packets, they can be easily extended to support a firewall service by including patterns in the route table for packets that should be explicitly blocked. This however requires a huge processing load. Most terminals support some sort of firewalling.

2.4.3 Network address translation (NAT)

Because there is a limited number of IP addresses available, NAT was developed to allow multiple devices on an internal, hidden network to appear as a single host (i.e. using a single IP address) on the Internet. Routers supporting NAT actively modify the source address of outbound packets and the destination address of inbound packet to maintain the illusion that the hidden hosts are directly connected to the Internet.

2.4.4 Tunneling

Prior to widely available Internet connections, larger companies wishing to connect multiple locations were forced to create their own private networks using, for example, leased telephone lines. With the advent of low-cost, wide-spread Internet service, companies are now able to create Virtual Private Networks (VPNs) that tie a subset of Internet hosts (those belonging to the company) together using encrypted tunnels. Routers implementing an overlay network tunnel packets through the Internet by treating the entire packet (including its header) as data, encrypting the data, and encapsulating the encrypted data into a new, larger packet. This outer packet is sent over the Internet in the regular way to another host in the VPN. Upon arrival, the packet is unwrapped, decrypted, and delivered to its destination. Note that the destination may be the host itself or a locally connected machine. In some cases, the packet is forwarded to another node in the VPN by re-encrypting and re-encapsulating the packet. A VPN is an example of an overlay network. While end-hosts often perform the tunneling, routers are well-placed to make optimizations based on their innate knowledge of the underlying network topology.

2.4.5 Mixed traffic

Today it becomes more and more common to use the same network for booth data transfer and voice or video. One reason is that network standards and quality have reached the level where it is economically beneficial to share the network resources. The main applications, except normal data traffic, for mixed traffic are:

- **Voice over ATM**
- **Voice over Frame Relay**
- **Voice over IP**

2.4.6 Application switching

Today organizations introduce multimedia and mission critical applications that require low latency with guaranteed network bandwidth to support response time and end-to-end transport constraints. Further, demands for increased QoS requires awareness of the content of the IP packets that are being switched in a router. Exploiting the information already contained within the IP packets traversing the network provides the necessary tools to differentiate the service for different users. This is commonly denoted as application switching.

Layer 4-7 information is crucial when balancing traffic loads across servers providing similar applications. Switches capable of supporting application switching keep track of each established session to individual servers. With a load balancing service, a collection of physical servers with different IP addresses, supporting the same application service (e.g. a web server farm) can be defined as a single virtual server. This virtual server is a "logical server" with a single IP address. Instead of communicating directly with the real IP addresses of the physical servers, users direct traffic to the virtual server address.

Application switching can be used to route packets to special servers. E.g. ftp-packets should go to the file server while jpg packets should go to the image server. Further, routers supporting encryption standards, e.g. IPSec etc., can investigate a packet to see if the application should be encrypted or not (e.g. https or http type of packets).

2.4.7 Fragmentation and reassembly

In TCP/IP, fragmentation refers to the process of breaking an arbitrary size packet into the smallest maximum size Packet Data Unit (PDU) supported by any of the underlying networks. This may be necessary because of restrictions in the communication channel or to reduce latency. The pieces are joined back together in the right order at the receiver terminal (reassembly). Reassembly can not take place in intermediate routers since different fragments (packets) may use different paths (sets of routers) while traversing the network. Fragmentation may also be performed by a router when routing a packet to a network with a smaller maximum packet size.

The term segmentation is used in ATM. In TCP/IP it is called fragmentation and is performed at the IP layer before the fragments are passed to the transport layer.

For a receiving terminal the reassembly task demands large memory bandwidth and size. Hence, a number of Network Processors for terminals have employed special-purpose Segmentation and Reassembly (SAR) engines to manage these performance critical memory operations.

2.5 Minimal Packet Processing

All forwarded packets undergo some amount of transformation when they pass through a router. We refer to the minimum amount of processing (the most common case) as IPmin.

Referring to the IP header, the minimum amount of processing is to decrement the Time-To-Live (TTL) field and recompute the header checksum. Most packets only require IPmin. However, if the packet header has options, is destined for the router itself (e.g. a routing protocol packet), or is otherwise exceptional, the router performs more processing on that packet than just IPmin. Because an important comparison metric for router manufacturers is the raw forwarding rate of packets only requiring IPmin processing (traditionally, the vast majority of packets), the path that these packets take through the router is often highly optimized. Such an optimized forwarding path is referred to as the fast path. Non-optimized forwarding paths are referred to as slow paths.

Note that terminals have to process (terminate) the entire protocol stack. Hence, there is no, or at least a totally different type of IPmin processing in terminals.

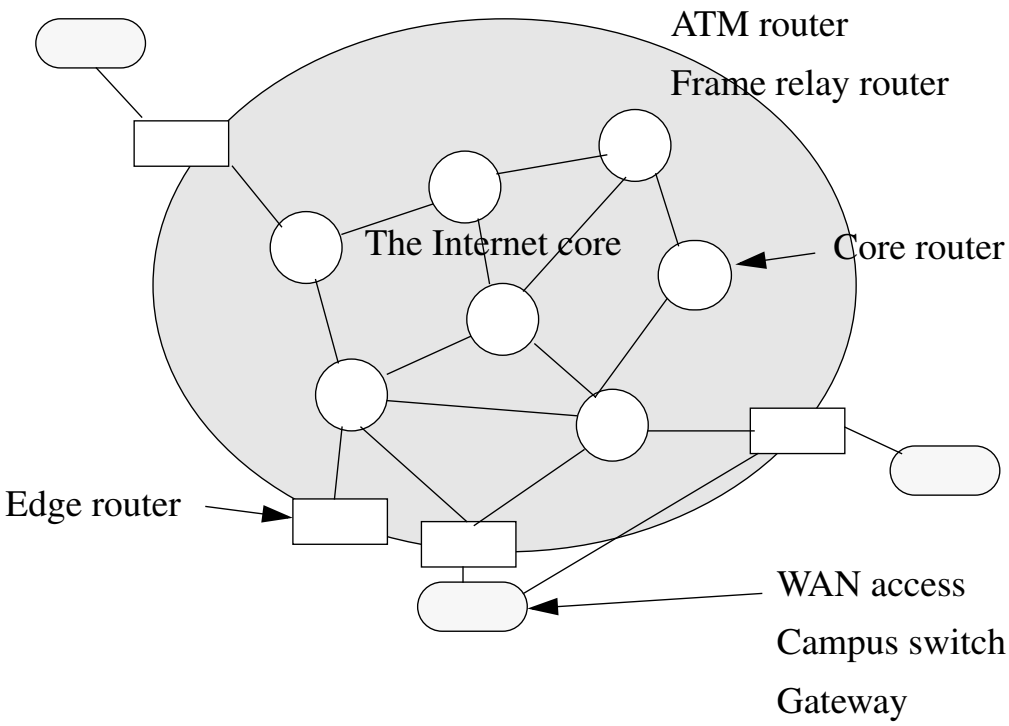

Figure 2.3: The network infrastructure consists of core routers, edge routers, and terminals. Core routers reside in the backbone network while edge routers provide access to the Internets pheripheral networks.

2.6 Network entities

In the Internet, which consists of a number of heterogeneous networks, there exist many different types of entities. The main groups are routers and terminals (i.e. hosts). Routers connecting other routers with high bandwidth in fiber based backbone networks are normally denoted as core-routers. Core-routers normally transport large amounts of data between networks in a packet-based fashion. Inside each network, at the edge of the internet, edge-routers provide an access point to a network. Edge routers transport data between one or more LANs and the ATM backbone network e.g. WAN. Edge routers does not gather routing information, they use it. Core-routers normally provides hundreds of Gbps in throughput while edge-routers can manage tens of Gbits/s. The network infrastructure is illustrated in figure 2.3.

Network Terminals (NT) can be of many kinds, for example desktop computers, laptop computers, IP phones, Internet radios, video conference systems, file servers, backup devices, and network attached printers. There also exist many wireless applications where the NTs are used, e.g. a mobile phone or a PDA, connected to a WAN.

2.6.1 Routers

The Internet is a packet-based, store-and-forward, computer-to-computer communication network based on the IP protocol. Rather than creating a dedicated channel from source to destination, transmitted data is transferred from device to device. At each device, the data is received, stored, and then forwarded to the closest device along a path toward the destination using a technique known as store-and-forward. To improve the efficiency of the process, transmitted data is broken into fixed-size chunks denoted packets which are sent from device to device in a pipelined fashion.

To support packet routing, IP specifies that every network interface have a distinct 32-bit identifier known as its IP address (128 bits for IP version 6). Devices with more than one network interface have more than one address. To send data across the Internet, a host divides the data into blocks, which form the data portion of the packets. For each block, the host attaches routing information in the form of an IP header to form an IP packet. Each packet is then encapsulated into a link-layer packet appropriate for the network interface (e.g. an Ethernet frame). Finally, the link-layer packet is sent to the router on the local network.

The two primary routing functions are:

- **Forward Packets:** This is the obvious job of a router, moving packets arriving on one port and sending them out on another port so that the packets eventually reach their destination. To forward an incoming packet, the router must examine the header. By using the destination address as an index into its routing table, the router can then send the packet to the appropriate next-hop device (either another router or the destination host) on the network connected to the appropriate output port. Forwarding packets is considered part of the data plane of a router.
- **Maintain Routing Table:** Because hosts and routers can join or leave the Internet at any time, routers must become aware of changes that effect their local routing tables. To support this dynamic aspect of the Internet, routers implement routing protocols to share connectivity information and maintain routing tables. The two primary routing algorithms currently in use are OSPF (Open Shortest Path First) and BGP (Border Gateway Protocol). While the details and application of these protocols are beyond the scope of this thesis, the important point is that these protocols are sufficiently complex that reasonable implementations require the capabilities of the general purpose control processor. Maintaining the routing table is considered part of the control plane of a router.

One distinction between the data and control planes (a.k.a. the fast path and slow path respectively) is that the former must process packets at line speed. The requirement that the data plane runs at line speed is based on the need to receive

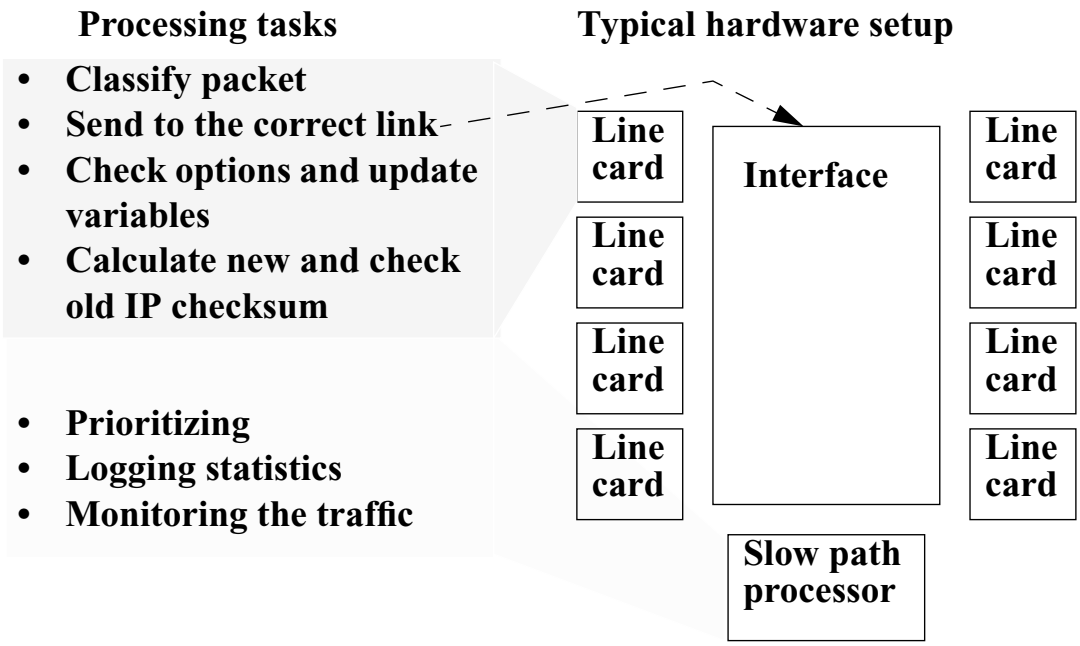

Figure 2.4: Example of processing tasks and allocation in a traditional type of router.

and classify packets as fast as they arrive, so as to avoid being forced to drop packets.

For hardware-based routers, enabling higher QoS means that more packets are being processed by the slow path since there are more exceptional packets. The reason packets move to the slow path is simply that the fast path does not have the capability (flexibility) to process these exceptional packets due to the lack of programmability. For both software-based and hardware-based routers, these exceptional packets require more cycles per packet as the number and complexity of the services increases.

Even though router manufacturers of today tend to include more and more intelligence in their devices, they normally do not handle protocols above the network layer in the protocol stack. The main reason for this is that a transport layer protocol such as TCP might have its payloads being segmented into many packets, which then are transmitted through separate network paths. Hence, it is normally only in the terminals the protocols in the transport layer and above will be processed.

Layer 2 routers are commonly denoted as switches. Switches makes all routing decisions based on layer 2 type of protocols (Ethernet header information). Layer 3 routers are the most common type. It makes all routing based on the layer 3 protocol header. The predominating protocol is IP.

There also exist upper layer routers (e.g. web-switches, TCP-routers etc.). Normally the processing complexity increases rapidly with the number of protocol

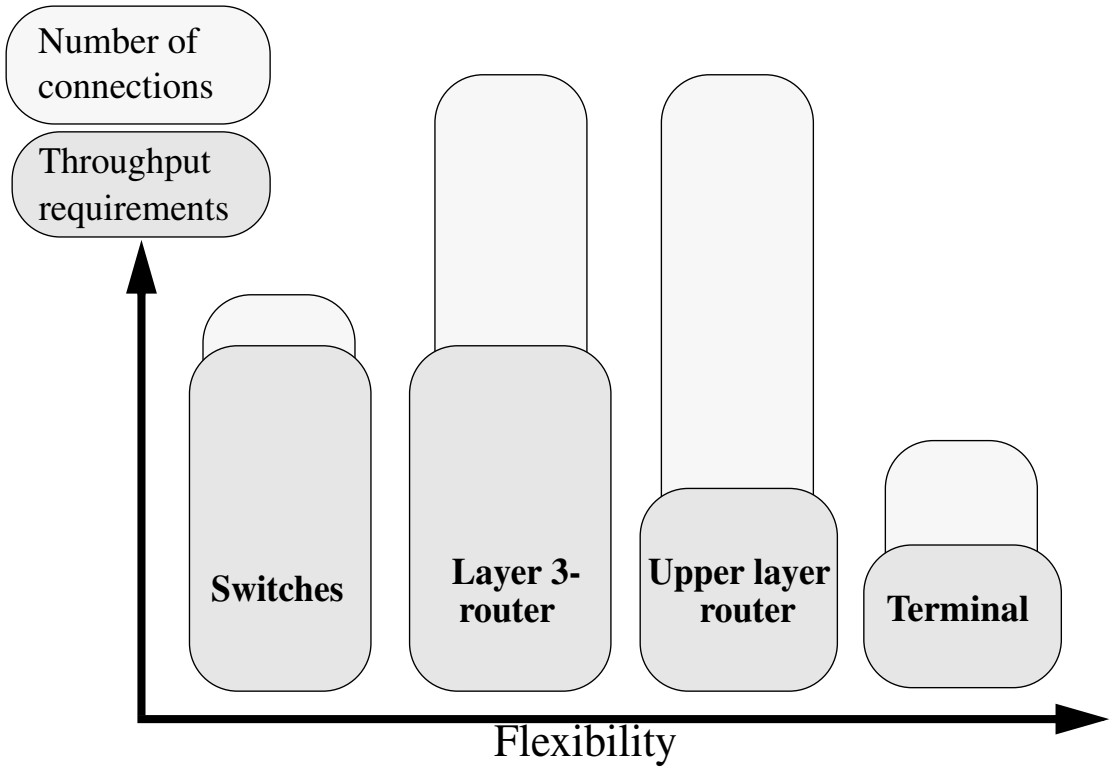

Figure 2.5: Requirements on bandwidth, flexibility and supported number of simultaneous connections for different network components. Note that there are large differences within each group. Especially the throughput requirements are application driven.

layers that must be processed in order to access the routing information. Hence, upper layer routers are not so common today but the use is steadily increasing.

Normally a router includes 3 basic components. They are line cards, interfacing backplane or switch, and a slow path processor. Some examples on processing tasks and a typical router hardware architecture is illustrated by figure 2.4.

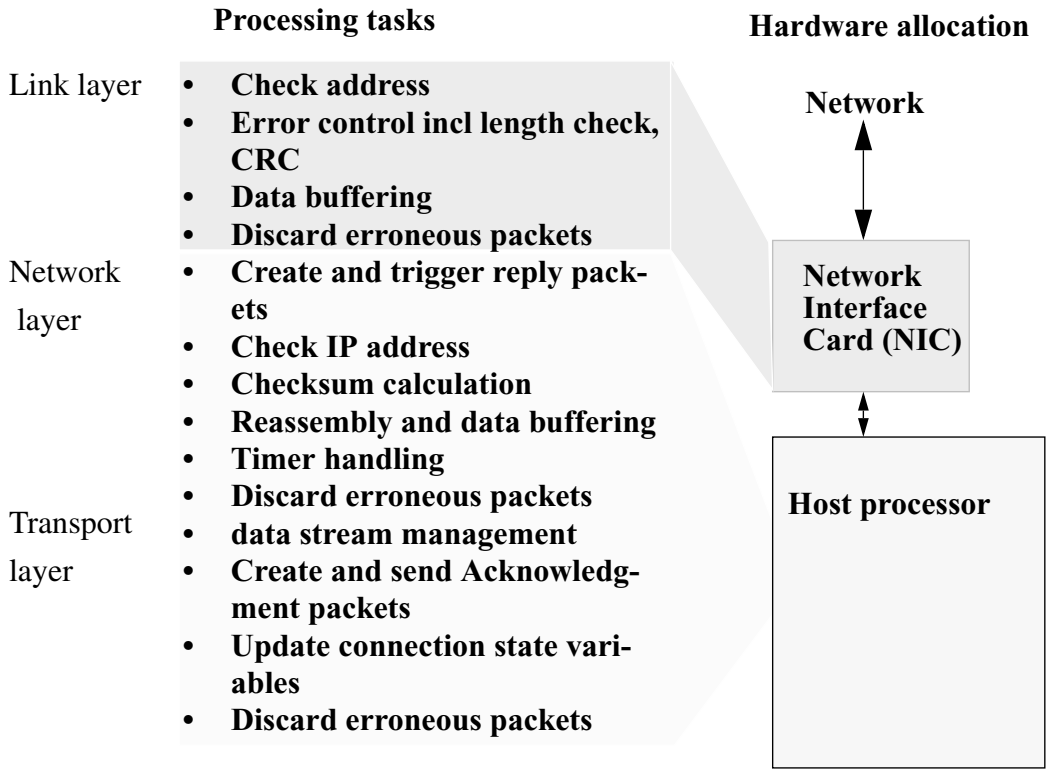

Figure 2.6: Examples of processing tasks and hardware allocation in a traditional type of desktop NT. As illustrated above the host processor has to do a lot of the processing while the NIC only process the link layer.

2.6.2 Network Terminals

Due to the diversity of the applications the requirements on the network interfaces are very different. The only common characteristics of a NT is that it terminates the packets.

The protocol processing that is required in terminals is not the same as in routers. In terminals, packets are generated and consumed (delivered to an application). In general it could be said that routers only use layers 1-3, but terminals must process layer 4 as well. There are two main differences between the protocol processing in a router and a terminal. First of all the number of connections are fewer and more stable in a terminal, and secondly terminals have to do more processing tasks since also the transport layer has to be processed. The second difference complicates the processing in a terminal while the first one relaxes it compared to a router. These differences indicate that the protocol processing in a terminal should be performed using a terminal specific hardware/software platform. The difference between requirements on network core components and a terminals is illustrated by figure 2.5.

As an example on the type of processing going on in a desktop PC we can consider the TCP/IP protocol stack processing introduced in section 2.2.2. An exam-

ple of some processing tasks in such a NT and the simplified hardware allocation, is illustrated by figure 2.6. Traditionally NICs consist of Physical layer ASICs and microprocessors while the processing in the host is performed in software.

Reference

[2.1] A. S. Tannenbaum, "*Computer Networks*", 3nd Edition, Prentice Hall PRT, ISBN 0-13-349945-6, 1996
[2.2] J. Kadambi et al, "*Gigabit Ethernet*", Prentice Hall PRT, ISBN 0-13-913286-4, 1998
[2.3] W. R. Stevens, "*TCP/IP Illustrated, Volume 1 The Protocols*", Addison-Wesley, 1994
[2.4] G. R. Wright, W. R. Stevens, "*TCP/IP Illustrated, Volume 2 The Implementation*", Addison-Wesley, 1994
[2.5] Technical Specification of BRAN and Hiperlan-2. Common part.", ETSI TS 101 493 - 1, V1.1.1, 2000
[2.6] "Technical Specification of BRAN and Hiperlan-2. Ethernet Service Specific Convergence Sublayer.", ETSI TS 101 493 - 2, V1.1.1, 2000
[2.7] Storage Networking Industry Association, on the internet, http://www.snia.org/home

Part 2

Design exploration

“It is not necessary to change. Survival is not mandatory.”
--W. Edwards Denning

3

Processor Architecture

There are currently three main implementation alternatives for protocol processing.

- The first is to map the processing algorithms directly to an ASIC (Application Specific Integrated Circuit). This implementation can support most requirements, but suffers from no (or almost no) flexibility. The flexibility is needed for two purposes. First, one single hardware design can be used for many different product groups if enough flexibility is available. Second, the hardware does not need to be redesigned when new protocols must be implemented.
- The second implementation alternative is to make use of regular general-purpose processors (of RISC type). These provide unlimited flexibility, but suffer from high silicon area cost, high power consumption and cannot support low latency or high throughput implementations.
- The third implementation alternative is specialized processors. The processor can be specialized by use of a dedicated instruction set (ASIP: Application Specific Instruction Set processor), or by adding accelerator units for challenging tasks. The foundation for the specialization of the processors is the analysis of the processing tasks of a number of protocols. The analysis includes instruction profiling, computation load distribution among the tasks and memory access pattern statistics.

Among the three implementation alternatives, only specialized processors can support both the flexibility and high efficiency needed in today's network processing equipment. This chapter will explain why.

3.1 Parallel processing

Normally parallelism is addressed as the key-contributor for increased performance in the area of computer architecture. There are several ways of exploiting parallelism in computers.

3.1.1 Exploiting parallelism using pipelines

By dividing the execution of tasks into smaller parts, each of the parts takes a smaller amount of time to perform. If parts of different tasks then are executed in parallel using pipeline scheduling, the performance can be dramatically improved. I.e. since only a part of the instruction has to be executed each clock cycle, the clock frequency can be increased compared to that of a non-pipelined processor. Pipelining is a technique used in almost all modern processors including processors dedicated for usage in computer networks. The number of pipeline stages is restricted by complexity and branchpenalty makes it hard, but up to 10-15 is possible. Examples on pipeline stages (instruction parts) are instruction fetch, decode, operand access, datapath execution, and write result.

The price to pay is the intermediate storage of data in pipeline registers and penalties for misspredicted branches causing stalls. Branchpenalty lowers the Instructions Per Clock cycle (IPC).

3.1.2 Instruction Level Parallelism

The main idea behind parallel execution is to extract parallelism from the inherently sequential nature of programs. A program can be thought of as a series of instructions executed in a logical order over time. Where later instructions do not depend on earlier ones, there is Instruction Level Parallelism (ILP). The goal in ILP is to find instructions that neither are data nor control dependent, so that they can be executed concurrently to achieve higher instruction throughput. The standard way to do this is to look for independent instructions within a window of instructions, usually of a fixed size. To find even more parallelism, a natural extension is to make the instruction window bigger. Examining more instructions together increases the chance of finding independent instructions, although there are diminishing returns. However, the logic necessary to check for independence becomes forbiddingly complex and makes large instruction windows impractical.

One architecture that supports for ILP is the Superscalar processor. In a multiscalar model, a program is partitioned into tasks, which are sets of basic program blocks. A task is a portion of the static instruction sequence whose execution corresponds to a contiguous region of the dynamic instruction sequence. Tasks are not necessarily control independent whereas threads are. A task is executed only if the dynamic instruction stream through the program includes that task; individual threads are executed unconditionally. The multiscalar processor executes multiple

tasks in parallel without checking for inter-task control dependencies. Individual tasks are assumed to be control independent and executed accordingly, then checked later and discarded if found to be dependent.

Another architecture that supports for ILP is the Very Long Instruction Word (VLIW) processor. It basically uses a complex instruction to control many functional units. These functional units then process the data in parallel.

The main difference between a VLIW and a Superscalar approach is that the first scedules the instructions during compilation while the later handles the bundling of instructions during runtime using special purpose hardware assist. Hence, the superscalar processor allows for out-of-order execution.

3.1.3 Data Level Parallelism

A Single Instruction Multiple Data (SIMD) architecture exploits parallelism by providing data to many execution units, all processing the same task. If the tasks are regular and data-intensive, this type of processors gives a low overhead due to the single control-path (i.e. single program counter, program memory, and instruction decoding hardware).

It is almost always possible to increase the width of the processor datapath to increase the performance. If, for example, the width of the datapath increases from 32 to 64 bit, the clock frequency can be reduced by a factor of two if it is possible to use the same number of instructions. This however increases the area and complexity of the data path and if the instructions and operands are not suitable for 64-bit execution it may not increase the performance. In protocol processing there are many operations that are not suitable for 64-bit datapaths. Instead bit and byte-based instructions are suitable.

3.1.4 Exploiting parallelism using datapath acceleration

In many application domains, the need for high throughput has been addressed using datapath acceleration. E.g. DSP processors make use of multiple Multiply-and-ACcumulate (MAC) units to manage high-throughput filtering. Another example is processors containing one datapath for floating-point instructions and one for fixed-point instructions. Actually, most modern processors (general purpose or not) uses datapath acceleration. In a domain-specific system it is possible to analyze the workload and identify kernel operations. These operations can then be accelerated using datapath extensions or standalone accelerators. The main difference is the use of centralized and distributed control respectively. By adjusting the datapath so that individual bits and/or bytes can be addressed and operated on, a processor can increase its protocol processing performance significantly.

3.1.5 Exploiting parallelism using accelerators

By using accelerators, i.e. dedicated hardware, the processing performance can be substantially increased. The accelerators are normally self-contained and do not require a lot of control signaling. Hence the overall control overhead can be reduced which reduces the memory cost and increases the efficiency. The accelerator approach is suitable if there exist processing tasks that are common and non-suitable for general purpose hardware processing. If the tasks are not very common the accelerator will just remain in idle mode. In protocol processing, search engines (i.e. CAM) are often used as accelerating coprocessors. Coprocessors used in protocol processors today, are either of look-aside type or flow-through type. Look-aside type of coprocessors are invoked by the main processor when needed. I.e. they function as procedure calls. Flow-through coprocessors operate on data-streams in a pipeline fashion. The main advantage with flow-through type of coprocessors is that they minimize the control information the main (host) processor needs to send to control coprocessor operations. The main drawbacks with flow-through coprocessors are the requirement on wire-speed processing and lack of flexibility due to the high degree of integration.

3.1.6 Exploiting parallelism using multiprocessor approaches

It is of course possible to exploit parallelism by the use of multiple processors. The speculative mechanism normally works as follows. Speculation is done on either the thread level (multithreading) or the loop level. In thread level speculation, a processor starts executing a thread. When a subroutine thread is encountered, one processor continues executing the subroutine thread while the next processor speculatively executes the code after the subroutine thread in parallel. In loop level speculation, subsequent iterations of a loop are executed in parallel by several processors. If a speculation violation is detected, the erroneous thread will be killed. Normally a multi-processor approach is not very efficient if the application does not imply multi-threading by its nature. The problem is that there will be too much shared data and inter-processor communication.

3.1.7 Exploiting parallelism using software techniques

Loop unrolling and software pipelining are common techniques used to improve the processing performance of a computer. Loop unrolling is done at compile time and it means that branch decisions can be avoided if the number of iterations are known at compile time. Software pipelining means that instructions from different instruction streams are interleaved. Since instructions that belong to different instruction streams normally are not dependent, they can be executed without hazards. Dependent instructions must be separated by a distance (in clock cycles) equal to the length of the pipeline latency for the source instruction in order to

avoid pipeline stalls. Software pipelining might be suitable for multi-port processing in a single protocol processor, since the instructions are normally unrelated.

3.2 Managing hazards

There are two types of hazards that may occur: data and control hazards. Data hazards occur when two data instructions are dependent, and close enough to overlap due to pipelineing. This means that program order must be conserved between data dependent instructions. There are three types of data hazards: write-after-read (WAR), read-after-write (RAW), and write-after-write (WAW).

The other type of possible hazard is control hazard. They occur because program flow selecting instructions (conditional branches) are dependent on the outcome of earlier instructions. This means that the correct instruction cannot be fetched and issued before the earlier instruction has been executed. This may force the processor to stall its processing until the result is available to the earlier instruction.

To reduce the number of stalls (branch penalty), branch prediction can be used to predict which instruction to fetch and issue. If the guess is wrong the instruction pipeline must be flushed and the correct instruction fetched. This introduces an uncertainty in processing latency, which is bad for wire-speed processing. Some computers also use speculative execution, i.e. they not only fetch and issue the predicted outcome of the branch, they also execute that predicted instruction.

Delayed execution is another approach that tries to reduce pipeline penalty. One example is to always process the two instructions following a branch instruction. This requires these two instructions not to be dependent on the outcome of the branch decision. If the compiler (or assembly programmer) cannot find this type of instructions to insert, it must insert NOP instructions after each conditional branch instead.

3.3 Optimization

By simply noting that some intrusions (tasks) are more common than others when a processor only is used for a specific application area (e.g. protocol processing), it is possible to improve the performance by optimizing the architecture for this kind of instructions. There are several ways of doing this.

One way is to optimize the Instruction Set Architecture (ISA). This include optimization of the instruction size, memory addressing schemes and creation of more complex application-specific instructions. A processor with an application-specific ISA is normally denoted as an ASIP. ASIPs normally have an improved performance and use less program memory than more general architectures. The reason is that complex (and common) tasks normally requiring several instructions can be executed using one instruction during only one clock cycle. One example is the

MAC instruction, which is normally used in filter applications. Normally the ISA and the hardware architecture have to be co-optimized. While optimizing the ISA it is also very important to co-optimize the compiler so that the improved ISA can be efficiently used.

The second stage of optimization is to optimize the micro architecture in order to be able to raise the number of IPC. This includes optimization of the critical path to be able to run at a higher clock frequency, acceleration, memory organization, branch prediction strategy etc. This is normally done after a specific way of exploiting parallelism has been chosen and the ISA and data width have been decided.

3.4 Implementation Alternatives

3.4.1 Controllability

Hardware components in a network interface can have different kinds of control. The three main alternatives are:

- **Fixed function.** E.g. ASIC with no flexibility.
- **Configurable.** The function of the data path can be changed but it cannot be changed every clock cycle. The controlability and flexibility can be high (e.g. in an FPGA).
- **Programmable.** The function of the data path can be changed in every clock cycle.

In a network processor the need for configurability and programmability can be reduced by the use of many different fixed function blocks, each capable of processing a small part of the tasks. Different blocks are then used only for specific tasks and do not need any configuration. Many protocols at the higher layers in the protocol stack have very high requirements on flexibility. The amount of flexibility and the type of control a hardware platform needs is an important design parameter.

3.4.2 Integration

Processors and memories in a network processor ASIC chip are sometimes integrated in the same silicon chip, which means almost all the processing work can be done internally without having to wait for slower external memory access. On-chip memory is a major advantage since many protocols require extensive memory access when being processed.

An ASIC can have multiple processors integrated into the chip to handle heavy workloads. This means that a single chip may be simultaneously working on many different processes for many independent protocol sessions. Parallel processors

within an ASIC (SoC) provide enormous performance advantages beyond those achievable with single-processor board-level products.

One particularly noteworthy example of parallel processing in a network processor ASIC is the implementation of timers. TCP processing depends on session timers to manage flow control and identify transmission errors. At gigabit and higher transmission rates, the accuracy of flow control and error detection becomes increasingly important to the health of the network. Board-level solutions have to implement TCP timers in software or use one or two general purpose hardware timers provided in a general processor core. This means that the events and timers are processed sequentially by a single CPU. Obviously, multiple hardware-based timers running in a custom ASIC add a great deal of efficiency as well as accuracy, resulting in the most consistent and predictable network operations.

Beyond accessing memory in silicon, ASICs also facilitate the use of advanced memory technologies that have been developed for high speed networking applications. Specifically, where TCP processing is concerned, a special memory technology for high throughput networking called CAM (Content Addressable Memory) can be used very effeciently. While CAM can be implemented in both board-level as well as ASIC solutions, it is less expensive and more efficiently utilized when implemented in an ASIC. In general, the content-based indexing of CAM virtually guarantees that each connection table lookup only needs a single memory operation. With a high volume of lookup operations occurring every second on a Gigabit Ethernet link, it is easy to see why an ASIC approach with integrated CAM is so efficient.

3.4.3 FPGA implementation

To implement parts or the whole of a network processor in a Field Programmable Gate Array (FPGA) would give a very high degree of flexibility due to the configurability of the FPGA. Since the cost of FPGAs today is acceptable for low volume products, it would be a cost effective solution even if the number of units sold is small. There are however four major drawbacks with FPGA implementation.

First of all the throughput of an ASIC implementation is significantly higher than the FPGA solution can manage. Secondly the power consumption is much higher in the FPGA. The third drawback with a standard FPGA is the limitations in size and complexity of the design that can be implemented on one FPGA. The memory communication and use of distributed embedded memories are also benefiting from an ASIC chip implementation.

3.4.4 Memories

The memory structure is very important for protocol processing because the processing is all about moving data. A specialized processor can have a dedicated

memory structure with adequate addressing modes, which can improve the efficiency dramatically.

A scalable multi-processor system using a shared memory hierarchy will encounter coherence problems. I.e. how and when should local caches be updated. If shared data is stored in multiple caches some sort of cache coherence technique must be used to avoid conflicts. The alternative is to only store private (processor-specific) data in local caches. This issue must be considered in multi-processor router systems using multiple memories for storage of routing tables.

Using on- or off-chip memories can be a very important for the performance of a protocol processor. Off-chip memories normally have lower bandwidth due to pin limitations. They always have higher latency than on-chip alternatives. The problem with on-chip memories is that the chip area is too small to accommodate big enough memories. Off-chip memories also benefits from use of special manufacturing processes dedicated for memories.

Since the demands on low latency are so high in protocol processing equipment new types of memories have recently emerged to address this problem. These reduced latency memories are used primarily for classification. In the future new memory types may emerge to address specific protocol processing problems but there is no reason to believe that this alone will solve all problems.

3.5 Summary

When designing high speed Programmable Network Interfaces (PNI) for terminals there are a number of challenges that the designer has to overcome. Some of these challenges are common to all micro-electronic designs, e.g.:

- Data transfer to/from external memories
- Pin limitation
- Packaging
- Verification

Others are specifically important in protocol processor designs, e.g.:

- Line-rate processing (fast path processing performance)
- Link-rate processing (slow path processing performance)
- Device integration (accelerators, memories, ASIC:s)
- Shared resources management (e.g. data and program memories)

To overcome these challenges three main approaches exist today. Their common goal is to provide sufficient processing power so that the host is efficiently offloaded. The three main alternatives are:

- **Application Specific Logic**

  Special Instruction Set

  On- or Off-chip accelerators

- **Advanced Processor Architectures**

  Data level parallelism

  Instruction Level Parallelism

- **Multi processor solutions**

  Task level parallelism and/or pipelined multi-processor systems

Combinations of these design approaches are also possible. Before selecting design methodology and architecture a number of design considerations and performance requirements have to be examined. Depending on application, cost sensitivity, and other factors, the optimal solution may vary. Using information about processing requirements it is always possible to improve the architecture using different types of optimization. This optimization process includes making the ISA and micro architecture dedicated for specific aplication. For further information on the design challenges and considerations when designing network processors, I recommend the book referenced as [3.1]. In the Ph.D. thesis [3.2] a deep discussion on memory architectures can be found. [3.3] may also be useful.

Reference

[3.1] Crowley, Patrick, et al, “Network Processor Design”, first edition, Morgan Kaufman Publishers, ISBN: 1-55860-875-3

[3.2] Mattias Gries, “*Algorithm-Architecture Trade-offs in Network Processor Design*”, Ph.D. thesis, Diss. ETH No. 14191, Swiss Federal Institute of Technology Zurich, 2001

[3.3] D. E. Comer, “Network System Design using Network Processors”, Pearson Education Inc., ISBN: 0-13-141792-4

4

Protocol Processing

4.1 Protocol processing trends

In the semiconductor industry it is a well known fact that the device production scales according to Moores law illustrated by table 4.1. The scaling factor S has been 0.7 since 1974 which means that the feature size becomes half as big every second year. Further we can see that the clock frequency scales almost with S and that the number of transistors / chip will scale as S^2. Historically the design community has been able to take advantage of this development to improve the processing bandwidth according to Moore's law by using improved design methodologies and architectures. As shown in the roadmap there will however be difficult to fill the chips with useful content in the future. To deal with this problem, re-use methodologies are addressed as the key issues for success. Together with the cost issue this means that future platforms must provide flexibility enough to survive over several product generations. Making the situation even worse, today a new problem has emerged when it comes to communication processing.

Networking technologies, have historically increased data rates in 10 times increments according to the Fibre Channel Industry Association [4.2]. Gigabit Ethernet (GE) today and 10 Gigabit Ethernet (10 GigE) tomorrow, together with high-speed back-bone networks, provides the network bandwidth overhead to accommodate the rapid growth in organizations and traffic today [4.3]. Further, more and more services are requested to be provided to the network. This makes

the processing more complex and increases flexibility demands. It becomes harder and harder to improve the devices so that they provide the speed and functionality specified by the network standards.

