Linköping Studies in Science and Technology

Thesis No. 998

A Programmable Network Interface Accelerator

Ulf Nordqvist

INSTITUTE OF TECHNOLOGY
LINKÖPINGS UNIVERSITET

LIU-TEK-LIC-2002:
Department of Electrical Engineering
Linköpings universitet, SE-581 83 Linköping, Sweden
Linköping 2002

Linköping Studies in Science and Technology

Thesis No. 998

A Programmable Network Interface Accelerator

Ulf Nordqvist

INSTITUTE OF TECHNOLOGY

LINKÖPINGS UNIVERSITET

ISBN: 91-7373-580-9 ISSN: 0280-7971

LIU-TEK-LIC-2002:71

Department of Electrical Engineering

Linköpings universitet, SE-581 83 Linköping, Sweden

Linköping 2002

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.

In the emerging research area of programmable network interfaces, there exist many hardware platform proposals. Most of them aim for router applications but not so many for terminals. This thesis explores a number of different router design alternatives and architectural concepts. The concepts have been examined to see which apply also to terminal designs.

A novel terminal platform solution is proposed in this thesis. The platform is accelerated using a programmable protocol processor. The processor uses a number of different dedicated hardware blocks, that operates in parallel, to accelerate the platform. The hardware blocks have been selected and specified to fulfill the requirements set by a number of common network protocols. To do this, the protocol processing procedure has been investigated and divided into processing tasks. The different tasks have been explored to see which are suitable for hardware acceleration and which should be processed in other parts of the platform.

The dedicated datapath, simplified control, and minimal usage of data buffers makes the proposed processor attractive from a power perspective. Further it accelerates the platform so that high speed operation is enabled.

Acknowledgments

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 PhD student Tomas Henriksson for valuable discussions, for coauthoring papers and for sharing information regarding many different topics (in many different languages).

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 PhD student.

The interesting discussions and especially the software related support of my former office-mate Daniel Wiklund has been very valuable to me.

Sharing the working environment with a inner circle of PhD students like, including Daniel Eckerbert, Henrik Ericsson, Mikael Olausson, Erik Tell, Stefan Andersson, Peter Caputa and Kalle Folkesson has been a true pleasure.

All of the former and present members of the Electronic Devices and the Computer Engineering group are acknowledged for the technical and administrative support. You all deserve a hats-off from me.

I would like to thank ECSEL graduate school for killing my time schedules, financing the work and broadening my technical perspective.

Finally I would like to thank my family for supporting (but not understanding) my work.

Preface

This thesis presents the results of my research during the period 1999-2002. The following three publications are included in the thesis:

- Dake Liu, Ulf Nordqvist, Christer Svensson "Configuration based architecture for high speed and general purpose protocol processing", *in proceedings of SIPS 1999*, Taipei, Taiwan, pp 540-547
- 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
- Ulf Nordqvist, Dake Liu, “Packet Classification and Termination in a Protocol Processor”, *submitted to the HPCA / NP2, to be held in Anaheim February 2003*

Other publications, not included in the thesis:

- Ulf Nordqvist, Dake Liu, “Configurable CRC Generator”, *in proceedings of the IEEE Design and Diagnostics of Electronic Circuits and Systems, DDECS*, Brno, 2002, April 17-19, pp. 192-199
- Ulf Nordqvist, "On Protocol Processing", *in proceedings of CCSSE 2001, Norrkoping,* Sweden, 2001, Mar 14-15, pp. 83-89
- Ulf Nordqvist, “A Comparative Study of Protocol Processors”, *in proceedings of CSSCE 2002,* Norrkoping, Sweden, Oct 23-24 2002, pp. 107-113
- Tomas Henriksson, Ulf Nordqvist, and Dake Liu, “Specification of a configurable General-Purpose Protocol Processor”, *Invited submission to a special issue of IEE Proceedings on Circuits, Devices and Systems.*
- 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

- 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
- 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

Contents

Abbreviations

ASIC	Application Specific Integrated Circuit
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
iSCSI	Internet Small Computer System Interface
LAN	Local Area Network
MTU	Maximum Transmission Unit
NIC	Network Interface Card
NP	Network Processor
NT	Network Terminal
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
SAN	Storage Area Network
SAR	Segmentation And Reassambly

SSL	Secure Socket Layer
TCAM	Ternary Content Addressable Memmory
TCP	Transport Control Protocol
TOE	TCP Offload Engine
UDP	User Datagram Protocol
ULP	Upper Layer Protocol
XAC	eXtract And Compare
XDR	External Data Representation

1 Introduction

1.1 Background and motivation

In the semiconductor industry it is a well known fact that the device production scales according to Moores law illustrated by the table below.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, normally 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. To make the situation even worse, today a new problem has emerged when it comes to communication processing. Historically, I/O data rates increased at approximately the rate of Moore's law, which allowed servers to maintain I/O processing performance from one product generation to the next.

Table 1: Projections of the ITRS Semiconductor Roadmap

	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 Mtransistors	276	439	697	1106	2212	8848

Networking technologies, however, have historically increased data rates in 10 times increments according to the Fibre Channel Industry Association [1.2]. 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 today. 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.Using traditional design methods, we are already experiencing the I/O processing gap problem illustrated by figure 1.1. The obvious solution to this problem is to offload the communication processing from the application processing device and instead use dedicated devices.

Example:

Consider a general purpose RISC machine in a 10 Gbps network. Assume min-sized packets (64 bytes), no gap between the packets arriving (worst case) and 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.

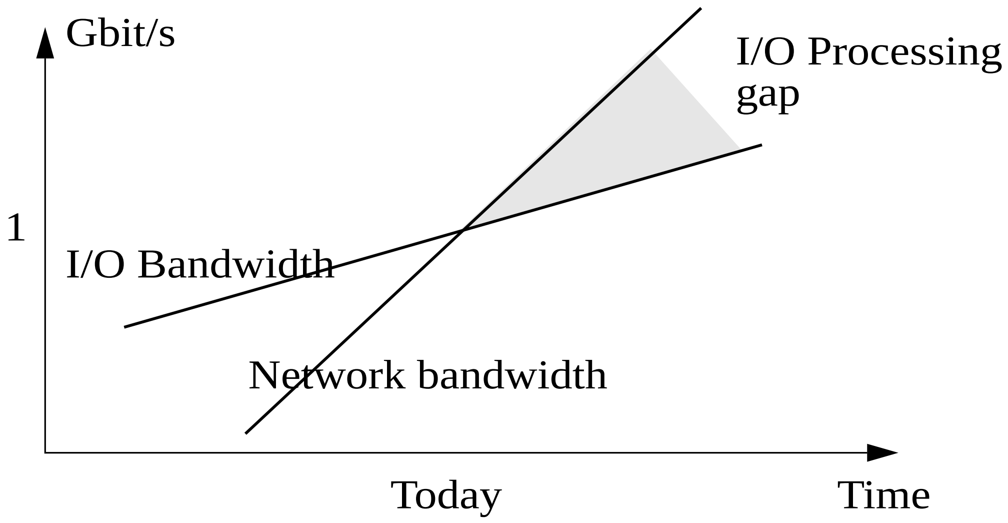

Figure 1.1: The I/O processing gap has started to become a problem using traditional CPU architectures. The reason is that while the I/O bandwidth approximately follows Moores law (1.5-2X) the Network bandwidth has a 10X improvement for each generation.

1.1.1 Research project

This thesis as well as the research project behind tries to attack the problem described in the previous section. The Ph.D. student project has been restricted to only deal with packet reception because of the complexity of the problem. The reason for this is that we consider the packet reception area more challenging and important due to the very hard real time requirements. Hence, the process of packet creation and sending is not very well described in this thesis. No architectural discussion regarding sending will be included, but all the architectures included in the network processor survey in chapter 4 does include sending functionality. Interested readers are encouraged to search for sender related information in the reference list of chapter 4.

The contribution of my work described in this thesis, is to explore the further architectural for network processing. The ongoing research, conducted by me and my colleagues, is partly focusing on defining problems, determine the requirements on a network hardware platforms, attacking the challenging problems described earlier in this section. Based on requirements we defined from the reality, a programmable network interface platform is proposed in this thesis. The project is going on and the architecture is constantly improving.

1.2 Outline

This thesis consists of two main parts, organized as follows. The first part including this and the 3 following chapters, describes available solutions and applications. Chapter 2 describes the basic concepts of computer networks, including common protocols, applications, processing tasks and equipment. Chapter 2 should be regarded as a introductory tutorial for readers with no or little background in the area of computer networks.

In chapter 3, a number of different hardware design considerations important for the design of programmable network interfaces (PNI) are included. These design considerations applies to any types of applications, protocols and networks. The chapter also lists a number of ways to classify and compare, the type and performance of PNI hardware platforms.

Chapter 4 consists of a survey of available PNI solutions. The survey covers many different applications and architectures, both from industry and university research groups.

The second part of the thesis contains my research proposals, results and the three included papers. In chapter 5 a set of protocols is presented. Based on this protocol set, a number of required processing tasks have been determined. The tasks are listed in chapter 5. Finally the chapter includes a proposal of a hardware architecture and methodology, dedicated for high-speed and flexible processing of the required processing tasks. The architecture is described in detail and measured using the performance parameters introduced in chapter 3.

Finally the last three chapters consist of the three papers included in the thesis.

References

[1.1] International Technology Roadmap for Semiconductors, on the internet, http://public.itrs.net/
[1.2] FC Magazine, Fibre Channel Industry Association - Europe, on the internet, http://data.fibrechannel-europe.com/magazine/

2

Network basics

2.1 Packet based networks

This chapter includes a brief introduction to the concept of packet based networks including computer networks. For readers seeking deeper understanding in this area, I recommend the following books. A good first encounter of the area and an excellent starting point for further reading is provided in [2.1]. In [2.2] the most common physical layer protocol, Ethernet and Gigabit Ethernet are explained. TCP/IP is described in depth in [2.3] and [2.4]. Readers with a background knowledge in the area of computer networking, may go directly to the next chapter for further reading.

2.1.1 ISO/OSI Protocol layers

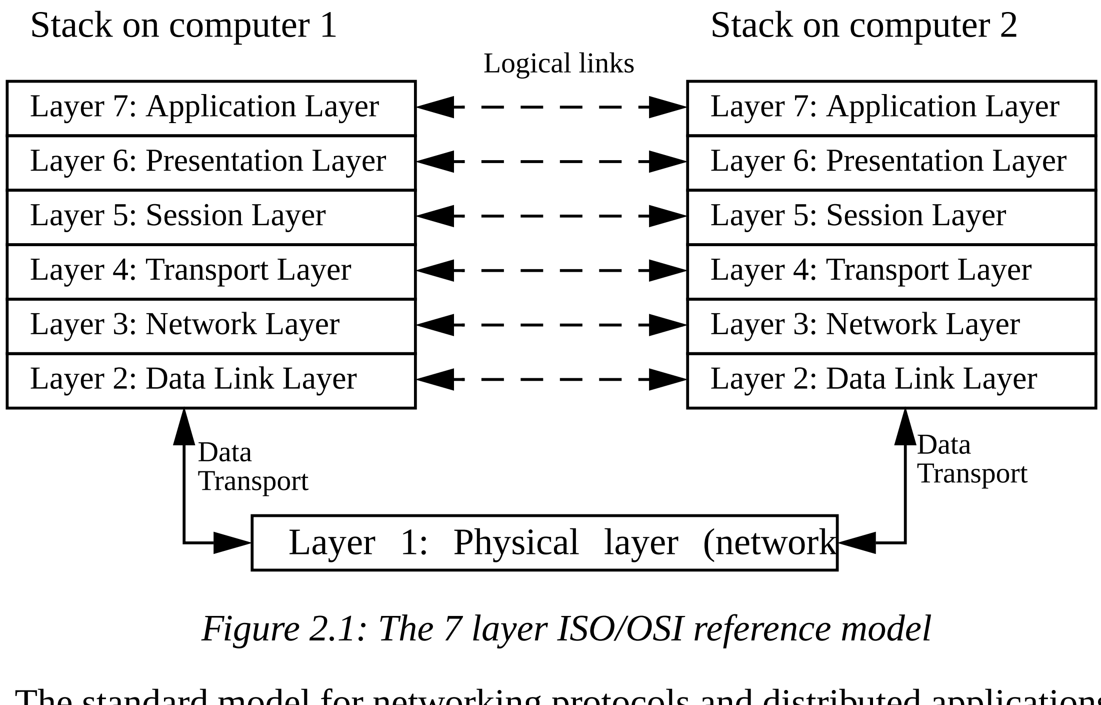

Figure 2.1: The 7 layer ISO/OSI reference model

The standard model for networking protocols and distributed applications is the International Standard Organization's Open System Interconnect (ISO/OSI) model. It defines seven network layers.

- Layer 1 - Physical

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

Data Link layer defines the format of data on the network. A network data frame, a.k.a. packet, includes 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 uses Internetwork Protocol (IP) as its network layer interface. IP is responsible for routing, e.i. 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. The network portion of the IP is assigned by InterNIC Registration Services, under the contract to the National Science Foundation, and the host portion of the IP is assigned by the local network administrators. For large sites, the first two bytes represents 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 it 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 difference 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.

2.1.2 TCP/IP Protocol layers

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 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 four network layers defined by TCP/IP model are as follows.

- 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.

2.1.3 Traditional layer 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 service entities, e.g. devices or pieces of software. The problem is the waisted 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 application processing host. The different services are further discussed in section 2.2.

2.1.4 Local Area Networks

Local Area 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 that covers a relatively small geographic area. It typically connects workstations, personal computers, printers, servers, and other devices. Devices commonly used in LANs include repeaters, hubs, bridges, LAN switches, and routers. A repeater is a physical layer device used to interconnect the media segments of an extended network. Repeaters receive signals from one network segment and amplify, retime, and retransmit those signals to another network segment. Repeaters are incapable of performing complex filtering and other traffic processing. In addition, all electrical signals, including electrical disturbances and other errors, are repeated and amplified.