Table 4.1: Projections of the ITRS Semiconductor Roadmap [4.1]

	2001	2003	2005	2007	2010	2016
Feature size nm	90	65	45	35	25	13
On-chip clock GHz	1.68	3.09	5.17	6.74	11.5	28.75
IO Speed GHz	1.68	3.09	5.17	6.74	11.5	28.75
# signal pins (ASIC)	1500	1700	2000	2200	2400	3000
total # pins micro processor	480 - 1200	500 - 1452	550 - 1760	600 - 2140	780 - 2782	1318 - 4702
Functions / chip	276	439	697	1106	2212	8848

Looking at the current growth rate of the performance of networking, computing, and volatile storage technology, a diverging development can be recognized.

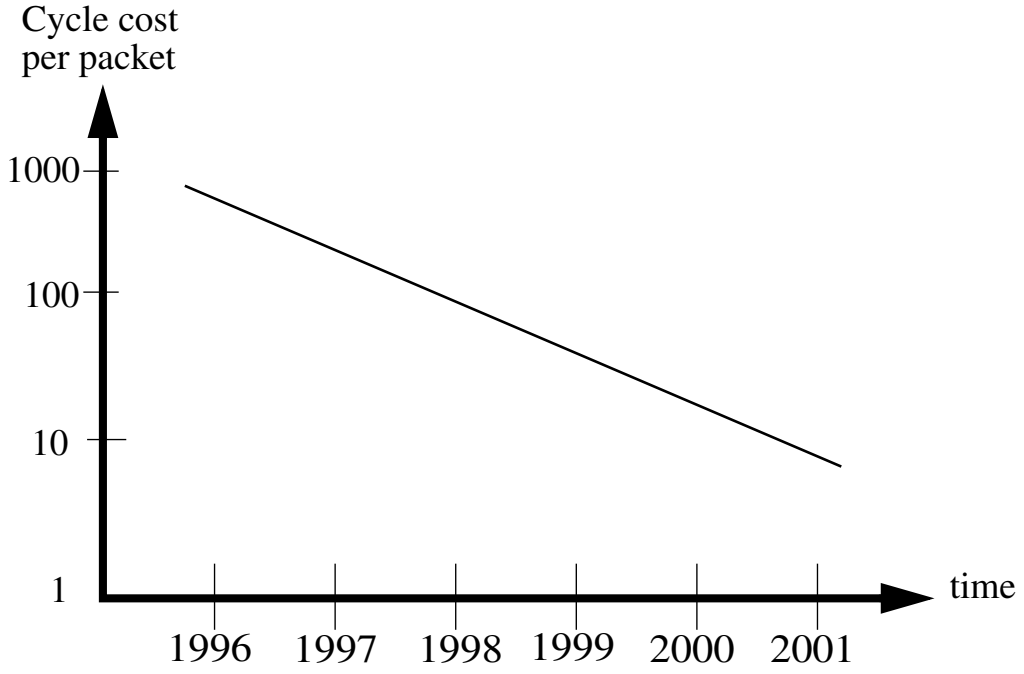

Figure 4.1: Average number of clock cycles available for the processing of a min-sized packet on a state-of-the-art general-purpose CPU at state-of-the-art link speed.

The random access time of dynamic RAMs only halves approximately every ten years [4.4] and [4.5]. Contrary to that, the computing performance of CPUs doubles every 18 to 24 months [4.6], [4.7], and [4.8]. Moreover, although the maximum link bandwidth used in the Internet increases at almost the same speed as computing performance, the volume of Internet traffic currently doubles every six months [4.3]. Therefore, protocol processing tasks will no longer be performed by general computing resources but must be accelerated by domain-specific protocol processors.

Table 4.2: Processing performance growth

	growth per 18 months
DRAM access time	1.1
Moores law	2
CPU Processors performance	~1.75
User traffic	~3
Router capacity	2.2
Line capacity	~6

The figures in table 4.2 can be compiled in to a figure describing the problem in terms of number of clock cycles available per min-sized packet (introduced in the first chapter). This is depicted by figure 4.1.

There are also terminal specific problems. Historically, I/O data rates increased at approximately the rate of Moore's law, which allowed servers and other types of terminals to maintain I/O processing performance overhead from one product generation to the next. According to table 4.1 this is changing since the network bandwidth increases much faster than the I/O processing performance of general purpose devices. Using traditional design methods, we are already experiencing the I/O processing gap problem, depicted by figure 4.2, in network terminals. The obvious solution to this as well as the other problems described in this section, is to offload the communication processing from the application processing device and instead use dedicated devices.

4.2 Design space exploration for terminals

Today, there are a number of different hardware platforms available for use as protocol processors. In order to investigate the need for, and compare different types of network processing hardware, an exploration and a classification of the different solutions is useful.

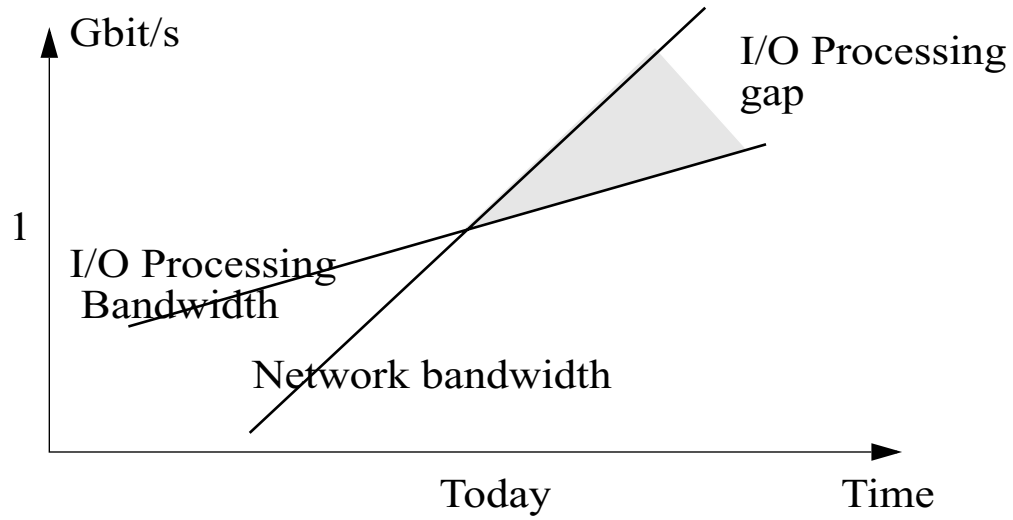

Figure 4.2: The I/O processing gap has started to become a problem using traditional CPU architectures. The reason is that while the I/O bandwidth in network terminals approximately follows Moores law (1.5-2X) the Network bandwidth has a 10X improvement for each generation.

4.2.1 Inter- or intra-layer processing

Intralayer processing means that each protocol layer is processed separately according to figure 4.3. This way of processing and conceptual thinking is a result of the invention of computer networks and protocol stacks more than 30 years ago. In the seventies, communication was considered to be a precious resource while the processor had infinite processing power. Today, the opposite is true.

Intralayer processing gives a processing overhead since a lot of intermediate results and data transports must be performed. However, the well established protocol standards support verification when intralayer processing devices are designed. There is also a need to support all the peer services stipulated by the different layer standards. This is the only reason why intralayer processing should be considered.

The main advantage with interlayer processing is the reduced amount of data-transportation and -processing due to the reduced need for intermediate results. Another advantage that the interlayer processing gives is that the processing can be divided and then distributed to different computing devices depending on the type of processing rather than layer type. The coarse separation is normally into tasks to be performed in hardware or software. Traditionally the physical layer was implemented in hardware while the rest was processed in software. Today, architectures where parts of all layers are accelerated in hardware emerge.

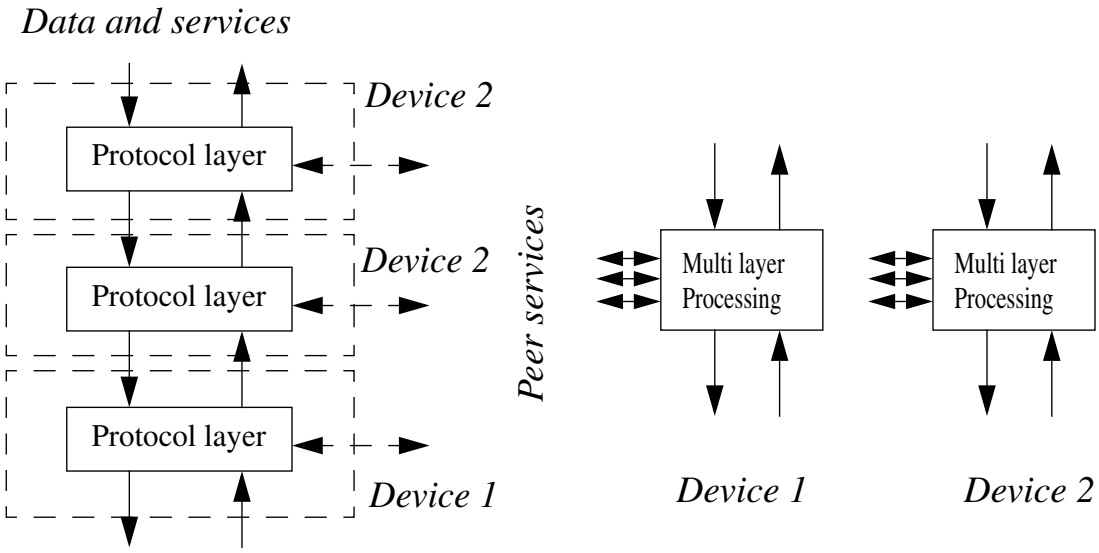

Figure 4.3: Interlayer (to the right) processing means that all or parts of several protocol layers are being processed simultaneously on one device. It does not mean that all processing is done on one piece of hardware or software. Several devices can still share the processing. Intralayer processing means that each protocol is processed sequentially, in order and on one single device. Higher layer protocols will not be processed until the lower has been finished.

To distribute the processing according to processing requirements and type in an interlayered way results in an orthogonal description of the processing tasks, compared to the traditional protocol stack.

4.2.2 Offloading coverage in terminals

Solutions available today from the academic research community and the industry are extremely diverse. Despite this diversity, terminal network processing platforms can be divided into four main groups according to their offloading strategy illustrated by figure 4.4.

Depending on application, throughput requirements, power awareness, and customer cost sensitivity different platforms selects one of the four different offloading strategies while offloading the host processor. The offloading device then typically accelerate and offload protocols at layer 2 up to layer 7. It is not certain that all parts of the protocols are offloaded from the host processor. Hence, the offload efficiency can vary within the four main groups in figure 4.4. Consequently, it is very important to clearly examine both which protocols and how big part of the protocols should be offloaded in order to optimize the entire offloading platform. Protocols not offloaded must of course be processed by the host.

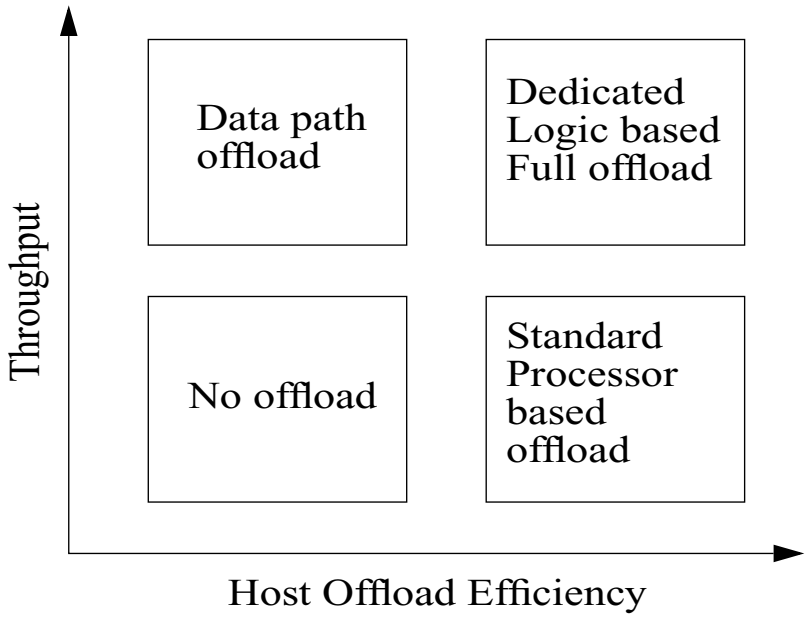

Figure 4.4: Host offloading strategies

4.2.3 Application coverage

The ability to run a certain set of network applications on the host using a PNI-based interface in certain networks is described by the application coverage of the protocol processor. The application coverage defines the workload of a protocol processor. Complexity and hardware cost grows as the application coverage grows. Hence, by modifying (reducing) the application coverage it is possible to achieve higher throughput. In order to compare two protocol processors, the first step must therefore be to carefully define their workload. The main problem is that there are currently no common standardized workload benchmarks available. Normally it is useless to compare the performance figures of two different protocol processor architectures that have different application areas.

The basic requirement for a large application coverage is that the bandwidth is sufficient for processing of the received data. The higher bandwidth the protocol processor can provide the more applications can be supported.

The second requirement is that the protocol processor has enough flexibility so that it can process all different applications (protocols) it is intended for. The application coverage defines what the protocol processor can be used for. Therefore it is the single most important classification parameter.

4.2.4 Host integration

The main purpose of a dedicated terminal PNI is to offload the host in order to accelerate and relaxe the application processing. Hence, it is important to find an efficient interface between the offloading device and the host Operating Systems (OS). Thereby, interrupts and memory conflicts can be minimized and application

processing bandwidth maximized. It is especially important to find a standardized interface if one consider the variety of operating systems, e.g. Linux, Windows etc., and offloading devices. A common interface also enables common programming models which simplifies compiler construction and reduces design time and verification efforts.

Since the year 2000, a number of different TCP Offloading Engines (TOE) have emerged. Most of these process all of the TCP protocol stack in a hardwired fashion which means that some tasks may be duplicated in the host, i.e two TCP/IP protocols are implemented in parallel. The reason for this is the lack of a standard OS interface for terminal hosts. The only successful market share so far has been special purpose servers, e.g. file servers, backup media servers, and complex imaging system servers, where proprietary OS interfaces are possible. In order for the TOE market to grow a common standard interface is needed. Such an interface denoted Chimney architecture has recently (May 2003) been proposed by Microsoft in [4.9]. The Chimney architecture provides a common interface for manufacturers of TOEs, HBA, NICs, and other protocol processing ASICs and it is expected to be of general availability 2005.

The Chimney architecture departure from the prevalent approach of stand-alone self-contained all-inclusive TOEs. Instead it proposes that some (and specify wich) processing tasks should be processed by the host mainly because of security reasons.

According to the Chimney architecture, the data path protocol processing should be offloaded to a TOE while the host is responsible for setting up and tearing down these accelerated TCP connections. Further, control oriented protocols such as DHCP, RIP, and IGMP should be implemented within the host OS (Windows based) TCP/IP stack. Chimney proposes that low-latency retransmission should be handled by the offloading device while reassembly should handled in the application buffer by the host. To perform the reassembly in the host is a significant processing load but it does provide protection against security vulnerabilities according to Microsoft.

The most important benefit with Chimney is that services using underlying TCP/IP (e.g. iSCSI) will automatically utilize acceleration benefits from Chimney-enabled TOEs. It also reduces the development and verification cost of ASIC TOEs which today is roughly 10 times as expensive as Gbit Ethernet ports. The drawback is that some of the ASIC-driven performance offered is lost since parts of the stack will be processed on the hosts general-purpose hardware.

4.3 Terminal specific requirements

Requirements on the protocol processing in a terminal are varying, however low silicon area and low power consumption are always desired. The latency of the processing is more important in streaming systems than for file transfer. For file transfer on the other hand, high throughput is required. Flexibility of the implementation is also desired since protocols evolve, for example IPv6 may take over after IP. A terminal does not have any IPmin processing since all protocol layers must be terminated. A protocol processor that is intended to offload more than Ethernet and IP must therefore have a very high degree of programmability.

4.3.1 Flexibility

A programmable network interface must provide flexibility and adaptability to the changing environment it might operate in. This results in some flexibility requirements that all PNIs have to meet to some extent:

- **Reconfigurable media adaptation.** In order for a PNI to be used in different networks and survive over time it must be capable of adaptation for different medias.
- **Programmable connection policy.** A PNI must support on-line change and control of the traffic flow.
- **Programmable host interface.** The interface between the PNI and the host system must be operating in real time and be highly flexible in order to avoid unnecessary interrupts in the host.
- **Data controlled datapath selection.** The datapath must be configurable or selectable depending on packet header information.

Providing the flexibility bulleted above gives a large protocol coverage but it increases the complexity of the hardware. There is always a trade-off between flexibility and throughput since flexible general purpose hardware never can reach the same throughput as dedicated hardware blocks. Hence, flexibility is an important performance parameter.

4.3.2 Throughput

The need for bandwidth is ever increasing and is not going to disappear. Further it is a fact that an increased bandwidth supports larger application coverage which is very attractive. The conclusion is that throughput is and will continue to be a very important performance parameter when a hardware platform is designed.

4.3.3 Inter-operability

In order to reach an optimal way of integrating the PNI device into the system, the PNI interface and the host operating systems must be co-optimized. To optimize the PNI interface is much easier than the host operating system since it is a

proprietary architecture. I have chosen to call this integration of the PNI with the host operating system, inter operability and it is very important for the overall system performance.

4.3.4 Cost

The cost of the protocol processor chip or board is a very important performance figure. The cost is important for any customer but network terminal users are especially cost sensitive. The cost is always an important part of architectural design trade-off. The cost of a protocol processor chip mostly depends on the package and the number off chips manufactured.

In order to make the package cheap, the area, power dissipation, and number of pins must be minimized. The power dissipation is normally an important optimization criteria in all micro-electronic systems but it is especially important for network terminals. Power figures are of course even more important in portable systems. The number of pins is hard to lower since a protocol processor by nature includes a lot of communication over the chip edge. But the number of accesses to off-chip resources can be optimized. The area is possible to minimize by architectural exploration. Minimizing the design can be either used for cheap packaging and/or to allow for more resources on-chip, e.g. memory.

The number of chips that can be manufactured is strongly connected to the flexibility of the design. A general design can be used for more applications and can also stay longer on the market. Hence, it is important that the design is reusable and flexible enough for a long life-time.

Reference

[4.1] International Technology Roadmap for Semiconductors, on the internet, http://public.itrs.net/

[4.2] FC Magazine, Fibre Channel Industry Association - Europe, on the internet, http://data.fibrechannel-europe.com/magazine/

[4.3] Internet Software Consortium, on the internet, http://www.isc.org

[4.4] W. A. Wulf, and S. A. McKee, "*Hitting the Memory Wall: Implications of the Obvious*", Computer Architecture News, 23(1):20--24, March 1995.

[4.5] Betty Prince, "*High Performance Memories: New Architecture DRAMsand SRAMs - Evolution and Function*", revised ed. John Wiley & SonsLtd., 1999.

[4.6] Lawrence G. Roberts, "*Beyond Moores law: Internet growth trends*", IEEE Computer, 33(1):117119, January 2000.

[4.7] Robert R. Schaller, "*Moores law: past, present and future*", IEEE Spectrum, 34(6):5259, June 1997.

[4.8] Standard Performance Evaluation Corporation. Open Systems Group, "*CPU benchmark suite*", on the internet, http://www.spec.org.

[4.9] Microsoft Corp, "*The future for TCP Offloading engine and RDMA*", on the internet, http://www.microsoft.com/whdc/winhec/pres03.mspx,

5

Network Processor Survey

As stated in earlier chapters specialized network processors have emerged both from industrial and academic initiatives. The processors range from pattern processors and Ethernet classifiers ([5.1]-[5.3]) to systems consisting of general purpose RISC cores ([5.4]-[5.7]) or reconfigurable hardware with additional special-purpose coprocessors designed to assist in common network packet processing tasks ([[5.8]-[5.10]]). This short example clearly shows the diversity of the network processor research community, both in terms of application and architectural approaches. This chapter will investigate a number of different architectures to see if there are any trends and lessons to be learned before focusing on designing programmable network interfaces (PNI) for terminals. But before we examine the new type of router implementation we should go back and look at the traditional implementation strategies of routers.

5.1 Traditional router implementation

Traditionally, there are two ways of mapping the control plane and data plane functions to the underlying hardware. A software-based router is characterized by a single processor implementing both the data plane and the control plane. All packets in a software-based router are handled by the processor under software control. It contains a microprocessor, a shared bus (implementing the switch), and multiple line cards (implementing the ports). The actual hardware may be packaged in a desktop PC or as a stand-alone, custom-engineered device. Open-source operating systems such as Linux include all the necessary software to implement a software based router on a PC.

A hardware-based router is characterized by an optimized fast path implementing all or part of the data plane using ASICs. The switch is often implemented as a high-bandwidth cross-bar switch rather than a bus. The line cards perform the route lookup and IPmin processing using dedicated hardware. Any portion of the data plane not implemented in dedicated hardware is handled by the control processor. Because the throughput of the custom hardware is significantly higher than the throughput of the control processor, hardware-based routers often have a large difference in throughput between the fast path and the slow path. Their overall performance is sensitive to the number of packets which cannot be handled by the fast path.

Software-based routers are generally less expensive than hardware-based routers due to the high-volume general-purpose components used. Hardware-based routers have higher-performance (IPmin) than software based routers.

5.2 Naming conventions

Depending on application coverage and marketing reasons, platforms dedicated for processing of packet based communication channels have different names. Common names on various communication network platforms are:

- **Network Processors (NP)**
- **TCP Offload Engines (TOE)**
- **Protocol Processors (PP)**
- **Programmable Network Interfaces (PNI)**
- **Network Interface Cards (NIC)**
- **Packet processors (PaP)**

The two most general names are NP and PNI. The other ones are normally regarded as subsets of the NP type, but no naming convention has been agreed upon. The application coverage may vary a lot between two architectures within the same group. For example a TOE may process parts or the entire TCP protocol stack. It may, or may not, support TCP flags and SAR. Regardless which, it will still be presented to the customers as a TOE.

5.3 Commercial architectures

5.3.1 Intel IXP processors

After acquiring Level One Communications Inc. including the IXP1200 network processor Intel has been one of the most important actors in this field. Intel today offers a number of chips to solve different tasks when it comes to what they call Network Infrastructure Processing [5.23]. First of all they have the Internet eXchange Architecture (IXA) which includes different NP. The NP family uses Xscale instruction set (improved Strong-ARM) and the peak capacity is today

10 Gbit/s using high-end MAC interfacing chips, while the normal IXP 1200 uses Fast Ethernet. The original IXP 1200 is today offered in 162 or 232 MHz versions. In later versions (1240 and 1250) Intel has integrated more and more hardware assist for data intensive protocol processing tasks.

The IXP 1200 datapath includes 6 different micro engines which supports multi-thread programmability. The second generation NP IXP2400 includes 8 microengines. The microengines in the IXP 2400 are connected in two clusters of four engines. The microengines uses an application specific instruction set. The microengines share memory resources and have private connections to its neighboring engines. Each microengine contains a 4096 times 40 bits program memory. Each microengine can process 8 different contexts, e.i. threads. There are 128 general purpose registers and 640 data transfer registers available in each microengine. Further it includes a local memory capable of storing 640 32-bit data values. This memory is used for cashing of packet header information. The microengines also includes the following dedicated hardware blocks:

- **CRC unit for 16 and 32 bit computations.**
- **Pseudo Random Number generator (used for QoS in congestion algorithms).**
- **Fine granularity hardware timers**
- **Multiplier**
- **16-entry CAM used for cache search and assists software pipelining.**

A TCAM can be connected as an external accelerator working in parallel with the IXP2400.

The IXA chip is mainly intended for packet processing for switching, protocol conversion, QoS, firewalling and load balancing. Further Intel offers Control Plane Processors in the IXC family. IXC is mostly efficient when they are being used for exception handling and connection states processing. They are normally used in high end systems, e.g. Base Transceiver Stations, Radio Network Controllers, and MAN servers. They normally operate together with an IXA type of chip handling the control plane processing. The IXS family contains Media Processors used for acceleration of voice, fax, and data-communication. In a big server a number of these IXS could be used together with one IXA chip. Finally Intel offers I/O processors (IOP) that is a quite general architecture, which can be used for SAN acceleration.

Even though the IXP family is fully programmable it is very hard to program. The difficulty lies in the fact that the programmer and/or compiler must put an even computational load on all micro-engines in order to take advantage of the multi-thread optimized architecture. Further, efficient programming models have been

identified as a key-issue. This IXP programming problem have been addressed by several research groups ([[5.30]] and [[5.31]]).

5.3.2 Motorola C-Port C-5e Network Processor

Developed in the late 1990s by C-Port Corporation the C-5 digital communications processors where similar to the IXP from Intel. The C-5e NP is currently a part of Motorolas C-Port family. It supports the use of 16 line interfaces, each controlled by a RISC-based Channel Processor (CP). The CP contains a receive and a transmit processor. They are serial data processors (SDP), which can be used for various layer-2 implementations. Further the CP contains a dedicated channel processor RISC core (CPRC) with a dedicated 32-bit instruction set. Each CPRC uses a 8 kB instruction memory and a 12 kB data memory. Each channel processor can manage 156 Mbps line cards but when used in clusters, much higher bandwidth is supported.

Further the C-5e NP includes an eXecutive Processor (XP) for control plane operations. C-5e NP also includes a number of dedicated co-processors:

- **A table lookup unit (TLU) classifies incoming packets based on information in an external SRAM memory.**
- **A buffer management unit that controls the payload data storage while the header is being processed.**
- **A queue management unit that is shared between all the processors to provide QoS.**
- **A fabric processor provides a high-speed network (PHY) interface.**

The SDP in the CP is responsible for the bit- and byte-wise processing and can be considered as the fast path. The SPDs are responsible for the layer-2 interfaces, e.g. GMII. They also handle encoding/decoding, framing, formatting, parsing, and error checking (e.g. CRC and header checksum calculation). The SPD may also initiate a classification search in the TLU. The receive SPD include two FIFO buffers. The first one is a small FIFO storing incoming data before the bit processing. The other FIFO is larger and it stores the data before byte processing. The SPD are also responsible for framing and synchronization of the incoming packets. Several CP can be concatenated using the very high bandwidth interface bus (35 Gbps) for pipelined processing.

During 2002 Motorola released a C-3e version, which manage 3 Gbit/s data rate [5.6].

5.3.3 iSNAP

The IP Storage Network Access Processor from Silverback [5.11] terminates and process IP-based storage traffic in a GEth network with full duplex. It separates

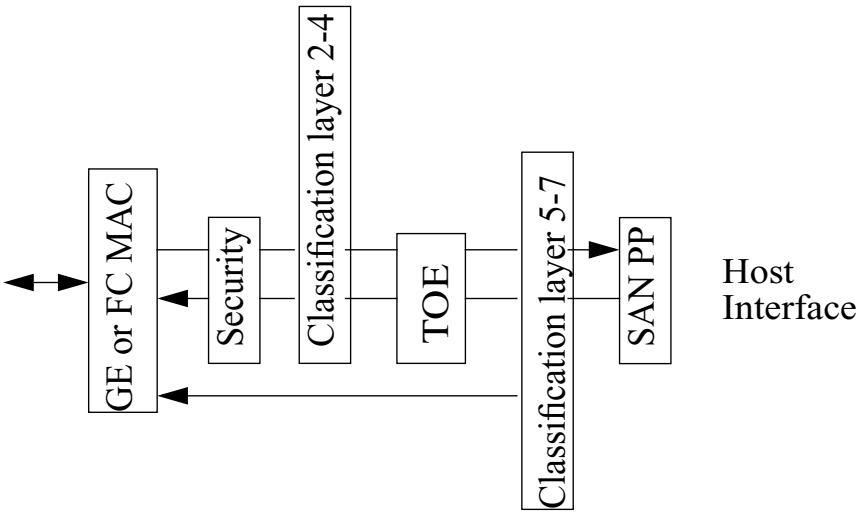

Figure 5.1: Trebia SNP architecture.

the header and data traffic processing. The header processing generates an event which is placed in a queue that communicates via DMA to the host. Meanwhile the packet data is stored in a DRAM until the event is finally created. At the host level the data can then be stored in separate application buffers depending on the upper layer protocol (ULP). This is called PDU awareness. ULP covered are iSCSI, NFS, CIFS and main application areas are servers, storage devices and Network Area Storage (NAS) appliances.

5.3.4 IBM PowerNP

First of all the PowerNP consists of a number of interfaces to memories (control and data) and networks (PHY/MAC ASICs). The packet processing is performed in the programmable Embedded Processor Complex (EPC) assisted by co-processors. The EPC contains 16 programmable engines known as picoprocessors. The picoprocessors operate in pairs called DPPUs. Each DPPU has a shared set of coprocessors that operates in parallel. The picoprocessors are essentially 32-bit scaled-down RISC machines with a dedicated instruction set. The DPPU also contains a small (4 kB) shared memory. The co-processors handle tree search, data storage, control access, queues, checksums, string copy, policy, counters, buses and system semaphoring.

5.3.5 Trebia SNP

This architecture [5.12] include a MAC block for mixed medias (wired and fibre-based), a security accelerator, various classification blocks, a TCP offload engine and a Storage Area Network (SAN)[1] protocol processor as illustrated by figure 5.1. The TOE can operate stand alone, terminating TCP connections without involving

1. Storage Area Networks (SAN) today sees a rapidly increasing use of NP to offload the host. The host is then typically acting as a file server. SAN was previously discussed in chapter 2.

the host processor. For IP storage applications Trebia claim that their TCP offload engine manage up to 10 GigE. The SAN PP is optimized for processing of storage I/O flows and especially iSCSI termination. According to CommsDesign.com Trebia is now out of business due to an immature iSCSI market.

5.3.6 iReady EthernetMAX

The Media Access Xccelerator [5.14] from iReady is intended for transport offload [5.13]. It fully terminates TCP/IP at GE speed. The TCP/IP accelerator uses a streaming data architecture similar to the one proposed by the author of this thesis. The data is not stored but instead processed while it is streaming through a 64 bit wide pipeline. The 64 bit wide datapath then processes the data using multiple dedicated hardware blocks implementing different state machines. Each state machine block processes a specific part of the incoming headers. The processor also uses hardware acceleration of iSCSI and IPSec. Since the complexity of the IPSec processing is 2 to 3 times higher than TCP/IP this architecture is not suitable from a power and cost point-of-view if the use of IPSec packets is not large in the network. The implementation does not use standard programmable devices. Instead dedicated logic for optimal performance is used.

5.3.7 Alacritech Internet PP

Alacritech [5.15] provides a Session Layer Interface Card (SLIC) [5.17] that includes accelerators for GE, network acceleration [5.16], storage acceleration and dual-purpose server and storage acceleration. Especially their Internet Protocol Processor (IPP) which offloads TCP/IP and iSCSI processing is interesting. The IPP offers acceleration of non-fragmented TCP connections. This means that data transfers to and from the TCP/IP stack is handled by the IPP while the host system must take care of the connection state processing. Parts of the TCP that IPP does not handle are:

- **TCP Connections and breakdowns (SYN segments)**
- **Fragmented segments**
- **Retransmission timeout**
- **Out of order segments**
- **Finish segments (FIN)**

Despite this down-sized functional coverage in the accelerators, Alacritech claims that 99.9 percent of the TCP/IP traffic is handled by the IPP while the other 0.1 percent is processed by the host processor. Alacritech further stresses the low power and low cost figures of their architecture.

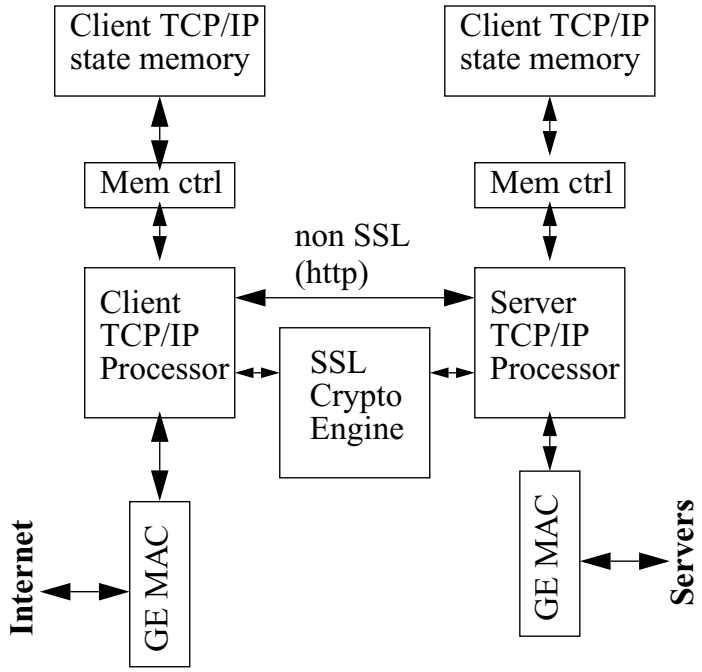

Figure 5.2: The UltraLock provides acceleration for SSL connections. Ordinary http packets are passed on without any processing in the SSL engine.

5.3.8 LayerN UltraLock

The UltraLock [5.19] illustrated by figure 5.2 uses a patented architecture named SIGNET [5.18]. The UltraLock chip offloads both the Network processing, including packet classification, and provides acceleration of Secure Socket Layer (SSL). The UltraLock also includes GE MAC accelerators.

In the TCP/IP processor the tasks are distributed among several different dedicated functional blocks in order to improve the throughput. These TCP/IP processors are also pipelined.

5.3.9 Seaway Streamwise NCP

Seaway Networks [5.20] offers a streamwise Network Content Processor (NCP) capable of multi-gigabit layer 4 (TCP) termination. The NCP also examine, modifies and replicate data streams based on their content (Layer 5-7). The NCP uses a streamwise switch to send data streams to different content processing devices, e.i. co-processors or general purpose CPUs.

5.3.10 Emulex LightPulse Fibre HBA

The Host Bus Adapter (HBA) from Emulex [5.21] includes an ASIC controller, a RISC core and a SAN accelerator. The SAN accelerator uses a context cache hardware so that context (PDU information) not must be transported to and from the host and thereby offloading the server PCI bus. The systems have 1 Gbit/s perfor-

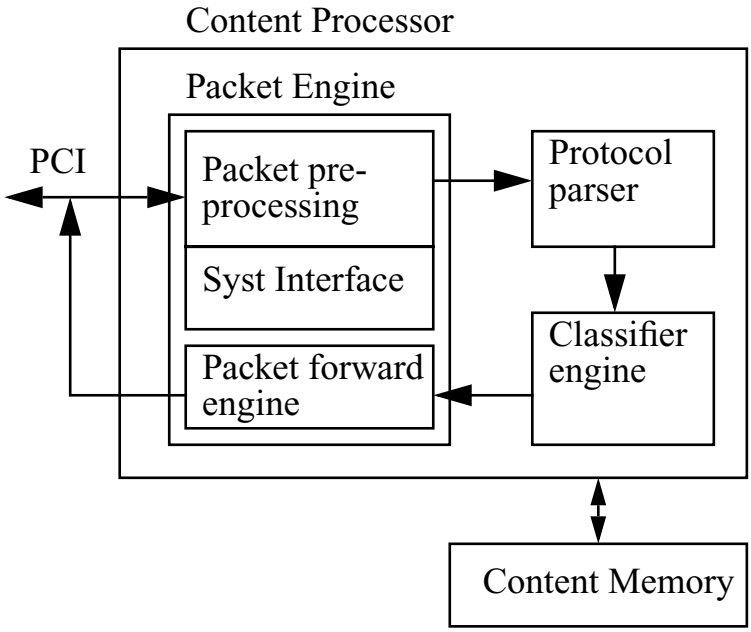

Figure 5.3: LeWiz content processor. The Packet pre-processor is a TOE. The Protocol parser examines the ULP data (layer 5-7) and based on this it start a search for a classifier using the classifier engine. The classifier then decides priority and is used for re-direction of the traffic according to the QoS policy.

mance and the main feature is the implementation of a strong SAN accelerator for high end servers.

5.3.11 LeWiz Content processor

LeWiz processor [5.22] processes layer 3-7 with hardware acceleration with a line rate capability of Gbit/s. Among other things it performs table lookup for connections, controls an external header data memory, supports different types of connections based on URL/source address, and handles XML and URL switching. LeWiz sells both hard and soft cores. The content processor architecture is further described in figure 5.3.

5.3.12 Qlogic SANblade

The SANblade [5.24] manage a 2 Gbit/s line rate using GE or fibre channel medias while performing iSCSI as a HBA. It completely offloads the TCP/IP protocol stack from the host. The SANblade also handles all I/O processing. The SANblade contains internal on-chip memory which they claim to be faster, cooler and moore scalable than using shared memory architectures.

5.3.13 Agere Systems - PayloadPlus

PayloadPlus provides a complete solution for OC-48c (2.5 Gbps) networks. The board solution includes 3 chips, capable of up to layer-7 processing. They are the

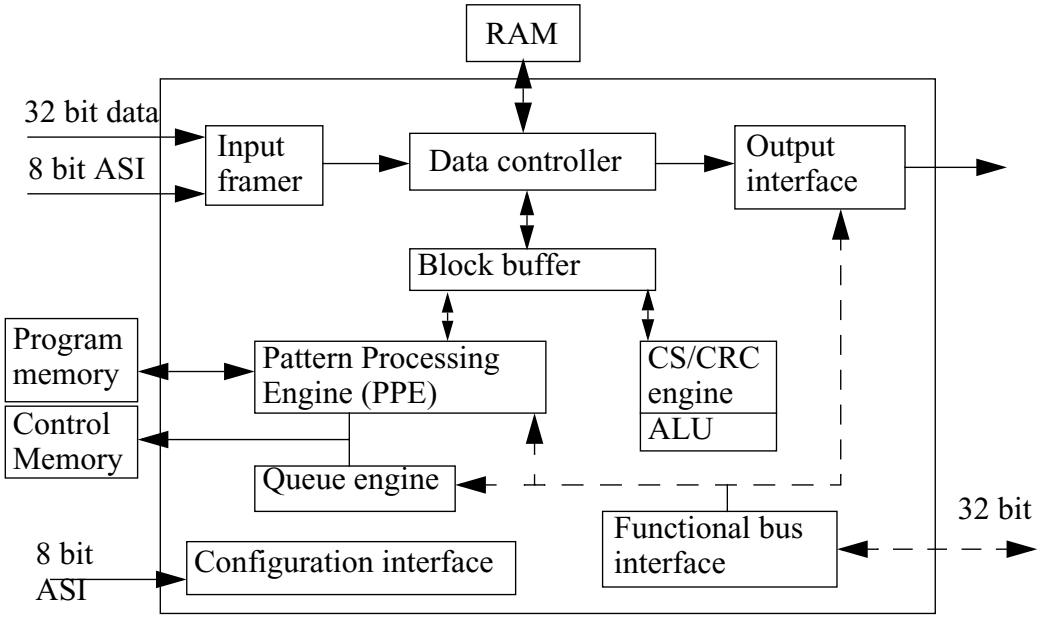

Figure 5.4: FPP architecture.