- Wireless LAN

Today there exist a number of different protocols for wireless LAN applications. They differentiate a lot in terms of performance, cost-figures, cod-

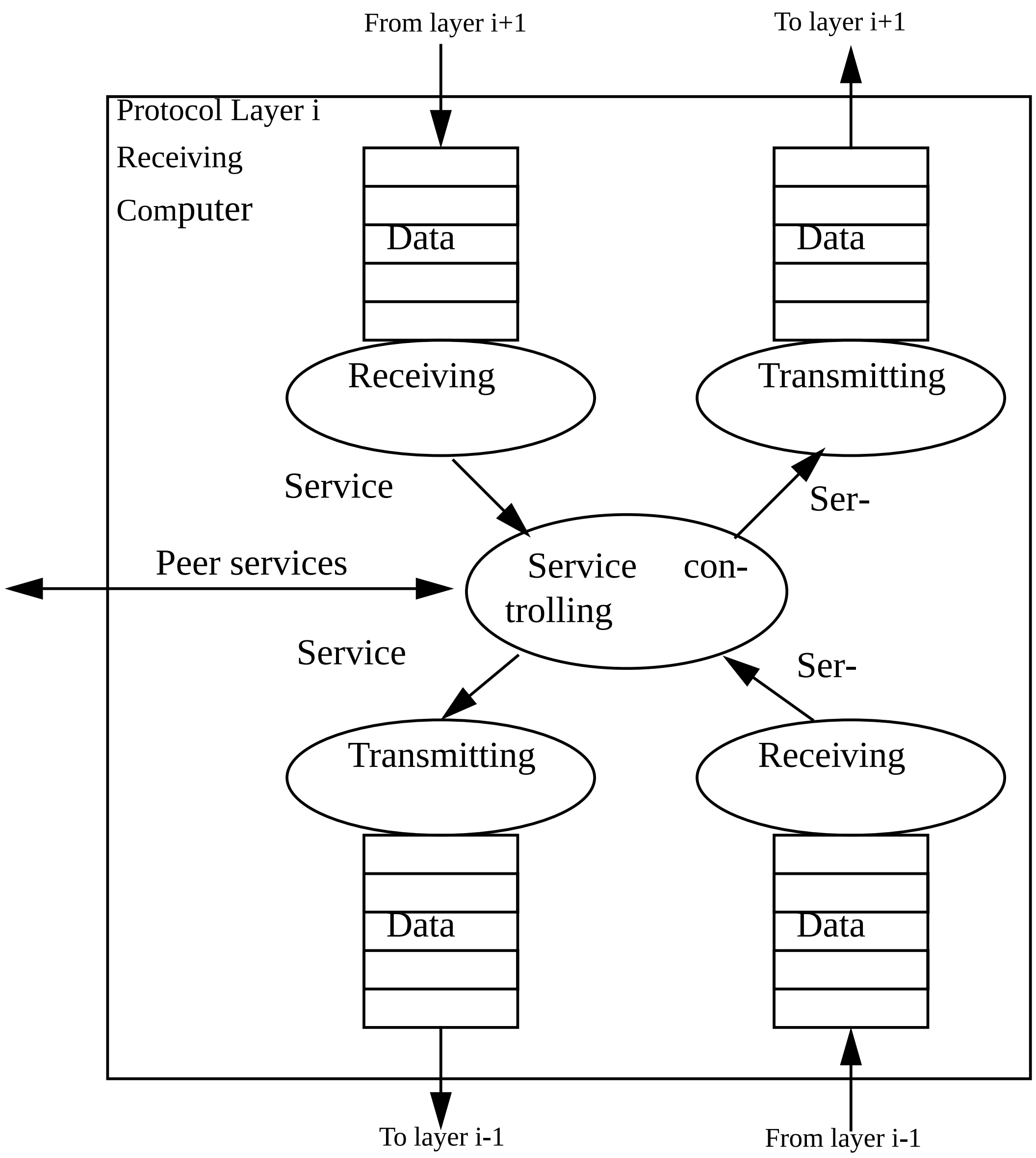

Figure 2.2: Traditionally layered protocol processing concept. During reception each protocol layer receives data and other services from the layer below. The data is processed in order to provide the peer services to the transmitting computer. In the same way all protocol layers provides services to 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.

ing schemes and connection orientation. The main alternatives are IEEE standard 802.11 and HiperLAN ([2.5] and [2.6]).

2.1.5 Storage Area Networks (SAN)

The usage of SAN is currently growing very fast. SAN is normally used for connections to and from file servers. They provide a very high bandwidth and the dominating protocol is Internet SCSI (iSCSI). Normally the SAN protocols are used on top of TCP/IP. Some examples on host bus adaptors (HBA) and SAN accelerators will be discussed in chapter 4. More information regarding SAN protocols, devices and applications can be found in [2.7].

2.1.6 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

2.1.7 Quality of Service

Fundamentally, QoS enables the possibility to provide better service to certain flows. 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. The QoS concept has been a huge research area for several years by 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.

2.1.8 Network performance figures

Some common networks and their performance figures are listed below.

Table 2: 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 2 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.

2.2 Protocol services

Regardless of the protocols used in a computer network, there exist 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 give a unique set of requirements on the processing this common set can not be a bit level correct processing description. Instead they 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.

2.2.1 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, coding 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.

2.2.2 Control flow selection

Decisions on how to process the packet can be made based on the parsed information. This decision making 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

```
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 2.3: Control flow selection pseudo-code.

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 2.3.

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

2.2.3 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

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. The decisions are then made based on the parsed information.

2.2.4 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 only is 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
- Other Coding/Decoding
- Encryption/Decryption

2.2.5 Datastream management

In network terminals the datastream 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 re-assembled 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 send packets.

2.3 Traditional network components

2.3.1 Network Terminals

Network terminals (NT) exist for many different applications. Some examples are desktops, printers and IP phones which are normally connected to the network using wired connections, e.g. Ethernet LANs. There also exist many wireless applications where the NTs are used, e.g. a mobile phone or a PDA, connected to a WAN. 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. This means that no routing decisions have to be made.

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 2.1.2. An example of some processing tasks in the NT and the simplified hardware allocation, is illustrated by figure 2.4.

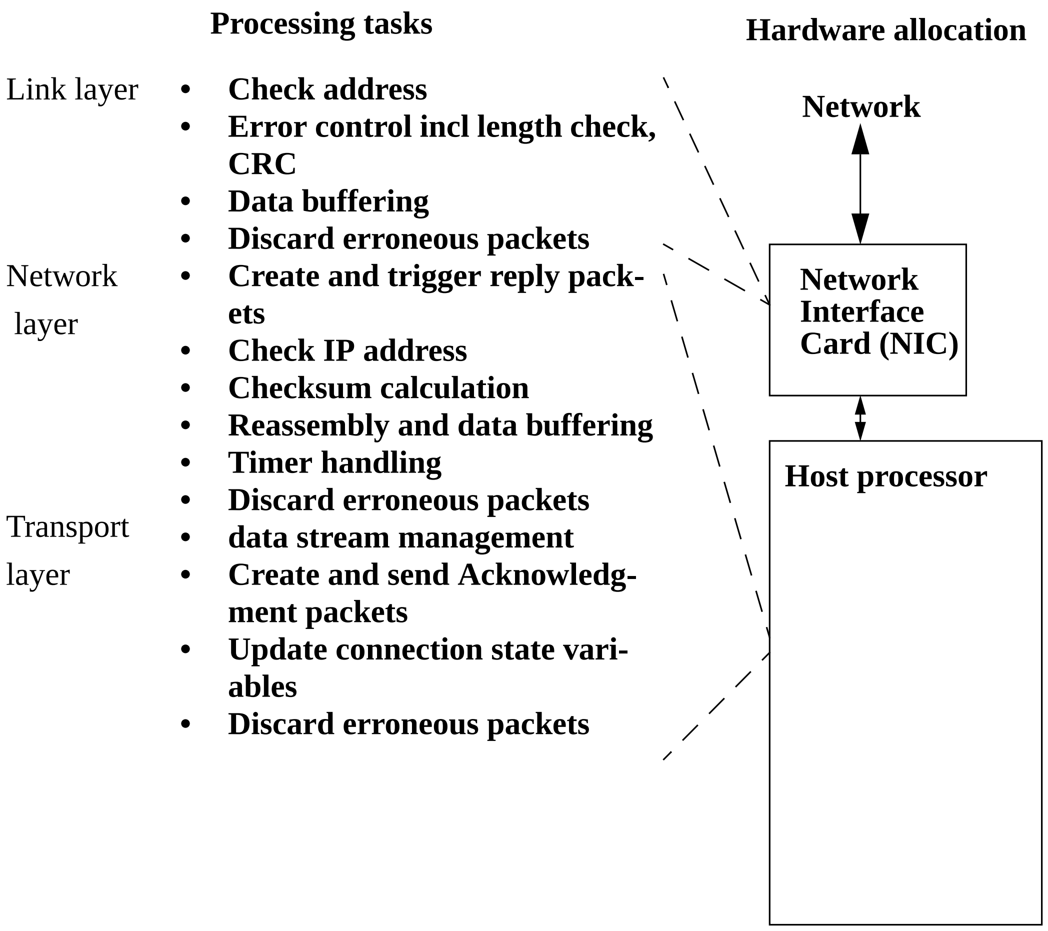
Processing tasks
Hardware allocation
Link layer
Check address
Error control incl length check, CRC
Data buffering
Discard erroneous packets
Network layer
Create and trigger reply packets
Check IP address
Checksum calculation
Reassembly and data buffering
Timer handling
Discard erroneous packets
Transport layer
data stream management
Create and send Acknowledgment packets
Update connection state variables
Discard erroneous packets
Network
Network Interface Card (NIC)
Host processor

Figure 2.4: 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.

2.3.2 Routers

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 only in the terminals the protocols in the transport layer and above will be processed. There are also simpler routers, only capable of lower layer processing. Layer 1 processing routers are normally called repeaters. Layer 2 routers are known as switches.

The main goals of a router are:

- To pass on incoming packets to the correct network link.
- To provide error control and security to the communication channels established.

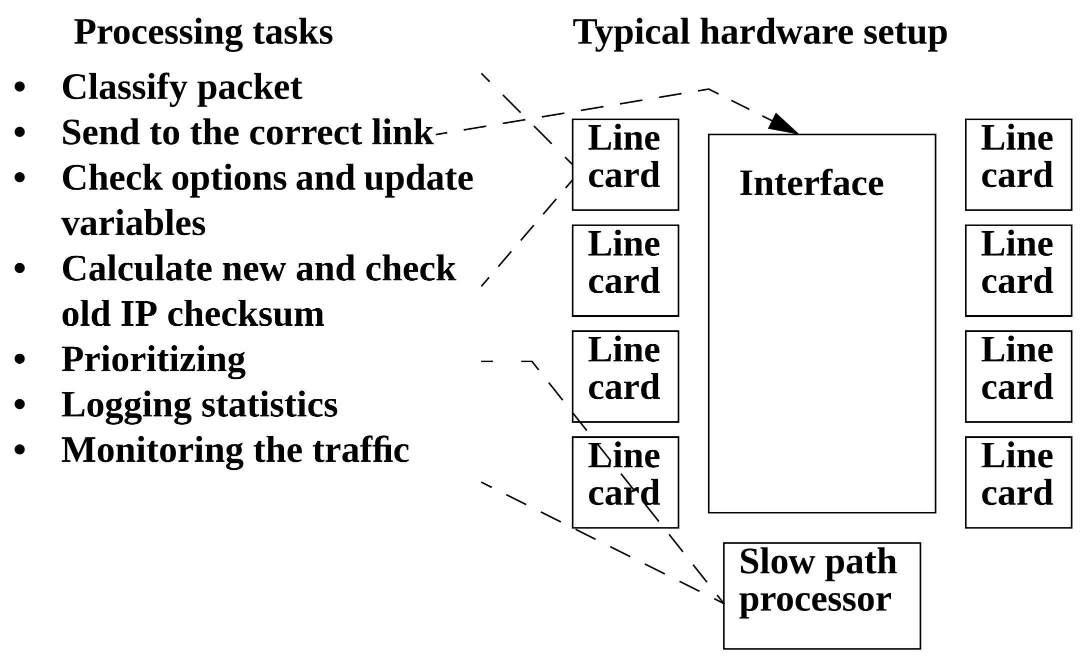

Figure 2.5: Example of processing tasks and allocation in a traditional type of router.

- To monitor and control the traffic flow so that it is optimal from the Internet Service Providers (ISP) point of view. E.g. management of billing and bandwidth resources.

Normally a router includes 3 basic components. They are line cards, interfacing backplane and a slow path processor (normally a PowerPC). Some examples on processing tasks and a typical router hardware architecture is illustrated by figure 2.5.

References

[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

3

Hardware platforms

3.1 Architectural challenges

When designing high speed programmable network interfaces (PNI) 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**
- **Power dissipation**
- **Pin limitation**
- **Packaging**
- **Verification**

Others are specifically important in PNI designs, e.g.

- **Line-rate processing (fast path processing)**
- **Link-rate processing (slow path processing)**
- **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 (ILP)

- **Multi processor solutions**

  Task level parallel or pipelined architectures

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. For further information on the design challenges and consideration when designing PNIs, I strongly recommend the new book [3.1]. In the Ph.D. thesis [3.2] a deep discussion on memory architectures can be found.

3.2 Design alternatives

Today, there are a number of different hardware platforms available for use as PNI. In order to investigate the need for, and type of PNI hardware a classification of the different solutions is useful. When selecting hardware solution the first step is to analyze the requirement and then select the type of hardware platform to use. Some of the most common PNI hardware platform design alternatives are discussed in the following subsections.

3.2.1 Inter- or intra-layer processing

Intralayer processing means that each protocol layer is processed separately according to figure 3.1. This way of processing and conceptual thinking is a result of the invention of computer networks and protocol stacks for more than 30 years ago. In the seventies the 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 reason why so many companies and research groups propose intralayer processing to be considered.

The main advantage with interlayer 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

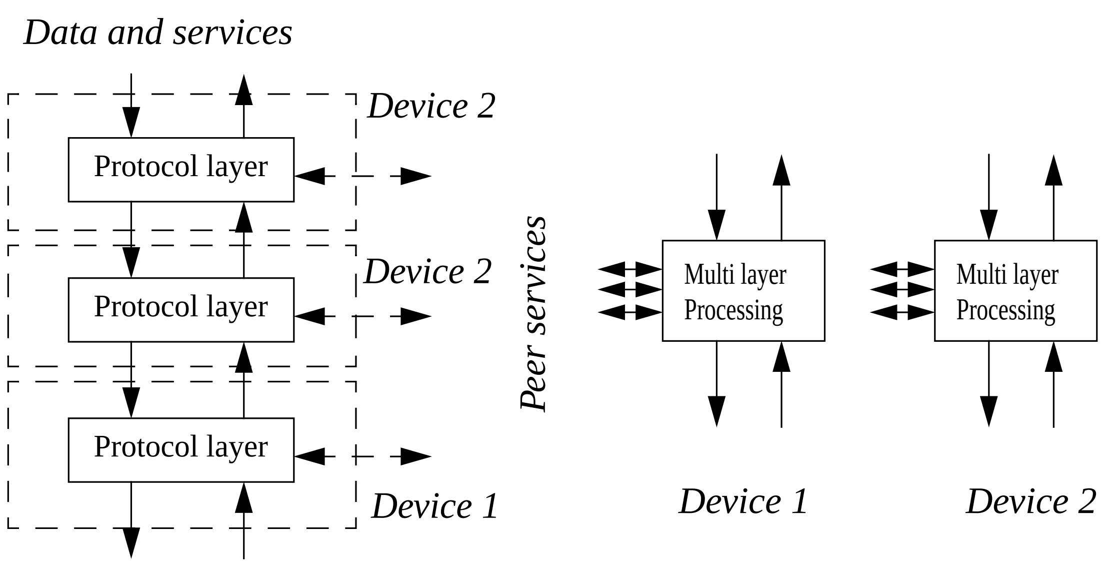

Figure 3.1: Interlayer (to the right) processing means that all or parts of several protocol layers are being processed simultaneously on one device. It does not mean that all processing is done on one piece of hardware or software. Several devices can still share the processing. Intralayer processing 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.

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.

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.

3.2.2 Type of control

The 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 can not be changed every clock cycle. The control ability 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 PNI 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. The 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. Hence, the amount of flexibility and the type of control a hardware platform uses, is an important design parameter.

3.2.3 Application coverage

The ability to run a certain set of network applications on the host using the interface in certain networks is described by the application coverage of the PNI. The problem is that the complexity and hardware cost grows as the application coverage grows.

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

The second requirement for a PNI to have a large application coverage is that it is flexible enough so that it can process all the different applications (protocols). The application coverage defines what the PNI can be used for. Therefore it is the single most important classification parameter. Normally it is useless to compare the performance figures of two different PNI architectures that have different application areas.

3.2.4 Offloading coverage

The solutions available today from the academic research community and the industry are extremely diverse. Despite this diversity, the communication network platforms can be divided into four main groups according to their offloading strategy illustrated by figure 3.2.

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 PNI can then typically process 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 3.2. Consequently it is very important to clearly examine both which protocol and how big part of the protocols should be offloaded. Protocols not offloaded must of course be processed by the host.

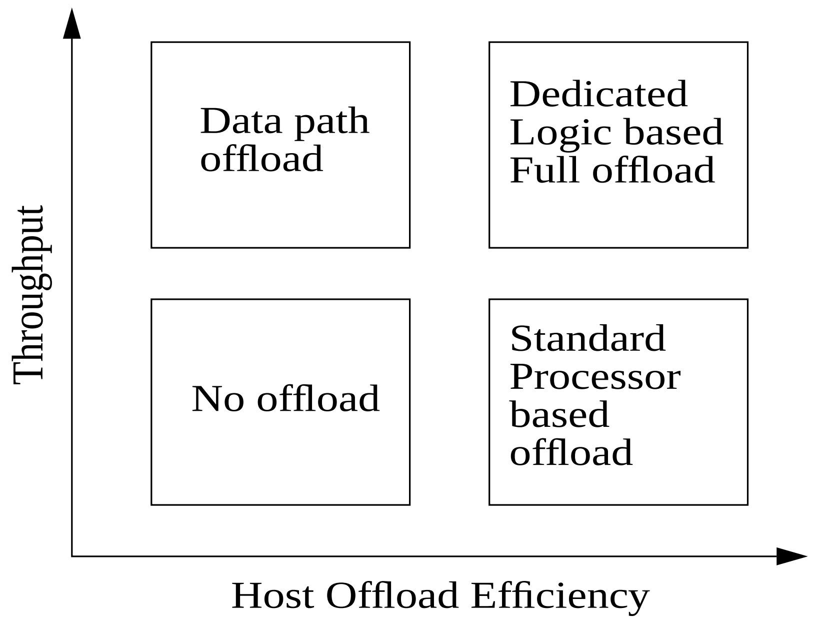
Data path offload
Dedicated Logic based Full offload
Throughput
No offload
Standard Processor based offload
Host Offload Efficiency

Figure 3.2: Host offloading strategies

3.2.5 Chip or board integration

Processors and memories in a PNI ASIC chip are 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 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 effectively. 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. More on accelerated memory access using CAMs is presented in paper 3, chapter 8.

3.2.6 Configurable logic

To implement parts or the whole of a PNI 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 very cost effective solution 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 will always be 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.

3.3 Performance measures

A number of different performance figures must be compared in order to evaluate and compare different hardware platforms to find the most suitable one for use as a PNI. The most important ones are described in this section.

3.3.1 Flexibility

A PNI must provide flexibility and adaptability to the changing environment it might operate in. This results in some flexibility requirements that all PNIs has 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 the data header information.

Providing the flexibility bulleted above gives a large protocol coverage but it increases the complexity of the hardware. There is always a tradeoff 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.

3.3.2 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.

3.3.3 Inter operability

The main purpose of a PNI is to offload and relax the host processor as much as possible. Hence, it is very important that the interfacing communication between the PNI and the host does not disturb or interrupts the processing on the host CPU. Further it is important that the host operating system can manage and access the data buffers as well as communication with the PNI in an efficient and non-interruptive way. In order to reach an optimal way of integrating the PNI device into the system, both the PNI interface and the host operating systems must be 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 a very important for the overall system performance.

3.3.4 Cost

The cost of the PNI chip or board is very important performance figure. The cost is important for any customer but network terminal user are especially cost sensitive. The cost is always an important part of architectural design tradeoffs. The cost off a PNI chip mostly depend 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 system but it is especially important for network terminals. The power figures are of course even more important in portable systems. The number of pins is hard to lower since a PNI by nature includes a lot of communication over the chip edge. 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.

3.4 Application Specific Accelerators

In order to improve the performance of PNIs used either in network nodes or terminals, dedicated hardware blocks are often used. The main purpose is to offload the offloading devices (PNI) by taking care of the computational heavy data intensive processing. New accelerator types for higher layer offloading emerges every year. Some of the accelerator types available today are:

- Two- or one-dimensional classification engines. Could be CAM, TCAM or RAM based.
- Storage Area Networks (SAN) Engines. Used in file servers.
- PHY and MAC layer ASICs
- Segmentation and reassembly (SAR) engines.
- Crypto engines
- Hardware timer assisting engines

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

4

Programmable Network Interfaces - A Survey

4.1 Naming convention

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 hole TCP protocol. Regardless which, it will still be presented to the customers as a TOE.

4.2 Commercial architectures

4.2.1 Motorola C-Port C-5e Network Processor

The C-5e NP is a part of Motorolas C-Port family. It supports the use of 16 line interfaces, each controlled by a 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 bandwidths 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 a external SRAM memory.
- Buffer management unit that controls the payload data storage while the header is being processed.
- Queue management unit that is shared between all the processors to provide QoS.
- Fabric processor provides a high-speed network 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.

4.2.2 iSNAP

The IP Storage Network Access Processor from Silverback [4.1] terminates and process IP-based storage traffic in a GE with full duplex. It separates 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

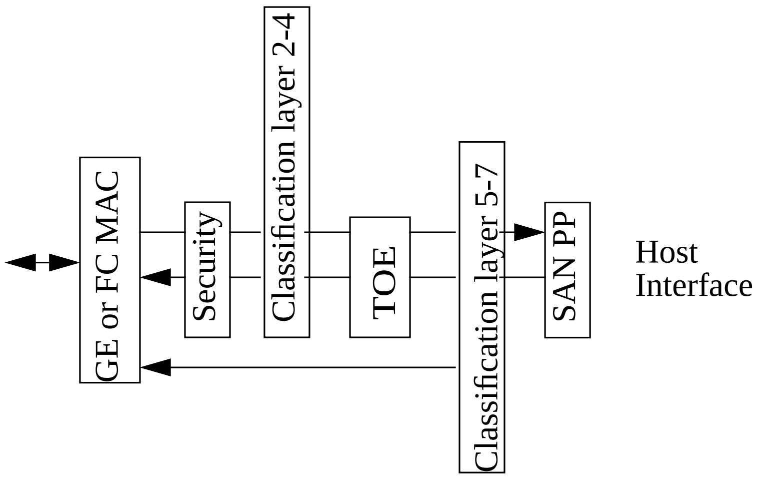

Figure 4.1: Trebia SNP architecture.

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.