Fast Pattern Processor (FPP), the Routing Switching Processor (RSP), and the Agere System Interface (ASI).

The FPP is programmed with a dedicated protocol processing language (FPL). The FPP does not contain any accelerators for classification and reassembly such as CAM or Segmentation and Reassembly (SAR) devices.

The Pattern Processing Engine (PPE) matches fields in the data stream based on the program stored in the program memory. The program is written in FPL. The FPP operates on 64 PDU (i.e. threads) at a time. Each PDU is processed by a separate processing thread denoted context. The CS/CRC engine performs 4 different checksums based on the FPL program, generic checksum (1-complement), IP v4 checksum, CRC-10 and CRC-32. The input framer can be configured for 8, 16, or 32 bit wide datastreams.

The RSP handles the traffic management and flow modifications in a programmable way.

The ASI is a PCI like standard bus. ASI is used for exceptions and management tasks. The main applications are layer 2-3 routing and switching. The PayloadPlus architecture also supports voice and data processing (e.g. over IP, AAL5, AAL2), access control and enables QoS functionality.

The ASI, the RSP, and the FPP is connected to the same 8 bit configuration bus. The configuration bus is used for updating of routing tables and programs during runtime. Each of these processor could be described as fixed-function processors dedicated for a specific task. They need to be combined in order to support all of the network processing tasks.

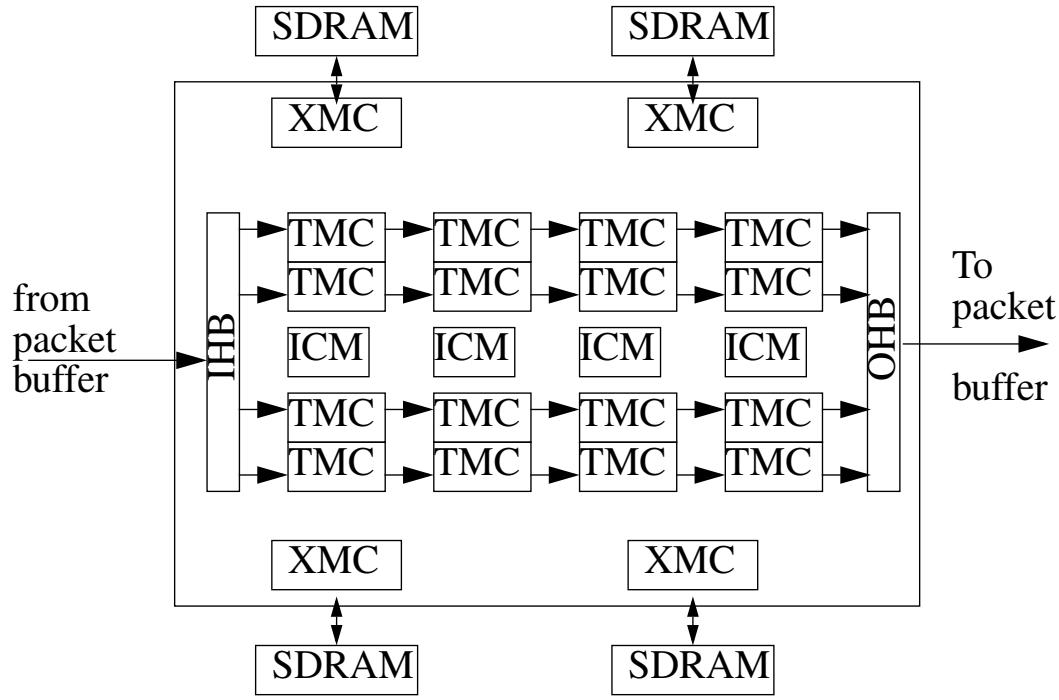

Figure 5.5: The Toaster2 architecture. IHB/OHB are uni-directional bus interfaces that are 64 bit wide and can operate at 100 Mhz and above. The TMC blocks are memory controllers that controls the access to each Internal Column Memory (ICM) while the XMC handles access to external memory devices.

5.3.14 Cisco - Toaster2

Toaster2 is a multiprocessor ASIC solution for layer 3 data path calculations. The chip includes 16 uniform processors each including a dedicated microcontroller (TMC). The 16 processors are organized in a 4 by 4 matrix. Each node also includes a program memory and a memory controller. The Toaster2 is typically used together with other Toaster2 chips, a packet buffer ASIC, PHY/MAC ASICs, and a routing processor. The routing processor is typically a general purpose RISC machine. The packet buffer stores the payload data while the header is being processed.

The TMC is essentially a SIMD architecture that uses a 64 bit instruction to operate on multiple 32 bit data. The architecture schedules ILP in software and then 4 stages of Toaster microcode is processed in a pipelined and parallel way by each row of four TMC.

5.3.15 PMC-Sierra

The PMC-Sierra ClassiPI is not really a network processor. Instead it is a DSP like classification device that can assist many different NPs with the complex task of packet classification. The ClassiPI architecture consists of two main engines.

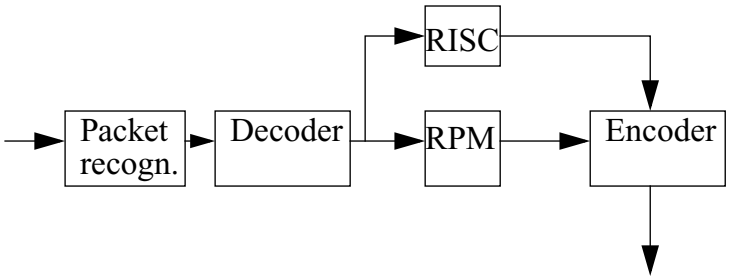

Figure 5.6: The PRO^3 architecture.

One is the Field Extraction Engine (FEE) and the other one is the Classification Engine (CE). The FEE can extract IP, UDP, and TCP header data from an incoming packet. The extracted data is then passed on to the CE for classification search operations. The CE is a RAM based classification engine that includes four ALUs and other processing logic. The CE uses an external memory for storage of programs and control state variables, e.g. counters and time stamps.

The RM9000x2 is their current next-generation processor. It contains dual 1-GHz MIPS processor cores, a 160 Gbps multi-port memory fabric, and a 500-MHz transport interface that allows for 16 Gbps data I/O traffic.

5.4 Academic architectures

5.4.1 EU Protocol Processor Project PRO^3

The architecture proposed by PRO^3 [5.25] consists of 5 parts. Most interesting is the Reconfigurable Pipelined Module, which processes data intensive tasks, and the embedded RISC core which takes care of the signaling processing. An illustration of the PRO^3 can be found in figure 5.6.

5.4.2 UCLA Packet decoder

This decoder [5.27], decodes packets on layer 2-4. The decoder architecture illustrated in figure 5.7 consists of one datapath for each layer, i.e. three data paths totally. It only uses one control path for the signaling processing. It operates on streaming data using an application-specific instruction set and the intended application area is routers.

5.4.3 TACO processor from Turku University

Based on a Transport Triggered Architecture (TTA). The TTA architecture only uses one instruction (move). The architecture uses the move instruction to transport data between different dedicated functional blocks. The main focus has been on optimization of the distribution of tasks and data between different dedicated hardware blocks. To do this, a development and simulation framework has been devel-

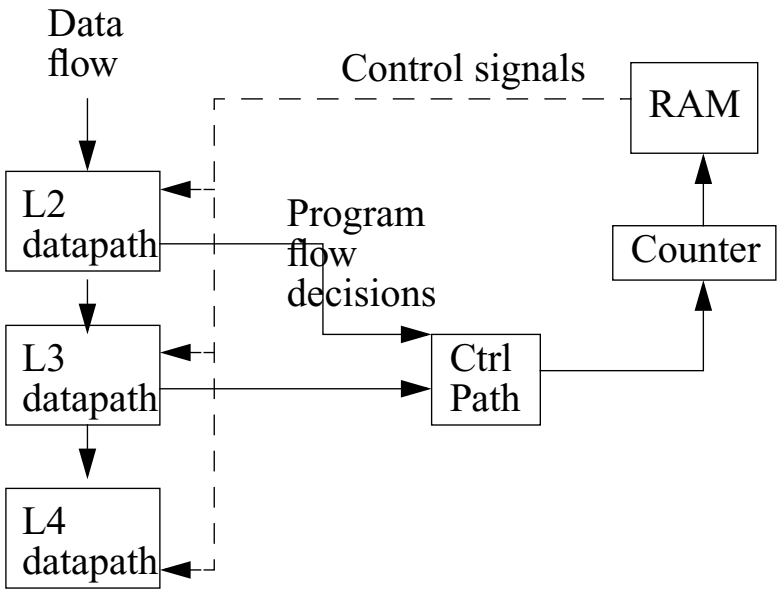

Figure 5.7: Simplified view of the UCLA processor architecture proposal showing how to accelerate case-jump functions.

oped. The intended application area is primary the ATM protocol. The framework and the architecture is described in [5.28].

5.4.4 PICO project from Berkeley

The PICO project is focused on low-power terminal processing for wireless networks. Examples on protocols covered are Bluetooth and Home RF. Sensor based networks has also been part of the project. The design consists of a fast path implemented in FPGAs and a slow path implemented in Programmable Logic Devices (PLDs). The PICO processor is further described in [5.29].

5.4.5 Washington University Field Programmable Port Extender

The Field Programmable Port Extender (FPX) [5.10] is a switching system including an FPGA which allows the functionality to be remotely reconfigured over the network. This off course gives a high degree of flexibility. The FPX module also includes a fixed function network interface.

5.5 Conclusions from survey

A number of different NP solutions are included in the survey. They all are focused on different application areas. Some fully offloads complex protocols, while others mainly focus on high-speed fast path operation. As one can see from the survey, a trend against separation of the network processor area into more dedicated specialized network processors optimal for a certain application area emerges. As illustrated by table 1 different architectures targeted for different

applications have different performance, programmability, and architectural approach.

Table 1: Summary of survey. Architecture (1 - MCU with NP feature, 2 - DP acceleration, 3 - ASIP, 4 - ASIC/Coprocessor)

NP name	Throughput	Flexibility	Architecture	Homogenous or Heterogeneous
SW-based router	*	***	1	-
ASIC-based router	***	*	4	-
Intel IXA/IXC/IXS/IOP	***	**	2, 4	Homogenous
Motorola C-Port C-5e	**	**	4	Heterogeneous
iSNAP	**	***	3	Homogenous
IBM PowerNP	?	***	3, 4	Homogenous
Trebia SNP	***	**	2, 3, 4	Heterogeneous
iReady EthernetMAX	**	*	4	Heterogeneous
Alacritech Internet PP	**	*	1, 2, 3	Heterogeneous
LayerN UltraLock	*	**	2, 4	Heterogeneous
Seaway Streamwise NCP	***	?	1, 4	Heterogeneous
Emulex LightPulse	**	**	1, 2	Heterogeneous
LeWiz Content processor	**	**	2, 4	Heterogeneous
Qlogic SANblade	**	**	?	Pipelined Multithread
PayloadPlus	**	***	2, 4	Heterogeneous
Toaster2	***	***	1, 4	Homogenous
ClassiPI	***	*	2, 4	Heterogeneous
PRO3	?	**	1, 2	Heterogeneous
UCLA	**	*	3	Heterogeneous
Turku TACO	?	**	3, 4	Heterogeneous
Berkeley PICO	*	**	1, 4	Homogenous
Washington FPX	?	***	4	Heterogeneous

It is interesting to note that the first generation of NPs (IBM and Intel) where quite similar, composing one or more processing element and a number of co-processors. Now there has emerged diverse solutions such as:

- **GP CPU**
- **Dataflow processing architectures (Cisco)**
- **DSP processing (PMC-Sierra)**

Processing elements are used in two different ways.

- **Pipelined:** Each processor element is designed for a particular packet processing task and communicates in a pipelined fashion. This dataflow based processing is used by Cisco, Motorola etc.
- **Parallel:** Each processing element is performing similar functionality. Normally arbitration units and extensive use of co-processors are normally required. Examples are Intel IXP and PowerNP.

There is a general disagreement on how much functionality to include in a PNI and how much should be left for the host in TOEs. Instead it is clear that TOE, MAC, Encryption accelerators, and SAN control accelerators are being designed and optimized independently. Hopefully this means that we soon can have standardized interfaces between different communication accelerators. In the future we will surely see new protocol processing application areas, where area specific PNIs are worth using. SAN is just the first one becoming commercially interesting. There is no clear trend on the amount of offloading needed in a TOE for NT so further exploration is needed.

One big question that remains unanswered is where the reordering and reassembly of the incoming application data should be done. The question is if the data should be delivered to the main memory unordered or if it should be stored in order in the application buffers. The second alternative demands an embedded data memory to be used. The data delivery format has off course a big impact on the host operation. The comparison clearly shows that there exist solutions to the various new PNI specific implementation problems and considerations discussed in earlier chapters.

Reference

[5.1] Agere Systems, "*Smart Processing for Terabit Networks, PayloadPlus Processor Family Overview*", Agere Systems Incorporated, Allentown, PA, 1999
[5.2] Agere Systems, "*10G Network Processor Chip Set (APP750NP and APP750TM) Product Brief*", Agere Systems Incorporated, Allentown, PA, November 2002.
[5.3] PMC-Sierra, "*RM9000x2 Integrated Multiprocessor Data Sheet*", PMCSierra Incorporated, Burnaby, B.C., 2001.
[5.4] Chandra, V., Selecting a network processor architecture, IBM MicroelectronicsTechnology Group, August 2002.
[5.5] port Corp., "*C-5 Network Processor Architecture Guide, C-port technical library C5NPD0-AG/D*", C-port Corporation, North Andover, MA, May 2001
[5.6] Motorola, "*C-5e/C-3e Network Processor Architecture Guide*", on the internet, http://e-www.motorola.com/files/issue_files/cat_not_blade/ C5EC3EARCH-RM.pdf

[5.7] Vitesse Semiconductor, "*IQ2200 Product Brief*", Vittesse Semiconductor Corp, Camarillo, CA, 2002.
[5.8] Wirbel, L., "*Network processors take reconfigurable approach*", EE Times, May 22, 2000.
[5.9] Solidum Systems, "*PAX.port 2500 Product Brochure*", IDT Canada Inc., Ottawa, Ontario, 2002.
[5.10]Lockwood, J.W., "*An Open Platform for Development of Network Processing Modules in Reprogrammable Hardware*", IEC DesignCon 01, Santa Clara, CA, January 2001, paper WB-19.
[5.11]Silverback Systems homepage, *on the www*, http://wwwsilverbacksystems.com
[5.12]Trebia Networks homepage, *on the www*, http://www.trebia.com
[5.13]National Semiconductors, "*Enabling Next Generation Ethernet*", *on the www*, http://www.trebia.com/EthernetMAXweb.pdf
[5.14]Minami, et al, "*Multiple network protocol encoder/decoder and data processor*", *US patent, no. 6 034 963*
[5.15]Alacritech Inc. homepage, *on the www*, http://www.alacritech.com
[5.16]Boucher, et al, "TCP/IP offload network interface device", US patent, no. 6 434 620
[5.17]Boucher, et al, "*Intelligent network interface system method for protocol processing*", US patent, no. 6 434 620
[5.18]LayerN Networks, "*SIGNET - Secure In-line Networking*", *on the www*, http://www.layern.com/SIGNETWP020419.pdf
[5.19]Omura, et al, "*The Evolution of Modern Digital Security Techniques*", *on the www*, http://www.layern.com/EvolutionWhitePaper.pdf
[5.20]Seaway Networks homepage, *on the www,* http://www.seawaynetworks.com
[5.21]Emulex homepage, *on the www,* http://www.emulex.com
[5.22]LeWiz Communication,Inc. homepage, *on the www*, http://www.lewiz.com
[5.23]Intel Corp., "Network Infrastructure Processors - Extending Intelligence in the Network", *white paper on the www,* http://www.intel.com/design/network/papers/251690001.pdf
[5.24]QLogic Corp. homepage, *on the www*, http://www.qlogic.com
[5.25]G. Konstantoulakis, V. Nellas, C. Georgopoulos, T. Orphanoudakis, N. Zervos, M. Steck, D. Verkest, G. Doumenis, D. Resis, N. Nikolaou, J.-A. Sanchez-P., "*A Novel Architecture for Efficient Protocol Processing in High Speed Communication Environments*", ECUMN 2000, pp. 425-431
[5.26]C. Georgopoulos et al, "*A Protocol Processing Architecture Backing TCP/IP-based Security Applications in High Speed Networks*", INTERWORKING 2000, Oct. 2000, Bergen, Norway
[5.27]M. Attia, I. Verbauwhede, "*Programmable Gigabit Ethernet Packet Processor Design Methodology*", ECCTD 2001, vol. III, pp. 177-180
[5.28]Virtanen, Seppo A., et al, "*A Processor Architecture for the TACO Protocol Processor Development Framework*", in the Proceedings of the 18th IEEE Norchip Conference, Turku, Finland, 2002, pp. 204-211
[5.29]T. Tuan, S.-F- Li, J. Rabaey, "*Reconfigurable Platform Design for Wireless Protocol Processors*", ICASSP 2001, pp. 893-896
[5.30]M. Tsai, C. Kulkarni, C. Sauer, N. Shah, and K. Keutzer, "*A Benchmarking Methodology for Network Processors*", Network Processors 2002: Design Principles and Practice, Morgan Kaufmann Publishers Inc., 2002., ISBN: 1-55860-875-3
[5.31]Gokhan Memik and William H. Mangione-Smith, "*NEPAL: A Framework for Efficiently Structuring Applications for Network Processors*", HPCA/NP2, Anaheim, 2002,to appear in Network Processor Design: Issues and Practices, Volume 2, Patrick Crowley, Mark A. Franklin, Haldun Hadimioglu, Peter Z. Onufryk (editors), Morgan Kaufman, 2003. (Also appeared in Second Workshop on Network Processors - NP2)

Part 3

Proposed architecture

“A journey of a thousand miles begins with a single step.”
--Confucius

6

Protocol Processor for Terminals

6.1 Proposed architecture

This chapter describes a hardware architecture proposal which is a result of my research during 1999-2003. The architecture is a dual-processor network interface dedicated for packet reception in a network terminal.

This chapter will give an overview of the architecture. In the following chapters a more detailed discussion on memory issues, data path, and control path will be included.

6.1.1 Research idea

The main research idea is to use a data-flow architecture that can partly process packets before they are stored. This means that the size and usage of packet buffers can be reduced. In order to support usage in high-speed networks we must use hardware acceleration and thereby enable the execution of packets as they stream through. Even if the bandwidth requirements of a terminal is not as high as for a core router, the terminal may be connected directly to a high-speed network and therefore require high speed processing.

Processing tasks have been analyzed and allocated to processing resources depending on their requirements. By combining different architectures and exploiting their benefits the proposed architecture can support both high perfor-

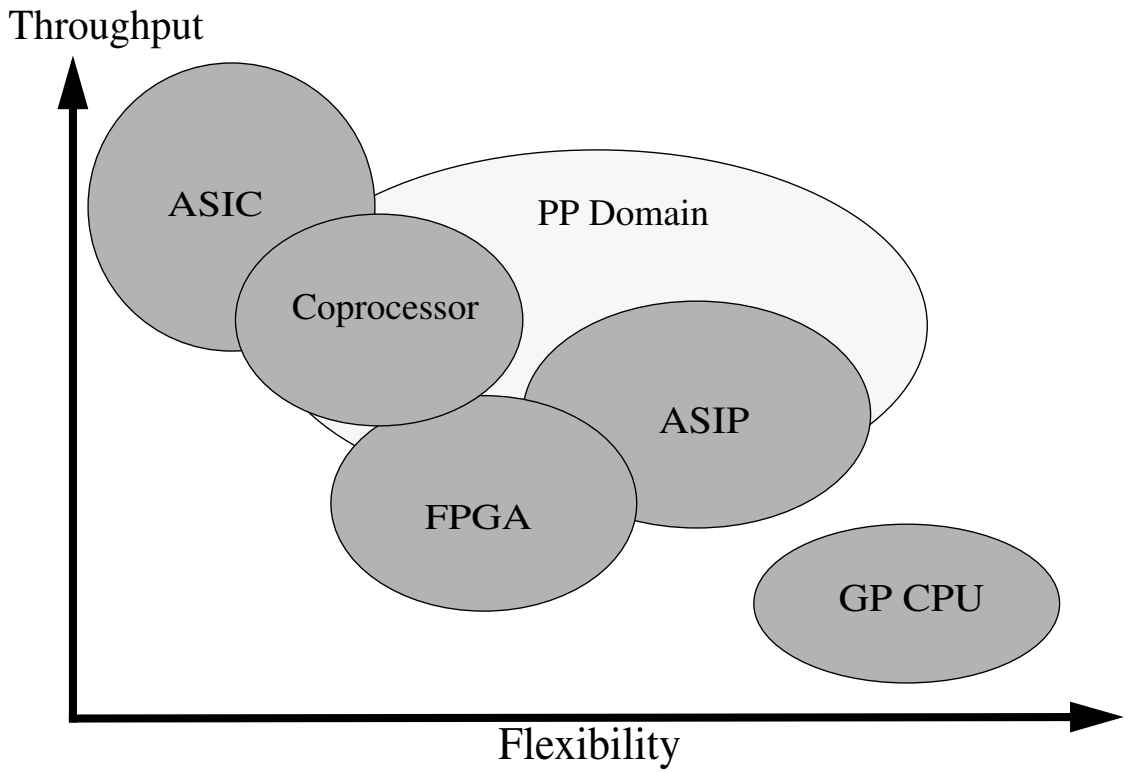

Figure 6.1: A protocol processor for terminals must have a highly optimized and dedicated architechture if both the requirements on performance and flexibility should be met.

mance and flexibility as depicted in figure 6.1. The flexibility needed is supported by configurability and programmability.

How do we know that our flexibility is sufficient to enable the processing of future protocols? The answer is: we do not know, we can only guess based on current standards and trends. Adaptation for new protocols, products, and standards is possible but not guarantied by providing flexibility.

6.1.2 Dual-processor solution

As mentioned earlier the proposed terminal network interface architecture consists of two parts, i.e two processors. The first part is the Programmable Protocol Processor (PPP) and the other is the micro controller (µC). The platform also includes several memories. They are used as program memory, data memory, and control memory. The control memory stores inter-packet control variables, e.g. connection state variables. Received payload data is stored in the hosts main memory, i.e. outside the PNI.

The architecture is intended to be a part of a SoC where one or several PPPs together with the µC act as a high speed PNI. An overview of the system is illustrated by figure 6.2.

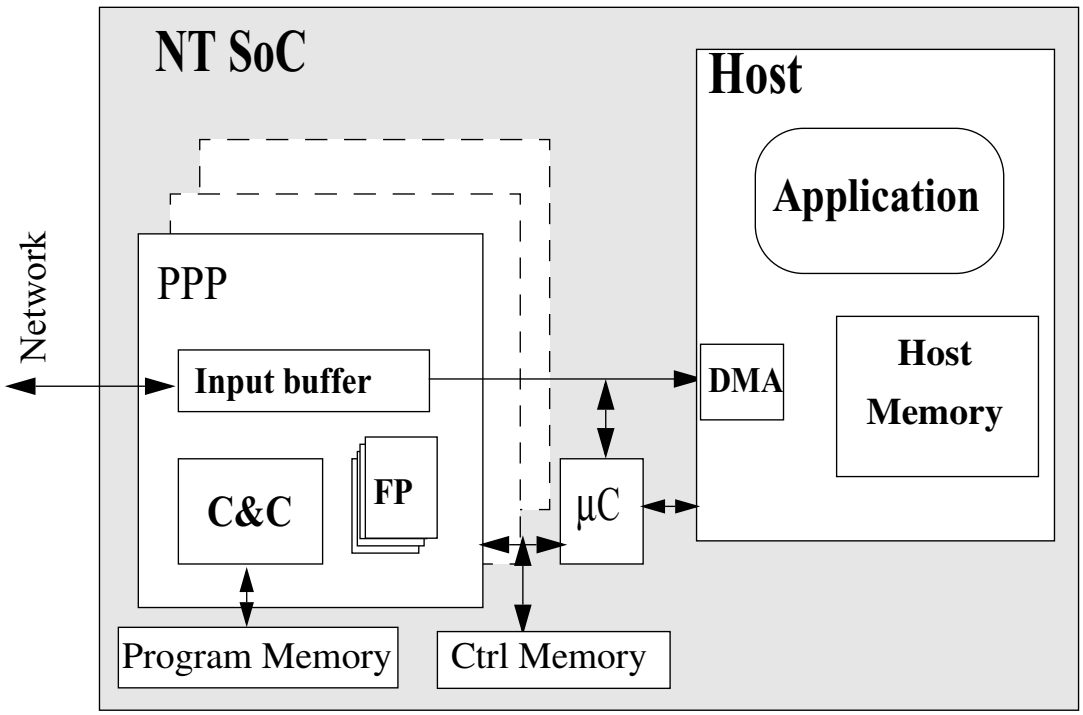

Figure 6.2: The PPP together with a general purpose micro controller (µC) handles the communication of one network media port. In a system on chip (SoC) many PPP can be used as port-processors in order to provide high bandwidth between the application and the network.

In order to reduce the overhead and support high-performance wire speed processing, processing tasks are allocated to the two processors based on their requirements on control, memory usage, and processing load. The coarse allocation is that control intensive tasks are being processed in the microcontroller and data intensive tasks suitable for hardware acceleration in the PPP.

During the progress of the research work, the dual processor architecture has changed name several times. The reason for this is that the research field is so immature that no naming convention has been agreed on. During the whole design process we have used the name Protocol Processor describing our PNI but this is so easy to mix with PPP so instead PNI will be used. The PNI consists of two parts. One is the general purpose micro controller. The micro controller hardware architecture has not been investigated in the research project. Instead the focus has been on the PPP implementing what can be considered the fast path of the protocol processing.

During my research, the fast path of the protocol processor has been denoted as Deep Pipelined Serial Processor (DPSP), Configurable Port Protocol Processor (CPPP) and Programmable Protocol Processor (PPP). The names reflects the ongoing rapid development, both in our research and in the research community by large. The name that will be used in the following chapters is PPP.

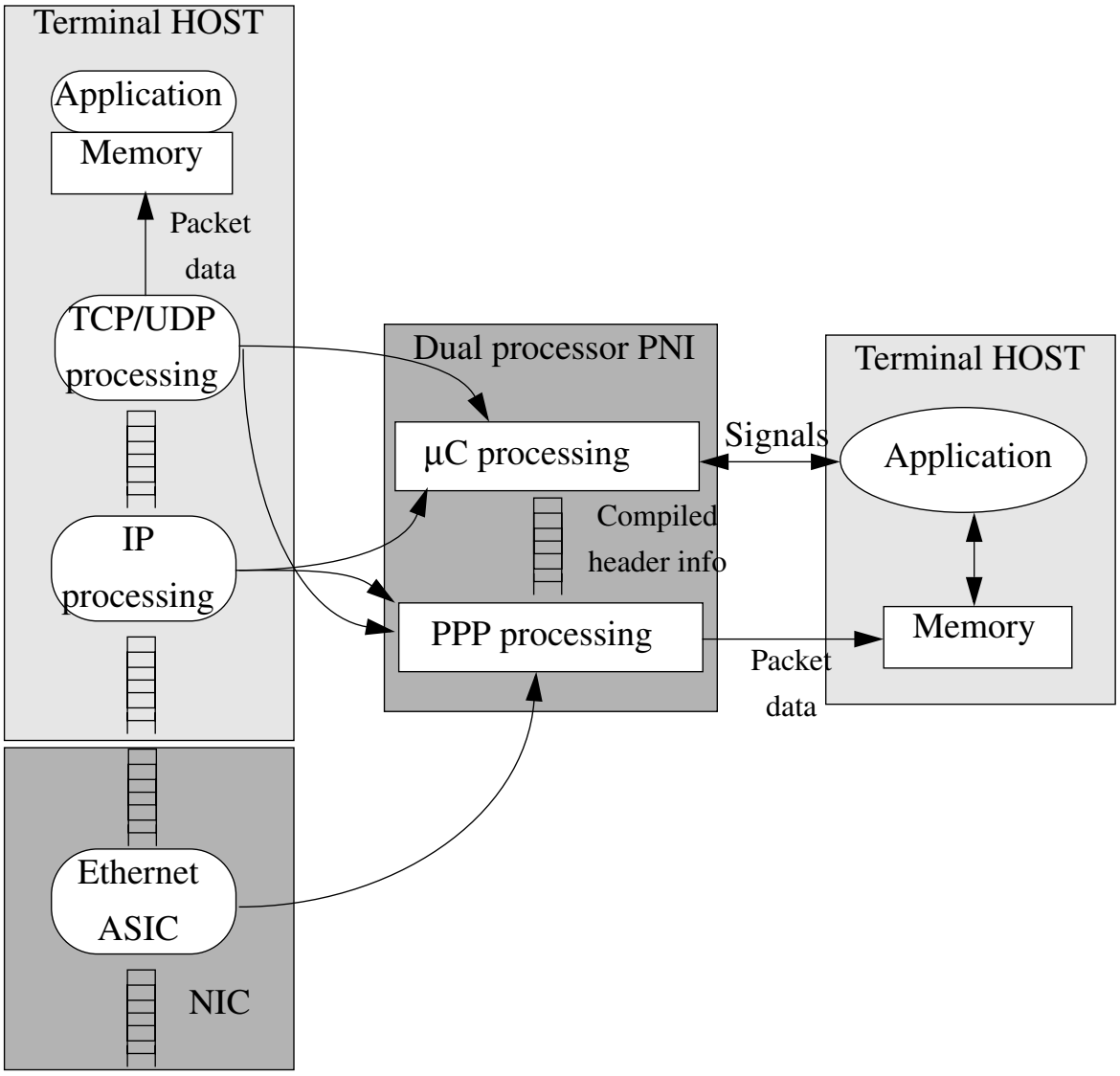

Figure 6.3: The proposed dual processor architecture acts as a programmable network interface. It offloads the host at high-speed still providing enough flexibility. By intra-layer processing in the two processors the usage of memory can be significantly reduced. The figure illustrates how the processing in a traditional type of terminal (to the left) can be mapped to the proposed PNI architecture (right).

6.1.3 Domain specific PPP

This thesis identifies two approaches available for wire-speed protocol processing. Either a dual clock system with a high-speed pipeline processor responsible for accelerator control or a synchronized data-flow machine with no pipeline penalty. Which to choose depends on the requirements on the protocol processor system, set by the terminal application and the protocol coverage.

While pipelined processors is the predominant solution in the network processor community this thesis proposes a synchronized data-flow architecture to be used as C&C if:

- **The supported number of connections is small**
- **The supported number of protocols is small**
- **Reassembly is not required from the PPP**

If these bulleted constraints are not applicable, the PPP should be implemented as a pipelined processor. Note, that the proposed dual-processor task partition supports both cases. This means that the PPP is domain specific while the micro controller is not. E.g. when offloading a file server, a pipelined solution should be used and when offloading a more simple terminal device, an efficient low-power synchronized data-flow implementation should be used. A deeper discussion on how the PPP should be implemented to meet different application domain specific requirements, is provided in chapter 7.

6.2 Dataflow based datapath

A datapath of the PPP has been developed and optimized based on tasks extracted from some common computer network protocols. These processing tasks are listed in section 6.6. The datapath of the PPP includes two types of components: the data-flow input buffers and Functional Pages (FP). An overview of the PPP architecture is illustrated by figure 6.4, The input buffer is used to access the data (with low fan-out) while it streams through the PPP. The buffer and other memory issues will be further discussed in chapter 8. FPs are a number of configurable accelerators dedicated for processing of data-intensive intra-packet tasks. The FPs will be further discussed in chapter 9.

6.3 Counter and controller

The C&C is the only programmable part of the PPP. It basically acts as the PPP control path. The C&C is responsible for starting and stopping FP processing, based on the program and the result flags from the FPs. The C&C is also responsible for the decision to discard or accept a packet.

The main tasks of the C&C are to control FPs using flags and to select program counter (PC) values based on the flags from the FPs. The PC value is for example used to select the correct program flow when the incoming packets protocol type have been checked.

The FSM top level packet reception control is illustrated by figure 6.5. The C&C produces start and stop flags for the FPs and simple instructions for the CMAA.

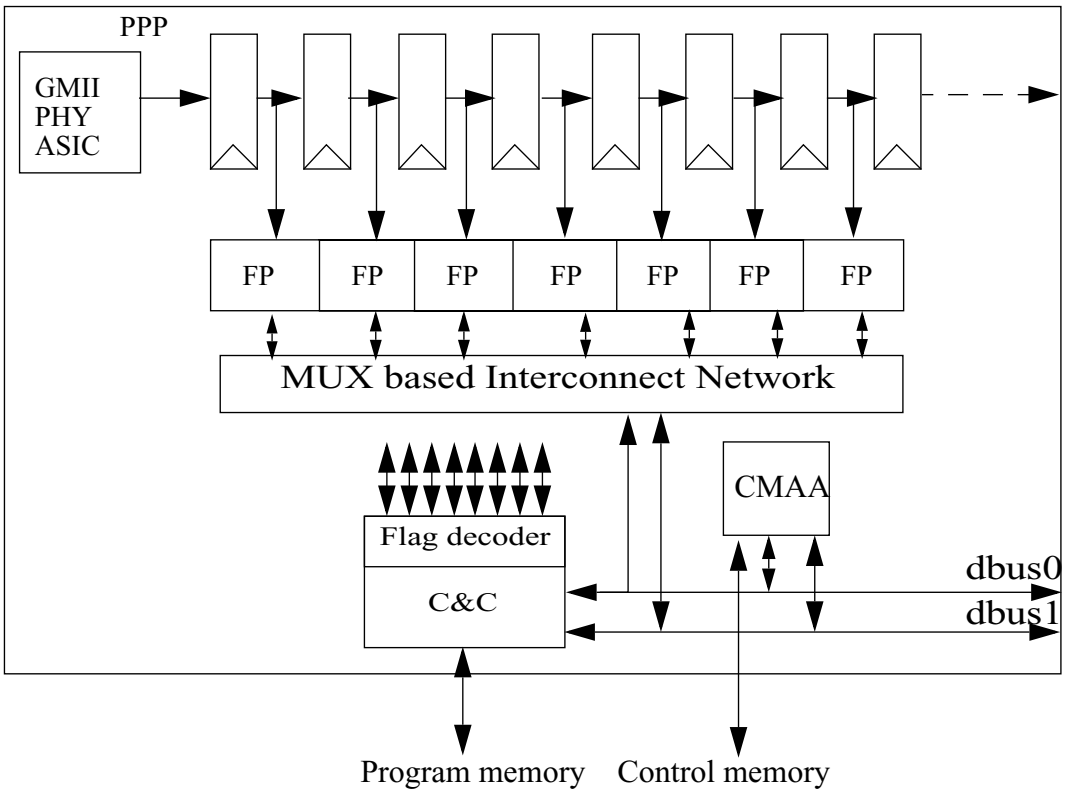

Figure 6.4: Overview of the PPP architecture.

C&C implementation details and the CMAA is discussed in chapter 10 and chapter 11 respectively.

6.4 Configuration

The proposed dual-processor architecture supports three levels of configuration.

- **Design time selection**

First of all it is possible to select and configure a number of FPs during the design phase, before manufacturing. At this phase one can also choose to implement the C&C as a pipelined processor or as a synchronized data-flow processor core. The application domain and protocol coverage of the PPP are defined during this phase.

- **Data path configuration**

Secondly the micro controller can configure the various FPs by writing configurations to configuration registers within each FP. This only requires a relatively small number of clock cycles. Hence, the data path of the PPP is configurable. The program flow of the C&C can also be fully configured during this phase by rewriting the contents of the program memory and CAMs. This means that the PPP can be configured for different sets of protocols on-line.

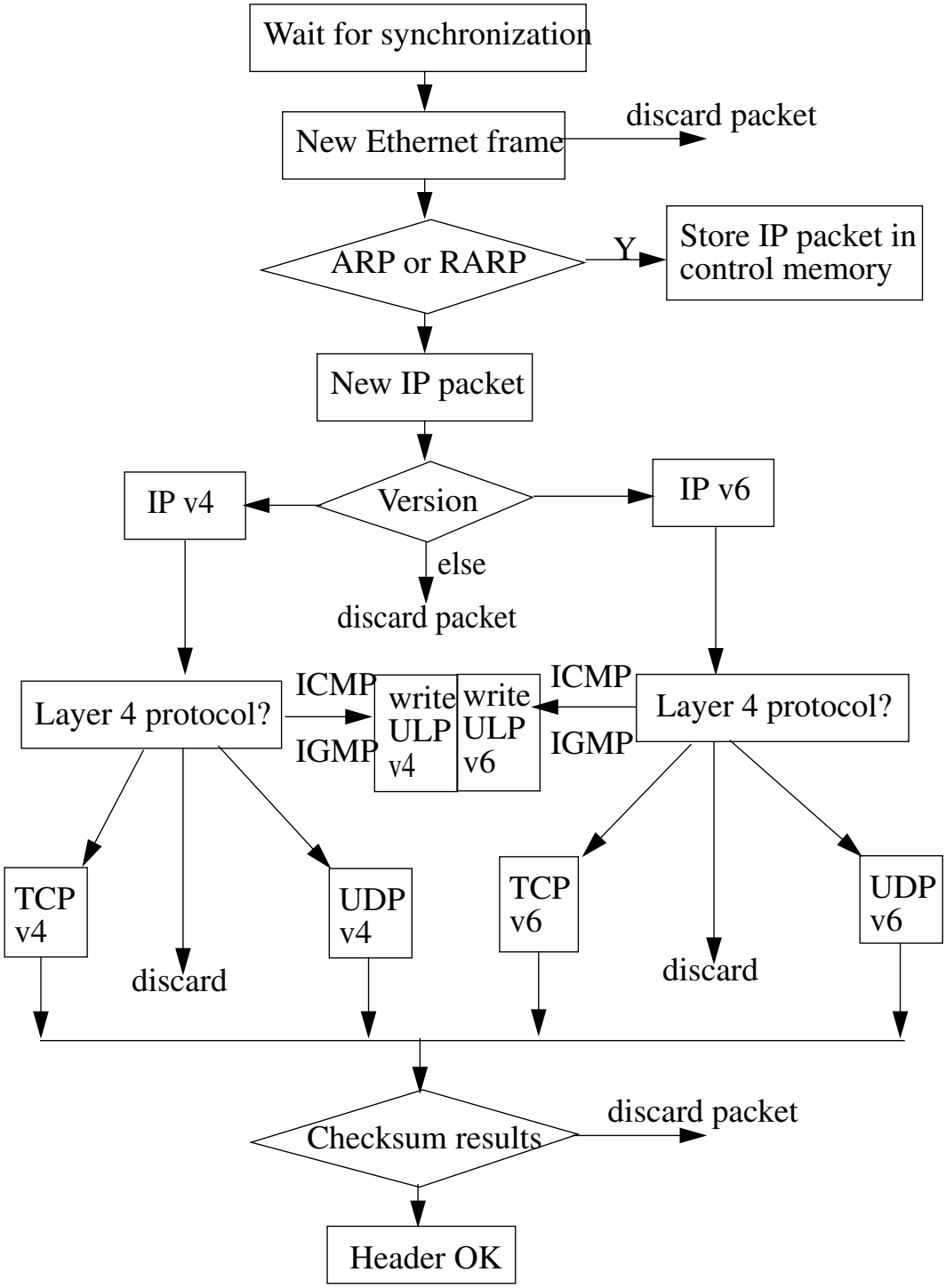

Figure 6.5: The control FSM controlling the PPP during packet reception will be implemented in the C&C in a programmable way. This figure illustrates such a FSM covering IPv4, IPv6, TCP, UDP, ARP and RARP protocols.

- **Programmable data path selection**

The data path can be controlled and selected in a programmable way using the C&C.

All together the three levels of configuration possibilities give the architectures a very high flexibility.

6.5 Interfaces

The PNI consists of two parts, the PPP and the μC. The interfaces between them and towards the surroundings can be divided into three parts.

6.5.1 Network interface

The interface between the network and the protocol processor consists of a PHY ASIC. Normally we consider ASIC interface to be the Gigabit Media Independent Interface (GMII), but MII, XGMII, or others could also be considered. Since the PPP is based on 32 bit accelerators, a simple parallelization interface is needed. The PHY ASIC is a part of the PPP and it produces 32 bit wide data that will be delivered to the input buffer. The use of such an interface means that the FPs do not need to handle the processing of the physical layer protocols even if it would be possible to integrate such FPs.

6.5.2 Micro controller interface

The interface between the PPP and the micro controller consists of two data busses, the shared control memory, and flags for control signaling. The micro controller also uses the two data buses when it configures the functional pages and the program memory of the C&C.

6.5.3 Host system interface

The interface between the host processor, including application, memory, DMA, and the PNI remains to be investigated. It is however clear that it will be the micro controller that will be responsible for this communication in the PNI. The micro controller will control the communication both with the DMA and the application through the hosts operating system. One might also consider using a standard back-plane bus (e.g. PCI) as interface between the two.

6.6 Protocol suite and processing task

To develop the proposed architecture, a common set of protocols has been used. The protocols are useful for investigations on architectural requirements and possibilities. This however does not mean that the architecture is not suitable for other application areas and protocols. These protocols represent a basic set needed in most computer networks. Hence, these protocols define the basic requirements on the processing in the protocol processor. The protocols included are:

- Fast Ethernet with PHY interface MII
- Gigabit Ethernet with PHY interface GMII

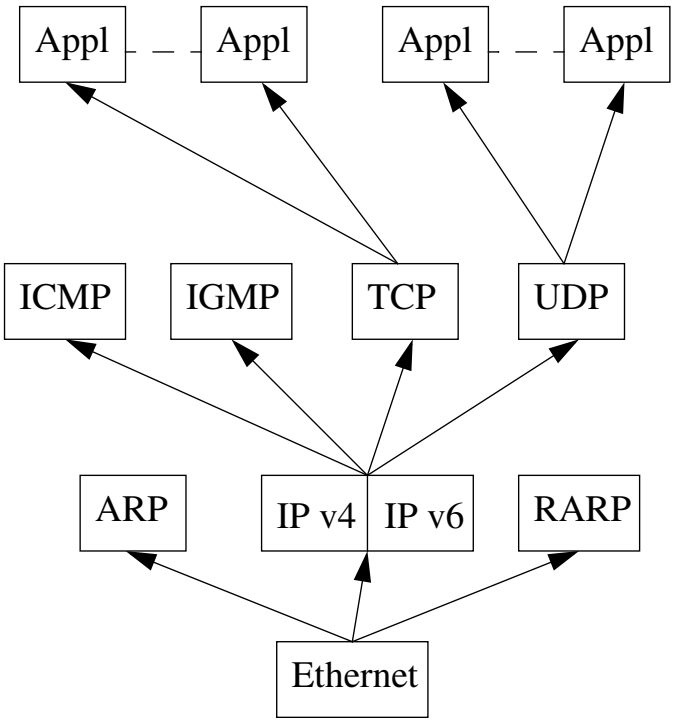

Figure 6.6: The packet demultiplexing of a received Ethernet frame.

- 10 Gigabit Ethernet with PHY interface XGMII
- IP version 4 and version 6 (IPv4 and IPv6)
- Address Resolution Protocol (ARP)
- Reversed Address Resolution Protocol (RARP)
- Internet Control Message Protocol (ICMP)
- Internet Group Management Protocol (IGMP)
- Transport Control Protocol (TCP)
- User Datagram Protocol (UDP)

The selected protocols are very commonly used today and there is no reason to believe that they will not continue to be used for a long time ahead. Further, the protocols are required for many of the existing application protocols used today. When a data frame from the network PHY interface is received, it will be passed on to different processing units. Each of these processing units perform a specific protocol processing algorithm which depends on which protocols have been used. After identification and demultiplexing of an incoming packet it is possible to check if it is valid according to the rules set by the specific protocol. The demultiplexing of incoming packets is depicted in figure 6.6.

Each header includes a number of header fields which have to be extracted and processed according to the protocol standard. The header encapsulation format is illustrated by figure 6.7 and figure 6.8.

Eth header	IP header	TCP/UDP header	DATA	CRC

Figure 6.7: One Ethernet frame encapsulate the IP packets. Each layer includes a header and data.

Ethernet Destination Address 47-16			
Ethernet Destination Address 15-0	Ethernet Source Address 47-32		
Ethernet Source Address 31-0			
Ethernet Type	IP ver	IP HL	IP ToS
IP Length	IP ID field		
IP Fragmentation	IP TTL		IP Protocol (TCP)
IP Header Checksum	IP Source Address 31-16		
IP Source Address 15-0	IP Destination Address 31-16		
IP Destination Address 15-0	TCPP Source Port		
TCP Destination Port	TCP Length		
TCP Checksum	Padding		
Data			
Ethernet Frame Check Sequence (CRC)			

Figure 6.8: Packet format example. Only packet header fields used for processing in the PNI is listed, e.g. not the Ethernet preamble.

In order to process all the headers and provide the services stipulated by the protocol standard, a number of processing tasks are required to be performed by the receiving terminal. This set of processing tasks is specific to the selected protocol suite. If new protocols should be included, new types of processing task may, or may not, be needed. The processing tasks are listed in the subsections following.

6.6.1 Ethernet

- **Calculate CRC**

Cyclic Redundancy Check is a error detecting code that is used to detect transmission errors. The CRC checksum is computed over the whole frame before it is compared with the transmitted CRC checksum. The transmitted checksum has been calculated using the same algorithm by the transmitter, and it is transmitted in the trail of the frame, after the data. The CRC computation is a very data intensive operation. In a simple RISC machine a 1500 Byte long frame require almost 44000 (non-optimized) instructions to process only the CRC checksum according to publication 2[1]. Hence, the CRC calculation is normally done using dedicated

hardware assist. Publication 3[1] describes such a hardware block dedicated for CRC acceleration.

- **Check Ethernet Destination Address (Eth DA)**

To be sure that the received frame is intended for the terminal, it must check that the destination address is correct. If the address is incorrect the packet will be discarded. Any failed test will result in a discard decision.

- **Check the type field**

The type field describes what sort of layer 3 packet is encapsulated in the frame. The valid options according to my protocol suite are ARP (0x0806), RARP(0x0835) and IP(0x0800).

- **Extract length field**

The length field must be extracted to know how long the packet is. It is especially important to know since the CRC value stored in the last 32 bits of the frame, must be extracted and compared to the computed CRC value.

- **Demultiplex data**

When the terminal has identified the layer 3 protocol used (ARP, RARP or IP) it can send the Ethernet data (including e.g. IP header) to the correct location for further processing.

6.6.2 Address Resolution Protocol (ARP)

- **Extract and check the ARP code**

The ARP protocol is used to query the network for a MAC address when we have an IP address but do not know the corresponding MAC address. The ARP code typically tells if the packet is a query or a reply.

- **Update ARP table**

We should update our table describing which MAC addresses belongs to which IP addresses.

- **Send reply**

If needed a reply packet should be triggered.

6.6.3 Reversed ARP (RARP)

RARP is typically used during a booting procedure. We know our MAC address from the NIC but do not have any IP address. To get an IP address we send out a RARP request. The header format and processing tasks are the same as for the ARP protocol.

1. Publications listed in the Preface.

6.6.4 Internet Protocol (IP)

- **Check the version**

The version field tells if it is IP version 4 or 6 that has been used. The main difference is that IP version 6 allows for a larger number of users since 128 bits are used for the addresses instead of 32.

- **Calculate header checksum**

The IP checksum is a 16 bit wide 1-complement addition of the header. The data is not included in the checksum addition since transport layer protocols (e.i. TCP, UDP, ICMP, IGMP) have their own checksums. This operation must be performed for all headers which can be a heavy load for a host processor.

- **Extract and check IP Destination Address (IP DA)**

The IP DA is unique for a terminal, no other terminal share the same address. Each network terminal can have several IP DA but normally it only has one. If the IP DA is erroneous the packet should be discarded.

- **Extract the IP Source Address (IP SA)**

The IP SA is used for checking if we should accept a packet or not. This procedure will be described in section 6.6.6.

- **Reassembly fragments**

An IP packet might be to big for some parts of the network. In that case, the servers will divide it into several smaller IP packets according to the Maximum Transmission Unit (MTU). This is called fragmentation. In order to obtain the original packet the receiving terminal must reassemble the packets. In order to do this the fragmentation offset and IP identification (IP ID) fields must be extracted and processed. The IP ID is the same for all the fragments and the fragmentation offset shows the order of the fragments. There are also flags saying if the packets has been fragmented or not. Another flag shows if the fragment is the last.

- **Handle time-outs**

If a fragment gets lost, a request for a retransmission must be sent after a certain time period.

- **Check protocol field and demultiplex data**

The protocol field shows the transport layer protocol used. The valid values in my protocol set-up are 1=ICMP, 2=IGMP, 6=TCP, 17=UDP. When the protocol field has been checked the data can be directed to the correct transport layer data buffer.

- **Check lengths**

There are two types of lengths involved in IP processing. One describes the header length, which is used to find out where the data starts. The other describes the total length which is used to see if all fragments have been received. The names of the length fields differ between the two IP versions, but the length information is essentially used in the same way.

- **Process options**

There are a number of different fields remaining that has to be processed. Among them are IP v6 extension headers, IP v4 options, and IP v6 flow labels.

6.6.5 ICMP and IGMP

ICMP normally communicates error messages, exceptions, and other conditions that require attention. IGMP is used for setting up and managing multicast groups.

- **Compute header checksum**

Same procedure as for IP checksum calculation.

- **Check ICMP version and type field**

The version field is normally 1. The packet is a query if the type is 1, and it is a reply if the type is 2.

- **Check IGMP type and code field**

This header information describes the type of request or reply. The parameter field should be processed if it is included.

- **Send ICMP payload to application**

Some control messages should be passed on to the application for further processing.

6.6.6 TCP

- **Extract Ports and check connection**

The Source Port (SP) and Destination Port (DP) together with the IP SA, IP DA and transport layer type define a connection. A receiving terminal should discard all packets not belonging to a valid connection. For some connections not all of the fields must be matched, instead these fields are wild-cards. The procedure is described in more detail in chapter 11.

- **Check Sequence number and reorder data**

The sequence number describes where in the data buffer the current payload should be placed.

- **Extract acknowledgment field and trigger a reply payload**

- **Check and process options and flags**

  Including the finish flag.

- **Update connection state variables and timers**

  This is the complex traffic flow management, controlling all traffic. This is done in software due to the high demands on flexibility.

6.6.7 UDP

The main difference between UDP and TCP is that UDP is connection-less.

- **Extract Ports and check connections**

  Similar to the TCP task. Called a connection although we only check if the port is open, not if a connection has been established.

- **Extract length field**

  To know when the whole payload has been received.

- **Calculate header checksum**

6.7 Processing task classification

Regardless of the protocols used in a computer network, there exists a common set of processing tasks that each node in the network must perform in order to make the network function correctly. There are also a number of tasks that are specific for the protocol. Since each protocol gives a unique set of requirements on the processing this common set can not be a bit level correct processing description. Instead it describes the nature of different processing tasks for different protocols on different layers. The main reason for grouping the processing tasks is to analyze flexibility and throughput requirements for a larger set of protocols, before deciding on resource needs. This results in a classification of a task based on its demand on the processing resources, not based on protocol or layer type. Consequently I have chosen to classify the processing tasks using five task groups.

6.7.1 Parsing

In order to perform any processing on a packet, the first step is to recognize the packet and the set of rules to apply on it. This identification of a packet and its rule-set is commonly known as parsing. During transmission, both the payload and the set of rules can easy be passed between different processes or processors. Hence, this group of processing tasks, mainly concerns protocol reception. During protocol reception, the first task is to detect a valid packet and its data alignment. To identify and detect a packet, software algorithms or hardware devices can be used. Secondly the information describing which rule-set to apply on the packet

must be extracted. The rule-set is normally stipulated by the protocol type and other parameters such as addresses stored in the packet header.

6.7.2 Control flow selection

Decisions on how to process the packet can be made based on the parsed information. This decision procedure normally consists of selecting a number of operations to perform. These operations can then be performed in hard- or software. The control flow selection is by nature very different from a standard Harvard architecture, where the program flow defines the operations to apply to the data in the data path. Here the program flow is selected based on the data extracted from the data path. The control-flow-selection can be implemented in software as a pseudo-code illustrated by figure 6.9.

If the protocol processing (or parts of it) is implemented in hardware the control flow selection does not (only) select the program flow. Instead the control flow selection is implemented as a configuration and selection of hardware that meet the requirements of the current protocol. The common tasks within this group can be listed as:

- Program flow selection
- Hardware configuration
- Hardware multiplexing
- Hardware scheduling

6.7.3 Transport control

The purpose of the transport control is to provide a secure and regulated communication between a sender and a receiver. In the telecommunication community this is commonly known as signaling. The transport control in a network terminal normally consists of two main types:

- Acknowledgement control including timer triggered events
- Receiver management e.g. policing, filtering, and QoS providing

```
case protocol_type is
    when A jump to flow1
    when B jump to flow2
    when C jump to flow3
    when D jump to flow4
```

Figure 6.9: Control flow selection pseudo-code.

The acknowledgment control must produce acknowledgments and send them back to the sender when packets have been received. It also includes keeping track of incoming acknowledgments to see if the transmitted packets have been successfully received. In a network terminal, the receiver management normally only consists of a decision to store or discard the received packet. It may also include a prioritizing of the incoming packets. Discard decisions are then made based on the parsed information.

6.7.4 Data processing

The purpose of data processing is to support the transmission control so that a secure and error-free channel is maintained. Since this type of processing tasks is only controlled by the packet type and is very throughput demanding, it has been given its own processing task group. These data intensive tasks are normally included in the lower layers in the ISO/OSI reference model. Some common types of data processing are:

- CRC calculation
- Checksum calculation
- Coding/Decoding
- Encryption/Decryption

6.7.5 Data stream management

In network terminals the data stream management consists of different kinds of buffer management. When transmitting a certain amount of data it may have to be divided into several packets and then sent to the correct address. The data must be reassembled and then stored in the correct memory location at the receiving terminal. In network infrastructure nodes (e.g. routers) the data stream management includes deciding where to forward packets.

6.8 Processing task allocation

The different processing tasks described in section 6.6, are allocated to different processing units within the PNI according to the table below. In general one can say that the C&C together with the XAC FP and CMAA is responsible for parsing while program flow selection is handled by the C&C by itself. Data processing is handled by FPs while the transport control is provided by the C&C. The datastrem management is handled by the C&C.

FPs, C&C, and CMAA are described in more detail in the following chapters.

Table 1: Allocation of processing tasks listed in section 6.6.

Protocol	Task	Processing hardware
Ethernet	Calculate CRC	CRC FP
	Check Ethernet DA	XAC FP, C&C
	Check type field	XAC FP, C&C
	Demultiplexing of data	C&C together with CMAA
	Extract length field	XAC FP
	Length counting	C&C, LC FP
ARP/RARP	Update ARP table	Micro controller
	Trigger ARP reply	Micro controller
IP	Check version	XAC FP, C&C
	Calculate header checksum	Checksum adder FP
	Extract and check IP DA	XAC FP, C&C
	Extract IP SA	XAC FP
	Reassembly fragments	CMAA, Micro controller
	Handling time-out of fragments	Micro controller
	Check protocol field	XAC FP, C&C
	Demultiplexing	C&C
	Check lengths	XAC FP, C&C, CMAA
	Process options	Micro controller
ICMP/IGMP	Compute header checksum	Checksum adder FP
	Check ICMP version and type	Micro controller
	Check IGMP type and code	Micro controller
	Demultiplexing	CMAA, Micro controller
TCP	Extract ports	XAC FP
	Check connection	CMAA (or C&C)
	Check sequence number and reorder data	Micro controller
	Extract acknowledgment and trigger reply	Micro controller
	Check and process options and flags	Micro controller

Table 1: Allocation of processing tasks listed in section 6.6.

Protocol	Task	Processing hardware
	Update connection state variables and timers	Micro controller, hardware timer
UDP	Extract ports	XAC FP
	Check connection	CMAA (or C&C)
	Extract and manage length	XAC FP, C&C
	Calculate header checksum	Checksum adder FP

6.8.1 Research methodology

The title of this thesis includes the word Processing, indicating that the main focus not has been on creation of a demo processor. Instead methods and architectures for protocol processing in general have been investigated with focus on terminals. Protocol processing has been possible to perform for many years by now. This thesis aims at improving the processing bandwidth of the data intensive tasks executed in the PPP. Hence, thourough analysis of the timing in performance critical parts of the PPP has been done. Using static timing analysis on the layout of such a performance critical part (e.g. FP, CMAA, or C&C) an upper bound on the performance of the entire PNI can be found. In order to verify the functional behaviour of each part, behaviour level VHDL code has been used as interfaces. One of the main problem with my structural models have been the lack of memory compilers for RAM, ROM, and CAM. Hence, only registers or logic have been used in the structural blocks. Remaining memories have only been modeled at behavioural level. Consequently no structural model nor a layout of the complete PPP have been completed. Neither have the micro controller been included in the timing simulations of the PPP.

Reference

[6.1] N. Person, "*Specification and Implementation of a Functional Page for Internet Checksum Calculation*", Master's thesis, Linköping University, March 2001, No.: LiTH-IFM-EX-959

[6.2] T. Henriksson,, "*Hardware Architecture for Protocol Processing*", Licentiate Degree Thesis, Linköping Studies in Science and Technology, Thesis No. 911, December 2001, ISBN: 91-7373-209-5.

[6.3] T. Henriksson, H. Eriksson, U. Nordqvist, P. Larsson-Edefors, and D. Liu, "*VLSI Implementation of CRC-32 for 10 Gigabit Ethernet*", In proceedings of The 8th IEEE International Conference on Electronics, Circuits and Systems, Malta, September 2-5, 2001, vol. III, pp. 1215-1218

[6.4] D. Liu, U. Nordqvist, and C. Svensson, "C*onfiguration-Based Architecture for High-Speed and General-Purpose Protocol-Processing*", Proceedings of IEEE Signal Processing Systems 1999, pp. 540-547, Taipei

[6.5] T. Henrikson, U. Nordqvist, and D. Liu, "*Specification of a Configurable General-Purpose Protocol-Processor*", Proceedings of CSNDSP 2000, Bournemouth
[6.6] M. Yang, A. Tantawy, "*A design methodology for protocol processors*", Proceedings of the Fifth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems, 1995, pp. 376 -381
[6.7] "*Building Next Generation Network Processors*", White paper, Sep 1999, Agere Inc., http://www.agere.com/support/non-nda/docs/Building.pdf
[6.8] D. Husak, "*Network Processors: A Definition and Comparison*", White paper, C-PORT, http://www.cportcorp.com/solutions/docs/netprocessor_wp5-00.pdf
[6.9] U. Nordqvist, T. Henriksson, D. Liu, "*CRC Generation for Protocol Processing*", in proceedings of the NORCHIP 2000, Turku, Finland

7

Application Domain Specific Processing

The proposed Programmable Protocol Processor uses highly dedicated data- and control-paths. Hence it is very important to carefully examine the application domain, including protocol set and traffic patterns, in order to find an optimal architecture. This optimization is done during the design phase. After the design phase the protocol processor can be used for different protocols and tasks within the specified application domain using configurable and programmable hardware. This chapter lists architectural design choices available. Further this chapter specifies how to make the protocol processor dedicated for a specific application domain.

7.1 Processor type

As stated earlier, the great challenge with wire-speed protocol processing lies in the decreasing time and number of clock-cycles available for processing of each packet. In a 10 Gbps TCP/IP/Ethernet connection with minimal packet sizes there are almost 15 million packets arriving every second, i.e. each packet has to be processed in 67 ns using a 32 bit wide data-path. While no general-purpose processor can meet this constraint there are two fundamentally different ways of implementing special-purpose processors for high-performance protocol processing.

Consequently there are two different ways of implementing the PPP. Which implementation method to choose depends on the application (type of network terminal) that the PNI is going to be used for.

- **Pipelined processors.** A pipelined processor can run at high clock frequency and therefore manage many instructions for each network clock cycle. In the experimental pipelined TIPP-processor [7.4] from Intel the execution core runs at a clock frequency which is 32 times higher than the network and classification memory speed. Hence, theoretically up to 608 clock cycles are available for each packet. This number is in fact reduced by the use of low-frequency CAM-memories (classification and reassembly). Further these 608 clock cycles include the cycles lost as pipeline penalties when conditional branch instructions are executed. The pipeline penalty is dependent on the depth of the pipeline. Since conditional branches are common instructions in protocol processing this may seriously limit the performance. Hence there is a trade-off between high-frequency (deep pipeline) and branch cost (few or no pipeline-stages).
- **Synchronized dataflow processors.** A synchronized data-flow processor runs at the same clock-frequency as the incoming data. This requires each instruction to be more complex compared to pipelined processors. Further pipeline penalties are not allowed since the data- and instruction-stream must be fully synchronized at all times. In order to achieve this the processor must move complex tasks to self-contained accelerators. Further, conditional branches must be executed in a single clock cycle. If these two conditions are met the processor can manage high-performance protocol processing operating at a relatively low clock frequency.

7.2 Software/Hardware partitioning

When application specific hardware is used for protocol processing tasks it is necessary to carefully profile the tasks covered. The primary goal is to find out which tasks to process in HW and which to leave to the SW. In most NPs, hardware accelerators are used for the most common tasks and protocols. Some tasks are easily recognized as suitable for hardware acceleration while others are not.

One of the tasks that can be implemented in hardware, software or in a hybrid, is classification. As illustrated by figure 7.1 classification can be implemented in different ways. From a performance point of view a hardware implementation is beneficial since the classification can be performed in parallel over several packet header fields or one field with several constant values. The number of comparisons that must be performed is proportional to the number of protocols covered times the number of fields included in the classification. Hence, the classification, which is a part of the program flow selection, should be performed in hardware in order

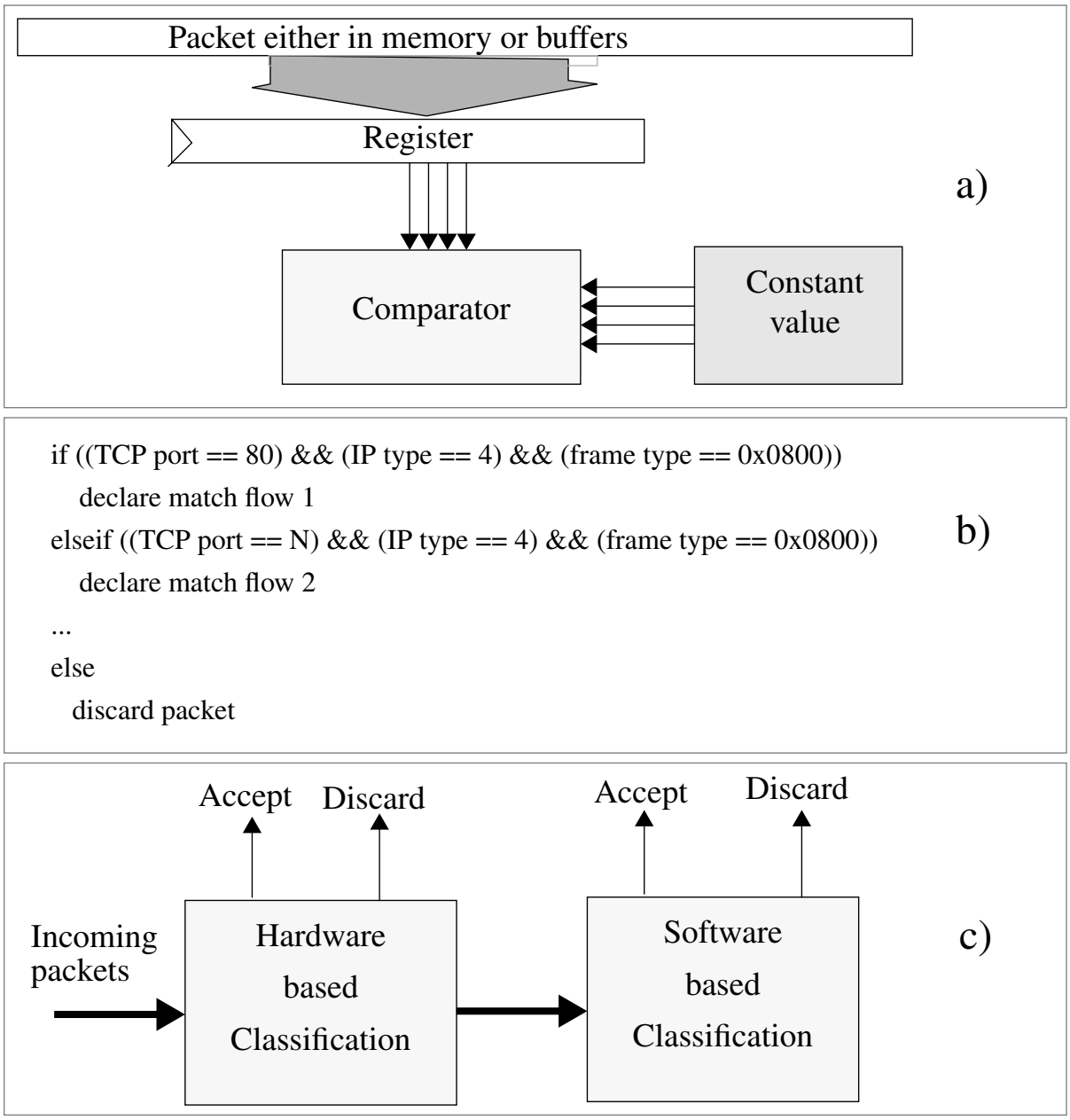

Figure 7.1: Packet classification using a) HW, b) SW, c) a hybrid implementation.

to achieve high bandwidth protocol processing. If however the number of protocols and fields tested in the classification process is too large, a hybrid system must be considered. The problem with a hybrid system is that the processing latency is data (i.e. packet) dependent. Packets classified in hardware have a different processing latency compared to other packets. Since the hybrid system only can be optimized for the average packet stream, buffering must be used.

7.3 General-purpose protocol processing

For general purpose protocol processing with a significant number of simultaneous connections and protocols, this thesis proposes a high-frequency pipelined

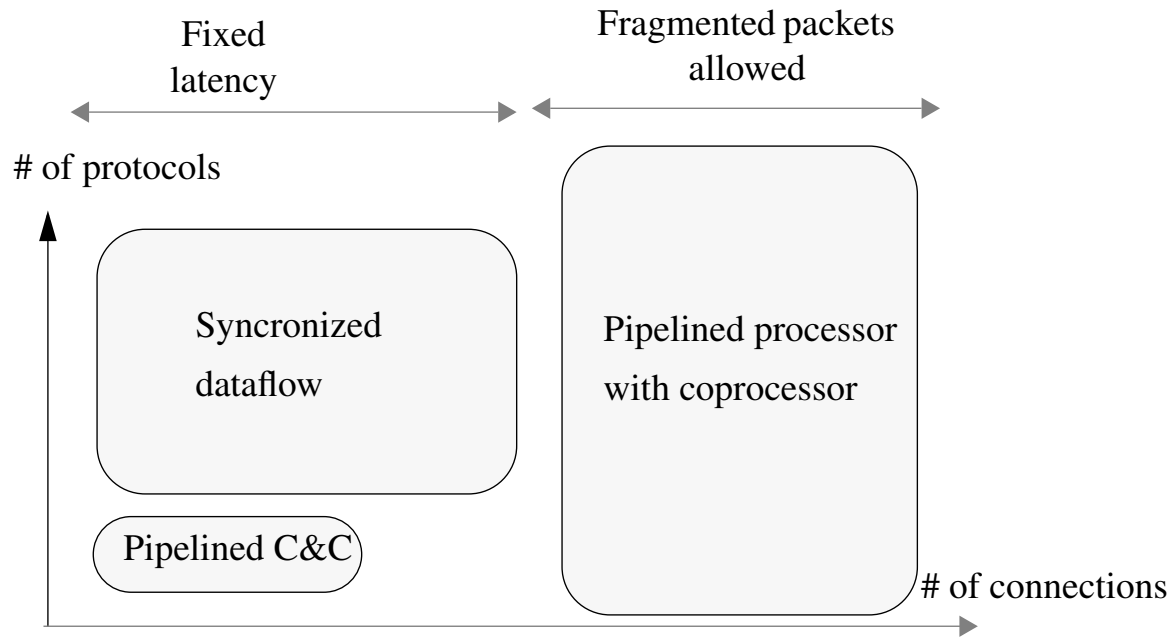

Figure 7.2: Aplication Domain Specific Protocol Processor implementation methodology.

implementation of the counter and controller. A pipelined processor, even with an efficient speculation methodology, gives larger area, power, and memory consumption compared to a syncronized dataflow implementation. For general purpose protocol processing, there are however no alternatives available.

Due to the large number of simultaneous connections a CAM assisted classification accelerator denoted CMAA is proposed in chapter 11. This accelerator can offload the reassembly handling of received fragmented packets. This accelerator is also responsible for controlling the PPP access to inter-packet information stored in the control memory.

7.4 Special purpose processing

In protocol processing a significant part of the tasks is program flow selection, i.e. determine what to do based on header information. If the number of entries to the case-statements is limited, the program flow selection can be implemented using parallel dedicated hardware components. This allow for the program flow selection to be performed in a single clock cycle. If all conditional branches are supported by this type of program flow selection hardware, all branch penalties can be eliminated. This means that the C&C unit can use the same clock frequency as the data-flow pipeline (i.e. the network clock). By using the same clock frequency the requirements on synchronization with the data flowing through the PPP becomes very strict. Since the data is only available to each FP during one clock cycle it is necessary to start and stop the processing in the FPs at the exact

clock cycle. The concept of synchronized protocol processing was introduced by my colleague Dr. Henriksson in ([7.1], [7.2], and [7.3]) even if the proposed implementation in chapter 10 differs.

The use of a single clock domain simplifies the layout and reduces synchronization problems between FPs and the control path (C&C) since the need for synchronization registers is eliminated.

Although synchronized dataflow implementation of the C&C normally is the best in special purpose terminals, chapter 10 shows that for extremely simple terminals with a minimum amount of program flow selection, the pipeline implementation outperform the synchronized dataflow alternative. This is depicted in figure 7.2.

7.4.1 Application domain restrictions

The synchronized data flow implementation of the C&C is suitable for certain network terminals. This type of terminals must be simple in order to use such a dedicated architecture. This means that synchronized data flow PPP can not support:

- **Reassembly of fragmented packets**

Reassembly of packets received out of order requires inter-packet information. Hence, this task is not suitable for processing in the PPP. Therefore it should be done either in the μC or in the host (as Microsoft proposes) even if the hardware architecture and memory access scheme of these two are not dedicated for reassembly.

- **A large number of simultaneous connections, i.e. complex address check**

The number of simultaneous connections determines the complexity of the program flow selection (case-statement). Since this case-statement processing is implemented in parallel dedicated hardware the number of connections can not be higher than the number of values that can be compared with packet header field data in the XAC FP. E.g. only one IP destination address plus multi/broad-cast addresses are allowed. Note that the addresses of course are configurable.

- **A large number of protocols**

The restriction on the complexity of program flow selection instructions also imply that the number of PPP supported protocols are kept low.

If these bulleted restrictions are acceptable for the intended terminal application the synchronized data-flow implementation gives a very efficient implementation. Due to the dedicated implementation with reduced control overhead, the C&C is both area and power efficient. Moreover it can operate at a relatively low frequency which is beneficial from many points of view.

These requirements on simplicity of the terminal does not mean that it is hard to find suitable special purpose terminals where a synchronized data flow processor can be used. E.g. reception of streaming media in a simple terminal connected directly to a high-speed network.

Reference

[7.1] T. Henriksson, U. Nordqvist, and D. Liu, "*Embedded Protocol Processor for Fast and Efficient Packet Reception*", in proceedings of International Conference on Computer Design, Freiburg, Germany, pp. 414-419, Sep 2002

[7.2] T. Henriksson, D. Liu, and H. Bergh, "*Method and apparatus for general-purpose packet reception processing*", US Patent Application 20030039247, February 27, 2003

[7.3] Tomas Henriksson, "*Intra-Packet Data-Flow Protocol Processor*", Doctor degree Thesis, Linköping Studies in Science and Technology, Thesis No. 813, May 2003, ISBN: 91-7373-624-4

[7.4] Y. Hoskote, V. Erraguntla, D. Finan, J. Howard, D. Klowden, S. Narendra, G. Ruhl, J. Tschanz, S. Vangal, V. Veeramachaneni, H. Wilson, J. Xu, N. Borkar, "*A 10GHz TCP offload accelerator for 10Gbps Ethernet in 90nm dual-VT CMOS*", IEEE International Solid-State Circuits Conference, 2003. Digest of Technical Paper, 14.7.

8

Memory issues

Using a data-flow based architecture puts the memory system in focus. Buffer design is very important for the overall performance of the PPP. While demultiplexing incoming packet streams, data transfers and memory accesses consume a lot of power. Except for the PPP, also the micro controller and the host access the memory system. Hence, it is neccisary to view the memory design as a system issue rather than just a part of the datapath optimization process.

8.1 Host memory

The host memory is the final destination for payloads of accepted packets received in the terminal. From this memory, payload data can be accessed by the host application. If the protocol processor cannot be accommodated on the same chip as the host, accepted packets need to be intermediately stored in a packet buffer memory on the NIC. This packet buffer then replaces the host memory in our memory hierarchy model. The size and throughput of this host memory/packet buffer must be sufficient for storage of all incoming payload data. Further, reassembly requires additional storage capacity for reordering buffers. Design of the host memory is not within the scope of this thesis.

8.2 Control memory

The control memory stores inter-packet information such as control state variables. It stores all data used by the micro controller for its control intensive computing. Further, control oriented packets (including headers) such as ARP, RARP,

ICMP, and IGMP is stored in this memory. They are then accessed and processed by the μC. The control memory will be further discussed in chapter 11 with focus on low latency access.

8.3 Program memory

The program memory is rather small due to the dedicated instruction set and extensive use of accelerators. Instead of storing many instructions which determine the PPP behavior, the program memory only contains conditions and flags. These flags are enough to control the FPs since they have internal (distributed) control.

The program memory is accessed by the C&C. Alternative ways of doing this is discussed in chapter 10. Due to the small size of the proposed program memory, low power and low latency access are possible.

8.4 Packet buffering

As described in chapter 7, there are three different ways of providing wirespeed protocol processing. Either the processing is perfectly synchronized with the network clock which means that no buffering of incoming packets is needed. This thesis does however suggest that a small number of registers are used in order to lower the fan-out from this register chain. This dataflow based packet buffering is further discussed in section 8.5.

8.4.1 Wire speed processing

Protocol processing in NPs is normally optimized the most common type of packets. While processing of such packets only requires a small amount of instructions, worst case (exception type of packet) may require a huge amount of instructions. Hence, all packets must be buffered and then processed when processing resources are ready. At average, the NP process the packets at the same time as they arrive but it can not handle bursts of worst case traffics if the buffers used is not very large. NPs capable of processing packets at average as fast as they arrive are known as wire speed NPs. A wire-speed NP must have enough buffering capacity to store a number of packets when a burst of worst case packets arrive. The main focus of this thesis is to avoid using large buffers for storage of packets that is not going to be used.