4.2.3 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 operates in pairs called DPPUs. Each DPPU have a shared set of co-processors that operates in parallel. The picoprocessors are essentially 32 bit scaled-down RISC machines with an dedicated instruction set. The DPPU also contains a small (4 kB) shared memory. The co-processors handles tree search, data storage, control access, queues, checksums, string copy, policy, counters, buses and system semaphoring.

4.2.4 Trebia SNP

This architecture [4.2] includes MAC block for mixed medias (wired and fibre-based), a security accelerator, various classification block, a TCP offload engine and a Storage Area Network (SAN) [1] Protocol processor as illustrated by figure 4.1. The TCP offload engine can operate stand alone, terminating TCP connections without involving the host processor. For IP

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

storage applications they 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.

4.2.5 iReady EthernetMAX

The Media Access Xccelerator [4.4] from iReady is intended for transport offload [4.3]. It fully terminates TCP/IP at GE speed. The TCP/IP accelerator uses a streaming data architecture similar to the one proposed by this papers author. The data is not stored but instead processed while it is streaming through a 64 bit wide pipeline. The 64 bit wide datapath then process the data using multiple dedicated hardware blocks implementing different state machines. Each state machine block process 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 not is large in the network. The implementation does not use standard programmable devices. Instead dedicated logic for optimal performance is used.

4.2.6 Alacritech Internet PP

Alacritech [4.5] provides a Session Layer Interface Card (SLIC) [4.7] that includes accelerators for GE, network acceleration [4.6], 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.

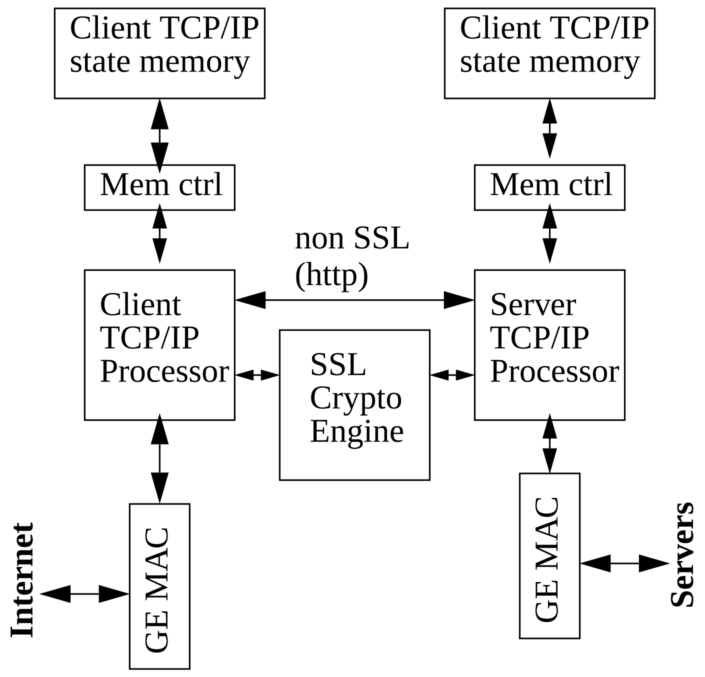

Figure 4.2: The UltraLock provides acceleration for SSL connections. Ordinary http packets are passed on without any processing in the SSL engine.

4.2.7 LayerN UltraLock

The UltraLock [4.9] illustrated by figure 4.2 uses a patented architecture named SIGNET [4.8]. 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.

4.2.8 Seaway Streamwise NCP

Seaway Networks [4.10] 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.

4.2.9 Emulex LightPulse Fibre HBA

The host bus adapter (HBA) from Emulex [4.11] 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 performance and the main feature is the implementation of a strong SAN accelerator for high end servers.

4.2.10 Intel IXA/IXC/IXS/IOP processors

Intel offers a number of chips to solve different tasks when it comes to what they call Network Infrastructure Processing [4.13]. First of all they have the Internet eXchange Architecture (IXA) which includes different NP. They 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. In for example the IXP 1200 the datapath includes 6 different micro engines which supports multithread 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 a application specific instruction set. The microengines shares 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 memory capable of storing 640 32 bit data values. 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).
- Hardware timers.
- Multiplier
- 16-entry CAM used for cache search and assists software pipelining.

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

The IXA type chips 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 operates together with a 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 pro-

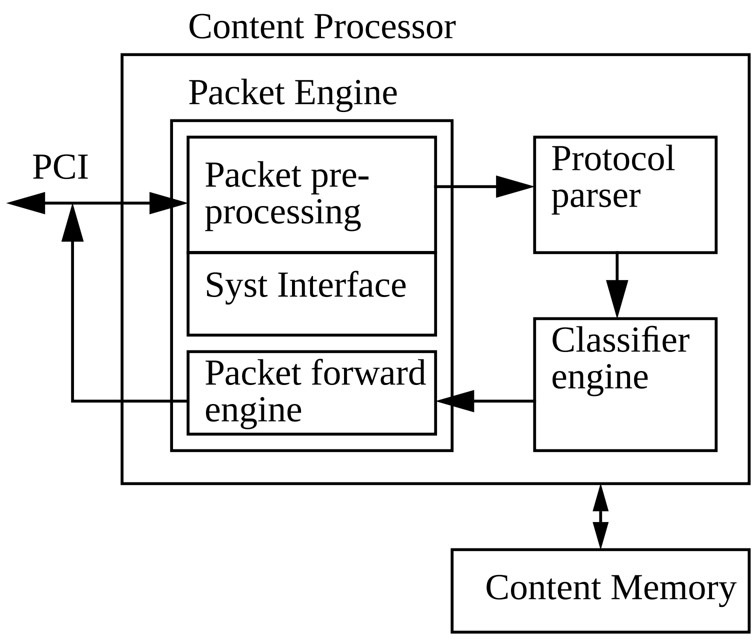

Figure 4.3: LeWiz content processor. The Packet pre-processor is a TOE. The Protocol parser examines the ULP data (layer 5-7) and based on this it start a search for a classifier using the classifier engine. The classifier then decides priority and is used for re-direction of the traffic according to the QoS policy.

cessors (IOP) that is a quite general architecture which can be used for SAN acceleration.

4.2.11 LeWiz Content processor

LeWiz processor [4.12] process layer 3-7 with hardware acceleration with a line rate capability of Gbit/s. Among other things it performs table lookup for connections, controls a external header data memory, support 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 4.3.

4.2.12 Qlogic SANblade

The SANblade [4.14] manage 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.

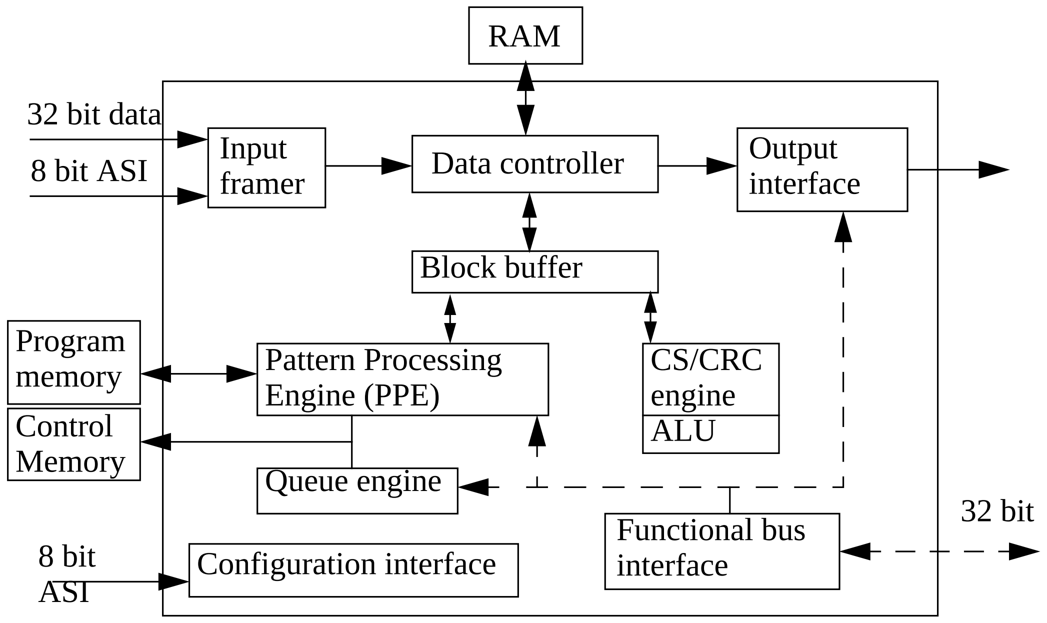

Figure 4.4: FPP architecture.

4.2.13 Agere Systems - PayloadPlus

PayloadPlus provides a complete solution for OC-48c (2.5 Gbps) networks. The board solution includes 3 chip, capable of up to layer 7 processing. They are the 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 at a time. Each PDU is processed by a separate processing threds called contexts. 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 standardbus. The main applications is 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.

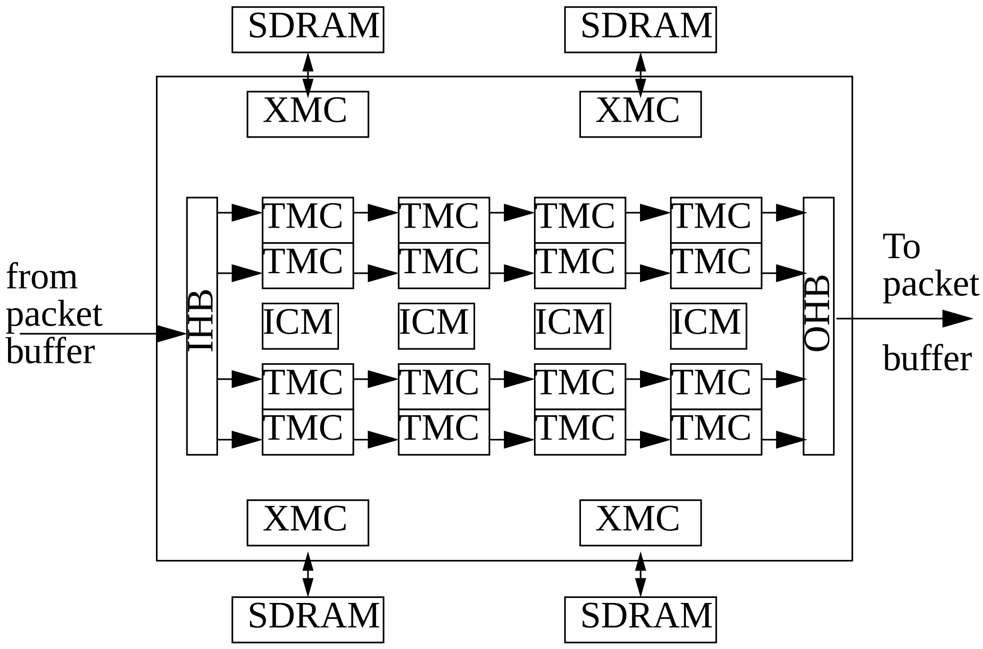

Figure 4.5: The Toaster2 architecture. IHB/OHB are uni-directional bus interfaces that are 64 bit wide and can operate at 100 Mhz and above. The TMC blocks are memory controllers that controls the access to each Internal Column Memory (ICM) while the XMC handles access to external memory devices.

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.

4.2.14 Cisco - Toaster2

Toaster2 is a multiprocessor ASIC solution. 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.

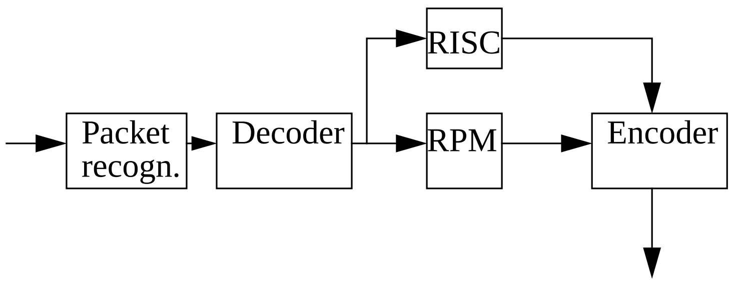

Figure 4.6: The PRO^3 architecture.

4.2.15 PMC-Sierra ClassiPI

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

4.3 Academic architectures

4.3.1 EU Protocol Processor Project PRO^3

The architecture proposed by PRO^3 [4.15] consists of 5 parts. Most interesting is the Reconfigurable Pipelined Module which process the 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 4.6.

4.3.2 UCLA Packet decoder

This decoder [4.17], decodes packets on layer 2-4. The decoder architecture illustrated in figure 4.7 consists of one datapath for each layer, e.i. 3 data paths totally. It only uses one control path for the signaling processing. It operates on streaming data using a application specific instruction set and the intended application area is routers.

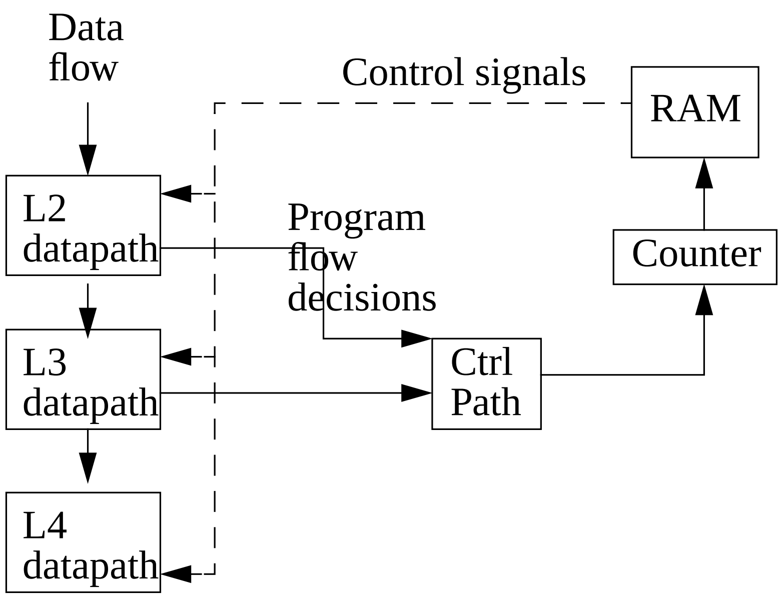

Figure 4.7: Simplified view of the UCLA processor architecture proposal showing how to accelerate case-jump functions.

4.3.3 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 developed. The intended application area is primary the ATM protocol. The framework and the architecture is described in [4.18].

4.3.4 PICO project from Berkeley

The PICO project is a 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 (PLD). The PICO processor is further described in [4.19].

4.4 Conclusions from survey

A number of different PNI solutions is included in the survey. They all are focused on different application areas. Some fully offloads complex protocols, while others mainly focus on high-speed 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. There is a general disagreement on how much functionality to include in the PNI and how much should be left for the host. 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 PNI:s 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 here further exploration is needed. One big question that remains unanswered is where the re-ordering 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 exists solutions to the various new PNI specific implementation problems and considerations discussed in chapter 3. It is also clear that the academic research community is far behind the industry and just have started to examine these implementation issues. Examples on interesting research areas still remaining are host OS interface, shared memory control etc.

References

[4.1] Silverback Systems homepage, *on the www*, http://wwwsilverbacksystems.com

[4.2] Trebia Networks homepage, *on the www*, http://www.trebia.com

[4.3] National Semiconductors, “Enabling Next Generation Ethernet”, *on the www*, http://www.trebia.com/EthernetMAXweb.pdf

[4.4] Minami, et al, “Multiple network protocol encoder/decoder and data processor”, *US patent, no. 6 034 963*

[4.5] Alacritech Inc. homepage, *on the www*, http://www.alacritech.com

[4.6] Boucher, et al, “TCP/IP offload network interface device”, US patent, no. 6 434 620

[4.7] Boucher, et al, “Intelligent network interface system method for protocol processing”, US patent, no. 6 434 620

[4.8] LayerN Networks, “SIGNET - Secure In-line Networking”, *on the www*, http://www.layern.com/SIGNETWP020419.pdf

[4.9] Omura, et al, “The Evolution of Modern Digital Security Techniques”, *on the www*, http://www.layern.com/EvolutionWhitePaper.pdf

[4.10]Seaway Networks homepage, *on the www*, http://www.seawaynetworks.com

[4.11]Emulex homepage, *on the www*, http://www.emulex.com

[4.12]LeWiz Communication,Inc. homepage, *on the www*, http://www.lewiz.com

[4.13]Intel Corp., “Network Infrastructure Processors - Extending Intelligence in the Network”, *white paper on the www*, http://www.intel.com/design/network/papers/251690001.pdf

[4.14]QLogic Corp. homepage, *on the www*, http://www.qlogic.com

[4.15]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

[4.16]C. Georgopoulos et al, “A Protocol Processing Architecture Backing TCP/IP-based Security Applications in High Speed Networks”, INTERWORKING 2000, Oct. 2000, Bergen, Norway

[4.17]M. Attia, I. Verbauwhede, “Programmable Gigabit Ethernet Packet Processor Design Methodology”, ECCTD 2001, vol. III, pp. 177-180

[4.18]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

[4.19]T. Tuan, S.-F- Li, J. Rabaey, “Reconfigurable Platform Design for Wireless Protocol Processors”, ICASSP 2001, pp. 893-896

5 Proposed Architecture

5.1 Introduction

This chapter describes a hardware architecture proposal which is a result of my research during 1999-2002. The architecture is a PNI Accelerator dedicated for packet reception in a network terminal. The architecture was originally introduced in the paper presented in chapter 6.

My and my colleagues have investigated and implemented different parts of the architecture to find optimal architectural solutions. These investigations have resulted in the two papers in chapter 7 and 8.

This chapter will give an overview of the architecture. It also includes a performance discussion based on performance parameters introduced in chapter 3.

5.1.1 Naming convention

During the progress of the research work, our architecture has changed name several times. The reason for this is that the research field is so immature that no naming conventions has been agreed on. During the hole design time we have used the name Protocol Processor (PP) describing our PNI. The PP 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 other part of the PP which implements the fast path.

In the papers included in the following 3 chapters, the PP fast path has been characterized 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. The name that will be used in this chapter is PPP. Note that the PPP is a part of the PP.

5.1.2 System perspective

As mentioned earlier the proposed PNI architecture is called protocol processor and it consists of two parts. The first part is the Programmable Protocol Processor (PPP) and the other is the micro controller (µC). The platform also includes two RAM memories. They are used as program memory and as a control memory which stores interpacket control variables. The PP is intended to be a part of a SoC where one or several PPP together with the µC acts as a high speed PNI. An overview of the system is illustrated by figure 5.1.

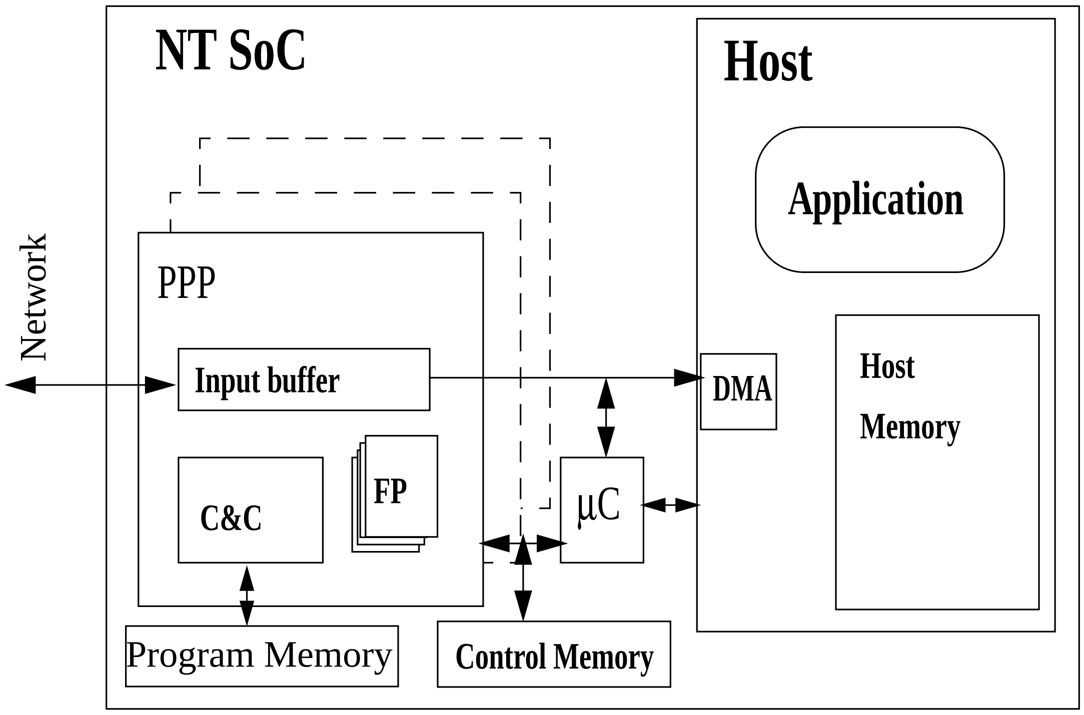

Figure 5.1: The PPP together with a general purpose micro controller (µC) handles the communication of one network media port. In a system on chip (SoC) many PPP can be used as port-processors in order to provide high bandwidth between the application and the network.

5.2 Processing tasks

5.2.1 Protocol suite overview

To test our architecture we have used a common set of protocols. The protocols are useful for investigations on architectural requirements and possibilities. This however does not mean that the architecture not is suitable for other application areas and protocols. The protocols we have chosen to include in our protocol coverage are:

- Fast Ethernet with PHY interface MII
- Gigabit Ethernet with PHY interface GMII