8.4.2 Packet level synchronization

If the processing is only synchronized with the data at packet level, the data must be stored so that the protocol processor can access the same data during several clock cycles. Normally protocol processing involves access of packet headers much more often than access of payload data. Hence, allowing certain (header)

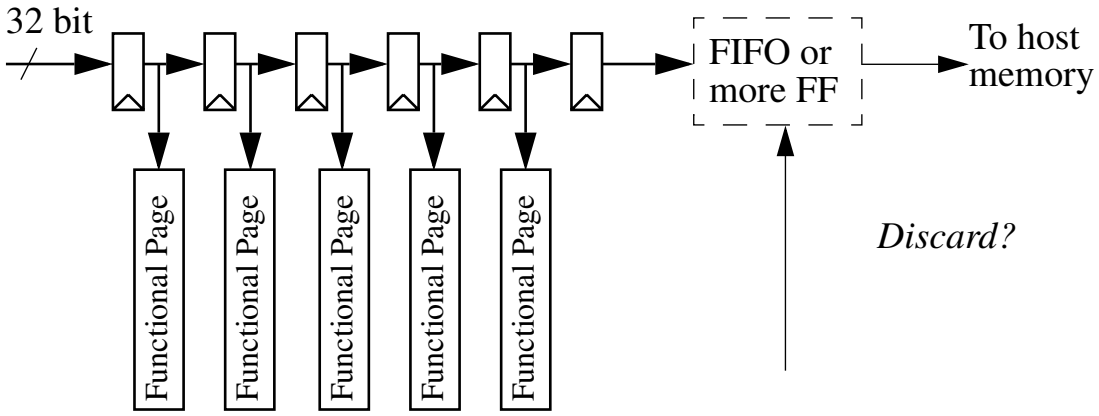

Figure 8.1: The input flip-flop chain. The chain of flip-flops enables access to the data stream with low fan-out. The chain can be eliminated if the number of functional pages is low. The RAM based FIFO in the end is used to save power.

fields to be accessed during several clock cycles can improve the performance of the protocol processor.

There is a hard deadline in the packet level synchronized protocol processing, the arrival of the next packet. If the processing of one packet is not finished the processor must discard the next packet. A protocol processor that can finish worst case processing of one packet before the next arrives uses packet level synchronization.

In order to relax the requirements and give the processor time to finish processing one packet, still avoiding this unacceptable discarding, the normal approach is to increase the minimal allowed packet size handled. This however deteriorates the overall network performance.

Using the proposed architecture it is possible to avoid packet level synchronization if fragmentation and complex classification is not supported. This issue will be discussed in chapter 11.

8.5 Dataflow based packet buffering

When data arrives from the PHY interface (GMII) to the PPP it streams through a chain of 32 bit wide flip-flop registers illustrated by figure 8.1.

The purpose of using a flip-flop chain instead of a normal RAM based buffer, is that we want to keep the fan-out from the registers as low as possible. If the number of functional pages (FP) is large, we have to use a larger number of registers in the chain, but if the number of FPs is moderate or low the chain can be minimized. The lower bound on the number of registers is then set by the decision latency of the header processing. We must know what to do with the data when it streams out of the register chain. One alternative is of course to store all incoming data until we receive a discard signal from the C&C. The problem is that we do not want to send

the payload data to the host memory before we know if it should be discarded or not. If the packet contains control information, it should be sent to the control memory for further processing in the μC. To write all packets in the host memory would introduce unnecessary interrupts and waste power.

8.6 FIFO buffer

To reduce these interrupts and the overall power consumption, a low power RAM based FIFO is inserted in the end of the input buffer. This FIFO will hold the packet until the PPP can decide if it should be discarded or accepted. The length (size) of this FIFO can be optimized to find the most power efficient architecture. This optimization must consider the number of flip-flops used, the processing latency, and traffic patterns. The number of FPs and the architecture (thereby the decision latency) are set by the protocol coverage at design time.

8.6.1 Memory hierarchy

According to figure 6.2, received packets will stream through the fast path (i.e. PPP) and if the packet is accepted, the payload data will be stored in the host (main) memory, where the application running on the host CPU can access the data for further processing. The memory where the packets are stored must be at least 1 MB in order for TCP datagrams to be accommodated, even if some fragments are delayed.

There exist many different ways of organizing the memory architecture in the terminal. The main memory organization alternatives are illustrated by figure 8.2.

- Case a): Off chip packet buffer memory on a NIC.
- Case b): On chip packet buffer memory on a NIC.
- Case c): Large shared off-chip memory on motherboard.
- Case d): On-chip packet buffer on the motherboard CPU chip.

The optimal solution depends mainly on the available chip area. On-chip memories will always be desirable if they are possible to accommodate. Minimal area will be used if the protocol processor and the host can use a shared memory. The problem with shared memories is that the host processing (i.e. the application) will be interrupted leading to performance degradation. Remember that the purpose of the protocol processor is to offload and accelerate the application processing running on the host. If the PNI can not be accommodated on the same chip as the host, an on-chip packet buffer should be used for intermediate storage of the incoming packets. The Packet Buffer Memory (PBMEM) should be able to store at least 100 ms of traffic, i.e. 1 MB in a 10 Gb/s network. The host memory organization is not a part of this research project but as illustrated in figure 8.2 the proposed PNI platform can be integrated with a wide variety of host architectures.

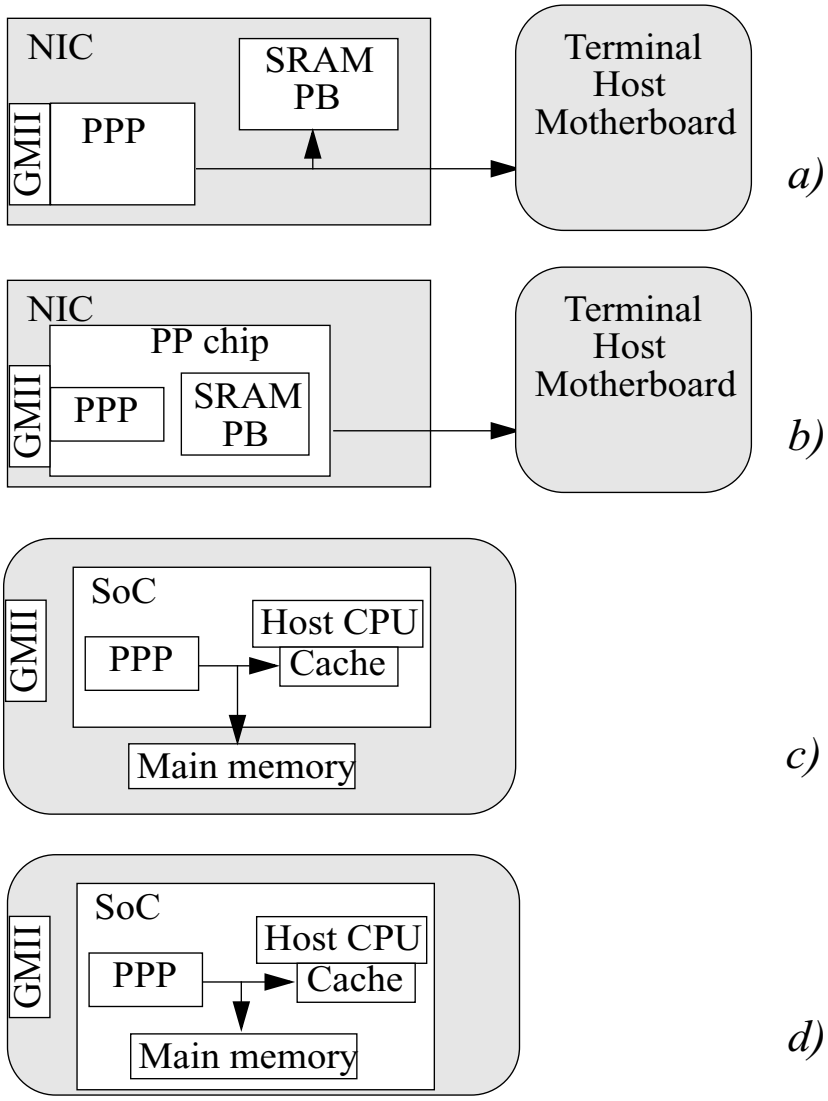

Figure 8.2: Packet buffer memory organization alternatives. The memory access energy cost is dependent on the selected memory organization.

8.6.2 Optimization strategy

As depicted in figure 8.3, the received packets streaming through the input buffer into the PB SRAM will be discarded if the protocol processor detects any error in the packet. Packets will be discarded if the address, port numbers, checksums, CRC, etc. is erroneous. This is the reason why it makes sense to add an extra FIFO buffer in the fast path input buffer (and use three different packet buffering components in the architecture) in order to lower the power consumption.

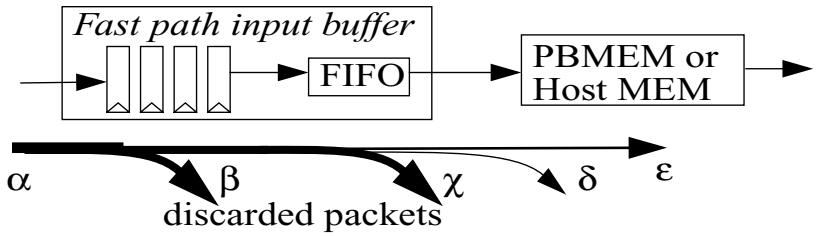

Figure 8.3: Packet flow during reception. Some packets are discarded while streaming through the three proposed packet buffers. These are the register chain and the FIFO in the fast path and the memory buffer. Accepted packet streams through all buffers and into the host main memory.

Example: *According to figure 8.3 b),* α *packets are received and streams through the registers in the input buffer. Then* β *packets will be discarded leaving* $\alpha-\beta$ *packets to be buffered in the SRAM FIFO. While streaming through the FIFO another* χ *packets will be discarded. According to this finally* $\varepsilon=\alpha-\beta-\chi-\delta$ *packets will be accepted. This means that only* ε *packets will be stored in the large memory. Hence, to add the FIFO buffer actually lowers the energy cost.*

8.6.3 Optimization parameters

Since the energy cost for the memory access will be dependent on which memory organization is chosen, the optimal input buffer architecture is dependent on the selected memory organization. Other parameters determining the sizes of the three input buffer stages for a certain memory system environment are:

- Network traffic. The length, protocol type, and transmission error rate.
- Discard decision latency. Depends on type of packet, type of error, and the fast path implementation.
- Buffer (dynamic) energy cost. Depends on size, clock frequency, activity, implementation method and process.

8.6.4 Simulations

In order to optimize the average energy consumption in the three input buffer stages, all the parameters listed in section 8.6.3 must be measured as accurately as possible.

Table 8.1: Packet size distribution in the average traffic flow.

1-10 B	11-490 B	491-510 B	511-1500B
40%	30%	20%	10%

Table 8.2: Discard decision latency for different protocol processing tasks.

Protocol type	Decision latency Address Check # Clock cycles	Decision latency Checksum # Clock cycles
Ethernet	4	packet length in B/4
ARP	5	packet length in B/4
RARP	5	packet length in B/4
TCP	10	packet length in B/4
UDP	10	packet length in B/4
IPv4	6	8
ICMP	6	10
IGMMP	6	10

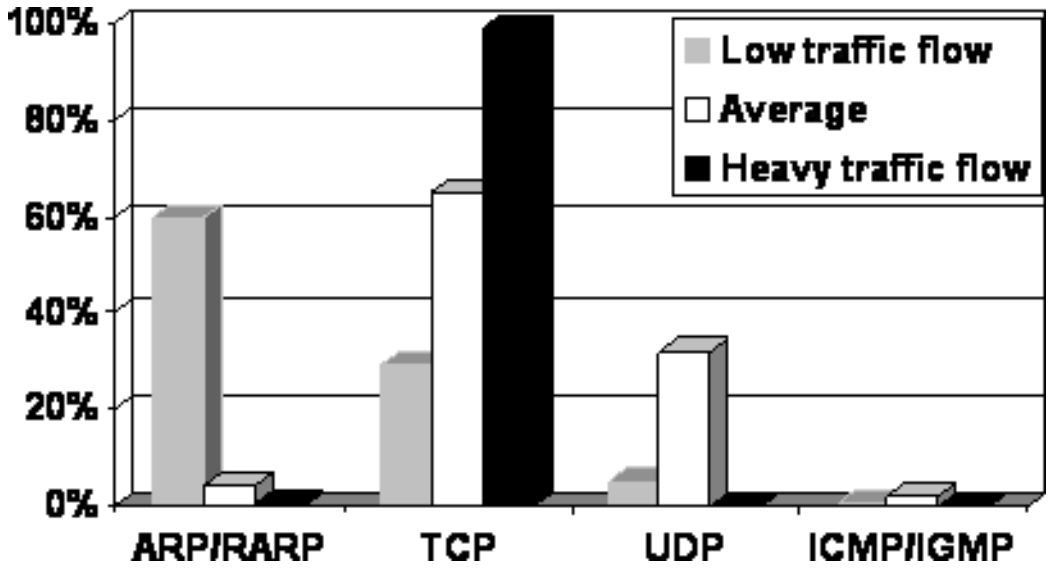

Figure 8.4: Packet type distribution in various traffic flows. At low traffic ARP/ RARP dominates while TCP dominates during heavy traffic flows. Note that no traffic from real time applications has been modeled.

The first parameter to investigate is the network traffic. We have used a network analyzing tool [8.10] to do this. Further we haven chosen to investigate three cases of network traffic flows. These are low traffic (no transmission, only reception), heavy traffic (file transfer application) and the average traffic flow (router traffic downscaled). The protocol type distribution for these three different traffic situations are illustrated by figure 8.4.

Further, packets length is distributed according to table 8.1, in the average traffic flow. The decision latency for different protocols and errors is specified by the program controlling the PPP operation. In the simulated architecture the decision

latency is given by table 8.2. The energy cost for access to the register chain, the SRAM based FIFO and the packet buffer SRAM memory has been modeled using cost functions mainly from [8.8]. In order to estimate the energy cost for access of a memory system (eq. (1)) accurately, it is essential to carefully model the effective capacitance according to eq. (2).

$$E_{MEM} = \frac{1}{2} Vdd^2 \cdot C_{eff} \quad \textbf{(1)}$$

$$C_{eff} = C_{onchipSRAM} + C_{offchipSRAM} + C_{interconnect} \quad \textbf{(2)}$$

We have assumed a 0.35-µm standard cell process for the flip-flops in the register chain and the use of a memory library optimized for low power. In order to reduce fan-out in the PPP and thereby enabling high speed operation, at least five 32 bit wide registers have to be used. Using MatLab a number of different simulations have been made. During the simulations the error-rates, packet buffer energy costs, and traffic flows have been used as input parameters to find the input buffer configuration that gives minimum average energy consumed per packet received.

8.6.5 Simulation results

Using information on energy consumption of different FIFOs, registers, and memories [3-10], the resulting optimal solution with minimal (dynamic) energy consumed per packet is displayed in table 8.3. Our simulations show that discarded packets will consume more energy if the number of register and FIFO stages are low. The reason is that not so many packets can be discarded before the data has streamed into the big packet buffer memory (1 MB SRAM). Meanwhile the accepted packets will consume less energy per packet if the number of register and FIFO stages are reduced. The reason is that all accepted packets (except the control oriented e.g. IGMP, ICMP, ARP, RARP, which are transferred to the control memory after the FIFO buffer) will be stored in the packet buffer memory anyway. The simulations states that in order to optimize the packet buffer power consumption, we have to choose a small register-based buffer, a medium sized FIFO buffer and a large PB SRAM. The average per packet energy consumed in the three buffer stages is in the magnitude of a few µJ (minimal 1.3). Further the table shows that the memory organization, which decides the energy cost for a memory access, has little effect on the result. Instead it is mainly the packet size and discard latency that determine the optimal buffer configuration.

Table 8.3: Optimal packet buffer organization for different traffic flows and memory organizations. Minimal number of register chain stages is 5. Memory organizations defined in figure 8.2.

Traffic flow type	Memory organization	# of FIFO stages	Energy savings
Low	a)	125	12%
	b)	125	13%
	c)	125	13%
	d)	125	14%
Heavy	a)	375	15%
	b)	375	15%
	c)	375	15%
	d)	375	16%
Average	a)	256	24%
	b)	256	27%
	c)	276	24%
	d)	264	29%

Table 8.3 also shows that the average energy consumption in the three buffer stages is significantly reduced compared to if no FIFO buffer would have been used. Note that the number of FIFO stages to use should be a factor of two. If a better distribution of packet lengths is used the results would be 128, 256, or 512 respectively. In a traditional type of input buffer the energy cost would increase instead. Simulations also indicate that the network error-rate has a low impact on the resulting buffer organization although it has a large impact on the energy savings.

From a general system power consumption perspective, it is natural to reduce the average power consumption instead of the peak power consumption. The average traffic flow in a network terminal is however hard to estimate since it is strongly depending on the applications running on the host CPU. So far we have assumed that the packets are non-fragmented. If fragmented packets are allowed we can expect the optimal number of register stages to grow since the decision latency is much higher.

Figure 8.5 illustrates how the energy consumption depends on the size of the FIFO and register chain.

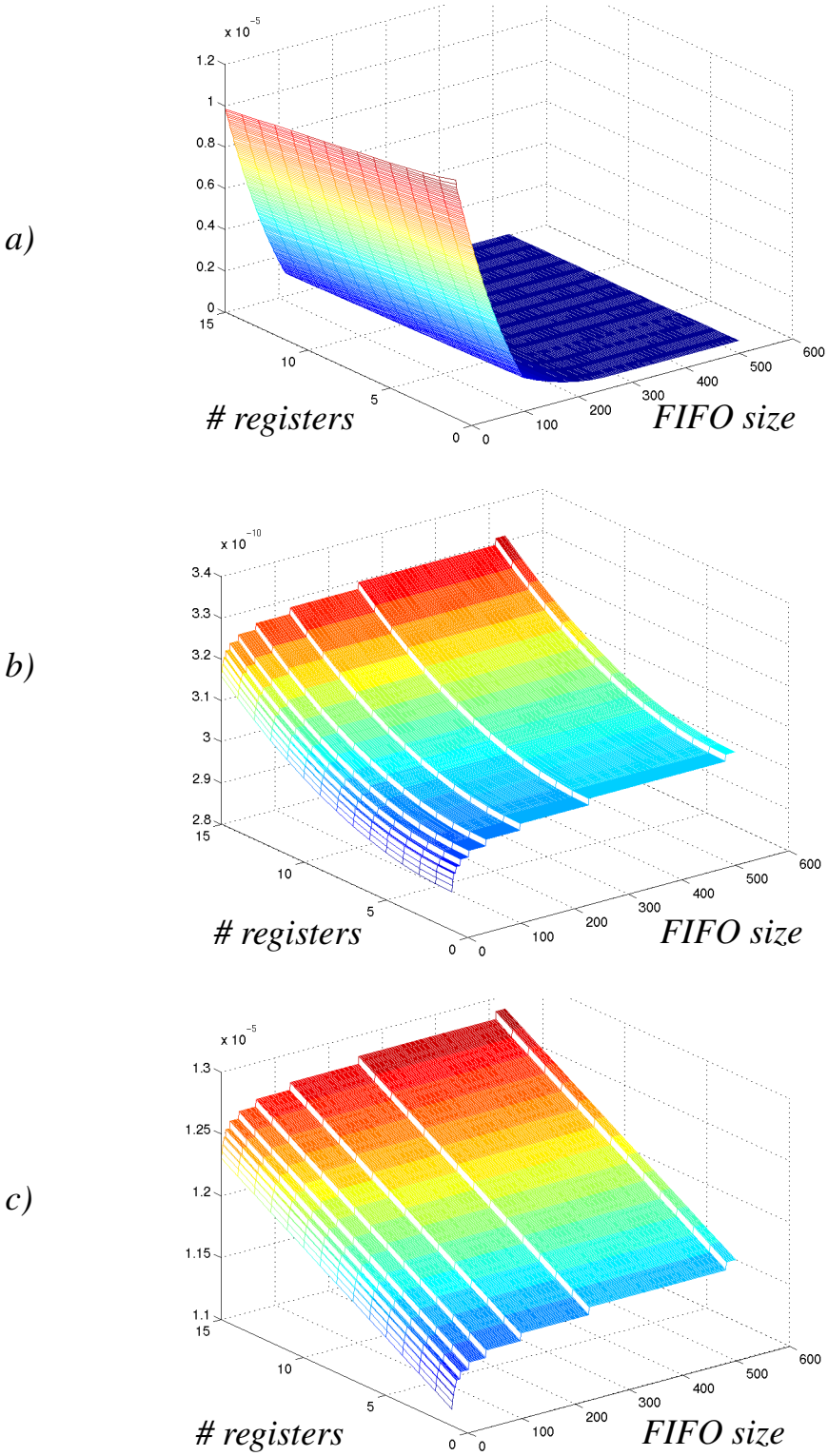

Figure 8.5: Average energy consumption for a) discarded packet, b) discarded packet due to lost fragments, c) accepted packets.

8.6.6 Architectural conclusions

This chapter proposes that an extra FIFO is added to the input buffer in the proposed fast path architecture. This reduces the power consumption because some

packets can be discarded before they are stored in a large energy consuming memory. Our simulation results indicate that the optimal input packet buffer organization is to use a minimal number of registers (maximally five) together with a relatively large RAM based FIFO (256 X 32bit).

Rough assumptions made on different error rates in a typical network are acceptable since they do not strongly effect the optimal buffer configuration. The same thing applies to the estimations of energy costs for access of the various buffer components.

Reference

[8.1] "IPTraf IP Network Monitoring Software", on the www, http://iptraf.seul.org/
[8.2] S. Barbagallo, M.Lobetti Bodoni, D.Medina, G.De Blasio, M.Ferloni and D.Sciuto, “A Parametric Design of Bui-in Self-Test FIFO Embedded Memory,” IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, pp221-229, 1996.
[8.3] A.R. Feldman and T Van Duzer, “Hybrid Josephson-CMOS FIFO,” IEEE Transaction on Applied Superconductivity, Vol.5, No.2, pp2648-2651, 1995.
[8.4] G.N.Pham and K.C.Schmitt, “A High Throughput, Asynchronous, Dual Port FIFO Memory Implemented in ASIC Technology,” Second Annual IEEE ASIC Seminar and Exhibit, pp, 1989.
[8.5] M. Hashimoto, etc., “A 20ns 256K*4 FIFO Memory,” IEEE J. of Solid State Circuits, Vol.23, No.2, pp490-499, 1988.
[8.6] Austria Micro Systems, “0.35 m CMOS Libraries (C35)”, on the www, http://asic.austriamicrosystems.com/data books/index_c35.html
[8.7] NEC, “Data-sheet mPD431000A”, on the www, http://www.nec.com
[8.8] F. Catthoor et. al., “Custom memory management methodology”, Kluwer Academic Publishers, 1998
[8.9] A. Berkeman, “ASIC Implementation of a Delayless Acoustic Echo Canceller”, Ph.D. Thesis, Lund University, Sweden, Dec. 2002
[8.10]Ethereal Network Analyzer, on the www, http://www.ethereal.com/
[8.11]T. Henrikson, U. Nordqvist, and D. Liu, “Specification of a Configurable General-Purpose Protocol-Processor”, Proceedings of the CSNDSP, 2000, Bournemouth

9

Hardware Acceleration

9.1 Functional pages

The functional pages are all dedicated hardware blocks with limited configurability. Since they are dedicated for the processing they do, they have very little control overhead, which saves power and allows for the FPs to have a very high throughput. The functional pages are responsible for the data intensive processing

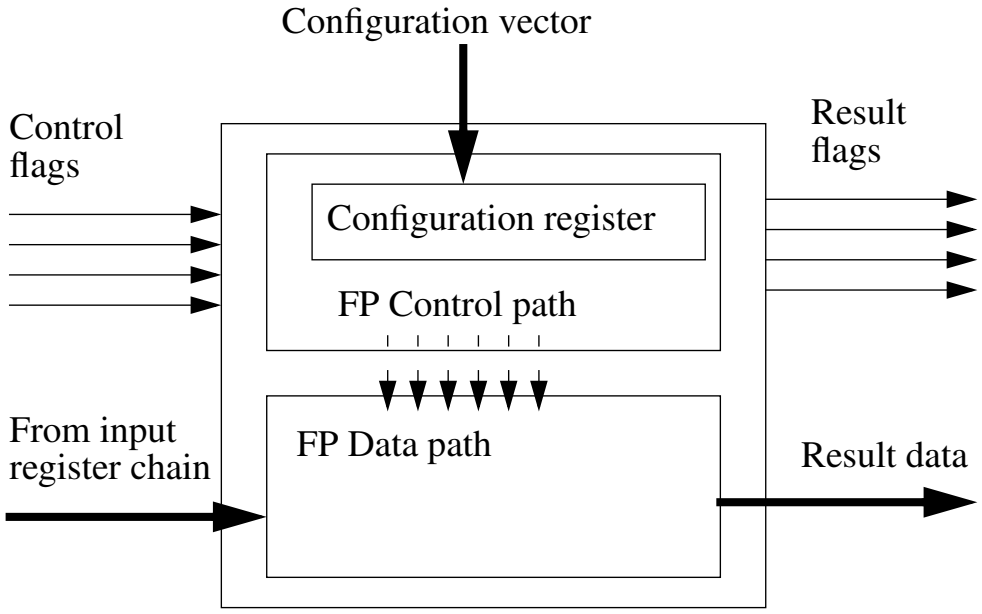

Figure 9.1: Funtional page interface. FPs are controlled by flags produced in the C&C. The primary output consists of result flags, e.g. discard flag. Some FP also produces result data.

in the PPP. Their processing tasks are very diverse both in terms of type and complexity. Hence, the FP hardware becomes very different. What they have in common is that they all have a limited configurability within their specific application area. Further they all are controlled by the counter and controller. The control normally consists of flags that start and stop the processing in the functional page. The typical FP interface is illustrated by figure 9.1.

The output from a functional page normally consists of flags. Some functional pages also produce result data that will be exported to other parts of PPP. The FPs can be configured using configuration registers. This configuration only takes one, up to a few clock cycles. Configuration vectors are produced in the micro controller, which also controls the configuration procedure. Using the protocol suite discussed earlier a small set of functional pages has been selected and implemented to process data intensive parts of the protocols. They are:

- **Extract and compare (XAC) FP**
- **CRC FP**
- **MII parallelization FP**
- **Checksum FP**
- **Length counter FP**

9.2 CRC FP

In order to provide error detection in communication networks a method called Cyclic Redundancy Check has been used for almost 40 years. This algorithm is widely used in computer networks of today and will continue to be so in the future. The implementation method has on the other hand been constantly changing. The CRC FP is very important for the overall performance of the PPP.

The CRC FP must support wire-speed processing. Further it must support all CRC polynomials defined by the protocols covered.

A configurable CRC FP has been implemented and manufactured using a standard cell process. The CRC solution proposed is very flexible and it can process a large set of CRC algorithms. If the bandwidth of such a configurable solution is not sufficient, a fixed logic, parallel data CRC implementation can be used.

9.2.1 The CRC algorithm

Cyclic Redundancy Check is a way of providing error control coding in order to protect data by introducing some redundancy in the data in an controlled fashion. It is a commonly used and very effective way of detecting transmission errors during transmissions in various networks. Common CRC polynomials can detect following types of errors:

- **All single bit error**
- **All double bit errors**
- **All odd number of errors, provided the constraint length is sufficient**
- **Any burst error for which the burst length is less than the polynomial length**
- **Most large burst errors**

The CRC encoding procedure can be described by equation 1.

$$V(x) = S(x) + x^{n-k}U(x) \quad \text{(EQ 1)}$$

V(x) is the n bit long data word transmitted and it consists of the original data and U(x) followed by a codeword S(x) called the CRC-sum. S(x) are the extra bits added to a message in order to provide redundancy so that errors during transmission can be detected. The length of the S(x) is denoted the constraint length. The constraint length of the most commonly used CRC polynomials are 8, 16, 24 and above all 32 bits. S(x) is computed according to equation 3.

$$X^{n-k}U(x) = a(x)g(x) + S(x) \quad \text{(EQ 2)}$$

$$\frac{X^{n-k}U(x)}{g(x)} = a(x) + \frac{S(x)}{g(x)} \quad \text{(EQ 3)}$$

S(x) is by other words the remainder resulting from a division of the data stream and a generator polynomial g(x). Since all codewords are divisible with g(x) the remainder of the left hand side of EQ 3 has to be zero for a real codeword.

The actual coding procedure is the same on both the receiving and transmitting end of the line. The CRC encoding/decoding principle is illustrated by figure 9.2.

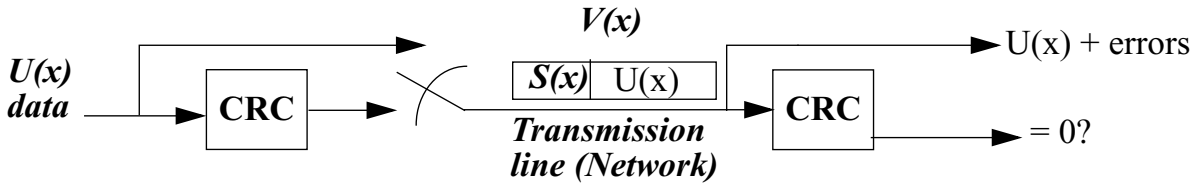

Figure 9.2: Principle of error detection using the CRC algorithm.

As can be seen in figure 9.2 the receiving NT perform a CRC-check on the incoming message and if the result (S(x)) is zero, the transmission was error free. One more practical way of solving this is to compute the CRC only for the first part of the message U(x), and then do a bitwise 2-complements addition with the computed checksum S(x) on the transmission side. If the result is non-zero the receiver will order a retransmission from the sender.

9.2.2 Traditional implementations

This section introduces the commonly used and presents one new architecture for implementation of the CRC algorithm.

- **Software (SW) Solution:** [9.4][9.2] The CRC algorithm can always be implemented as a software algorithm on a standard CPU, with all the flexibility reprogramming then offers. Since there in most communication network terminals exists a CPU, the SW-solution will be cheap or free in terms of hardware cost. The drawback is obviously the computational speed since no general purpose CPU can achieve the same troughput as dedicated hardware. The processing load might also be a problem for the host processor in many applications.
- **Serial ASIC Solution:** Linear Feedback Shift Register (LFSR) with serial data feed [9.21] has been used since the sixties to implement the CRC algorithm, see figure 9.3. As all hardware implementations, this method simply perform a division and then the remainder which is the resulting CRC checksum, is stored in the registers (delay-elements) after each clock cycle. The registers can then be read by use of enabling signals. Simplicity and low power dissipation are the main advantages. This method gives much higher throughput than the SW solution but still this implementation can not fulfill all the speed requirements of today's network nodes. Since fixed logic is used there is no possibility of reconfigure the architecture and change the generator polynomial using this implementation. Several loop-connections schemes and reset alternatives exist.

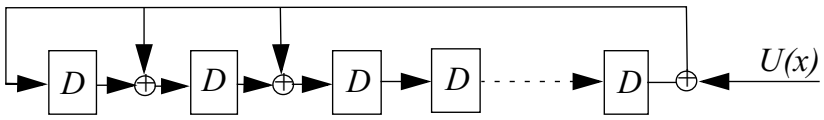

Figure 9.3: Linear Shift Serial Data Feed

- **Parallel ASIC Solution**: In order to improve the computational speed in CRC generating hardware, parallelism has been introduced [9.3], [9.5], [9.6], [9.10], [9.12], [9.13]. The speed-up factor is between 4 and 6 when using a parallelism of 8. By using fixed logic, implemented as parallelized hardware, this method can supply for CRC generation at wire speed and therefore it is the pre-dominant method used in computer networks. The parallel hardware implementation is illustrated by figure 9.4. If the CRC polynomial is changed or a new protocol is added, new changed hardware must be installed in the network terminal. That would be very expensive. The lack of flexibility makes this architecture unsuitable for use in a protocol processor.

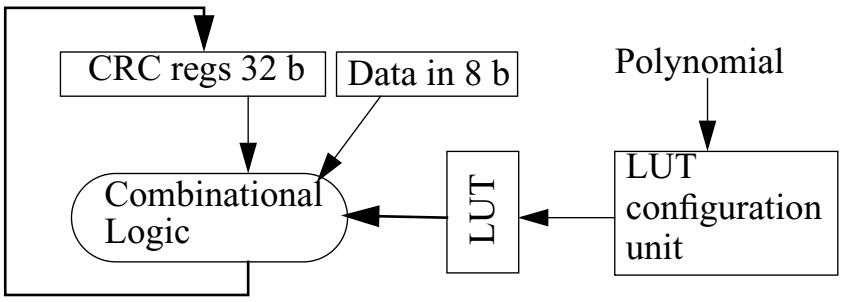

Figure 9.5: Look-Up Table based configurable hardware.

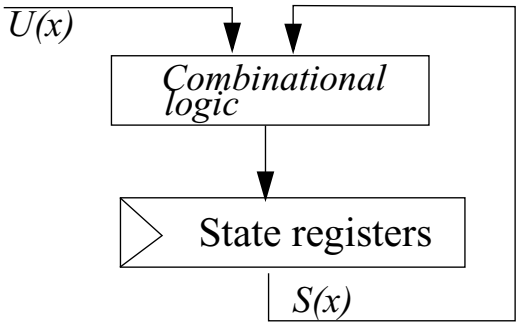

Figure 9.4: Parallel Fixed Logic Implementation

- **LUT based solution** One way of implementing configurable hardware is by using Look-Up-Tables (LUTs) as proposed by [9.4], [9.13] and [9.3]. The architecture is illustrated by figure 9.5.This implementation can be modified by using a larger or smaller LUT. If the size of the LUT is reduced the hardware cost in terms of power consumption and area will be reduced but in the same time the Combinational logic will be increased so the effect will be cancelled. The optimal solution has not been derived. This solution enables some configurability since we can change the polynomial by changing the content of the LUT memory. However there is no possibility to adjust for different constraint lengths.

9.2.3 Evaluation of implementation alternatives

10 different implementation alternatives of the CRC algorithm, including one CPU RISC-based SW-implementation, have been examined. They have been described using Mentor Graphics Renoir and VHDL, synthesized and optimized using Build Gates from Cadence and the place and route was done using Ensemble P&R from Cadence. The technology used is AMS 0.35 µm.

Since most network protocols are byte based, using a parallelism of more than eight, complicates the CRC implementation, even if other parts of a protocol processor might run on other clock frequencies using for example a 32 bit wide input stream. It is possible to use a higher degree of parallelism but if it can be avoided, it should be.

The fixed logic and parallel input implementation is the fastest, as seen in table 1 . This complies with results reported in earlier work. We can also see that the LUT-based method gives about twice the speed of the Configurable Radix 16 implementation at the cost of a 4.5 times higher area. A big part of the area in the LUT based architecture is the LUT registers, but the power consumption will anyway be considerably higher than the power consumption in the Radix-16 implementation. In many upcoming applications, such as HIPERLAN [9.16], [9.17], the power consumption will be crucial. The speed supported by the Radix-16 implementation exceeds 0.6 Gbit/s, which is sufficient since today NT applications do not demand higher troughput. Since the logic in that specific implementation dominates and the connection delay is quite small, there will be a considerable increase of the speed powered by downscaling in future technologies. The speed-up factor due to scaling s will be up to s^2 which means that even protocols as 10-GEthernet which will come in the future can be supported by the Radix-16 implementation [9.14] thanks to scaling.

Conflicts with other processes make interlayer processing difficult, not to say impossible when using the SW algorithm run on a CPU. This means that even if the SW algorithm alternative can be implemented on a high-performance CPU that provides the speed that is needed, it is not suitable for protocol processors such as those described by [9.7] and [9.8].

Because of the superior performance of a parallel ASIC implementation, it will be used for implementation of network-core components. The concept of using several ASIC implementation as Functional Units in a protocol processor and just letting the processor turn on the CRC that is currently used, as in VLIW architectures, might also be of interest although you then have no configurability for support of new protocols.

Software solutions for low speed protocols will also be used for low-speed applications, but an increasing area of applications demands high-speed configurable protocol processing, including CRC generation. Further, software solutions lacks power efficiency. A hardware architecture that can fulfill this specifications is the structure based on Look-Up tables as proposed in [9.5].

Table 1: Comparison between different CRC implementations. Pads are not included in the area computation.

CRC implementation	Polyn. Length	Area [mm^2]	Max Clk freq. [MHz]	Max Speed [Mbit/s]
Serial Input - fixed Ethernet Polynomial	32	0.014	413	413
Serial Input - any polynomial	32	0.017	369	369
Serial Input - any polynomial	16	0.011	355	355
Parallel(8) Input - any polynomial	32	0.061	109	875
Parallel(8) Input - any polynomial	16	0.038	130	1039
Parallel(8) Input - fixed Ethernet Polynomial	32	0.035	208	1663
Parallel(8) Input LUT Based	32	0.225	169	1358
Configurable Radix-16 CRC - any polynomial	**32**	**0.050**	**166**	**663**
Configurable Radix-16 CRC - any polynomial	**16,24,32**	**0.052**	**153**	**612**
SW Pure RISC (43893 clk cycles / 1500 Bytes)	any		600	164

9.2.4 Proposed CRC implementation

Another novel implementation method, the *Radix-32 Configurable CRC Unit*, has been implemented and manufactured. The architecture combines configurable and parallel hardware, which makes it suitable for use in a protocol processor.

- **Configuration**

By noticing that any polynomial of a fixed length can be represented by implementing the CRC using a LSR with switches on the loop back as illustrated by figure 9.6, a configurable hardware can be implemented using NAND-gates to implement the switches.

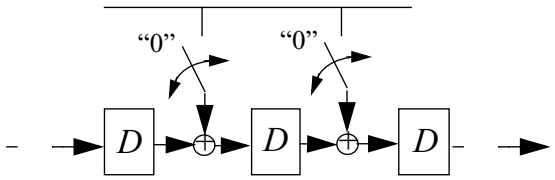

Figure 9.6: Configuration by use of switches in the circuit feed back wire.

This allows us to change the polynomial g(x) of a given length L by storing a bit description of the polynomial in a register. However some protocols uses CRC with

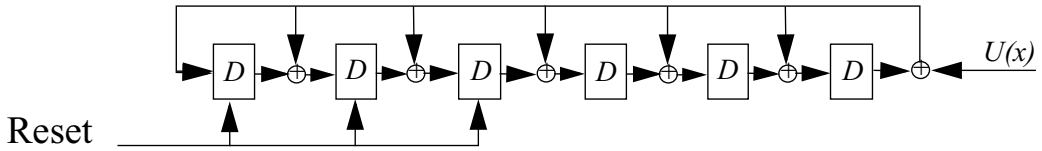

Figure 9.7: Changing constraint length using reset-signals. In this particular example the constraint length can be changed between 3 and 6 bits.

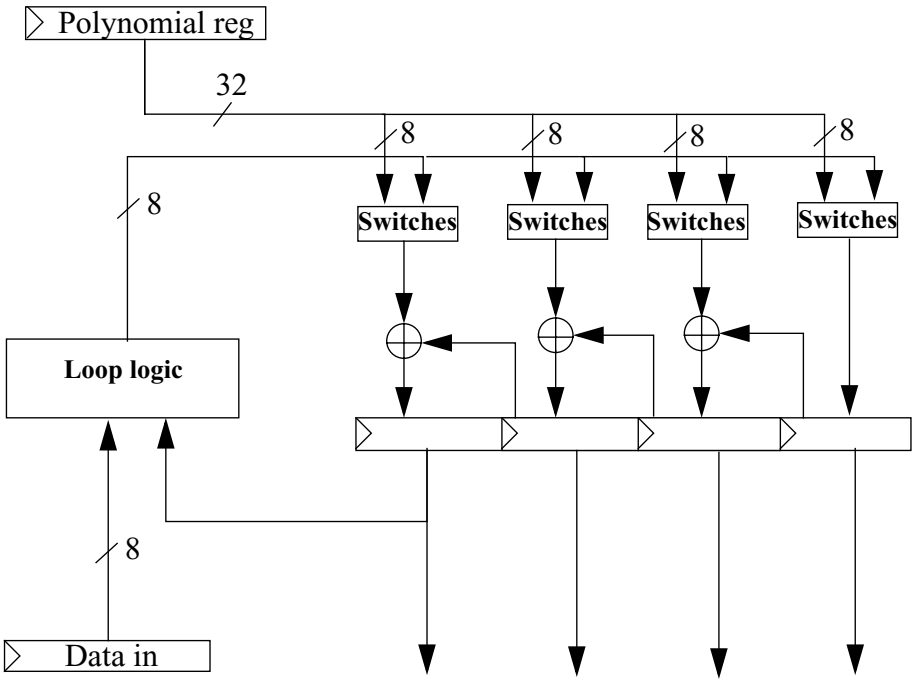

Figure 9.8: Bytewise calculations enhances the throughput of the CRC.

different constraint length L. This can be solved by using reset-signals for the flip-flops not used as illustrated in figure 9.7. All flip-flops also need a preset input since before the start of the computation all flip-flops must be set to "one".

- **Parallelism**

In order to improve the speed of the Radix-32 Configurable CRC, an 8-bit wide input data stream is used as can be seen in figure 9.8. The resulting bit in each position k in the CRC register then depends on the value of the k-8 CRC bit, the last eight CRC bits, the polynomial bit description and the input bits. The logic, which consists mainly of XOR and NAND-functions, provides the necessarily configurability.

For a fixed polynomial CRC, the throughput gets higher with wider inputs. The speedup in a configurable CRC is however not that significant. In simulations we have found it to be less than 20% when doubling the input width from 8 to 16 bits. Further it is clear that every network seen today are byte-based. This means that a

multi-byte calculation of the CRC must handle the last bits width special algorithms. That is why an 8-bit wide input has been chosen.

The polynomial registers make it possible to implement any given CRC algorithm of a given size. Using shut-down logic on parts of the circuit enables the CRC to be configured for 8, 16, 24, or 32 bit polynomials. This means that CRC polynomials for protocols such as HIPERLAN, UMTS, ATM and Ethernet are manageable.

9.2.5 Measurement results

A ***Radix-32 Configurable CRC Unit Radix-32 Unit*** was implemented using AMS 0.35 µm standard cell process.

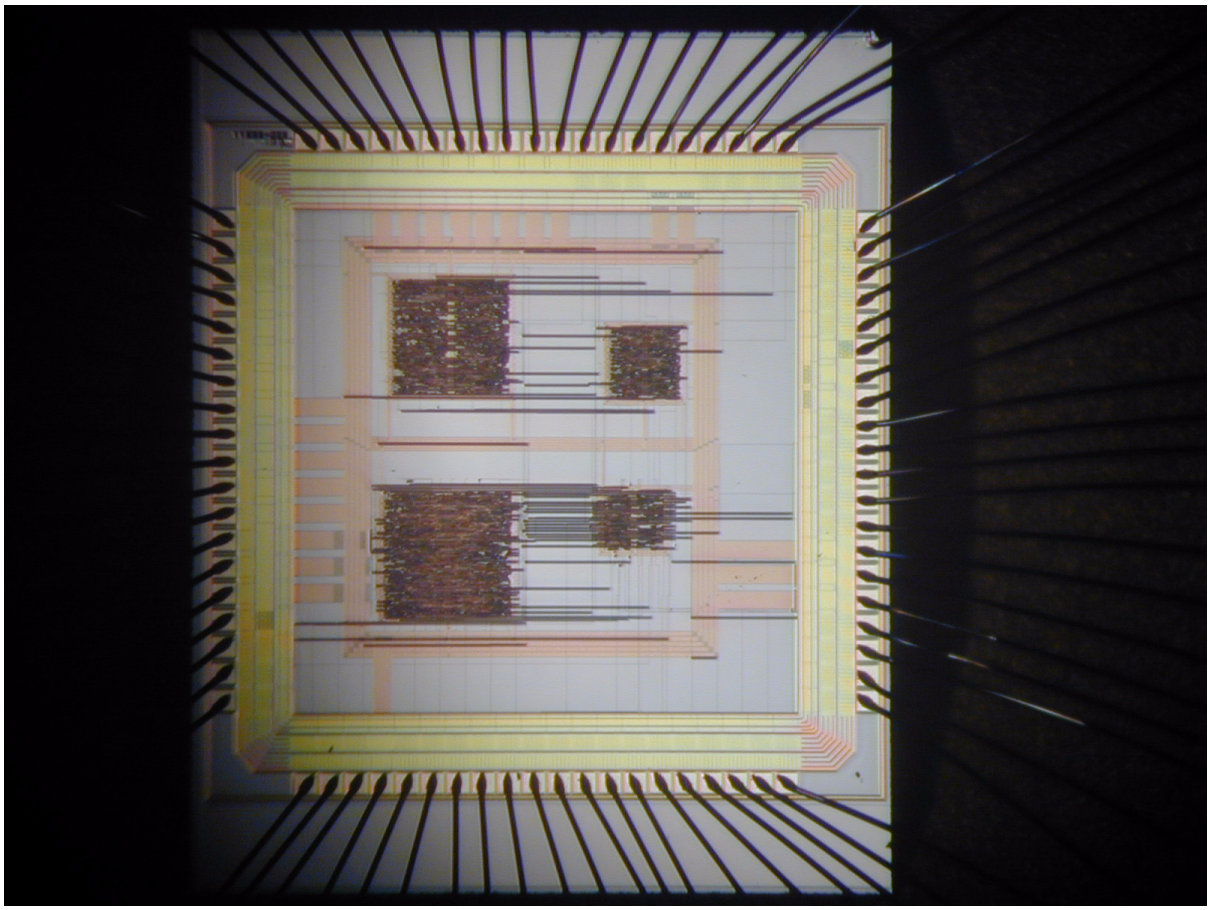

Figure 9.9: CRC test chip.

The chip manufactured contained three different CRC generators and one Parallel/Serial-converter for speedup of the input data feeding. The chip photo can be viewed in figure 9.9.

In order to use a minimum number of pads on the chip the polynomial register was implemented using a shift register. The output is serially shifted out from the chip. Due to the limited number of pads the controlability of the chip is limited. Therefore testing on random data has been used in order to verify the functionality. Another test limitation is the fact that the measurement equipment (pattern genera-

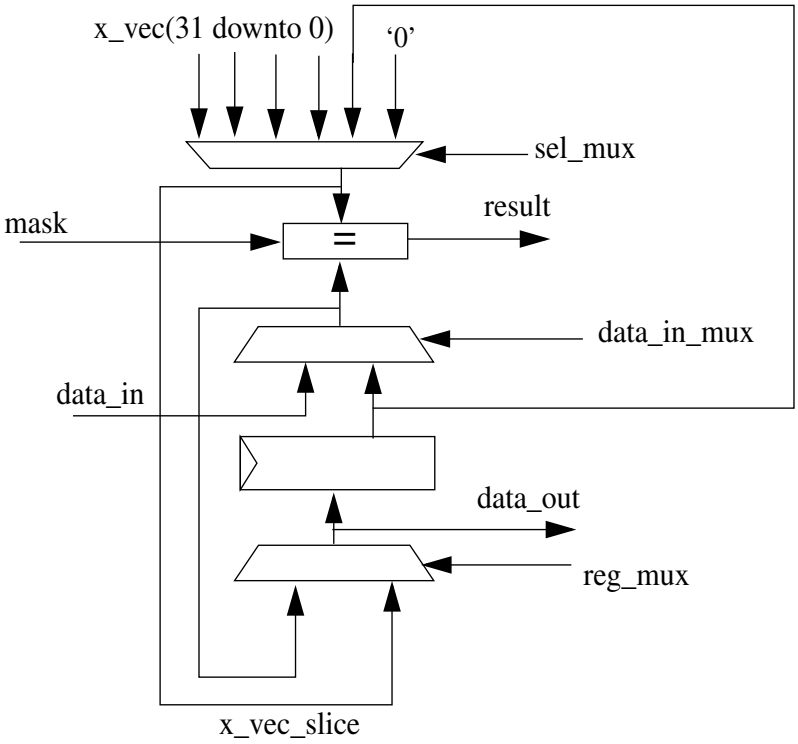

Figure 9.10: One out of four byte comparing slices in the XAC FP.

tor) available only allowed stepwise increment of the frequency. However, by reducing the supply voltage one can calculate the theoretical maximum throughput. Using this method the measured maximum frequency for the design is 189 MHz which gives a throughput of 1512 MB/s. The result from static timing analysis suggested 186 MHz, which is very close.

9.3 Extract and Compare FP

The XAC FPs are used for extraction of header information that will be used by other parts of the PPP. They are also used for comparisons between the data stream and a data vector stored in the FP. This is used when the destination address of a packet is checked. A XAC FP contains one or several 32-bit registers holding the values to compare with the extracted vector. It also contains a register holding the extracted vector. The XAC FP can compare four 8-bit values, two 16-bit values or one 32-bit value with the extracted vector. It generates a number of result flags based on the comparisons. The XAC FP is functionally divided into four slices each comparing one byte. One of these byte-slices is illustrated in figure 9.10.

In order to support more advanced program flow selection the use of more registers and compare units must be considered. Once the protocol coverage is set it is easy to decide the numbers of XAC FPs as well as the number of parallel comparators and registers in each.

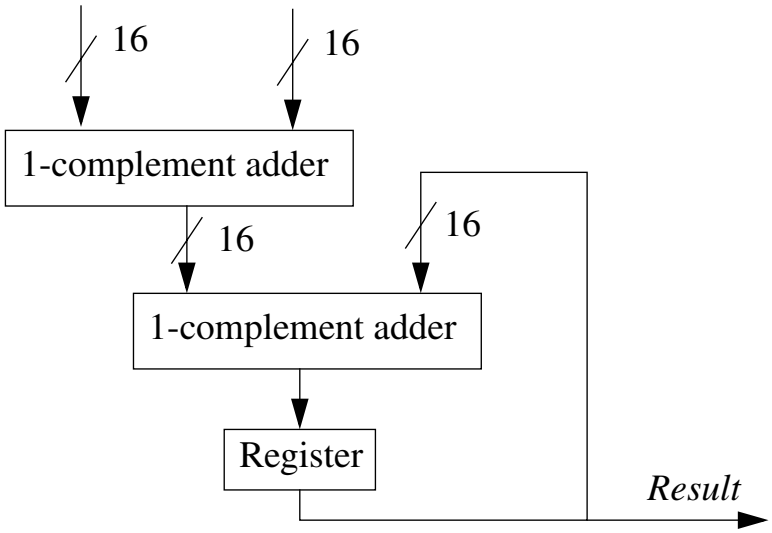

Figure 9.11: Checksum FP.

9.4 Parallelization FP

The MII parallelization FP must be included if the PNI is going to be used with the MII as interface. The MII produces 4 bit wide data. The FP is responsible for parallelization and alignment of the data, before it is passed on to the 32 bit wide input buffer chain.

9.5 Checksum FP

The Checksum FP essentially consists of a pair of 1-complement adders and is a simplified version of the FP described in [9.1]. An overview of the checksum calculating FP is illustrated by figure 9.11. According to the investigations done by my colleagues this FP can operate at 10 Gbit/s, which is unreachable using a general purpose processor.

For the IP header checksum, all the fields in the header are included in the computation. For UDP and TCP the checksum includes the IP pseudo header (figure 9.12) together with the whole UDP or TCP packet. Hence, the checksum FP must be configurable for these three types of checksum calculations.

There are basically two aproaches possible for the implementation of this accelerator. Either the control (including construction of pseudo headers), is handled inter-

IP Source address (32)		
IP Destination address (32)		
Zeros (8)	Protocol field(8)	Payload Length (16)

Figure 9.12: IPv4 pseudoheader

nally or by the C&C. If the C&C is responsible for the control, more general checksum computations are supported. In order to calculate checksums for packets divided into several fragments, intermediate checksums must be calculated, stored, and finally added together. Hence, the control is much simpler if fragmented packets are not accepted by the PNI.

9.6 Length counter FP

The length counter FP is responsible for counting the lengths of a packet. This is used to find out when all fragments have been received, and when to start and stop checksum calculations. The length counting is syncronized by the C&C, which uses it to schedule its actions. The length counter FP consists of one adder, and two registers. One register holds a stop value and the other is used as an accumulator register. When the two register values are equal, an output flag is generated. This flag is then used to stop the checksum calculation FP.

9.7 CMAA

In chapter 11 an acceleration engine named CMAA included in the PPP is discussed. The Control Memory Access Accelerator (CMAA) operates both as a memory controller and as a packet classifier. The CMAA also performs reassembly of fragmented packets. The CMAA accelerates the access to control variables stored in the control memory. This access is based on data extracted from the packet headers using the XAC FP. The core parts of the CMAA are two look-up engines (LUE). The LUE mainly consist of Content Addressable Memories (CAM). The throughput and latency of the CMAA are strongly dependent on the number of entries the LUE have in their connection tables. If the number of entries implemented after the bench-marking and optimization procedure is relatively low, e.g. 16 or 32, the latency, throughput, area, and power consumption will be acceptable. Also with this small number of entries the CMAA would significantly relax and accelerate the overall PNI processing. A deeper investigation of the application is however required before the final number of entries can be decided. This process also requires the use of network processing benchmarks. A behavioral VHDL model of the CMAA has been implemented by me. I have also implemented a structural model of the control path. The critical path of the CMAA consists of the LUE which remains to be implemented using full custom design techniques.

9.8 Hardware timer assist

Managing and updating the timers can become a large part of the processing of the TCP and IP protocols. Note that the number of timers is proportional to the number of network connections. Hence, the problem is not as severe in a network

terminal as it is in routers even tough one can imagine terminals that must support many simultaneous connections, e.g. fileservers. If the supported number of connections is high, timers must be considered to be offloaded from the micro controller since the hardware cost is limited and the hardware is very efficient. A hardware timer consist of a counter, a memory including all the timer events in an ordered (linked) list and some small control logic. Normally millisecond granularity is enough but simulations indicate that fine granularity improves the overall network performance. Since handling of timers is a task implemented by the µC, no timer FP has been implemented, but in complex terminals it may be considered.

9.9 Comments

All FPs can perform high throughput processing due to their relatively dedicated architecture. The slowest and most complex one is the CRC FP. It still manages a multi Gigabit throughput in such a mature standard cell process as the 0.35 µm AMS 3-M 3.3 V if the number of covered CRC algorithms are low.

Apart from the FPs mentioned above we can also consider other types of FPs if the protocol coverage would be changed. Examples on such FPs are cryptographic FP, coding FP etc. With a different set of FPs it would also be possible to cover wireless protocols and ATM protocols. This illustrates the generic nature of the architecture.

Reference

[9.1] N. Person, "*Specification and Implementation of a Functional Page for Internet Checksum Calculation*", Master's thesis, Linköping University, March 2001, No.: LiTH-IFM-EX-959

[9.2] .A. Perez, "*Byte-wise CRC Calculations*", IEEE Micro, Vol. 3, No. 3, June 1983, pp. 40-50

[9.3] G. Albertango and R. Sisto, "*Parallel CRC Generation*", IEEE Micro, Vol. 10, No. 5, October 1990, pp. 63-71

[9.4] T. V. Ramabadran and S. S. Gaitonde, "*A Tutorial on CRC Computations*", IEEE Micro, Vol.8, No. 4, August 1988, pp. 62-75

[9.5] T. B. Pei and C. Zukowski, "*High-speed parallel CRC cicuits in VLSI*", IEEE Transaction Communication, Vol. 40, No. 4, pp. 653-657, 1992.

[9.6] R. F. Hobson and K. L, Cheung, "*A High-Performance CMOS 32-Bit Parallel CRC Engine*", IEEE Journal Solid State Circuits, Vol. 34, No. 2, Feb 1999

[9.7] D. Liu, U. Nordqvist, and C. Svensson, "*Configuration-Based Architecture for High-Speed and General-Purpose Protocol-Processing*", Proceedings of IEEE Signal Processing Systems 1999, pp. 540-547, Taipei

[9.8] T. Henrikson, U. Nordqvist, and D. Liu, "*Specification of a Configurable General-Purpose Protocol-Processor*", Proceedings of CSNDSP 2000, Bournemouth

[9.9] C. J. Georgiou and C.-S. Li, "*Scalable Protocol Engine for High-Bandwidth Communications*", IEEE International Conference on Communications. ICC'97 Montreal, Volume: 2, 1997, Page(s): 1121 -1126 vol.2

[9.10]R. Nair, G. Ryan, and F. Farzaneh, "*A Symbol Based Algorithm for Implementation of Cyclic Redundancy Check (CRC)*", Proceedings VHDL International Users' Forum, 1997, Page(s): 82 -87

[9.11]J. Kadambi et al, "*Gigabit Ethernet*", Prentice Hall PRT, ISBN 0-13-913286-4, 1998
[9.12]G. Castagnoli, S. Brauer, and M. Herrmann, "*Optimization of cyclic redundancy-check codes with 24 and 32 parity bits*", IEEE Transactions on Communications, Volume: 41 6, June 1993, Page(s): 883 -892
[9.13]R. J. Glaise, X. Jacquart, "*Fast CRC calculation*", 1993 IEEE International Conference on Computer Design: VLSI in Computers and Processors, 1993, Page(s): 602-605
[9.14]A. P. Chandrakasan, R. W. Brodersen, "*Minimizing power consumption in digital CMOS circuits*", Proceedings of the IEEE, Vol, 83 No. 4, pp. 498 -523, April 1995
[9.15]M. Yang, A. Tantawy, "*A design methodology for protocol processors*", Proceedings of the Fifth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems, 1995, pp. 376 -381
[9.16]"Technical Specification of BRAN and Hiperlan-2. Common part.", ETSI TS 101 493 - 1, V1.1.1, 2000
[9.17]"Technical Specification of BRAN and Hiperlan-2. Ethernet Service Specific Convergence Sublayer.", ETSI TS 101 493 - 2, V1.1.1, 2000
[9.18]A. S. Tannenbaum, "*Computer Networks*", 3nd Edition, Prentice Hall PRT, ISBN 0-13-349945-6, 1996
[9.19]"Building Next Generation Network Processors", White paper, Sep 1999, Agere Inc., http://www.agere.com/support/non-nda/docs/Building.pdf
[9.20]D. Husak, "Network Processors: A Definition and Comparison", White paper, C-PORT, http://www.cportcorp.com/solutions/docs/netprocessor_wp5-00.pdf
[9.21]W. W. Peterson and D. T. Brown "*Cyclic Codes for Error Detection*", *Proc. IRE*, Jan 1961, pp. 228-235

10

Counter and Controller Implementation

10.1 Problem formulation

As stated earlier there are basically two approaches available for wire-speed protocol processing. Either a dual clock system with a high-speed pipeline processor [10.2] responsible for the protocol processing or a synchronized data-flow machine[10.1] with no pipeline penalty. Which to choose depends on the requirements on the protocol processor system set by the terminal application. This chapter will discuss implementation alternatives for the Counter and Controller unit.

10.2 Synchronized data-flow implementation

For normal instructions the synchronized C&C uses two instruction pipeline stages as illustrated by figure 10.1. It fetches instructions from the wide program memory using one pipeline stage and executes the fetched instruction in the next pipeline stage. Note that there is no Register File (RF) or write result stage in the instruction pipeline since the C&C only consumes and produces flags, not data. In order to avoid stalls, the instruction decoding stage has been eliminated and instead fully decoded instructions are stored in the program memory. Stored instructions are used directly as control signals.

The number of stored instructions can be very low due to the limited protocol coverage. In theory the number of instructions stored is maximally one plus the

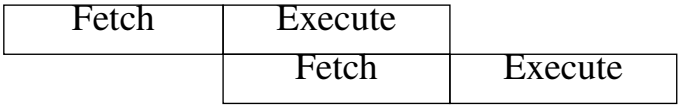

Figure 10.1: Instruction pipeline. The synchronized implementation of the C&C uses a two-stage instruction pipeline for most instructions. In order to avoid branch penalties a special unit supporting conditional branches has been added. This unit makes it possible to perform a program flow selection using a single clock cycle. If a conditional branch is taken the fetched instruction is discarded and the new instruction and fetch address are provided by the branch unit.

number of instructions required for protocol header processing times the number of protocols covered. The number of instructions required for the processing of one protocol header is equal to the length of the protocol header (in Bytes) divided by four due to the synchronized 32 bit wide data stream. Hence, nine protocols with 20 Byte long headers each, requires maximally 55 instructions. Normally this figure can be reduced significantly due to redundancy between different protocols and the fact that not all header fields are used. These instructions are stored either in the result memory of the branch unit or in the program memory. As the number of instructions are kept relatively small, the cost for these wide memories is acceptable. Further it allows the access to the program memory (fetch) to have low latency.

There are many alternative ways of implementing a hardwired case-statement. Using a simplified TCAM (STCAM: See “CAM implementation issues” on page 139) is one rather straight forward alternative. The STCAM based branch unit depicted in figure 10.3 uses the PC value and flags generated in FPs as inputs. If there is a match in one of the CAM entries, i.e. a conditional branch is taken, a new instruction and instruction fetch address are provided. Since this STCAM will be a part of the critical path of the PPP and thereby determine the maximum clock frequency it can operate at, it is very important to optimize this CAM search. The latency of a CAM search is mainly dependent on the size of the two search fields and the number of entries in the memory. The latency of the CAM search must be added to the latency of the XAC FP and Muxes to find the critical path of the PPP. An implementation of the C&C suitable for synchronized special purpose protocol processing is illustrated in figure 10.2.

Using AMS 0.35 µm standard cell library an implementation of the synchronized data-flow version of the C&C has been completed. The implemented branch unit supports 16 different conditional branches, each with four case entries. Static timing analysis of the implemented layout shows that the critical path is 10.9 ns long, which indicates that it can support wire speed processing at 2.9 Gbit/s. This

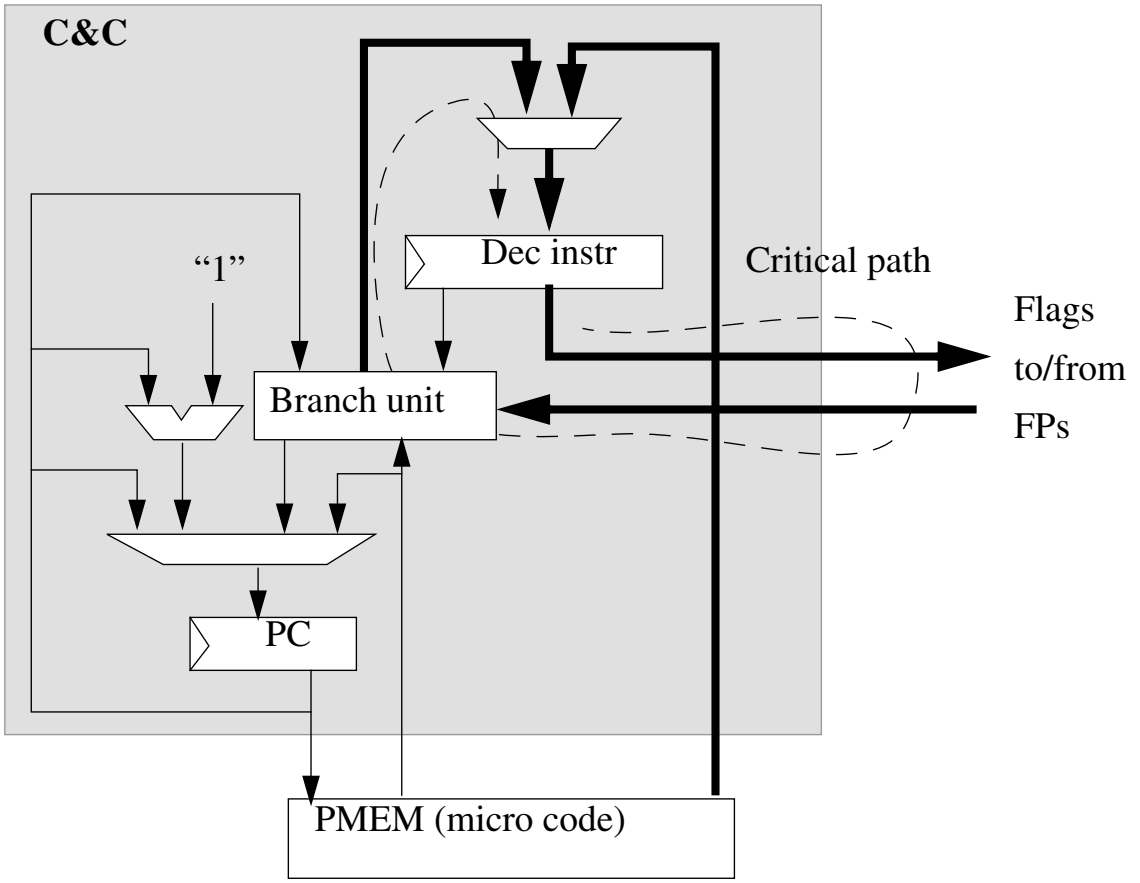

Figure 10.2: Branch unit supporting single clock cycle program flow selection. The critical path includes extraction of packet header data, comparison and branch decision.

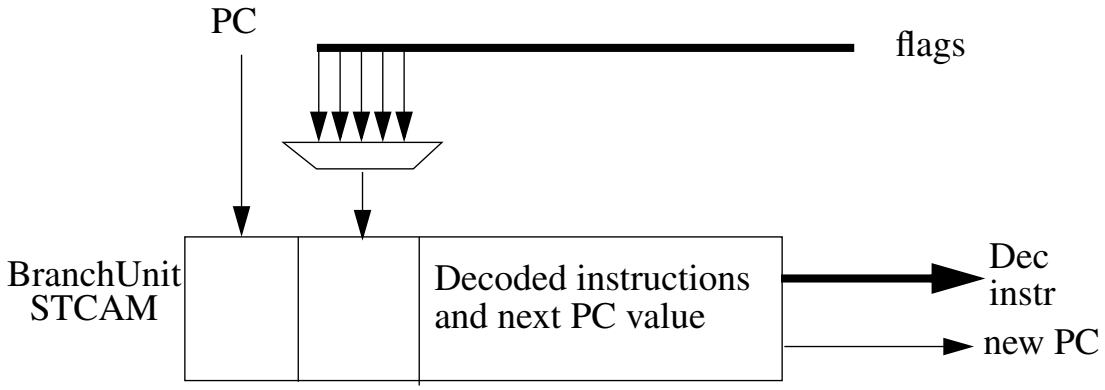

Figure 10.3: The branch unit makes its branch decision based on the program counter value and flags created in FPs.

is comparable with the performance of a configurable CRC FP implemented using the same process.

10.3 Pipelined C&C implementation

Using the set of protocols earlier described in this thesis a pipelined implementation methodology of the C&C has been done. The organization of the C&C is depicted in figure 10.4. The C&C datapath consists of a multiplier free ALU, a status register and a register file. The register file is used to transfer information between FPs, the CMAA and the C&C.

The pipelined C&C uses an application specific ISA. The ISA is very simple since most processing tasks are handled by FPs, the C&C essentially only functions as the PPP control path. The simple instruction set consists of 12 instructions. They are listed in table 1.

Table 1: ISA for the C&C

Type of instruction	Name	# of stages
Logic	And	4
	Or	4
	Not	4
Arithmetic	Add (16 bits)	5
	Sub (16 bits)	5
Transfer	Load (immediate)	4
	Move	4
Jump	Jump unconditionally	4
	Brneqz	4
	Breqz	4
	NOP	

Each instruction is 24 bit long. Logic instructions are used to extract and modify the flag information stored in the status register. Arithmetic instructions are used for checksum calculation of fragmented packets and control memory address calculation. The move instruction operate on the register file while load is used for setting of the status registers or the register file. These status registers are used to communicate with the FPs and the CMAA.

High frequency operation is enabled using pipelining according to figure 10.5. The number of pipeline stages used is five although only arithmetic instructions uses all five stages.

It requires the program memory to be pipelined since two stages are used for the instruction fetch. I have not implemented such a memory. Instead I used a behav-

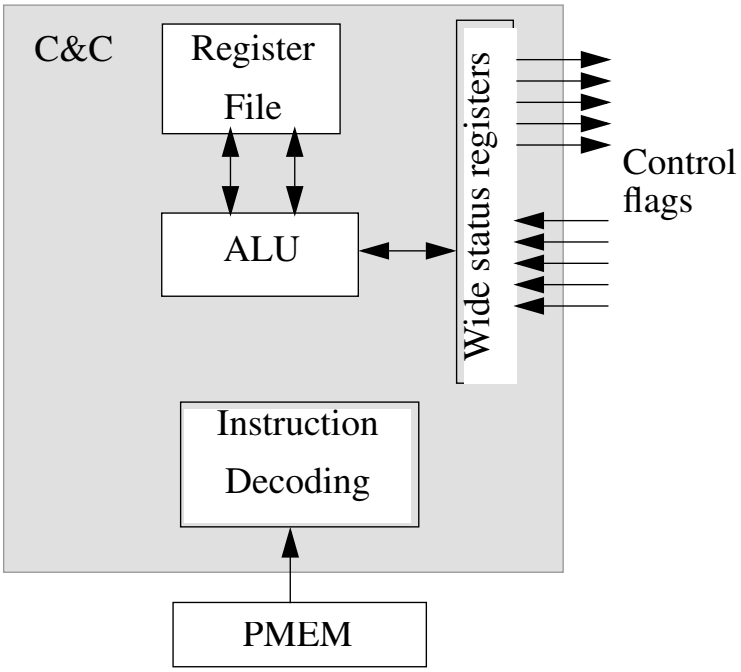

Figure 10.4: C&C organization for general purpose terminal processing.

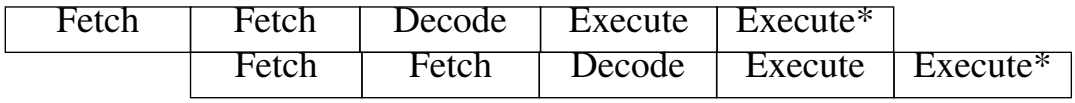

Figure 10.5: Five stage instruction pipeline. Only arithmetic instructions uses the second execute-stage.

ioral memory model. The size of the memory is small and using pipelining it should be possible to achieve very high speed memories using full custom design techniques.

Since the C&C operate at a different clock frequency compared to the FPs and the CMAA synchronization is needed. This takes place in the status register. Flags generated in FPs are written to some registers while control signals from the C&C (e.g. start, stop, discard) are written to others. Flags generated in the C&C uses high speed registers to temporary store the flags until the status register is updated using the network clock. Using clock dividing circuitry, these two clock regions can remain in phase with each other.

The program flow selection in a pipelined C&C is straight forward. Using masked logic operations, result flags from FPs are transformed into branch conditions. These conditions can then be tested using branch instructions.

These branch instructions is the kernel operation of the C&C. Hence, branch prediction is very important for the overall performance of the PNI. The optimal branch prediction is dependent on a number of parameters but most importantly the

Ethernet Destination Address 47-16			
Ethernet Destination Address 15-0	Ethernet Source Address 47-32		
Ethernet Source Address 31-0			
Ethernet Type	IP ver	IP HL	IP ToS
IP Length	IP ID field		
IP Fragmentation	IP TTL		IP Protocol field
IP Header Checksum	IP Source Address 31-16		
IP Source Address 15-0	IP Destination Address 31-16		
IP Destination Address 15-0	UDP Source Port		
UDP Destination Port	UDP Length		
UDP Checksum	...		
PAYLOAD DATA			
Ethernet Frame Check Sequence (CRC)			

Figure 10.6: Packet header fields used for packet decoding in example.

protocol set. Hence, the branch prediction scheme as well as the hardware implementation are application domain specific. Branch prediction is today a huge area in the research community and the problem is a common part of all computer architecture designs.

Due to the inherit uncertainty of the processing latency in a pipelined processor, the cycle by cycle synchronization of the data flow must be questioned. Instead packet synchronization or even buffered wire-speed processing may be considered in order to allow for higher network speeds. As will be shown in section 10.4.2, it is however possible achieve high speed protocol processing if the tasks are limited.

The critical path of the pipelined C&C is the ALU. Static timing analysis of the layout of the C&C datapath shows that the simple ALU can run at 588 MHz when implemented using AMS 0.35 µm standard cell library. If the C&C runs at a four times as high clock frequency, the PPP can support for a network speed of 4.7 Gbit/s. If the protocol coverage is increased, the C&C might have to run at more than 16 times as high clock frequency compared to the network interface (GMII). This however still allows for up to 1 GBit/s of wirespeed processing.

10.4 Simple packet decoding example

In order to illustrate the difference between different C&C implementations a very simple packet decoding example can be used. In this example the PNI

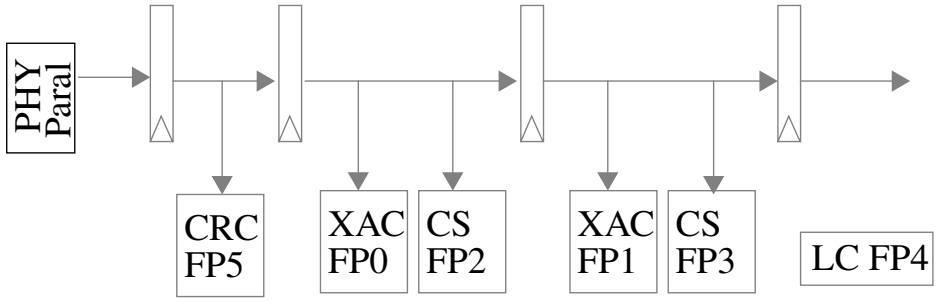

Figure 10.7: FP organization and numbering.

decodes a UDP/IP/Ethernet packet. The protocol processing includes checking destination addresses, destination ports, CRC, IP header CS, UDP checksum, and IP protocol field which is illustrated in figure 10.6. Only one specific port and address is checked.

In order to implement the C&C it is important to notice the placement of different FP accelerators. In this example the placement and the used set of FPs are illustrated by figure 10.7.

10.4.1 Synchronized dataflow example

In this example only one protocol type and destination is allowed at each layer. This means that the program flow selection is very simple. In fact it is possible to use only one XAC FP since packet header data never is compared to more than one value. During the packet decoding of one packet a number of instructions is executed. Some instructions are fetched from the program memory while branch instructions are fetched from the branch unit. As illustrated by figure 10.8 the branch unit contains 10 different branch instructions. They are used to check for packet arrival, to see when FPs are ready and to test the result. Further branch instructions are used for matching of packet header fields such as Ethernet destination address, type, protocol, IP destination address, and UDP destination ports. If any of these packet header fields does not match the value stored in the branch unit, the packet will be discarded. The branch unit checks if the PC is valid, if the flags match the entry and if the branch should be taken at match (E) or no match (N).

Instructions fetched from the program memory sends flags to FPs or the µC to control or acknowledge a packet respectively.

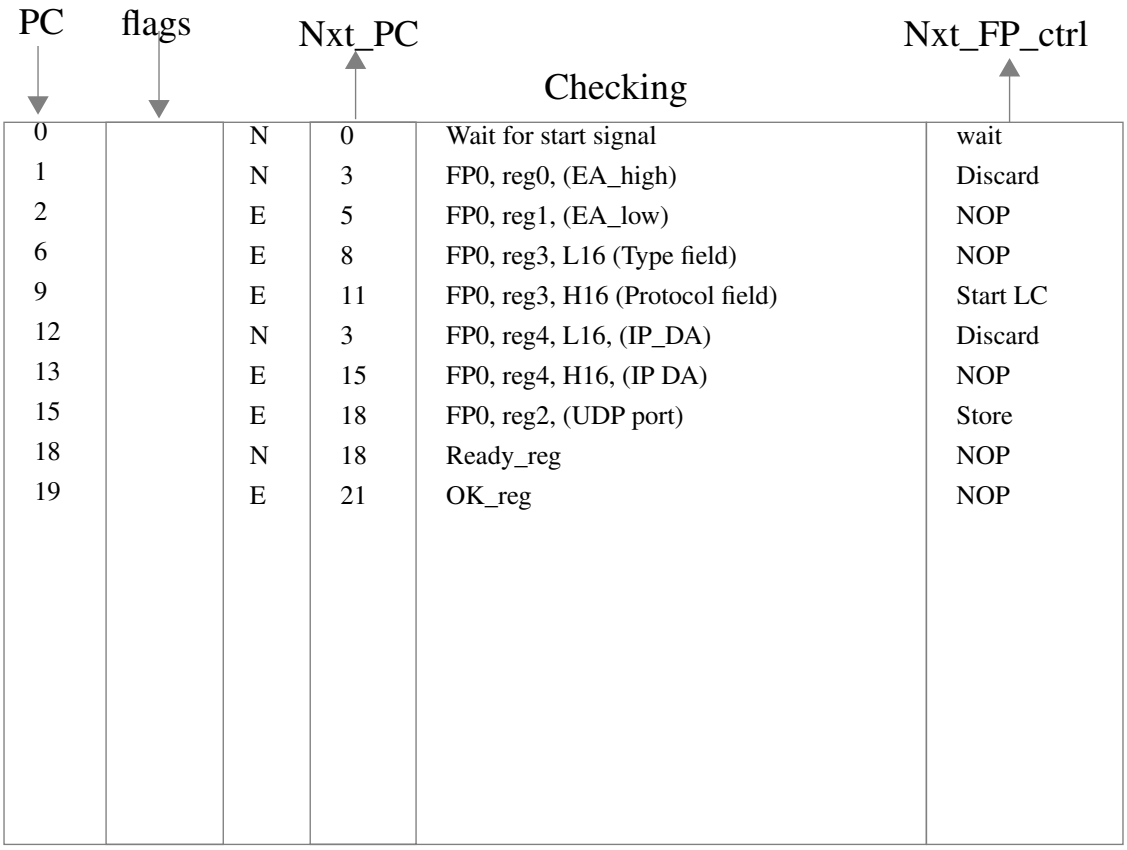

Figure 10.8: Branch unit content.