- 10 Gigabit Ethernet with PHY interface XGMII
- IP version 4 and version 6
- Address Resolution Protocol (ARP)
- Reversed Address Resolution Protocol (RARP)
- Internet Control Message Protocol (ICMP)
- Internet Group Management Protocol (IGMP)
- TCP
- 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 ethernet interface is

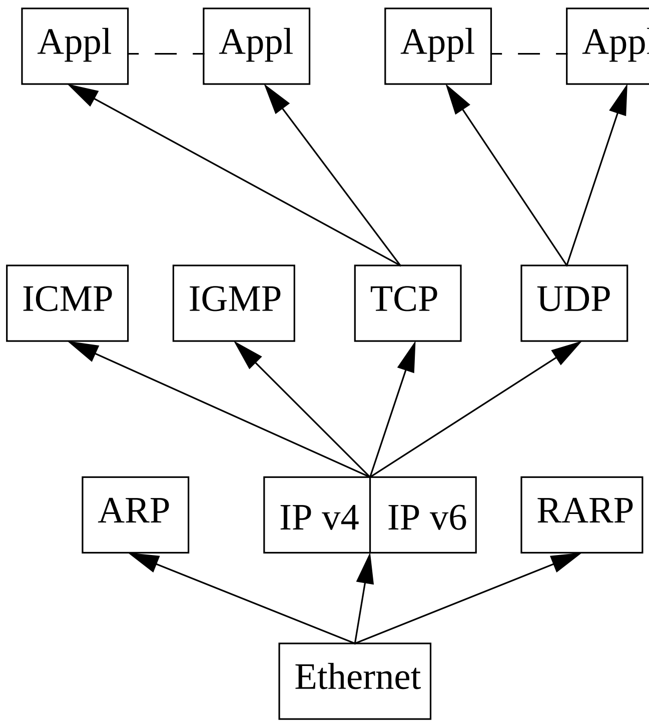

Figure 5.2: The data demultiplexing of a received Ethernet frame.

Eth header	IP header	TCP/UDP header	DATA	CRC

Figure 5.3: One Ethernet frame encapsulate the IP packets. Each layer includes a header and data.

received, it will be passed on to different processing units depending on which protocols that have been used, according to figure 5.2.

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 5.3.

In order to process all the headers and providing 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 are specific to the selected protocol suite. If new protocols should be included, new processing task types may, or may not, be needed. The processing tasks are listed in the subsections following.

5.2.2 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 hole 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 paper 2. Hence, the CRC calculation is normally done using dedicated hardware assist. Paper 2 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 we discard the packet.

- Check the type field

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

- 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 to the correct location for further processing.

5.2.3 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 a IP address but do not know the 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.

5.2.4 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 is the same as for the ARP protocol.

5.2.5 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 5.2.7.

- 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 segmentation. In order to obtain the original packet the receiving terminal must reassembly 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 is two types of lengths involved in IP processing. One describes the header length which is used to know when the data starts. The other describes the total length which is used to see if all fragments have been received. The names of the fields differs 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.

5.2.6 ICMP and IGMP

ICMP normally communicates error messages and exceptions, or 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. If the type is 1, the packet is a query, and if it is 2 it is a reply.

- 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.

5.2.7 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 defines 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 detail in paper 3.

- 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.

5.2.8 UDP

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

- Extract Ports and check connections

Similar to the TCP task. I call it a connection although we only check if the port is open.

- Extract length field

To know when the hole payload has been received.

- Calculate header checksum

5.3 Proposed datapath

A datapath of the PPP has been developed and optimized based on the processing tasks introduced in section 5.2. The datapath of the PPP includes 3 types of components, the input buffer, the functional pages and a control memory accelerator. An overview of the PPP architecture is illustrated by figure 5.4

5.3.1 Input buffer

When data arrives from the network interface (GMII) to the PPP it streams through a chain of 32 bit wide flip-flop registers illustrated by figure 5.5.

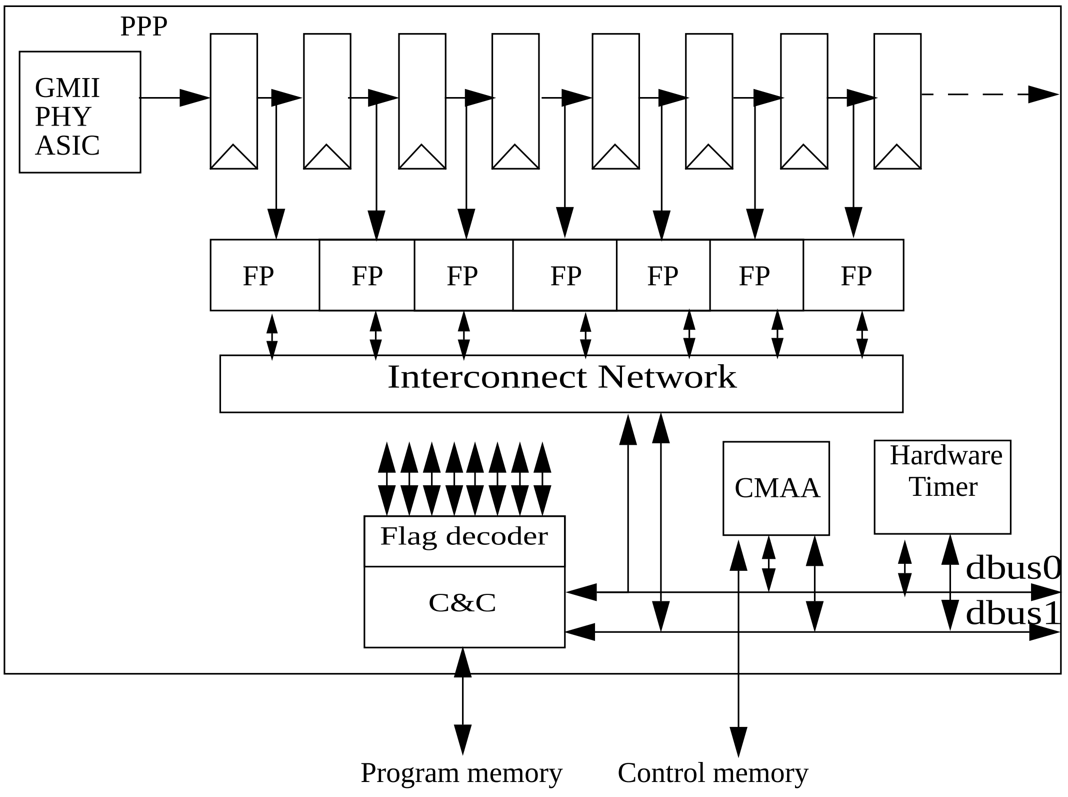

Figure 5.4: Overview of the PPP architecture.

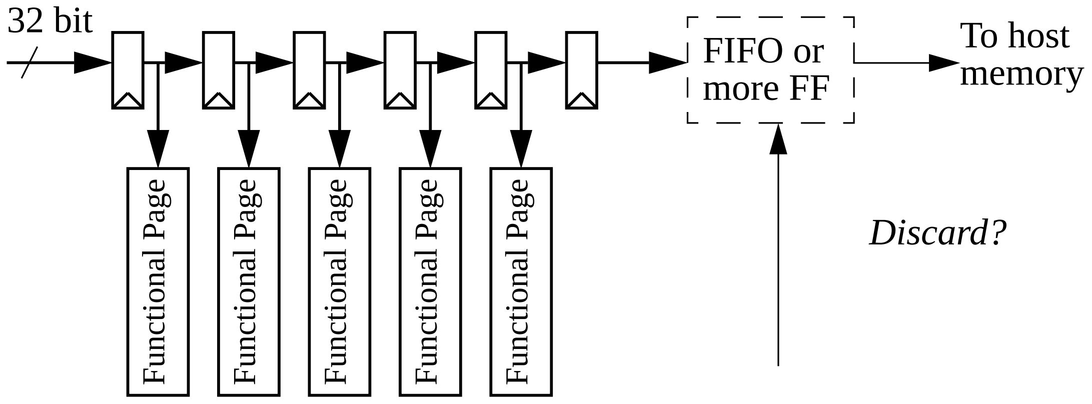

Figure 5.5: The input flip-flop chain. The chain of flip-flops enables access to the data stream with low fan-out. If the number of functional pages are moderate and the header processing delay high the chain might include a RAM based FIFO in the end to save power.

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 register as low as possible. If the number of functional pages (FP, see section 5.3.2) is large, we have to use a large number of registers in the chain, but if the number of FPs are 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 do not want to send the payload data to the host before we know if it should be discarded or not. But there is no reason not to use a low power RAM based FIFO in the end of the input buffer if the number of FPs is low and the decision delay is large. The number of FPs and the decision latency are set by the protocol coverage, at design time.

5.3.2 Functional pages

The functional pages are all dedicated hardware blocks with a 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 in the PPP. Their processing tasks are very diverse both by 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 (C&C discussed in section 5.5.1). The control normally consists of flags that starts or stops the processing in the functional page. The typical FP interface is illustrated by figure 5.6.

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. The configuration vectors are produced in the micro controller which also controls the configuration procedure. Using the protocol suit discussed earlier a small set of functional pages has been selected and implemented to implement the data intensive part of the protocols. They are

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

The CRC FP is very important for the overall performance of the PPP. This FP has been implemented and manufactured using a standard cell process. The CRC solution proposed in paper 2 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. Such a FP is described in [5.3] and it enables very high throughput. The CRC FP, CRC algorithms and design considerations are discussed in detail in paper 2.

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

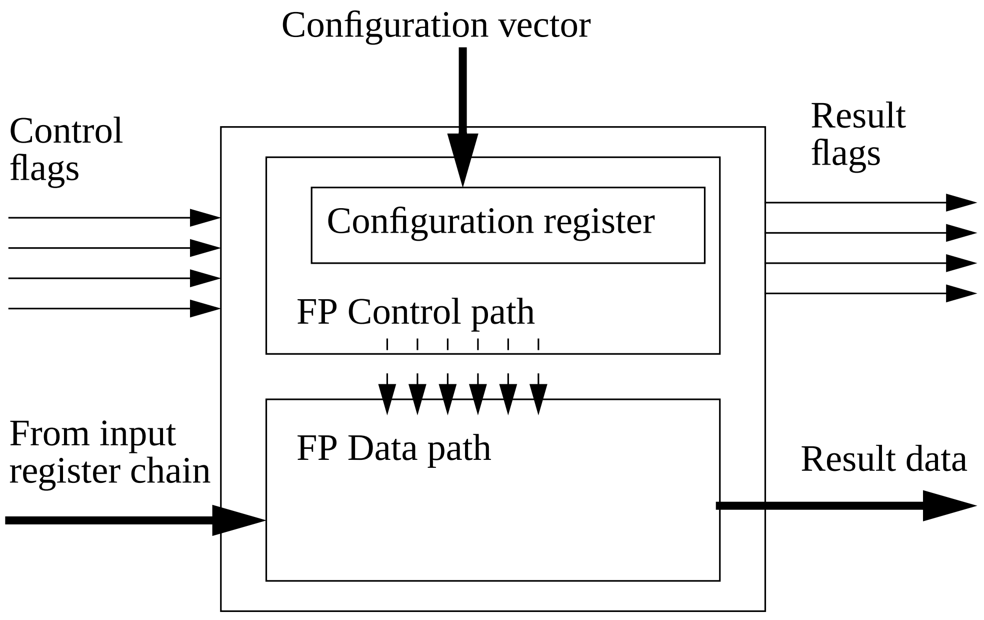

Figure 5.6: Funtional page interface.The FP are controlled by flags produced in the C&C. The primary output consists of result flags, e.g. discard flag. Some FP also produces result data.

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 a masked registers of 32 bits 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 values 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 5.7. The XAC FP has been implemented and verified in VHDL but the final layout remains to be implemented.

The MII parallelization FP is only included if the PP 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.

The Checksum FP essentially consists of a pair of 1-complement adders and is a simplified version of the FP described in [5.1]. An overview of the checksum calculating FP is illustrated by figure 5.8. According to the investigations done by my colleagues this FP can operate at 10 Gb/s which is unreachable using a general purpose processor. This FP has not yet been implemented by me.

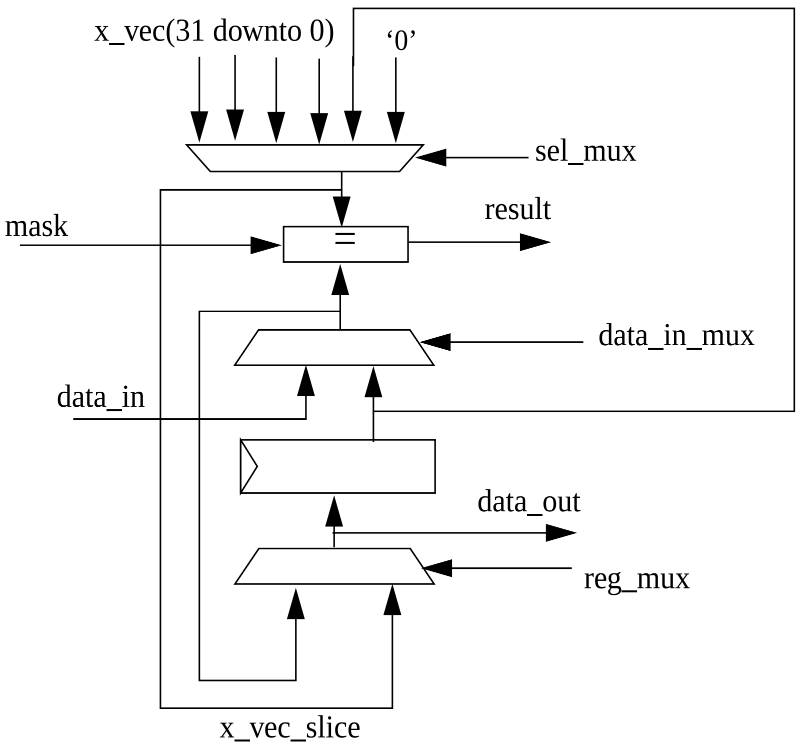

Figure 5.7: One out of four byte comparing slices in the XAC FP.

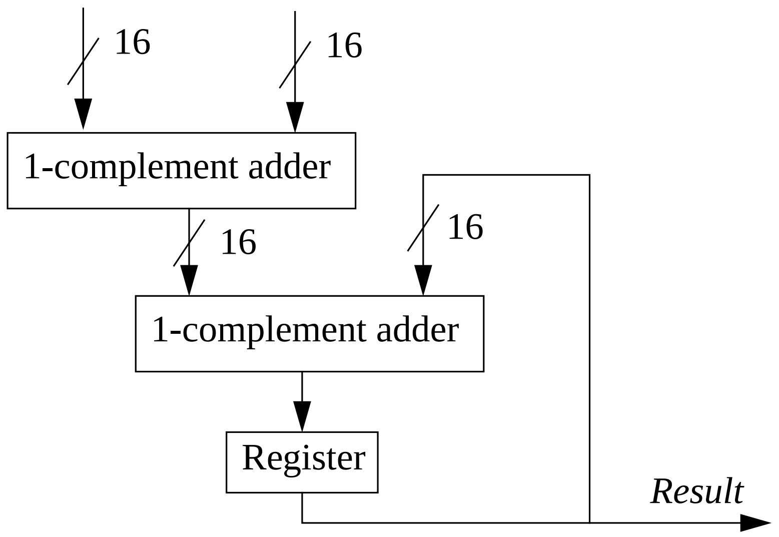

Figure 5.8: Checksum FP.

The Length counter FP is responsible for counting the lengths of a packet to find out when all fragments have been received. The length counter adder is closely controlled by the C&C which uses it to schedule its actions. The length counter FP consists of a adder, and 2 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. The FP also produces a flag when the content of the accumulator register is zero.

All of the FP can perform high throughput processing due to their relatively dedicated architecture. The slowest and most complex one is the CRC FP. It still manage 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.

In my research group a parallel project have found a different set of FP. The main difference between the two projects, is that the other does not handle fragmented packets and uses a more dedicated C&C architecture. The alternative set of functional pages are described in [5.2].

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.

5.3.3 CMAA

In paper 3 in chapter 8 an acceleration engine 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 performs reassembly of fragmented packets. The CMAA also accelerates the access to control variables stored in the control memory. The 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 depending 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 not be bad. Also with this small number of entries the CMAA would significantly relax and accelerate the overall PP 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.

5.3.4 Processing tasks allocation

The different processing tasks described in section 5.2, are allocated to different processing units within the PP according to the table below.

Table 3: Allocation of processing tasks listed in section 5.2.

Protocol	Task	Processing hardware
Ethernet	Calculate CRC	CRC FP
	Check Ethernet DA	XAC FP
	Check type field	XAC FP, C&C
	Demultiplexing of data	C&C together with CMAA
	Extract length field	XAC FP
	Length counting	C&C
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
	Extract IP SA	XAC FP
	Reassembly fragments	CMAA

Table 3: Allocation of processing tasks listed in section 5.2.

Protocol	Task	Processing hardware
	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
	Check sequence number and reorder data	Micro controller
	Extract acknowledgment and trigger reply	Micro controller
	Check and process options and flags	Micro controller
	Update connection state variables and timers	Micro controller, hardware timer
UDP	Extract ports	XAC FP
	Check connection	CMAA
	Extract and manage length	XAC FP, C&C
	Calculate header checksum	Checksum adder FP

5.4 Interfaces

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

5.4.1 Network interface

The interface between the network and the protocol processor consists of a PHY ASIC. Normally we consider it to be the Gigabit Media Independent Interface (GMII), but MII, XGMII or others could also be considered. The GMII 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 does not need to handle the processing of the physical layer protocols even if it would be possible to integrate such FPs.

5.4.2 Micro controller interface

The interface between the PPP and the micro controller consists of 2 data busses, the shared control memory and control signaling using flags. The micro controller also uses the two data buses when it configures the functional pages or the program memory of the C&C.

5.4.3 Host system interface

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

5.5 Control path

5.5.1 Counter and controller

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 the packets. The C&C is essentially a small RISC machine with a minimal internal datapath. It uses only an ALU. It also includes a register file, flag decoder, a program counter and a program flow controller. Further, a special conditional jump support must be included. The conditional branch support selects one out of four program counter values based on the flags from the FPs. This is used to select the correct program flow when the incoming packets protocol type has been checked. The C&C executes a set of programmed finite state machines. The FSM top level packet reception control is illustrated by section 5.9. The C&C produces start and stops flags for the FPs and simple instructions for the CMAA. To trigger the firing of the flags it uses the two counters. The C&C operates at a higher clock frequency compared to the rest of the PPP. The research on C&C is going on and its HW will be implemented after the licentiate defense.

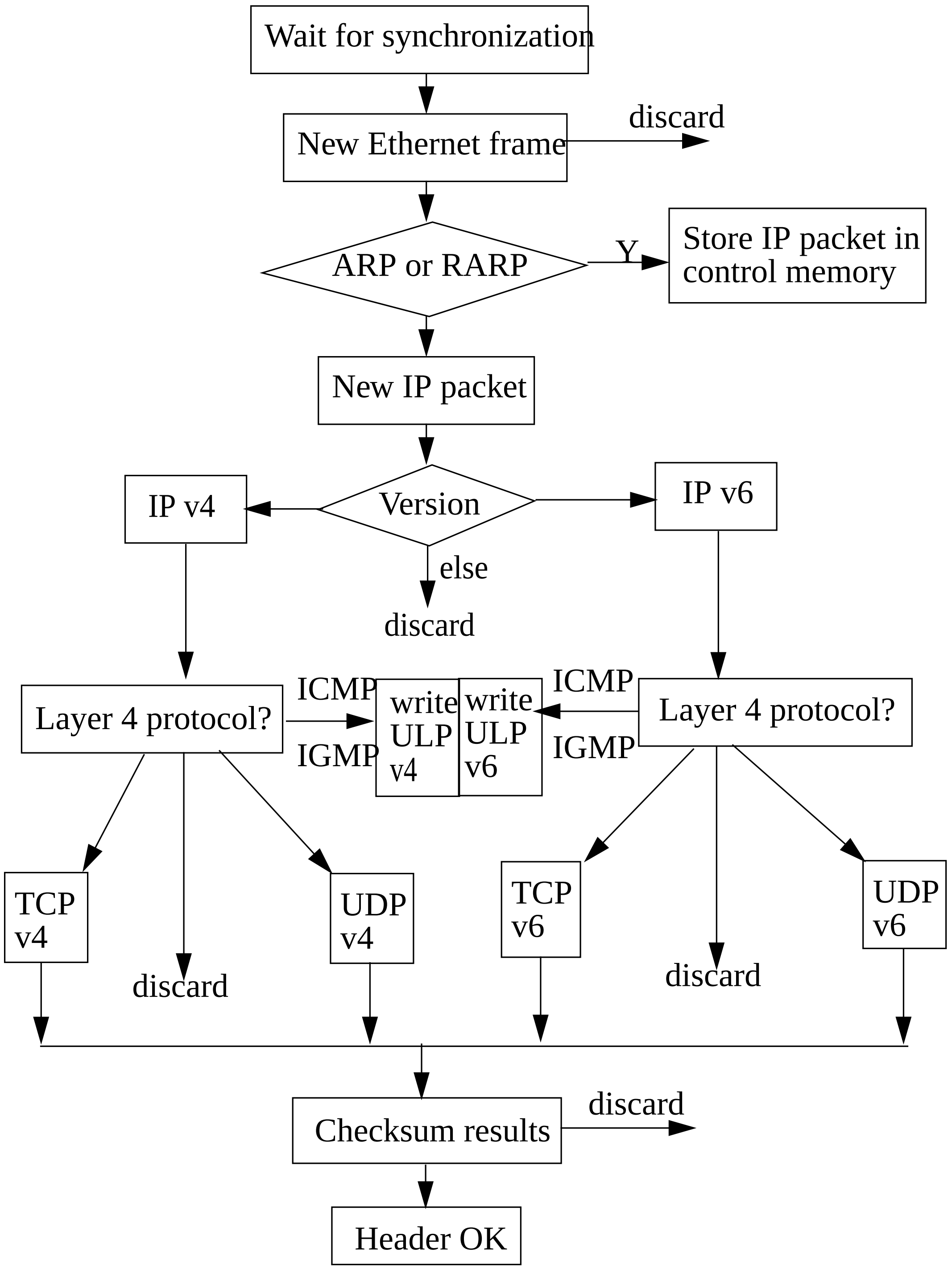

Figure 5.9: The control FSM controlling the PPP during packet reception will be implemented in the C&C in a programmable way.

5.5.2 Hardware timer

Managing and updating the timers can become a large part of the processing of the TCP and IP protocols. The number of timers is proportional to the number of network connections, so the problem is not as severe in a network terminal as it is in routers. Despite this, it is a task that must be considered to offload 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 order and some small control logic.

5.6 Configuration

The proposed 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.

- Data path configuration