Table 2: Program memory content

PC	Instr	Jump address	FP control
0	NOP		Ready_reg, Mask
1	BR	3	FP0: Mask, reg0
2	BR	5	FP0; Reg1
3	Set		Discard
4	Jmp	0	
5	Nop		
6	BR+Set	8	FP2: Start FP0: Mask, reg2
7	Jmp	3	
8	Set		FP3: Start Start: LC
9	BR	11	FP0: Mask, Reg3
10	Jmp	3	
11	NOP		

Table 2: Program memory content

PC	Instr	Jump address	FP control
12	BR	3	FP0: Reg4, Mask, Shift
13	BR	15	FP0: Reg4, Mask, Shift
14	Jmp	3	
15	BR	17	FP0: Reg2, Mask
16	Jmp	3	
17	Store		
18	BR	18	Ready_reg: Mask
19	BR/Set	21	OK_reg: Mask Stop store
20	Jmp 3		
21	Set		Set: Packet OK
22	Jmp	0	

10.4.2 Pipelined C&C

This example does not cover reassembly of fragmented packets. Hence, only seven instructions are needed. They are:

- **BRNEQZ**
- **BREQZ**
- **NOP**
- **JMP**
- **LOAD**
- **AND**
- **NOT**

All jump instructions (JMP, BRNEQZ, and BREQZ) executes the following instruction unconditionally. Hence, a misspredicted branch will result in a penalty of two instructions when the instruction pipelined is flushed. BRNEQZ is predicted as always taken while BREQZ is predicted as Not taken. In the simple example an erroneous speculation means that the packet should be discarded.

According to my behavioral simulations the C&C maximally needs three instructions per network clock cycle to handle simple packet decoding of UDP/IPv4/ Ethernet using the specified ISA. I.e. we need three times as many instructions compared to the synchronized implementation due to increased task complexity. Complexity comes from a more complex classification and reassembly handling. This number (three) is rather low compared to many other (e.g. [10.2]) pipelined protocol processors but the number of tasks compared in the C&C is also lower.

E.g. checksum calculations have been offloaded. This low number indicates that a network speed of 6.2 Gbit/s is supported.

The content of the program memory for this simple example is listed in table 3.

Table 3: Program memory content.

	PC	Instruction	Source	Adr/Dest	Network clock cycles
Wait:	**0**	NOT	Ready reg		0
	1	BRNEQZ		0	
	2	LOAD	Reg0	FP0	
	3	LOAD	Reg1	FP0	1
	4	AND	FP0		2
Eth Adr:	5	BREQZ		10	
	6	NOP			
	7	AND	FP0, Mask		3
	8	BRNEQZ		14	
	9	LOAD	FP0, Reg3		
Discard:	10	LOAD	Direct		
	11	JMP		0	
	12	NOP			
	13	NOP			
Type:	**14**	NOP			4
	15	NOP			
	16	NOP			

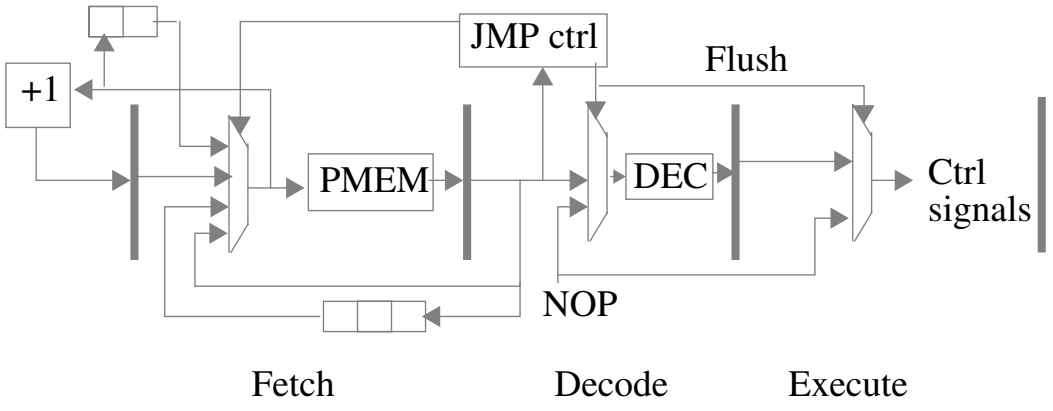

Figure 10.9: Instruction pipeline with a simple branch prediction. It always execute following instruction before predicting BRNEQZ taken and BRWQZ not taken.

Table 3: Program memory content.

	PC	Instruction	Source	Adr/Dest	Network clock cycles
	17	AND	FP0, Mask		5
	18	LOAD	FP2-start		
	19	BRNEQZ		24	
	20	NOP			6
	21	NOP			
	22	JMP		10	
	23	NOP			
Length counting:	24	LOAD	Start	FP4	6
	25	NOP			
Protocol:	**26**	AND	FP0, Mask		7
	27	BRNEQZ		32	
	28	LOAD	FP0: Reg4		
	29	NOP			
	30	NOP			
	31	JMP		10	
IP_DA:	**32**	NOP			8
	33	NOP			
	34	NOP			
	35	AND	FP0, Mask		9
	36	BREQZ		10	
	37	LOAD	FP0: Reg2		
	38	AND	FP0, Mask		10
	39	BRNEQZ		44	
	40				
	41	NOP			
	42	NOP			
	43	JMP		10	
UDP:	**44**	AND	FP0, Mask		11
	45	BREQZ		10	
	46	NOP			
	47	LOAD	Start store	µC	12

Table 3: Program memory content.

	PC	Instruction	Source	Adr/Dest	Network clock cycles
	48	NOP			
Check result:	49	AND	Ready reg, mask		
	50	NOT	Mask		
	51	BRNEQZ		50	
	52	AND	Ready reg, mask		
	53	AND	OK reg, mask		
	54	NOT			
	55	BREQZ		10	
	56	NOP			
	57	JMP		0	
	58	LOAD	Ack/reset		

Note that this example is the simplest possible. In a more complex terminal, the C&C would require more clock cycles and program memory to perform the packet decoding. Further, a different (general purpose) branch prediction algorithm would be used. In the general case it is possible to implement a branch prediction algorithm based on the most common type of protocols and connections. This optimization would however not improve the worst case processing latency.

10.4.3 Conclusions

This example illustrates the main difference between the two implementation alternatives for the C&C proposed in this thesis. The synchronized dataflow implementation have no synchronization registers and no need for buffering, but on the other hand, the branch unit and the XAC FP is a part of the critical path which limits the scalability.

The pipelined C&C can manage very high speeds when the processing does not involve complex program flow selection or classification. While classification can be offloaded from the C&C as shown in the next chapter, the number of protocols covered will directly effect the number of instructions needed to process the packet. But if the number of protocols grow, the usage of non-buffered processing is questionable anyway. The pipelined processor uses more memory and registers than the alternative implementation which is a drawback.

Reference

[10.1]T. Henriksson, U. Nordqvist, and D. Liu, "*Embedded Protocol Processor for Fast and Efficient Packet Reception*", in proceedings of International Conference on Computer Design, Freiburg, Germany, pp. 414-419, Sep 2002

[10.2]Y. Hoskote, V. Erraguntla, D. Finan, J. Howard, D. Klowden, S. Narendra, G. Ruhl, J. Tschanz, S. Vangal, V. Veeramachaneni, H. Wilson, J. Xu, N. Borkar, "*A 10GHz TCP offload accelerator for 10Gbps Ethernet in 90nm dual-VT CMOS*", IEEE International Solid-State Circuits Conference, 2003. Digest of Technical Paper, 14.7.

11

Control Memory Access Acceleration

11.1 Introduction

For general purpose protocol processing with a significant number of simultaneous connections and protocols, this thesis proposes a high-frequency pipelined implementation of the counter and controller. A pipelined processor with an efficient speculation methodology gives a larger area, power and memory consumption compared to the synchronized data-flow implementation. For general purpose protocol processing, there are however no alternative available. Due to the large number of simultaneous connections in such general purpose terminals, a CAM assisted classification accelerator denoted Control Memory Access Accelerator (CMAA) is proposed. This accelerator can also accelerate the reassembly handling of received fragmented packets. This accelerator is also responsible for controlling the PPP access to inter-packet information stored in the control memory.

11.2 Control Memory Access Accelerator

As mentioned earlier the micro controller is responsible for the communication control, i.e. signaling handling. Using a general micro controller is a straightforward method similar to the traditional way of slow path processing in a GP CPU. The problem with this solution is that the control information must be transferred between the micro controller, the PPP, and the control memory with low latency in

order for the PPP to process its part at wire speed and make the decision if the packet should be discarded. In general purpose terminal processing the address check becomes very complex due to the larger number of connections to check for in the receiving terminals connection table. The only possible solution for high speed general purpose classification is to use an accelerator, a.k.a. classification engine. Further, acceleration of slow path processing offloads the micro processor. Hence, a platform including accelerating hardware assist and control interfaces dedicated for packet recognition and control memory access have been developed. The Control Memory Access Accelerator (CMAA) presented in this chapter uses two Look-Up Engines (LUE) in order to recognize and classify the incoming packet. These LUE essentially consists of Content Addressable Memories (CAM) which are well known and commonly used in routing and switching applications. One of the early work in this area is [11.1].

11.2.1 Header data

The purpose of storing control information is to ensure that connection-oriented protocols (e.g. TCP) can perform protocol processing on the payload, which can be divided or segmented into many lower layer packets. These packets can arrive out-of-order and in case of connection oriented protocols the routing information is not included in all packets. Hence it is obvious that some information on the current status of a connection must be stored in order to be able to continue the processing when the next packet arrives. In the case of the protocol set discussed earlier in this chapter the following information is normally needed.

- **Protocol type**
- **Length (received so far)**
- **Total length (transmitted in the last IP packet)**

The length field(s) is provided to the length counter adder in the PPP which updates the number and finally sends the updated value to one of the XAC FP. There it is compared to the total length value which is stored in the control memory. If they are equal, the micro controller is notified that all packet fragments have been received and this entry will be removed from the search list. If unequal, the new length value is written back to the control memory.

- **Accumulated checksum results**

The checksum result is provided to one of the checksum calculating adders, which adds it to the recent packets checksum using a 1-complement addition, and produces a new checksum. If the length is equal to the total length which means that the whole payload message has arrived the updated checksum it is sent to one of the XAC FP for comparison with the received checksum.

- **IP Source and Destination Address.**

The source address is extracted from the data stream by the PPP. The address value is then used to construct a pseudo header. The pseudo header is used in the checksum calculation. Normally, only one destination address is used for unicast packets in a terminal. This means that it is not needed to be stored in the control memory.

- **TCP Source and Destination Ports**

The type, ports, and address fields identifies a specific connection. To see if a incoming packet should be discarded or accepted these fields must be checked. They are also used to identify which application the payload should be directed to.

- **Identification number**

The IP identification number is used to find the correct memory buffer in the control memory.

- **Pointers to the memory position of proceeding and succeeding packets/segments.**

In order to provide all of the services stipulated by the TCP standard, more connection related information than listed above needs to be stored. On the other hand the only information needed for the PPP to perform its processing is the information highlighted in bulleted text. The information stored in the control memory can also be used to calculate the host memory address. An algorithm for this type of memory address calculation remains to be implemented for the general case even if it is simple for special applications, e.g. VoIP. A general algorithm for in-order data-buffering in the host memory would significantly reduce the host processor interrupts. This type of algorithm would benefit from an accelerated access to the control memory. This issue will not be further discussed in this chaper.

11.2.2 Accelerator interface

The CMAA interface to the rest of the PPP and the micro controller is illustrated by figure 11.1.

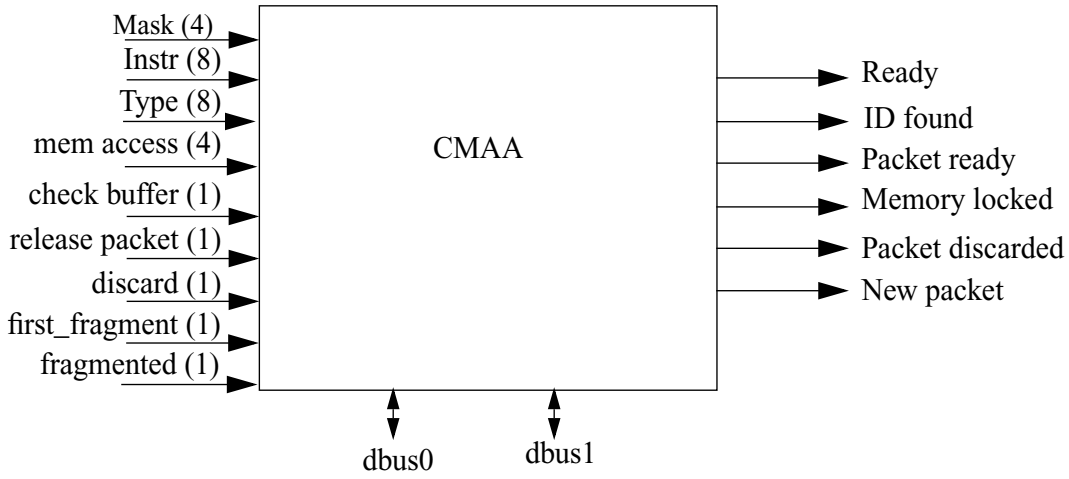

Figure 11.1: Accelerator interface.

Basically the input to the CMAA consists of flags and an instruction generated in the C&C. In table 1 the simple instruction set (6 instructions) is listed.

Table 1: Lightweight instruction set

Name	Source	Internal configuration
New packet	dbus0= IP ID field	Packet type
Load register	dbus0	Port or Address word
ID CAM operation	dbus0	write, read or remove
PA CAM operation	dbus0	write, read or remove
Release to micro controller		
Set memory buffer	dbus1	Packet type

As output the CMAA generates a number of flags. The two data buses are being used only for transport of packet header data.

11.2.3 Data path

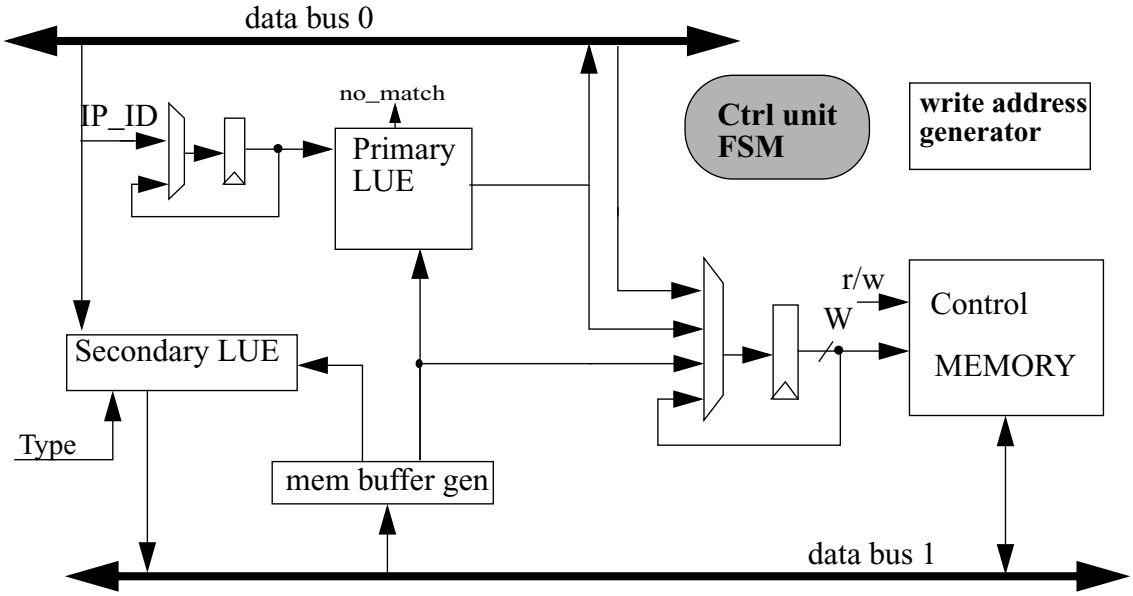

Figure 11.2: CMAA architecture. An accelerating hardware architecture for control memory access in the protocol processor. Based on traditional packet classification techniques it support low latency access to stored connection variables in the control memory

An overview of the CMAA architecture is illustrated in figure 11.2. The CMAA data path includes two LUEs, a buffer pointer generator, and a simple memory access selector. The Primary LUE (PLUE) only includes one CAM, which has 16 bit wide entries, M entries and a W bit wide result memory. The purpose of this unit is to check if we already have received a fragment of the incoming packet. This is checked using the IP Identification field (IP ID). If an arriving packet is fragmented, the fragmented flag will be produced in the C&C and provided to the CMAA. Then the fragment is checked in the PLUE to see if a packet buffer exists in the Control memory. If the CAM in the PLUE does not have a matching identification field entry, a new packet buffer will be created and the IP ID will be written to the PLUE CAM. In the packet buffer, inter-packet variables such as length and checksums will be stored. If the packet is non-fragmented there is no need to store its IP ID so the packet buffer is created directly on the control memory address provided from the mem buffer gen unit in figure 11.2. The SLUE is a classification engine including 6 CAMs and its purpose is to check for valid connections. The two data buses are 32 bit wide. The mem buffer gen generates new buffer addresses for both packet buffers and connection buffers. The address generation is controlled from the µC.

As the other accelerating devices in our protocol processor, e.g. FPs, the CMAA remains in idle mode while not in operation. Power-up will be performed when a

new packet arrives. This reduces the power dissipation significantly in a network terminal due to the un-even time distribution of the packet reception.

In this paper we leave the final CAM design and implementation to be further investigated and optimized. The reason behind this is that they are extremely important for the overall performance and they require different design techniques, tools, and expertise than the rest of the PPP. Final implementation of the LUE will of course have a huge impact on the performance of the CMAA. This issue is further discussed in section 11.2.7.

A layout of the CMAA excluding the two LUE and the buses has been produced. The number of standard cells and the area of the CMAA excluding the input registers and the two LUE are 716 and 0.105 mm^2 respectively. This part of the CMAA has been simulated, using static timing analysis on the layout, to run at almost 300 MHz. This means that it is not included in the critical path of the PPP. Since we use registered inputs and outputs in the CAMs, it is the SLUE that will be the critical path of the CMAA.

11.2.4 Control procedure

The normal packet reception procedure of operation in the CMAA, is illustrated by figure 11.3. The procedure is controlled by the control unit finite state machine (FSM) in the CMAA.

If a new packet arriving is fragmented, the PPP provides CMAA with the IP ID number and gives a new-packet instruction to the CMAA. The IP ID is then stored in the input registers to the PLUE. Next 2 clock cycles, the CMAA continues to load ports and IP addresses while the PLUE checks if a fragment of this payload has already been received. If there is a match in the PLUE search, the corresponding address pointer to the buffer in the control memory, which is stored in the PLUE result memory, is stored in the input register to the control memory. While the PPP continues the packet processing, it can then access the control memory directly. If the new fragment contains the layer 4 header, the port, source and type fields are loaded from the PPP and then checked in the SLUE. If this loading is completed after the PLUE search, i.e. it is a IPv4 packet, the SLUE can immediately check the connection information. Otherwise the control unit remains in the check connection state while the loading continues. Based on the SLUE result, the packet is either discarded or the matching connections address pointer is provided to the data bus 1. Next clock cycle, the data bus 1 value will be stored at the packet buffer address which is already stored in the input register to the control memory. This means that the µC easily can access the connection information. Then the CMAA hands over to the PPP using the packet-ready flag.

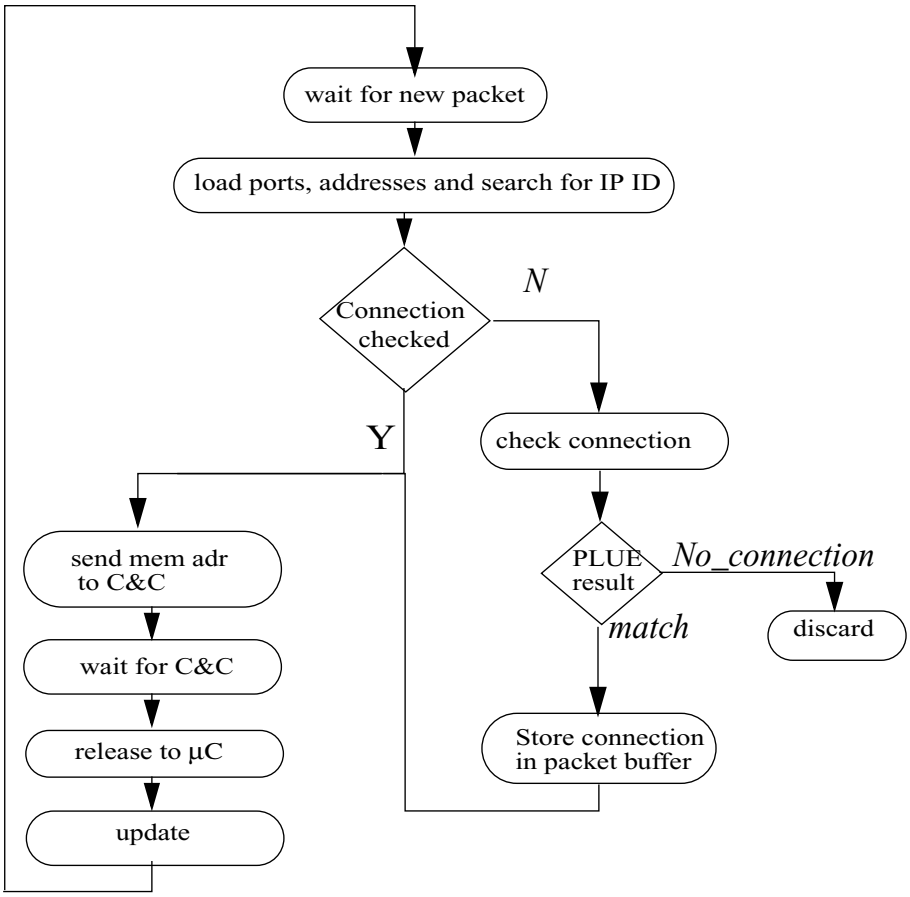

Figure 11.3: Control handling procedure within the CMAA.

After the PPP has received the packet-ready flag, it continues to process the packet and updates the control memory.

After successful packet processing, the PPP releases the packet to the CMAA. Next clock cycle, the CMAA releases the lock of the control memory, starts buffer pointer updating, and sends the new-packet flag to the µC. During the update state, the CMAA also updates the write address for new entries to the two LUEs. This is only done if a write operation has been performed. During the write address search, the CMAA uses one of the generic adders in the PPP to search for empty entries. When the pointer updating and the CAM write search is finished the CMAA returns to the wait-for-new-packet state.

11.2.5 Control memory organization

The control memory is organized according to figure 11.4. As illustrated the control memory consists of a number of different buffers storing inter-packet information. Further the memory includes all the control oriented packets that is going to be processed in the micro controller software. Since these protocols are completely processed by the micro controller, the payload of these packets is also stored in the control memory. For TCP and UDP type of packets only preprocessed header infor-

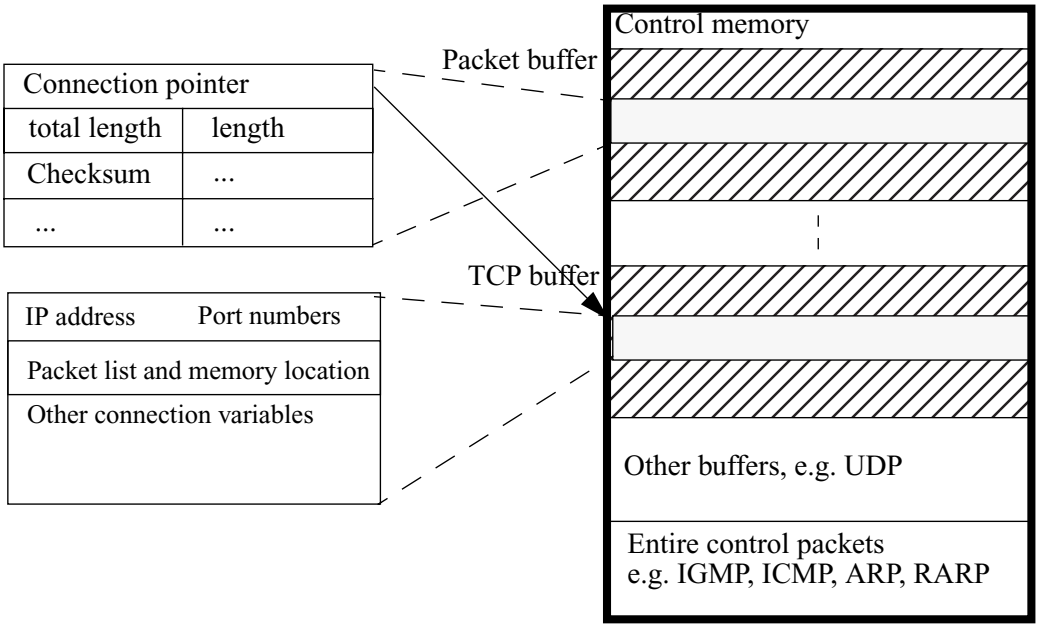

Figure 11.4: Memory organization in the control memory.

mation is stored. In the packet buffers, layer 3 information needed for reassembly is stored. Each packet buffer is deleted when the entire layer 3 packet has arrived.

11.2.6 Look-up Engine architectures

The SLUE consists of 6 CAMs as illustrated by figure 11.5. The outputs generated by the CAMs are vectors containing zeros or ones describing table matches. These are used to select the address pointer in the result memory, i.e. a pointer to the control memory address for the received packet.

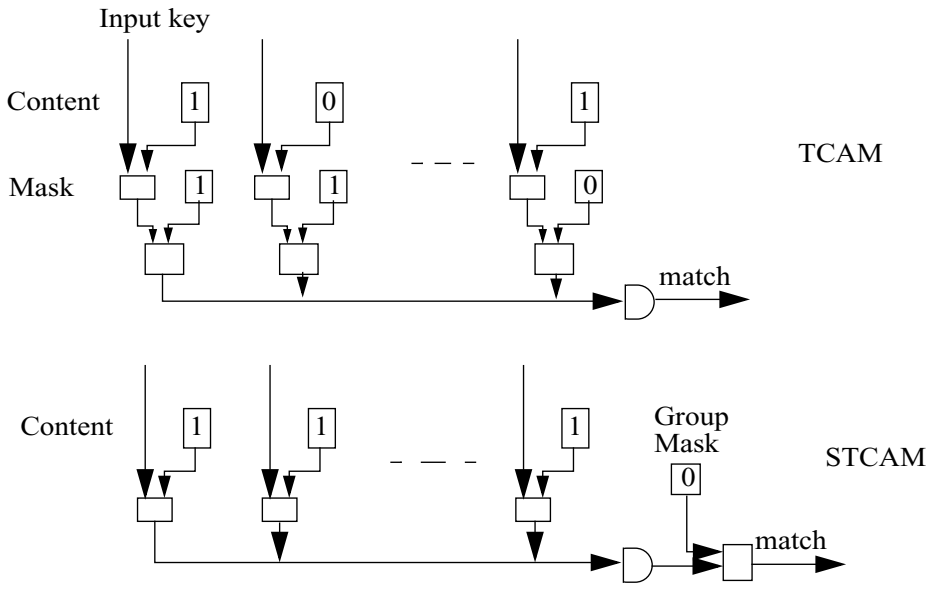

Figure 11.6: Simplified TCAM principle.

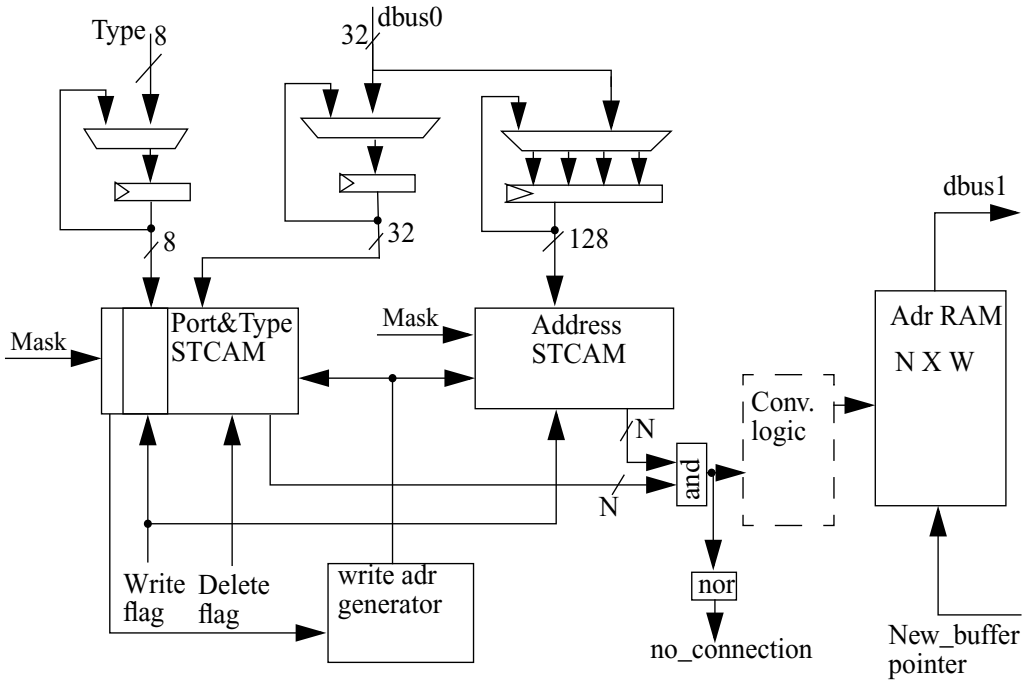

Figure 11.5: Secondary Look-Up Engine (SLUE) architecture. Note that the conversion logic that converts the matching vector to a result memory adress can be eliminated if the matching vector is used directly as word lines in the memory. This however require that the RAM must be implemented in the same manufacturing process.

The 7 different CAMs we propose to be used in the CMAA architecture will have a huge impact on the performance, size and power figures of the entire design. Therefore they require a thorough investigation and optimization procedure in order to obtain the optimal system performance. Even if the optimization of these CAMs is not in the scope of this thesis, some characteristics and requirements on the CAMs can be noted. First of all we propose that CAMs should be used instead of TCAMs ([11.4] and [11.5]). This reduces the cell size and power dissipation. The primary LUE is a standard CAM memory with 16-bit content and M entries. The result memory is M times the length of the control memory address W.

In order to provide flexibility for different protocols we use a concept we call Simplified TCAM (STCAM) illustrated by figure 11.6 in the secondary LUE. Instead of using ternary bit comparisons as in TCAMs we only provide a wildcard function to the entire CAM. In figure 11.7 there is an illustration showing how the secondary LUE uses the STCAM principle.

The mask input enables a wildcard functionality for different fields when recognizing an incoming packet according to table 2. The table shows that the proposed SLUE architecture can be used for various types of protocols. A careful use of these wild cards is needed in order to avoid multiple matches. By using the type

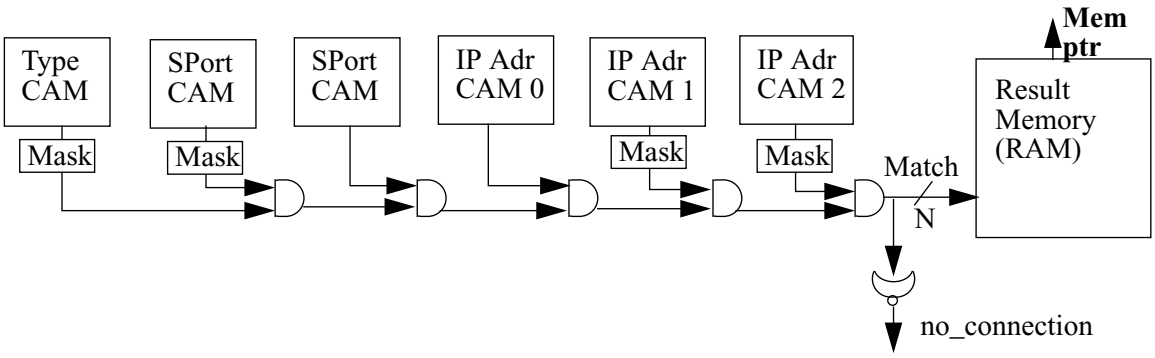

Figure 11.7: The two different STCAM in the SLUE each consists of three ordinary CAMs and some masking functions. Each of the CAMs uses N entries.

field, which is an internal type, it is possible to avoid multiple matches which means that the priority logic in the SLUE can be eliminated. Further it enables the connections to be written in the CAM in an arbitrary order.

Table 2: Configurations using masking for different packet types and applications.

Protocol examples	Type	Source Port	Destination Port	Source Address	Destination Adr
IPv6 Unicasting	Optional	Optional	16	128	*
IPv6 Broad or multi casting	Optional	Optional	16	*	128
IPv6 alt	Optional	Optional	16	64	64
IPv4	Optional	Optional	16	32	32
IPv4	Optional	Optional	16	32	*
UDP	Optional	16	16		32

It can always be discussed how much the IP version 6 (IP v6) protocol will be used in the future but we have chosen to include it since the penalty is not as severe in network terminals as it is in routers. The reason for this is that in network terminals we only have one destination address to check for unicasting. This can be done in other parts of the PPP. Hence 128 bits can be excluded from the CAMs entries. For broad and multicasting packets a different type field is generated and only the destination is checked (instead of the source address). This reduces the penalty we have to pay in forms of larger CAMs when including IP v6. There exist however routers where only 64 bits out of the 128 in the IP v6 address are used for packet classification. The reason is that in such networks the other 64 bits is just a copy of the MAC address. If such method would be applied the CAMs can reduce

the wordlength of the content with additionally 64 bits by eliminating the IP Adr 2 in figure 11.7. Since this architecture will be used in a network terminal the activity will not be as high in the CAMs as it would be in a router. The reason is that we only do a load and search operation when a new packet arrives, not every clock cycle. The low activity significantly reduces the power consumption in the CAMs.

11.2.7 CAM implementation issues

The total size of the 7 CAMs and their result memories will be a major part of the systems chip area. It is very hard to make predictions on the sizes of these CAMs since that is a matter of optimization effort and implementation strategy. Further the complex placement and routing requires a full-custom approach even for standard cell based designs. Even without a final layout, a lower bound on the chip-area can be estimated. Using standard cells from our design process (AMS 0.35 µm 3.3 V 3-M) an optimized bit-slice cell in a CAM is approximately 350 μm^2 which results in a lower bound on the combined CAM area according to EQ 1. The result memories must store M + N times W bits using approximately 180 μm^2 each.

$$A_{CAM} = (16 \times M + (128 + 40) \times N) \times 350 + (M + N) \times W \times 180 \qquad \mu m^2 \tag{EQ 1}$$

As an example M=16, N=64, and W=20 can be considered. The chip-area for the two LUEs would then be at least 4 mm^2. This figure is acceptable but if more entries are to be considered, a process migration to smaller geometries is natural. The number of entries to implement is a matter of optimization. This optimization procedure requires a careful analysis of application requirement and network traffic. Never the less it is clear that in NT, the required number of network links is not as high as in routers. Hence M and N does not need to be very large for most applications and networks.

In order to examine our architectural performance, it is crucial to know how many clock cycles each search operation in the two LUE requires. We expect the system clock to have a period of maximally 7.5 ns in a 0.35 micron process, based on timing analysis on other parts of the PPP. Hence the maximum network speed is 4.3 Gbit/s using the specified 32 bit wide input buffers. Since we are sure that there is only one packet being processed at any given time, we do not necessarily need the LUEs to be pipelined, i.e. we do not need any internal intermediate results to be stored. Instead a multi-cycle-path design technique can be used. To use pipeline stages or not is an implementation issue for the CAM designers. Simulations show that the small PLUE will not require more than 2 clock cycles to complete one search, i.e. it has a critical path shorter than 15 ns. Then we assume M is maximally 64.

The number of clock cycles required for a search operation in the SLUE is equal to the critical path divided by 7.5 ns. The critical path consists of circuit delays and wire delays. If the SLUE are being implemented using standard cells the logic delay is simple to calculate. For N=64 there will be approximately 15 logic cells in the critical path which leads us into believing that 2 clock cycles is enough. The problem is that in larger CAMs a big part of the critical path is wire delay. In my research design (N=256) I have used synthesis and P&R tools from Cadence. The resulting implementation result is very far from optimal and does not meet my requirement of 3 clock cycles. The design is simply to large and hence the P&R problem to complex. Therefore the conclusion is that the design strategy must be changed to something more custom oriented even if the CAM is rather small compared to the one used in routers. Clearly a bitslice manipulating placement strategy has to be used for efficient CAM design regardless of the size. Anyway the conclusion after studies of other comparable CAM designs and discussion with industry CAM designers is that, for N less than or equal to 256, a search operation will require maximally 4 clock cycles (or pipeline stages). For N=64, 3 clock cycles are definitely enough. These figures apply to standard cell based designs.