Secondly the micro controller can configure the FPs using a relatively small number of clock cycles. This means that the data path of the PPP is configured. The program flow of the C&C can also be fully configured during this phase by rewriting the contents of the program memory.

- 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 gives the architectures a very high flexibility.

5.7 Performance

Using the performance parameters introduced in chapter 3 we can discussed the performance of the proposed architecture.

- Flexibility

The architecture is programmable with a configurable data path. It is capable of processing up to layer 4 packets and handles fragmented packets. The FP are selected to be as general as possible. That way they might be reused for other protocols. The micro controller provides all the flexibility needed.

- Throughput

Using dedicated hardware blocks enables a very high performance. My simulations indicates that the PPP functional pages and CMAA manage more than 4 Gb/s throughput, using a mature, not to say old standard cell process. The throughput of the processing in the micro controller and the interface remains to be investigated.

- Inter-operability

The general purpose micro controller can be programmed to interact with the host operating system, but that is not a part of this research project. However, it is clear that the flexibility, provided by the PPP and micro controller programmability, makes it possible to optimize the user interface. Further investigations on host operating systems optimization must be performed.

- Cost

CMAA cost depends on the number of entries that will be used. The power must be considered low since there is so little processing overhead. Further the dedicated datapath is only performing tasks they are dedicated for which increase the power efficiency. The SoC approach means reduced number of pins. The PPP will not use much area as long as the CMAA does not include to many entries. This means that the packaging cost is small. The flexibility of the design will also help to keep the cost low since it both increase the time on market and the application coverage.

References

[5.1] PERSSON, N.: ‘Specification and Implementation of a Functional Page for Internet Checksum Calculation’, *Master’s thesis*, Linköping University, March 2001, No.: LiTH-IFM-EX-959

[5.2] 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.

[5.3] 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

6

Paper 1: Configuration-Based Architecture for High Speed and General-Purpose Protocol Processing

Dake Liu, Ulf Nordqvist, and Christer Svensson

In the Proceedings of SIPS 1999

Abstract

A novel configuration based general-purpose protocol processor is proposed. It can perform much faster protocol processing compared to general-purpose processors. As it is configuration based, different protocols can be configured for different protocols and different applications. The configurability makes compatibility possible, it also processes protocols very fast on the fly. The proposed architecture can be used as a platform or an accelerator for network-based applications.

6.1 Background

Networking has been developing very fast and more and more protocols are emerging for different applications. Higher processing performances are requested by applications. Requirements could be recognized as:

- Multiple ports and multiple Gigabits per second real-time framing and de-framing.

- To pre-process as much protocol jobs as possible before a memory access.
- A general, simple, fast, and flexible architecture for different kinds of protocols.
- A built in protocol recognition and automatic configuration capability.
- Low power, high speed, and memory (size and access) efficient architecture.

Two kinds of protocol processors are available on the market nowadays, one is the specific single protocol-limited ASIC (we call it SPASIC in this paper), the other is the processor-based general-purpose CPU (we call it GPCPU in this paper). None of them can fit the requirements for future computer communications. The first one, SPASIC, is only used for one protocol or a few specific protocols included in the design. Obviously, it does not support future protocols. The second one, GPCPU, cannot work at very high speed because of the general architecture. As a redundant and speed-limited architecture, it is not the satisfactory solution for a relatively stable and control-extensive flow. From another point of view, the protocol processor must be compacted because it is often used as a pre-processor and as a small part in a certain kind of application. Therefore, the redundant architecture is not suitable for embedded or integrated solutions.

Most solutions available now use a specific circuit to process the protocol flow, and use a GPCPU for switching, routing, and other applications. Because of the limited SPASIC architecture, future flexibility is limited. For multiple applications, more SPASIC cores are integrated to cover more protocols and this makes the system redundant.

We need to recognize the protocol of the incoming package and then configure the processor to fit the protocol because the system might be used in a variable environment. Therefore, a new architecture is strongly requested, which is as fast as a SPASIC, as flexible as a GPCPU, and as simple as possible.

6.2 Functional coverage of DPSP

The system proposed is a new architecture for control-extensive processes, e.g. protocol processing. One example is to take the data package from AUI (Attachment Unit Interface of 10Mb Ethernet), or MII (Medium-Independent Interface of 100Mb Ethernet), or GMII (for G-bits Ethernet). Fast pre-process for different level of protocols is performed, for example, from Ethernet to IP and even up to TCP on the fly.

We can solve all problems mentioned above by introducing the *Deep pipeline serial processor DPSP*. It executes the protocol processing based on a booted and predefined configuration. Since the control is based on the configuration instead of software programs, DPSP can process protocols in real speed, e.g. Gbit Ethernet. After booting, the configuration HW can be shut down, which gives possibilities of low power. Following this way, the application, e.g. IP telephone or IP switching can be separated from the protocol framing and de-framing. The advantages are:

- Framing and de-framing are performed in a separate core; it acts as a platform or an accelerator and makes more application integration possible.
- Separated the DPSP as a stand-alone machine working at high speed with a standard implementation.
- All functional blocks inside the DPSP are self-contained and configured, therefore the adaptation to a long-term unpredictable future protocols is possible.
- The protocol can be recognized by this solution and a correct configuration can be booted to the DPSP after the recognition process. We define this feature as the self-learning and self-adaptation for any product used for different environments, e.g. home RF.

The architecture performs protocol processing based on both pre-configured setting and a real time control program. The pre-configured setting processes the protocol in every cycle inside each field of a data frame. The real-time control program only works on the higher level such as branch decisions, macro selections, and job hand over. Thus, the processing speed can be much higher because there is no program (which is slow in principle) involved in sub level processing. By planning the configuration, the architecture can supply as good flexibility as that supplied by a GPCPU.

6.3 Application overview

The goal is to make a platform for all possible network applications. Part of the possible applications and features supported by the platform can be listed:

- Fast framing, de-framing for the Internet switching: G-bits Ethernet source, and destination address extraction, fast IP DA and SA extraction etc.
- Predict the memory allocation: relax memory traffic, payload reordering, etc.
- Fast queue and priority check for the real time network applications.

- For certain applications the products recognize the protocol of the coming data, and boot the protocol configuration after learning.
- The user can boot different protocols for different applications.
- For fast prototyping or SoC integration.

6.4 Architecture

We introduce a new architecture that can work towards the physical limits of CMOS [3]. It can be implemented using conventional ASIC design flow, and can be configured by a program to suit different kinds of protocols and applications. The proposed architecture is divided into two parts. The first, which is the key part, namely Deep Pipeline Serial Processor (DPSP). Serial does not mean bit serial, it is a byte or a word based serial architecture. The second part is a normal micro-controller, the C. The C supports the DPSP configuration, the interface between DPSP and the application, and the real-time high-level job control. The DPSP can work much faster than the micro controller can.

The proposed architecture executes the protocol process based on both programs and pre-set configurations. The program only controls macro jobs, which are based on the frame rate instead of the byte rate. The pre-set configuration controls real time protocol processing at high speed with a relatively fixed control and working mode. Therefore, the program control induced speed limit is completely eliminated.

The proposed architecture is configured for a specific protocol before the protocol process. The configuration is performed by writing coefficients and control codes into control registers in a Functional Page, FP. All Functional Pages are scheduled in the order in which the protocol is processed in sequence.

For implementation convenience, data coming into every functional page is pipelined. Functional pages are connected one by one following the job schedule. Each FP manages its process in its own sub field. For example, the FP for CRC manages only the CRC check on the fly. Another example, the FP for header matching only matches the protocol header for its synchronization.

The System block diagram is given in figure 6.1. The left part is DPSP and the right part is the C for configuration, applications, and for supporting applications. Different protocols can be executed according to the configuration given by C. The C performs the service support. Which is divided into three parts. The first part is booting, including the boot of configurations for all FP and programs in the counter and controller. The sec-

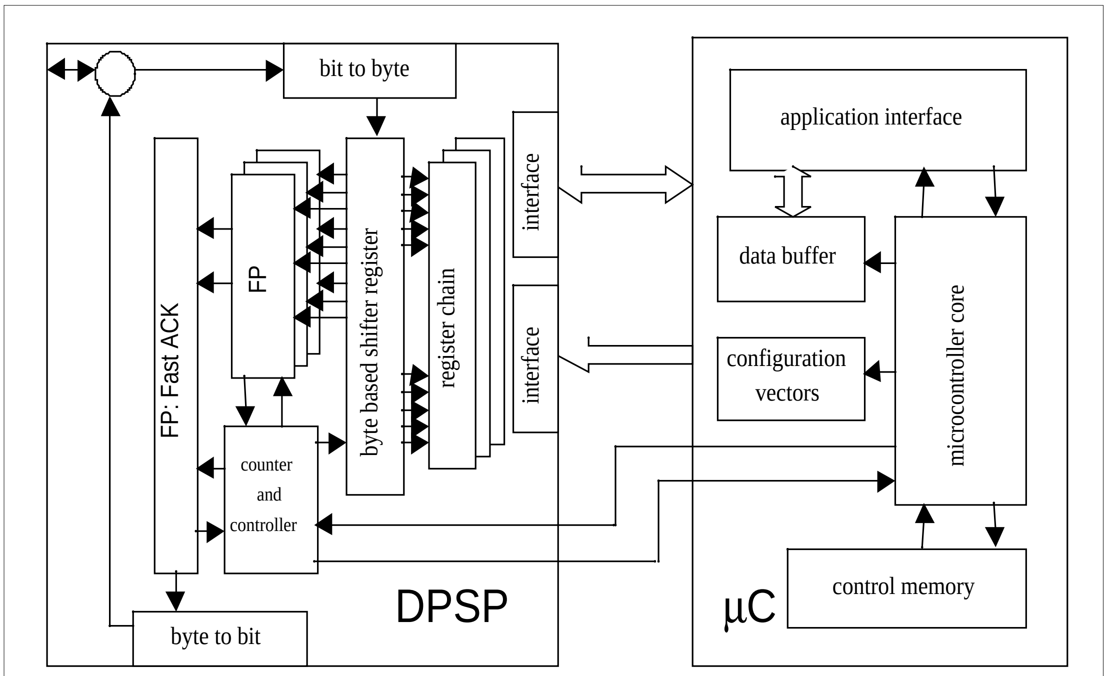

Figure 6.1: The system block diagram.

ond part is the DPSP monitoring, including checking the DPSP executing status, receiving and transmitting payload data, and sending interactive control. The third part is to coordinate DPSP with the application hardware. The configuration is performed during the power-on boot. When the protocol of the incoming data is unknown, the booting is performed for the protocol recognition first, and secondly, the normal protocol specific configuration according to the result of the recognition is booted. The DPSP top level architecture is given in Fig. 2. Following functions will be allocated as FP's in the DPSP given in the above figure:

Matching: It sets up the synchronization by recognizing the preamble.

Error checking: Check errors according to the coding of the protocol.

The field extraction: It extracts fields and accelerates processes further.

Level hierarchy transparent process: The HW can make levels of network hierarchies transparent. The upper level payload can be extracted.

Payload management: To measure the length of the payload and to validate the correctness of the data. Then allocate the data into a suitable position.

Other QoS options: According to the applications, QoS can be supported.

Application interfacing: Before data allocation, check the application, find the possibility to send data to the application on the fly.

Fast acknowledgement: The acknowledgement can be compiled in an easy and fast way according to extracted fields.

Fast ACK as an important function is performed on the fly in DPSP. Necessary messages such as DA and SA are kept for building the fast ACK. The FP for ACK is allocated between the shift-in and shift-out. The fast ACK packet can get TCP ACK, IP address and LAN address, e.g. Ethernet address from the buffer.

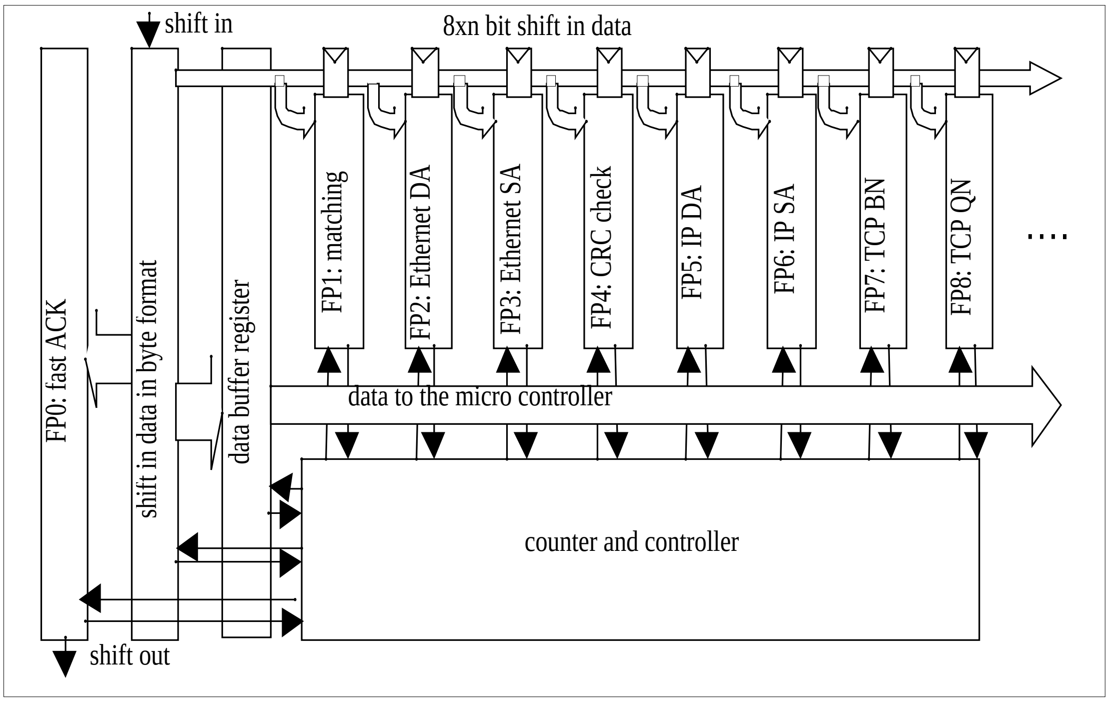

Figure 6.2: Dataflow of the DPSP.

The data flow is given in figure 6.2. The data coming from the physical level has been converted to byte level format and data rate is one eighth of the bit rate. Control signals (single pins) are handover start-finish strobes from the counter and controller. Control signals coming to the counter and controller gives timing status. Shift in and out are 8 bits input-output data of DPSP. Other width of data busses can be configured.

6.4.1 Functional Pages

Simple FP implementation can be done by custom design. Complicated FP will be implemented using synthesis. The flags are outputs from the sythesised logic using the configuration, the incoming data, and the control conditions as inputs. As an example, the matching unit uses configuration registers to save the header pattern. When the shifted input data matches

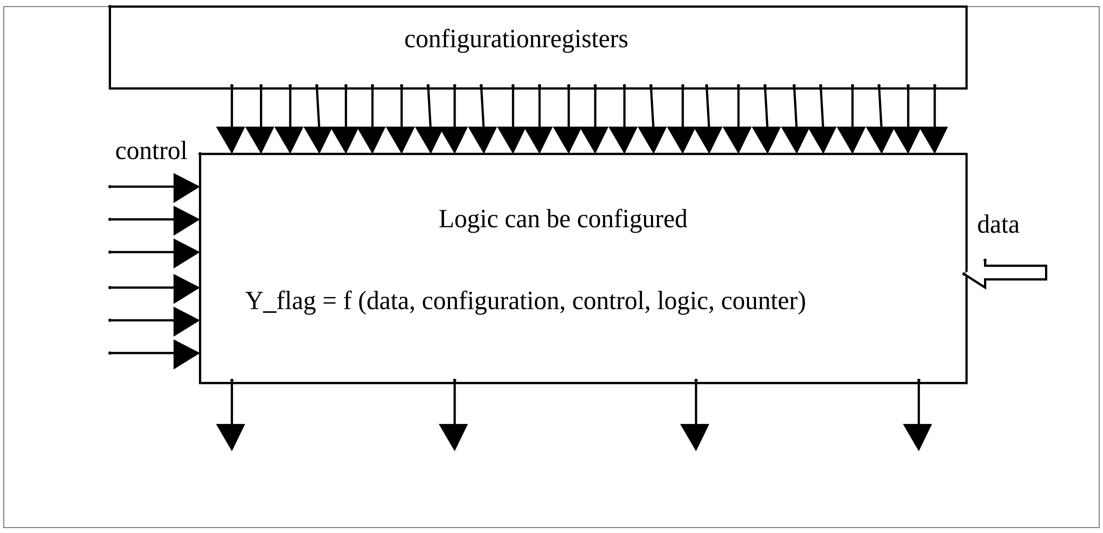

Figure 6.3: FP structure

the pattern at a certain point, a matching flag is given as Y_match = & (data, configuration_register).

The active period of a FP is decided by its function. Most FP's are only active part of the time. Some FP's are active all the time during a frame process, e.g. the CRC check.

6.4.2 Counter and Controller

The counter and controller is a counter based state machine (FSM) adapted by configurations. A complete configuration set will be written into a register file. Each one or few lines in the register file are configured for the control of a FP.

There are two levels of controls performed in the ”counter and controller”. The upper level control is specified as the handover process. The lower level control supports only the counting status. The upper level control is a kind of interactive control. The lower level control is not interactive because the FP uses the status as a control reference without giving feedback. The deep pipeline is scheduled inside each FP. The control of the deep pipeline is given by the lower level control from the ”counter and controller”. Status of the state machine is configured according to the recognized protocol. A group of control vectors for a specific FP is selected (addressed) by the counter. Therefore, the control procedure is scheduled following the configuration. The deep pipeline data path performs the protocol jobs in N+ cycles. Here N is the number of bytes (or words, according to which protocol is used) and is the number of cycles used for hand over one job from one FP to another FP.

The control is scheduled in the following way:

- Start a FP
- Let the FP run itself
- Monitoring flags coming from all active FP’s.
- Make new control decision according to flags.
- Monitor the control interface between the micro controller and the DPSP.
- Change the control procedure if the micro controller gives a new request.
- Inform the micro controller to that the data is available.
- Responde to the micro controller to accept data.
- Send the accepted data to a FP responsible for the acknowlegement.

6.5 Conclusion

We have described a configuration based DPSP architecture as a platform for network applications. The architecture implements the infrastructure of an accelarator which gives the necessary framing and de-framing, and a fast acknowledgement. Most protocol processes can be supported by DPSP architecture because of the flexible configuration. The configuration-based architecture can also support protocol recognition based on predefined protocol preambles. As the DPSP is a specific architecture for protocol processes, it can accelarate protocol processing on the fly for high speed applications.

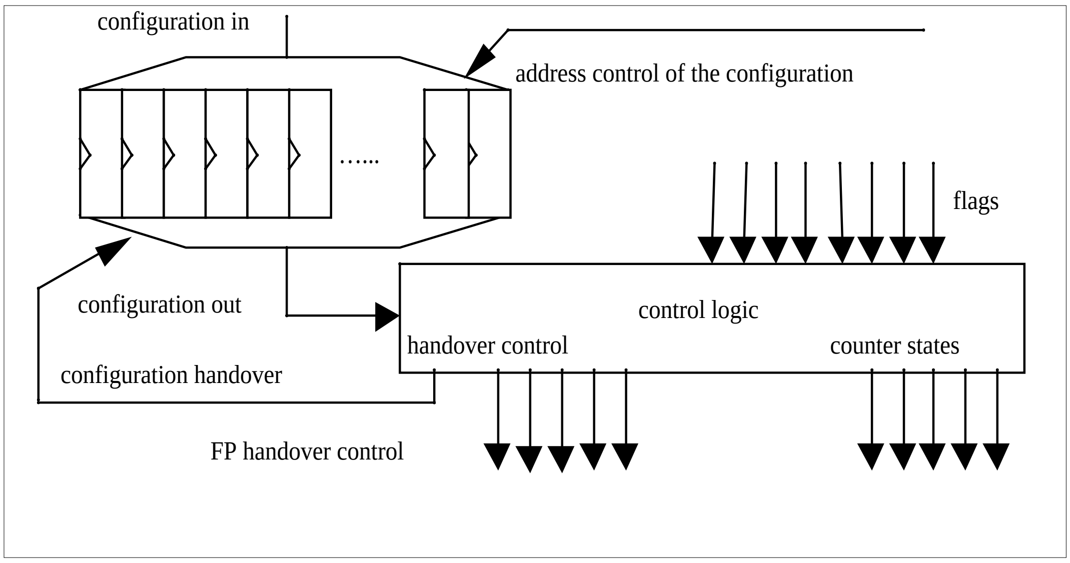

Figure 6.4: Counter and Controller

6.6 Acknowledgments

Authors would thank to the useful discussions with Dr. Kenny Ranerup Switchcore, Sweden. The research is supported by the Center for Industrial Technology at Linköping University (CENIIT) and Exellence Center in Computer Science and Systems Engineering in Linköping (ECSEL).

REFERENCES

[6.1] Andrew S. Tanenbaum, Computer Networks, 3rd Edition, Printice Hall PRT, ISBN 0-13-349945-6, 1996.

[6.2] Jayant Kadambi et al, Gigabit Ethernet, Printice Hall PRT, ISBN 0-13-913286-4, 1998.

[6.3] Anders Edman, and Björn Rudberg, SDH 10Gb/s regenerator frame in 0.6m CMOS, 97 IEEE ISSCC, pp 156-157, 1997.

[6.4] Axel Jantsch, Johnny Öberg, and Almed Hemani, Is there a Niche for a general protocol processor? Proceedings of the 16th IEEE NORCHIP conference, pp 214-221, Lund, Sweden, Nov. 98.

7

Paper 2:
CRC Generation for Protocol Processing

Ulf Nordqvist, Tomas Henrikson and Dake Liu

In the Proceedings of NORCHIP 2000

Abstract

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 methods has on the other hand been constantly changing.

A comparative study of different implementation strategies for computation of Cyclic Redundancy Checks has been done in this paper. 10 different implementation strategies was examined. A novel architecture suitable for use as an IP in an protocol processor is presented. As conclusion, different implementation techniques have been divided into application areas according to their speed, flexibility and power-consumption.

7.1 Introduction

Both computer and human communication networks, uses protocols with ever increasing demands on speed, cost and flexibility. In the market segment of hardware for network nodes such as routers, switches and bridges, the performance needs can be fulfilled by using Application Specific Integrated Circuits (ASIC) or Application Specific Standard Products (ASSP). This will probably be the case also in the future due to there relatively cost-insensitive costumers. In order to let the end-user take advantage of the bandwidth enhancement in today networks, tomorrows Network Terminal (NT) hardware must support transmission speeds of Gbit/s [7.10. Hardware for such NT components is on the other hand sold on a cost-sensitive market share with high demands on flexibility and usability.

Traditionally NT has been implemented as ASIC:s for the lower layers in the OSI-Reference Model [7.17 with an CPU-RISC based SW implementation of the upper layers [7.8, or completely implemented in software [7.1, [7.3, [7.17. In [7.6, [7.7 we presented a new architecture for configurable protocol processing that supports programmability on the upper layers and gives both configurability and high performance on the lower layers. This kind of solution is also supported by [7.18, [7.19 and [7.14. This architecture specifies that, the without any doubt most computational extensive task, Cyclic Redundancy Check (CRC) [7.3, [7.20, should be implemented as configurable hardware, supporting buffering free processing.

The speed requirement is very important since a protocol processor must buffer incoming data if jobs are not completed at wire-speed. This leads to high costs in terms of power consumption, area and manufacturing costs due to the usage of buffers.

The aim of this paper is to compare different implementations of CRC computational units in order to specify a suitable one for protocol processors.

7.1.1 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
- 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.

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

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) is computed according to equation 2.

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

S(x) is by other words the reminder resulting from a division of the data stream and a generator polynomial g(x).

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 1.

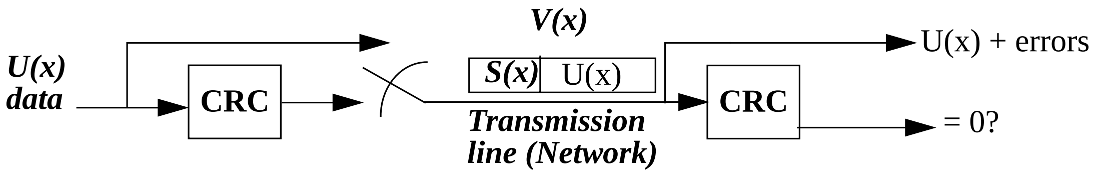

Figure 7.1: Principle of error detection using the CRC algorithm.

As can be seen in figure 1 the receiving NT perform a CRC-check on the incoming message and if the result 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.

7.2 Implementation theory

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

- **Software(SW) Solution** [7.3**,** [7.1: The CRC algorithm can always be implemented as an 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.

- **Traditional Hardware Solution**: Linear Shift Register (LSR) with serial data feed [7.20 has been used since the sixties to implement the CRC algorithm, see figure 2. As all hardware implementations, this method simply perform a division and then the reminder 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 network nodes. Since fixed logic is used there is no possibility of reconfigure the architecture and change the generator polynomial using this implementation.

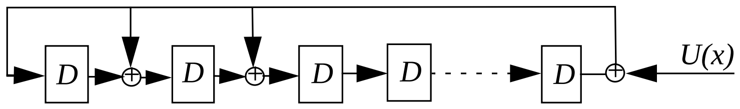

Figure 7.2: Linear Shift Serial Data Feed

- Parallel Solution: In order to improve the computational speed in CRC generating hardware, parallelism has been introduced [7.2, [7.4, [7.5, [7.9, [7.11, [7.12. 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 3. If the CRC polynomial is changed or a new protocol is added, new changed hardware must be installed in the network terminal. The lack of flexibility makes this architecture non suitable for use in a protocol processor.

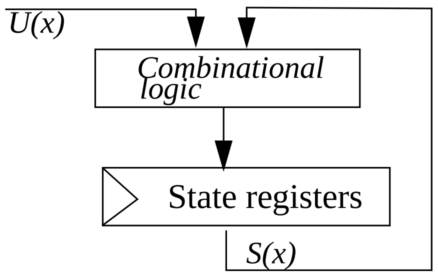

Figure 7.3: Parallel Fixed Logic Implementation

Configurable Hardware: One way of implementing configurable hardware is by using Look-Up-Tables (LUT) as proposed by [7.3, [7.12 and [7.2. The architecture is illustrated by figure 4.

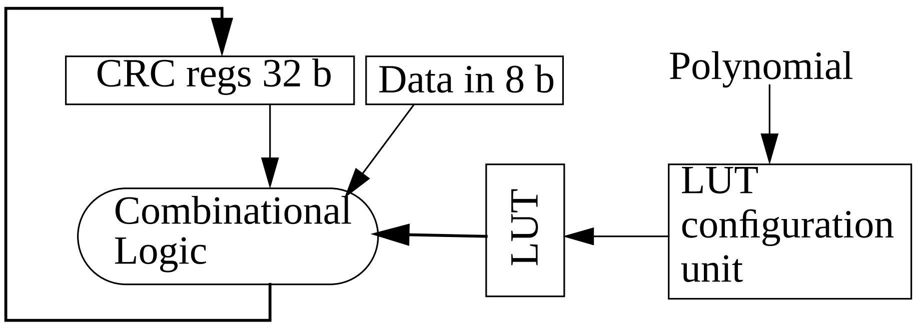

Figure 7.4: Look Up Table based configurable hardware.

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 Network will be increased so the effect will be cancelled. The optimal solution has not been derived.

Another, novel implementation method is the ***Radix-16 Configurable CRC Unit***, which is presented for the first time in this paper. By noticing that any polynomial of a fixed length can be represented by implementing the CRC using a LSR with switches on the reconnecting wires as illustrated by figure 5, a configurable hardware can be implemented using NAND-gates to represent the switches.

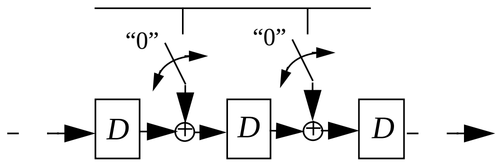

Figure 7.5: Configuration by use of switches in the circuit reconnecting wire.

In order to improve the speed of the Radix-16 Configurable CRC, a 4 - bit wide input data stream is used as can be seen in figure 6. The resulting bit in each position *k* in the CRC register then depends on the value of the *k-4* CRC bit, the last four CRC bits, the polynomial bit description and the input bits. The logic, which consists mainly of XOR and NAND-functions provides the necessarily configurability.

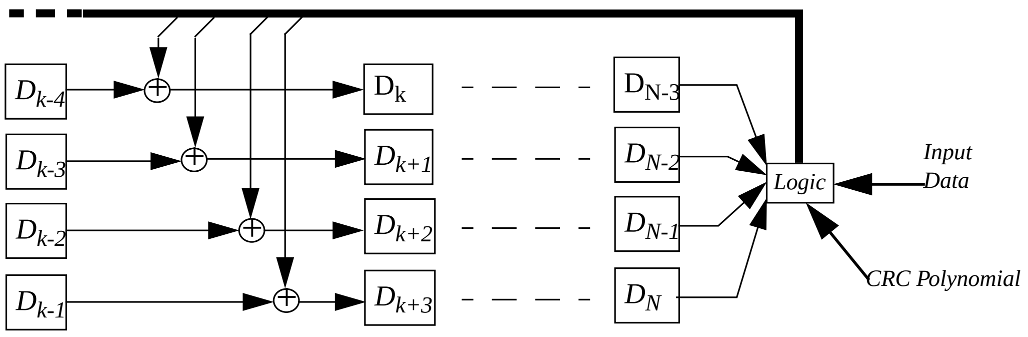

Figure 7.6: Radix-16 Configurable CRC engine

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

7.3 Experimental results

10 different implementations 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 & Route was done using Ensemble P&R from Cadence. The technology used is AMS 0.35 µm.

Since most network protocols are bytebased, there is no meaning in investigating a parallelism of more than 8 even if the other parts of a protocol processor might run on other clock frequencies using for example a 32 bit wide input stream.

As seen in table 1 the fixed logic and parallel input implementation is the fastest. That is in the order of what have been reported in earlier work. We can also see that the LUT based method gives about twice the speed as 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 [7.15, [7.16, 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 consider-able 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 [7.13 thanks to scaling.

Conflict with other processes makes 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 [7.6 and [7.7.

Table 4: Comparison between different CRC implementations. The 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

7.4 Conclusions

Because of the superior performance of a parallel ASIC implementation, it will be used for implementation of network-node-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 supporting new protocols.

Software solutions for low speed protocols will also be used for low-speed applications, but a increasing area of applications demands high-speed configurable protocol processing, including CRC generation. An hardware architecture that can fulfill this specifications is the Look-Up table -based structure proposed in [7.1 and implemented in this paper.