Even with a pessimistic feature size projection (Moores law), there is no reason to believe that scaling not can support the CMAA to run at clock periods around 3 ns using 3 clock cycles for one search operation. Hence the CMAA could be used in a 10 Gbit/s network such as 10 Gigabit Ethernet, using already available processes, i.e. 0.13 micron. The resulting latency for CMAA operations is further discussed in section 11.2.8.

The latency, critical path, and power consumption in the LUE are of course depending on M, N, and W. To optimize these variables simulation on real world network traffic is required. Until this optimization phase is completed the numbers M=16, N=64, and W=20 will be considered for further architectural development.

11.2.8 CMAA decision latency

Table 3: Examples on memory access latency for various packets received. (PLUE requires 2 clock cycles to perform a search)

Layer 3 protocol	# clock cycles latency for CMAA operation 3 stage SLUE	# clock cycles latency for CMAA operation 4 stage SLUE
IPv4 - new packet	9	10
IPv4 - old packet, new fragment	4	4
IPv6 - new packet	11	12
IPv6 - old packet, new fragment	4	4

The proposed architecture for access of the control memory reduces the control memory access latency to a few clock-cycles. The fast path latency determines how big the input buffer chain has to be. The latency of the CMAA must be added to the latency of the PPP in order to calculate the total fast path latency. We propose that the SLUE should use 3 clock cycles to perform a search. A 3-clock-cycle type of SLUE would give a maximum memory access latency of 11 according to table 3 when a new packet has been received. Further the table shows that a four cycle type of CAM architecture, will give a maximum memory access latency of 12 clock cycles. This of course have an impact on the pipeline register chain in the PPP and the total latency for a packet reception and delivery to the micro controller.

The PPP can start the processing of an incoming packet before the control data has been accessed from the control memory. Therefore this latency only sets a lower limit on the latency of the total packet reception. The total latency is however mainly dependent on the processing activities, including interrupts and stalls, in the micro controller.

11.3 Enabling flow based QoS

Using the fast control memory access, it is possible to enable quality of service (QoS) to the reception. Any kind of priority parameters or flow parameters can be stored in the different buffers in the control memory. These can then be used for multiplexing of the incoming data stream if a flow based operation is demanded.

11.4 Shared control memory

The motivation for separating the protocol processing into one PPP-part and one µC-part is of course to use the programmability of the µC when processing control intensive tasks and still have high-performance and low-power implementation of

the data intensive processing. This distributed architecture however requires an interface, and that interface is the control memory unit together with control flags to and from the C&C. As mentioned before, the PPP only needs to access the memory when a new packet is received and then only a limited part of the control information is used. Since the latency of this access directly effects the length of the input buffer chain, the PPP must have priority over the µC when it comes to memory access. In fact the µC only have access to the control memory when the CMAA resides in the update or wait-for-new-packet state according to figure 11.3.

11.5 Implications on system performance

Each hardware acceleration block in the PP has been seperatively implemented and simulated using static timing analasys.The conclusion is that they can operate at network speeds of moore than 170 Mhz. Since all parts of the fast path operates on streaming data it means that the network can run at this clock frequency. The fast path architecture processes each packet, delivers data and control signals to the micro controller and then returns to idle mode. When the fastpath, e.i. the C&C, has returned to idle-mode it can start processing the next packets. Hence, the proposed fast path architecture can operate in high speed networks as long as the gap between the incomming packets is sufficient for the processor to return to idle mode. This must be supported by the network protocol. Hence, better solutions are to either use an input packet buffer or multiple protocol processors that are hardware multiplexed.

The slow path consists of the micro controller which is enough flexible to fully offload the TCP and protocols alike. The micro controller is not capable of processing the packets at wire speed. This may limit the performance of the entire system for extreme traffic situations. The fast path does however offload some of the tasks traditionally processed on general purpose hardware. This will relaxe the slow path. The amount of offloading depends on the traffic flow and requires further simulations.

In order to verify the functionality of the CMAA block used as proposed in a fast path, a cycle-true and bit-true behavioral model has been simulated. The simulation model covers the fast path packet reception. The model includes GMII network interface (32 bit wide input). Further a behavioral model of the CMAA including 16-entry, 3-stage pipeline CAMs has been used.

So far the only protocols simulated are TCP and IPv4. Fast path tasks simulated includes CRC, IP reassembly, checksum calculation and data stream demultiplexing. The C&C is programmed to cover these protocols using a the program memory in the C&C model. The network traffic simulated is random.

The simulations verifies that the proposed CMAA architecture can be used in the protocol processor environment. Further it shows that when programmed for TCP the minimal distance allowed between 2 packets is 23 clock cycles (ideal is none). In addition to this limitation the simulations also shows that the minimal number of clock cycles required per received Ethernet packet processed is 38. The decision latency in the CMAA contributes to this figure. These architectural limitations strongly effects the performance compared to an ideal solution. Especially for small packets which is the worst case. The overall system performance is on the other hand much dependent on the network traffic. If the traffic does not include many fragmented packets the CMAA may not be worth using due to the performance degradation. In that case a PP without CMAA acceleration should be used.

11.6 Summary

A novel architecture for acceleration of control memory access in a protocol processor for network terminals was presented in this chapter. The architecture uses classification engines and concepts which has traditionally been used for network infrastructure components. The proposed architecture enables low latency access to connection state variables, partial checksum results and any other control information stored in the shared control memory. Hence inter-packet processing such as reassembly has been accelerated using our flexible protocol processor architecture for network terminals. Further it offloads the micro controller so that a wide variety of protocols can be processed in a programmable way, using the proposed protocol processor platform in high-speed networks. The proposed architecture can process the fast path in a multi gigabit network, implemented in a mature standard cell process such as AMS 0.35 µm.

For general purpose terminal processing the C&C must be implemented as a pipelined processor and use a CMAA to accelerate reassembly. classification and control memory access. Synchronized control of FPs and CMAA is enabled by use of a wide status register. It is however not possible to maintain synchronization with the data stream without use of buffers when minsized packets are allowed. Hence, terminals requiring the use of a CMAA to handle connections and reassembly is not suitable for data-flow processing. Instead packet synchronization, increased minimal allowed packet sizes, lower network speed, or flow synchronization must be considered. I.e. the proposed dataflow processing architecture must be modified if fragmented packets and a large number of connections are supported.

Reference

[11.1]McAuley A. et al., "*Fast Routing Table Lookup Using CAMs*", IEEE INFOCOM '93, March 1993

[11.2]P. Steenkiste, "*A Systematic Approach to Host Interface Design for High-Speed Networks*", IEEE Computer, v.27 n.3, pp.47-57, March 1994
[11.3]M. B. Abbott, L. L. Peterson, "*Increasing network throughput by integrating protocol layers*", IEEE/ACM Transactions on Networking, vol. 1, pp 600-10, 1993
[11.4]van Lunteren J., Engbersen A.P.J., "*Multi-field packet classification using ternary CAM*", Electronics Letters, Volume: 38 Issue: 1, 3 Jan. 2002, pp 21 -23
[11.5]Huang N-F, Chen W-E, Luo J-Y, Chen J_M, "*Design of multi-field IPv6 packet classifiers using ternary CAMs*", GLOBECOM'01. IEEE, Volume: 3, 2001, pp 1877 -1881

12

Conclusions

A dual processor architecture for protocol processing in network terminals was proposed. It uses configurable accelerators to offload data intensive processing tasks. These accelerators are controlled using a programmable Counter and Controller (C&C). Control intensive tasks are handled by a general-purpose micro controller.

The architecture was developed to meet increasing demands on bandwidth, flexibility, and cost in todays network terminals. Using accelerators the proposed architecture supports multigigabit network speeds when implemented in a mature process technology. Flexibility requirements are supported in three stages. At design time a suitable implementation method can be selected for accelerator and C&C depending on the intended application domain. Hardware accelerators can be configured online and the data path is controlled in a programmable way. Further the micro controller supports unregular slow path processing.

The proposed architecture is power efficient due to its reduced memory usage compared to traditional designs. The proposed architecture was developed with computer network terminals in mind but it may also be used for other types of packet based network entities, e.g. gateways, PDAs, and wireless terminals etc.

The proposed protocol processor architecture is based on synchronized data flow processing. This thesis concludes that the use of data flow based processing is suitable for special purpose terminals. In more complex terminals, which require fragmented packets and a large number of connections to be handled, the proposed architecture must be modified by adding a special control memory access accelerator. This accelerator is needed since fragmented packets require the processor to access inter-packet information stored in the control memory. In complex terminals, this control memory access leads to an increased decision latency. The increased decision latency will lower the performance if the synchronized processing scheme is used. Hence, this thesis proposes that when used in complex terminals, the protocol processor should use input packet buffering.

Future work

“Only the wisest and stupidest of men never change.”
--Confucius

13

Media Gateways

With the proliferation of Internet-connected devices including cellular telephones, 2-way pagers, and personal digital assistants (PDAs), there is a market for efficiently delivering web-based content to these devices. A transcoding media gateway is a specialized router that converts high-bandwidth or high-resolution data (e.g. video) to lower-bandwidth (suitable for wireless links) or lower-resolution (suitable for handheld displays). Thinning a data stream to match the capabilities of the end device can reduce latency and improve efficiency in the use of transmission bandwidth. The media gateway shares many characteristics with a normal network terminal which has been the focus of this thesis. Hence, this chapter is included in the thesis as a discussion around knowledge transfer with focus on finding suitable areas for future research.

13.1 Multi-processor issues

It is possible to divide gateway architectures into homogenous, specialized heterogeneous, and hybrid systems. A homogenous system consists of identical protocol processors as illustrated by figure 13.2. The internal architecture, the access to shared memory and the access to hardware accelerators are equal between the processors. A specialized heterogeneous system consists of protocol processors optimized more or less hardwired for a specific task like packet classification, routing, or QoS. This is depicted by figure 13.1. This solution is well suited for a system with a well defined problem space that is unlikely to change much over time. An ethernet switch is a typical example of such a system. A hybrid system can be con-

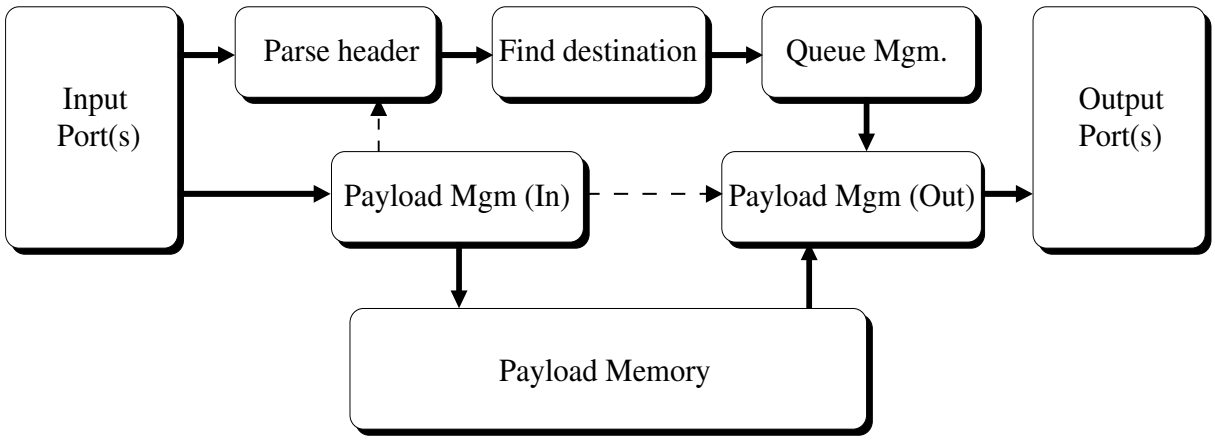

Figure 13.1: Heterogeneous Protocol Processing Architecture.

structed in several ways. One way is to combine several groups of protocol processors into a system. For example into one group that consists of identical protocol processors optimized to handle intra-packet operations and another group that is optimized to handle inter-packet operations. Another possibility is to have certain basic functionality common to all protocol processors but extra accelerators available on some.

According to earlier research on protocol processing, **heterogeneous or hybrid systems are the only choice for protocol processing**. Homogenous system does not support requirements on performance and power-consumption as good as heterogeneous systems.

13.1.1 Communication between protocol processors

The communication between protocol processors should be optimized to handle short header transfers. A classical protocol processor does not care about the payload data beyond storing it once in the ingress part and retrieving it once in the egress part. It is also possible to reduce the amount of shared memory accesses by connecting protocol processors in a serial manner. A buffer in between the proto-

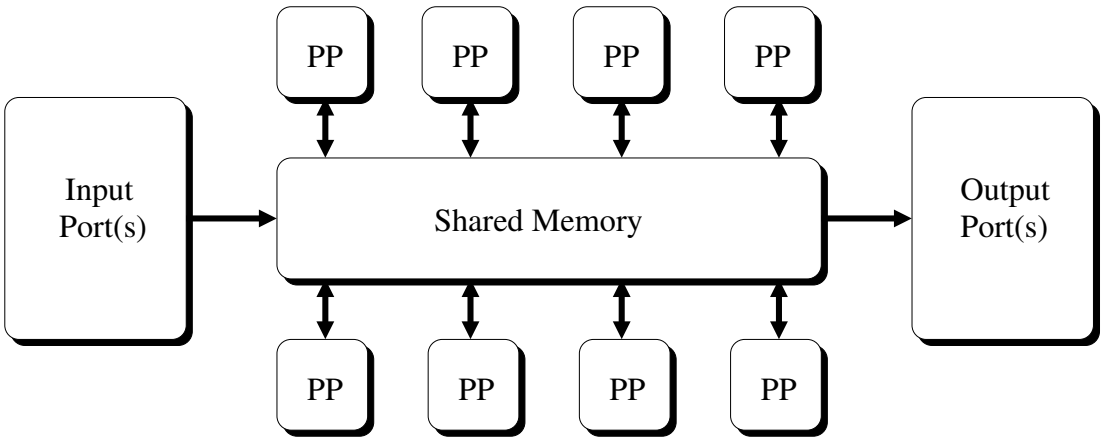

Figure 13.2: Homogenous Protocol Processing architecture.

col processors aids synchronization. As the message parsing is done in an inherent serial manner, many synchronization problems can be avoided.

13.1.2 Synchronization in multiprocessor systems

There are basically two types of data structures requiring synchronization: structures that are read often and seldom updated, and structures that are both read and updated often. A routing table is seldom updated whereas a data structure detailing the usage and allocation of the payload memory would be updated and read very often.

The general purpose CPU would typically update data structures fairly seldom. Care must be taken to either have the data structure in a consistent state during the entire update or make provisions to either duplicate the structure or stall readers during inconsistent times.

Data structures that are updated often require careful design. Some systems have dedicated units to control these structures, for example a unit to handle the allocation of incoming payload data.

13.1.3 Accelerators

By using accelerators, i.e. dedicated hardware, the processing performance can be substantially increased. The accelerators are normally self-contained and does not require a lot of control. Hence the over all control overhead can be reduced which reduces the memory cost and increases the efficiency. The accelerator approach is suitable if there exist processing tasks that are common and non-suitable for general purpose hardware processing.

A lot of different accelerators could be used for network processing in media gateways:

- **CRC: Required for both incoming and outgoing packets. A configurable solution supporting all polynomials is most efficient.**
- **Checksum accelerator: Required to deal with IP packets.**
- **RNG: A true random number generator could be beneficial to some cryptographic applications.**
- **PRNG: A pseudo random number generator could be useful in QoS.**
- **Timers: Flexible hardware timers offload timer management in tasks like shaping.**
- **Queue manager: Used to speed up QoS.**
- **Classification engine**

When deciding what kind of accelerators to use in a multi-processor gateway there, the coprocessor approach must be considered. Third party coprocessors and

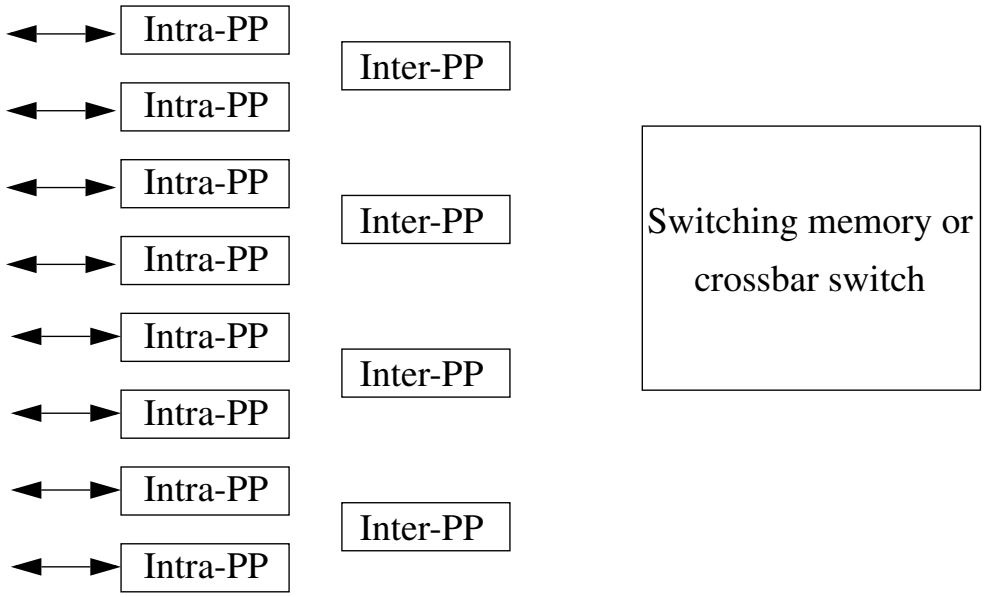

Figure 13.3: Multi-processor partition using the concept of intra- and inter-packet processors.

common interfaces to them becomes more and more common. Hence, they may be a cost effective solution.

13.1.4 Multi-port packet processing

In a gateway with multiple (physical) ports coming in and going out, the proposed PPP can be used for port-processing while the micro controller implements inter packet tasks for one or more ports. This is depicted by figure 13.3. In order to find a good partition between and organization of tasks, processors and memories extensive behavioral modeling and simulation are required.

13.1.5 Application processing

Normally a gateway must support three basic functions. First of all packets needs to be received (terminated) and transmitted (created) at speeds specified by standards defined for each protocol layer. Secondly a switching function must be supported. Finally, there are many application layer tasks which have to be supported. These include voice compression, tone processing, fax/modem communication etc. Since these tasks are so complex, the DSPs they normally run on are normally implemented as a separate chip. Normally off-chip memories are used to transport data to and from these DSPs. Using the technology scaling it might be possible to have all these DSPs and memories on-chip in the future but today it is not really feasible. Hence, this chapter does not further discuss the application integration on-chip.

13.1.6 Efficient task allocation

According to my previous research results, I strongly believe that task-allocation should be done according to processing requirements. I.e. dedicated hardware acceleration of throughput driven tasks and general-purpose programmable devices handling flexibility demanding tasks.

Intra-layer processing, i.e. the traditional way of processing network communication protocols, gives a processing overhead since a lot of intermediate results and data transports must be performed. The main advantage with inter-layer (the opposite of intra-layer) processing is the reduced amount of data transportation and processing since we reduce the need for intermediate results. Another advantage that the interlayer processing gives us is that the processing can be divided and then distributed to different computational units depending on the type of processing rather than layer (protocol) type. Hence, each of the two processors in our proposed dual-processor architecture is always optimized for the specific type of tasks that will be processed which gives both a high performance and a high degree of domain-specific flexibility.

13.1.7 Parallelization scheme

One of the most critical design decisions in a media gateway system is the selection of parallelization scheme. Hence, it is a very important research issue. Following we list a couple of possible parallelization alternatives.

- **Using multi-threading**

If it is possible to reduce Inter-processor and memory accesses, use more parallel data transfers to reduce the transfer latency (overhead), and make each processor more self-contained, the multi-threading approach would be more efficient.

- **Using port processing**

Parallelization could be applied if there is one protocol processor supporting each physical port. The shared resource is the control memory.

- **Using protocol-layer processing**

Another possible parallelization scheme is to have one processor process each protocol layer. This means that the packet data is shared.

- **Using connection-based processing**

Each processor handles one (or a group of) connections. This require a flexible processor and avoid control memory sharing, i.e. the connection states does not have to be shared. The drawback is saturation during bursty traffic.

- **Using task-based processing**

Each processor is dedicated for a specific (or group of) protocol processing tasks. A dedicated hardware is possible but it requires a lot of shared memories.

13.1.8 Inter-processor communication

Normally different types of processors used in a multi-processor gateway have different types of interfaces and bit-rates. They have different control-path and memory interfaces. Regardless, it is important to remove the overhead in the Inter-Process Communication and create:

- **Common data-path interfaces**

In network processors it is common to transport data in fixed-sized cells over the back-plane. The size of these cells and the tagging (control-information) that is connected to each cell must be designed and evaluated with a system-perspective. This data-path interface have a huge impact on the memory architecture. Hence, they must be co-optimized.

- **Low-overhead common signaling-interface**

By nature a heterogeneous system will have a heterogeneous and distributed control path. It is important that this signaling interface is kept simple (to avoid overhead) yet flexible.

13.2 Gateway memory systems

Memory is a major part of any modern SoC. Since memory technology is not scaling with Moores law, the part of the system that is memory, is currently increasing. Recently, there has been some new types of memories emerging. These are dedicated for network processing (search engines) but the increased bandwidth and reduced latency has not been enough. Hence, the main research focus has been (and should continue to be) on memory usage.

A transcoding media gateway is a specialized router that converts high-bandwidth or high-resolution data (e.g., video) to lower-bandwidth (suitable for wireless links) or lower-resolution (suitable for handheld displays). Thinning a data stream to match the capabilities of the end device can reduce latency and improve efficiency in the use of transmission bandwidth.

The processing in a media gateway can be classified according to the task-type which is illustrated by figure 13.5. There are some tasks suitable for processing on general purpose processors, some on DSPs and some on dedicated network processors.

During the processing of a packet in the media gateway different memories are accessed several times which is depicted in figure 13.4. These accesses is a major

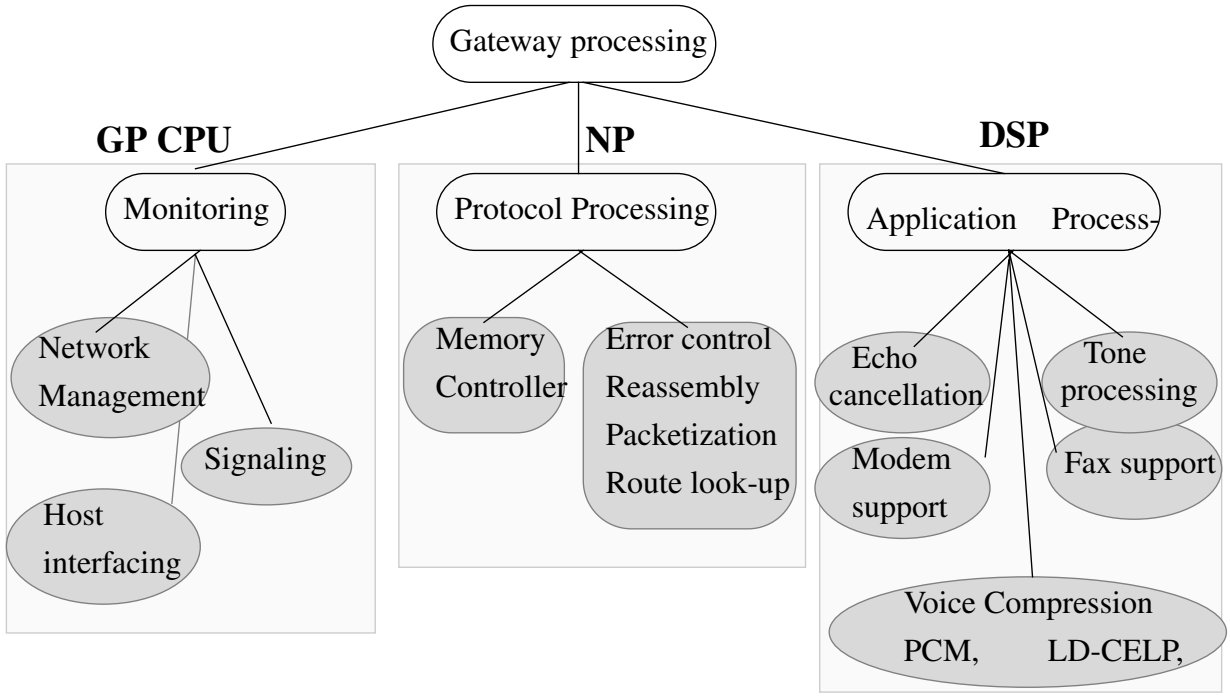

Figure 13.5: A heterogeneous system architecture is required due to the heterogeneous nature of the processing tasks in a media gateway. As a result of this heterogeneous nature of the system, the memory organization becomes very important. It is the key-issue for successful integration of all these processors (General Purpose CPUs, Protocol Processors, or DSPs) which may have different bit-rates and heterogeneous interfaces for control.

contributor to the overall power consumption and therefore important to minimize. One common way of doing that is to place the payload in memory when a packet is received and not to access it before the data is transmitted. Some times this is not possible because the algorithm needs to access the payload data, e.g. voice compression. In order to reduce the size and number of accesses to the memory, it is important to carefully schedule the memory usage and architecture.

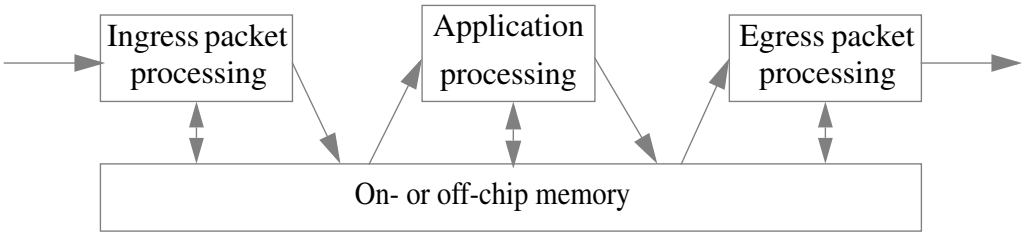

Figure 13.4: During the processing of a packet in a gateway, data normally have to be stored to and then accessed from memory several times before a new packet has been created and then sent away. If the number of memory accesses can be reduced it will result in a reduction both the memory size as well as the area and power consumption which will reduce the cost.

The basic requirements on the memory architecture in a media gateway includes:

- **Sufficient scalability/flexibility**
- **Sufficient throughput/bandwidth**
- **Low power/area consumption**

13.3 Application driven memory optimization

Packet information is normally stored and accessed several times during the processing (packet traversal) in a gateway. First of all, the packet is stored in an Input-buffer before the Ingress processing takes place. In our previous research we have shown how this input buffer can be eliminated by data flow based processing which reduces the memory access.

Secondly, the payload is stored in a shared memory acting as the switch of the gateway. Using cut-through switching, the size of the memory acting as a crossbar switch can be lowered. Cut-through switching is a method to improve the throughput (bandwidth) of the switch but it can also be used to provide QoS and reduce the memory cost. If the packet data have to be processed by an application protocol, e.g. voice compression or echo cancellation, additional memory access and data exchange is required. If this communication have to be off-chip due to the limited chip-area available, it is very costly in terms of power and latency. Memories are also used to store the program-code and control information (e.g. connection state variables). All of these heterogeneous (logical) memories can and should be optimized so that an efficient memory architecture is obtained.

13.3.1 System-level hand-over

Normally a gateway must support three basic functions. First of all packets needs to be received (terminated) and transmitted (created) according to standards defined for each protocol layer. Secondly a switching function must be supported. Finally, there are many application layer tasks which have to be supported. These include voice compression, tone processing, fax/modem communication etc. Since these tasks (and the DSPs they normally run on) is large and complex systems, they are normally implemented as a separate chip. Normally off-chip memories are used to transport data to and from these DSPs.

If all of the Media Gateway could be accommodated on-chip, this system-level hand-over memory access can be avoided. That would lower both the processing latency and the power consumption.

13.3.2 Switching memory usage

IP is based on best-effort while ATM supports a variety of QoS requirements. The drawback of IP is the low bandwidth while it does not require any setup delay

since it is connection-less. The ATM has the opposite characteristics. Today it becomes interesting to combine the two in order to find a high-performance switch-architecture that supports QoS. This is especially interesting in mixed networks. Consider for example a gateway with IP on one side and ATM on the other. By nature such a gateway must combine the properties of both an IP and an ATM device.

An ATM-packet is small (53 bytes). If the size of the payload-data is relatively moderate, ATM networks provide higher throughput compared to IP networks, due to the shorter header length (5 B compared to 20 B).

The drawback with ATM is that a Virtual Circuit (VC) has to be established between the sender and the receiver in order for the ATM packets to find their way to the destination. The main advantage of ATM networks is the inherent support for QoS. One example is low latency transmission using prioritization which is very important for certain applications, e.g. VoIP/VoATM.

There are fundamentally two ways of designing a switch. One is the store-and-forward method where a packet is basically stored, reassembled, and checked for different types of errors before it is forwarded to an output-link. This is the traditional way of implementing an IP switch. One drawback is that the latency can not be guaranteed although flows can be prioritized.

The other solution is to use some sort of cut-through switching which means that parts of a payload or packet is forwarded before it has been assembled in the switch. This means that the reassembly and other processing tasks in the switch are bypassed. This can for example be used if a long IP packet comes in to the switch and thereafter is forwarded as ATM packets. When the IP header is received the switch knows where to send the data. Hence, it can be transmitted to an output link as soon as enough data to fill one ATM packet (48 B) has been received. This normally increases the throughput since the routing procedure is simplified because the VC has already been setup. Cut-through switching also means that the memory position where the beginning of the packet has been stored, is free to be used to store new payload data coming which is depicted in figure 5. If the packet-size is 500 B this reduces the memory size with 90% as illustrated by figure 13.6.

There are two situations where the use of cut-through switching is especially beneficial:

1) If a certain flow of packets have many large packets the latency can be reduced if packets are sent out before all of the packets have arrived. This also reduces the memory size since the entire payload is not stored in the memory at the same time. If the switch for example identifies a TCP Syn request and an ftp message in the upper-layer payload it is natural to guess that this flow will contain many large packets.

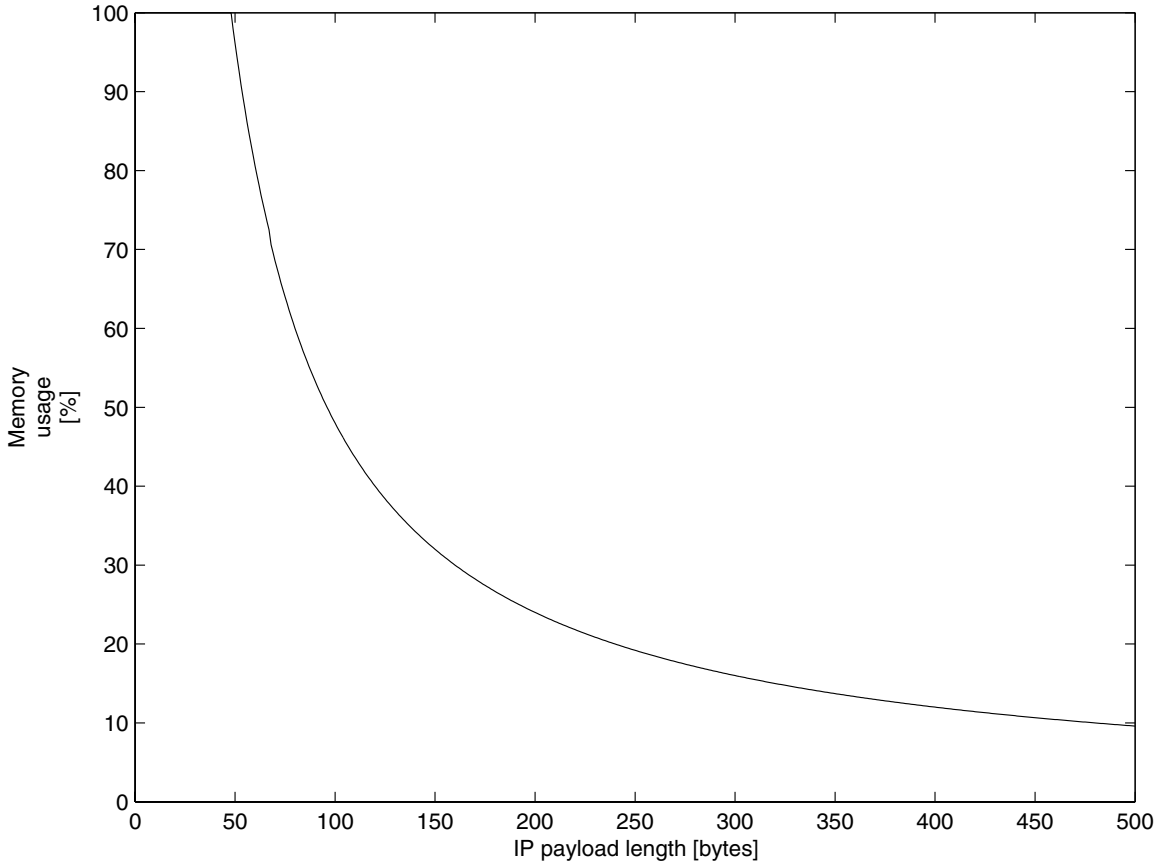

Figure 13.6: The amount of memory used in a cut-through switching system (IP to ATM conversion) compared to IP store-and-forward switching as a function of IP payload length. If the payload size is greater than the size of an ATM packets payload (48 B), the memory-size can be reduced.

2) If a certain flow/session (host-host pair) have higher priority, e.g. demands lower latency during a VoIP session, it is possible to provide low latency using cut-through switching.

Cut-through switching can be applied to some flows or to all. If only some flows uses cut-through switching the others use normal store-and-forward type of switching. A switch typically has a mode where the operating mode of a port will change from cut-through to store-forward if the error rate exceeds a certain (configurable) threshold.

13.3.3 Control memory

The proposed control memory stores inter-packet connection variables. This information is used to support reliable end-to-end communication between network entities. It basically tells the gateway what to do with received packets (ATM or IP). This information is normally shared between the ports of the gateway, e.g. routing tables. Hence, this type of memory is very difficult to distribute because of coherence problem. This is however an interesting topic for future exploration.

13.3.4 Program memory

The proposed PPP has an optimized ISA which is dedicated to control a number of configurable hardware accelerators (Functional Pages). The optimized ISA and the configurable hardware reduces the control-overhead. This means that the size of the program memory can be significantly reduced which saves area, power and consequently money.

13.3.5 Caches

Host Address Caching is an existing solution [5], but in general packet data lacks both temporal and spatial dependencies and therefore the use of data-caches is not very useful in network processors. On the other hand the application processing DSPs in a gateway system may be efficiently supported by data-caches. Instruction caches is normally very useful. But if a small code-memory (including branch-units) is used the use of instruction caches becomes questionable.

In order to successfully distribute caches to different processors in a multi-processor system, it is very important to keep all the caches coherent. E.g. if a routing-table is updated in one NP all other routing tables must be updated. In instruction caches the coherence is not as important.

13.4 Implementation issues

Once a memory organization and hierarchy has been decided, a second stage of optimization starts. The physical and logical implementation have a huge impact on the memory cost and performance.

13.4.1 Memory technology mapping

The selection and development of SRAM, DRAM, and new types of low-latency-access memories is important for bandwidth, latency and power-consumption. Normally all of the memory can not be accommodated on-chip, so a selection of which parts of the memory architecture to be On/Off-chip is important.

13.4.2 Search engines

Search-engines are commonly used to identify the actions to perform in a gateway based on header information of received packets. There are three ways of implementing these search engines. Basically each way represents a special type of memory, i.e. CAM, TCAM, and RAM-based memories. Using combination of this a very efficient classification and filtering can be supported. Besides the design of these search-engines, one interesting research topic is the interfacing between these accelerators and other processors in the system, e.g. look-aside interfaces.

13.4.3 Payload management

A payload manager is essentially a hardware-based memory controller. It keeps track of payload data stored and have three basic functions:

A read-pointer controls which packets should be read from the payload memory for transmission on output links. A write functions allocates and point out memory positions where new payloads are written. Finally the payload-manager should have a free-list function that point out free memory positions. It is especially tricky to keep track of multi-cast messages. They should be kept in the payload memory until the message has been transmitted onto a number of output links.

13.5 Conclusion

As a conclusion the following items are listed as possible and perhaps even suitable future research topics in the area of Protocol Processors for Media Gateways.

- Task analysis to find suitable accelerators
- Evaluation of parallelization schemes
- Development of data-path and signaling interfaces for inter-processor communication
- Memory controller (accelerator) design for fast path processing
- Search-engine design including interfaces using benchmarks
- Evaluation and design of a cut-through switching system using benchmarks
- Cache design
- Input buffer optimization through wire-speed processing
- Memory technology evaluation from a system perspective

References

[13.1]D. Liu, U. Nordqvist, and C. Svensson, “Configuration-Based Architecture for High-Speed and General-Purpose Protocol-Processing”, Proceedings of IEEE Signal Processing Systems 1999, pp. 540-547, Taipei

[13.2]Ulf Nordqvist, “A Programmable Network Interface Accelerator”, Licentiate Degree Thesis, Linköping Studies in Science and Technology, Thesis No. 998, January 2003, ISBN: 91-7373-580-9.

[13.3]Tomas Henriksson, “Hardware Architecture for Protocol Processing”, Licentiate Degree Thesis, Linköping Studies in Science and Technology, Thesis No. 911, December 2001, ISBN: 91-7373-209-5.

[13.4]Tomas Henriksson, “Intra-Packet Data-Flow Protocol Processor”, Doctor degree Thesis, Linköping Studies in Science and Technology, Thesis No. 813, May 2003, ISBN: 91-7373-624-4.

[13.5]T.-C. Chiueh, Pradhan, P., “Cache memory design for network processors”, High-Performance Computer Architecture, 2000. HPCA-6. Proceedings. Sixth International Symposium on, 8-12 Jan. 2000, Page(s): 409 -418

Index