A novel architecture for this application area has also been presented, which has a superior power-delay product. The architecture implemented can be configured for CRC encoding/decoding using any 16, 24 or 32 bit polynomial. Power consumption will be kept low using shut-down logic. The architecture support the speed requirements of today protocol processing in NT:s. For upcoming protocols used in NT network processing, scaling will provide necessarily speed-enhancements.

REFERENCES

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

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

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

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

[7.5] 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

[7.6] 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

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

[7.8] 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

[7.9] 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

[7.10]J. Kadambi et al, "Gigabit Ethernet", *Prentice Hall PRT, ISBN 0-13-913286-4*, 1998

[7.11]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

[7.12]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

[7.13]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

[7.14]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
[7.15]“Technical Specification of BRAN and Hiperlan-2. Common part.”, *ETSI TS 101 493 - 1*, *V1.1.1*, 2000
[7.16]“Technical Specification of BRAN and Hiperlan-2. Ethernet Service Specific Convergence Sublayer.”, *ETSI TS 101 493 - 2*, *V1.1.1*, 2000
[7.17]A. S. Tannenbaum, “Computer Networks”, *3nd Edition, Prentice Hall PRT,* ISBN 0-13-349945-6, 1996
[7.18]“Building Next Generation Network Processors”, *White paper, Sep 1999, Agere Inc.*, http://www.agere.com/support/non-nda/docs/Building.pdf
[7.19]D. Husak, “Network Processors: A Definition and Comparison”, *White paper, C-PORT*, http://www.cportcorp.com/solutions/docs/netprocessor_wp5-00.pdf
[7.20]W. W. Peterson and D. T. Brown “Cyclic Codes for Error Detection”, *Proc. IRE*, Jan 1961, pp. 228-235

8

Paper 3: Packet Classification and Termination in a Protocol Processor

Ulf Nordqvist and Dake Liu

Submitted to the HPCA Network Processor Workshop

Abstract

This paper introduces a novel architecture for acceleration of control memory access in a protocol processor dedicated for packet reception in network terminals. The architecture enables the protocol processor to perform high performance reassembly and also offloads other parts of the control flow processing. The architecture includes packet classification engines and concepts used in modern high-speed routers. The protocol processor combined with a general purpose micro controller, fully offload up to layer 4 processing in multi gigabit networks when implemented in mature standard cell processes.

8.1 Introduction

Both computer and human communication networks use protocols with ever increasing demands on speed, cost, and flexibility. There is also a strong development towards an increased use of network protocols for applications that traditionally used other implementation techniques, e.g. voice and video. One reason is that packet based network protocols can normally handle a mixture of any kind of traffic. For network node components such as routers, switches and bridges, the performance needs have been fulfilled using Application Specific Integrated Circuits (ASIC) or Application Specific Standard Products (ASSP) since these applications traditionally have had quite moderate demands on programmability. These traditional approaches will probably continue to co-exist with more programmable solutions such as network processors (NP) in the future, due to their relatively cost-insensitive and performance demanding consumers. Having said this, it is clear that the networking industry is requesting moore programmable devices in tomorrows network.

In order to let the end-users take advantage of the bandwidth enhancement in todays networks, tomorrows Network Terminal (NT) hardware must support transmission speeds of Gbit/s. Hardware for such NT components is on the other hand sold on a cost-sensitive market share with high demands on flexibility and usability. Traditionally NT has been implemented using ASIC:s situated on the network interface card processing the lower layers in the OSI-Reference Model [12] and a CPU-RISC based SW implementation of the upper layers. Usage of standard, general purpose CPU:s, is expensive in terms of cost, space and power due to their lack of dedicated hardware. There is also an upper capacity limit, set by the I/O capacity and the instruction rate of the CPU. Today it is easy to find Network Interface Card (NIC) supporting multi-gigabit networks but such bandwidth can not be utilized by the host since it requires the host to be fully loaded processing layer 3 and 4 protocols, leaving nothing for the application and system processing. The research focus has mainly been on router and switching applications so far, but in the future the terminals will also require offloading using programmable high-speed solutions.

To meet these new requirement a new area of communication handling hardware platforms has emerged. These are commonly denoted as TCP Offload Engines (TOE). One of these TOE solutions is called programmable protocol processor (PPP) and it was introduced by this papers authors in [9] and [10] 1999. As most of the TOE it consist of programmable parts that can accelerate and offload a terminal host processor by handling the communication protocol processing. The protocol processor platform is a domain specific processor solution with superior performance over a general purpose CPU, that still provides flexibility through programmability within the limited application domain. The PPP architecture is intended for integration on a ASIC chip, it is not a board-level integrated programmable NIC. The protocol processor hardware platform is further discussed in chapter 2. In chapter 3 a novel methodology and architecture for handling and distributing, control flow information to and from our protocol processor is introduced.

8.2 Programmable protocol processor

The main task of the protocol processor is to process the packets transferred between the application on the host processor and the network, so that a secure and reliable connection is provided between the sender and transmitting function. The protocol processing architecture (and the research project behind) discussed in this paper, only deals with the reception of packets. Since the transmitting of packets is limited by the applications construction of packets and have lower demands on low latency, we have chosen to concentrate our research on the packet reception problem before discussing packet creation acceleration. The goal of this research project is to process as much of the protocol stack as possible before storing the data payload to the systems main memory. By reducing the memory access and buffering, illustrated by figure 8.1 , both memory bottlekneck problems and power consumption can be reduced. In order to achieve this, the protocols must be processed at network speed and multiple layers are being processed simultaneously as proposed in [18].

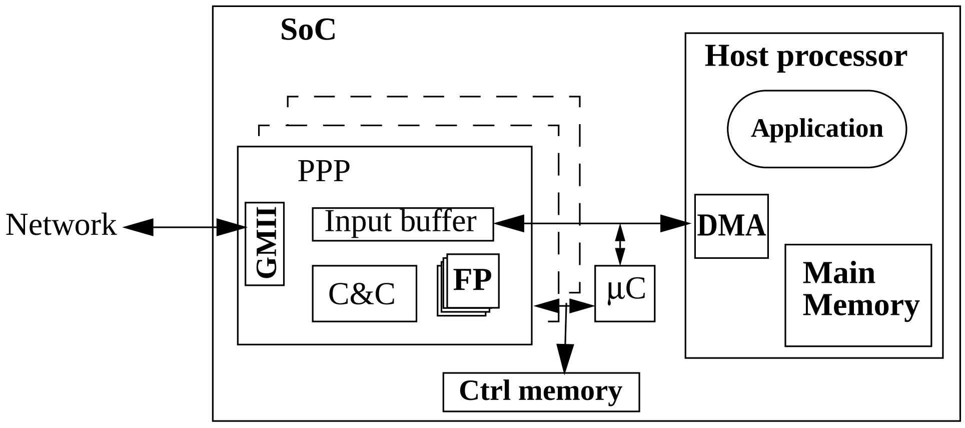

Figure 8.2: The PPP together with a general purpose micro controller handles the communication of one network port. In a system on chip (SoC) many PPP can be used as port-processors in order to provide high bandwidth between the application and the network. A control memory is used for storage of inter packet control variables.

In order to deal with the fact that the nature of the different processing task in a protocol processor is very versatile the hardware platform has been divided into two parts. This is illustrated by figure 8.2 . The first part is the Programmable Protocol Processor (PPP) which is dedicated for data intensive processing task mainly originating from the lower level protocols in the OSI-protocol stack standard. Examples on such tasks are checksum

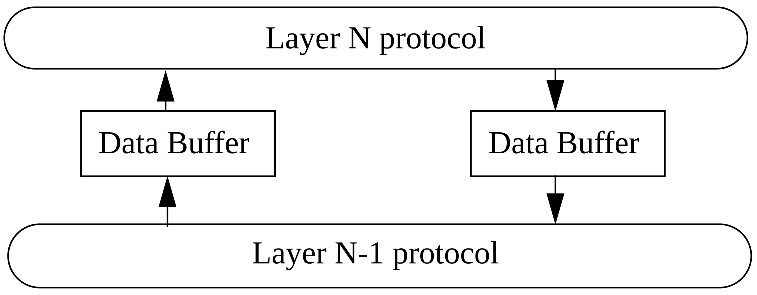

Figure 8.1: Using inter-layer processing the power consumption and required memory usage in the protocol processor can be reduced since all buffering of data between different protocol layers can be eliminated.

calculations, address checks, length counters etc. Normally they are intra-packet processing tasks that have to be performed even if the protocols covered are very simple. The other part off the platform is a general purpose micro controller (µC) that deals with control intensive protocol processing tasks such as connection state handling and other inter-packet processing tasks. The micro controller is also used for the configuration of the PPP for different type of protocols as well as firewall updates. Further the micro controller handles the control communication with the host processor and the DMA, e.g. setting up and closing sockets etc. Using DMA communication between the PPP and the host reduces the interrupts compared to bus-communication [17]. In the NP research community there is today a clear trend towards a separation of the processing in a slow and fast-path similar to our approach. In figure 8.3 there is an illustration showing how different layers in the protocol stack are distributed to different processing resources.

The micro controller is very suitable for implementation of the various finite state machines (FSM) which contributes to a big part of the control processing. Never the less, there are other tasks within the inter-packet processing domain, which the micro controller efficiently can be offloaded from. One of the main operations is a search and access of control data based on header information in a receiving packet. This operation is comparable to the bind and in_pcblookup C-functions used in software implementations. In a receiving situation the PPP will process the packet and then discard it or hand it over to the micro controller.

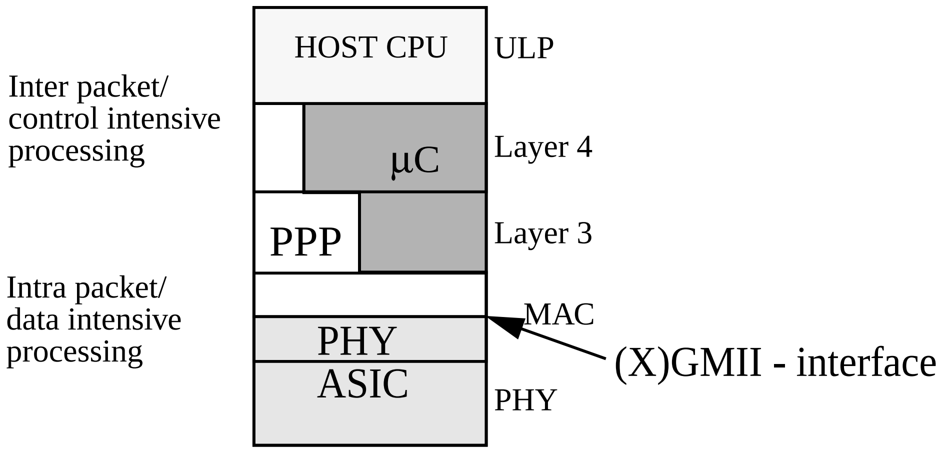
HOST CPU
ULP
Inter packet/ control intensive processing
μC
Layer 4
PPP
Layer 3
Intra packet/ data intensive processing
MAC
PHY ASIC
(X)GMII - interface
PHY

Figure 8.3: Offloading the host using various types of accelerators for different types of processing tasks and protocols. Typically higher layer protocols require more flexibility through programmability.

The proposed platform, including a PPP together with the μC, is essentially a TCP offloading engine (TOE) dedicated for network terminals. A TOE for NT does not make any routing decisions. It only discards packets or accept them before they are passed on to the correct host memory buffers. Further the number of connections is much less than in a layer 4 router. Hence the architectural design of such an offloading device have other goals and requirements. Consequently the research on such devices must divert from the network processor research area.

As illustrated by figure 8.4 the PPP hardware architecture for protocol processing consists of four main parts. One is the input buffer chain that provides the data to the accelerating functional pages (FP). By using a 32 bit wide chain of flip-flops, the fan-out from the flip-flops can be kept on a tolerable level even if the number of FP increases with new protocols and an increased protocol coverage. Using a RAM based FIFO buffer instead of flip-flops would decrease the activity but the fan-out would be a huge problem. As long as the fan-out is kept low it is still possible to replace the last flip-flops in the chain with a minimal RAM-based FIFO. The total buffer size is dependent on the decision latency of the PPP. The decision answer is normally discard or send packet to memory for further processing. Some payloads should be sent to the host memory and some to the control memory.

The control of the various accelerators (FP) in the PPP, mainly consists of start and stop flags. These flags are provided from the Counter and Controller (C&C). The flags are generated based on an internal program in the C&C, result flags from the FPs and counter values generated in the C&C.

The C&C is responsible for sceduling the start of the processing in the FPs at the correct clock-cycle, as the data streams through the register chain. Based on the result from the FPs, the C&C can either discard the packet or continue the processing by configuring and starting new FPs. If a packet is discarded, e.g. because the destination address was errornous, all the FPs are immediately shut down in order to save power.

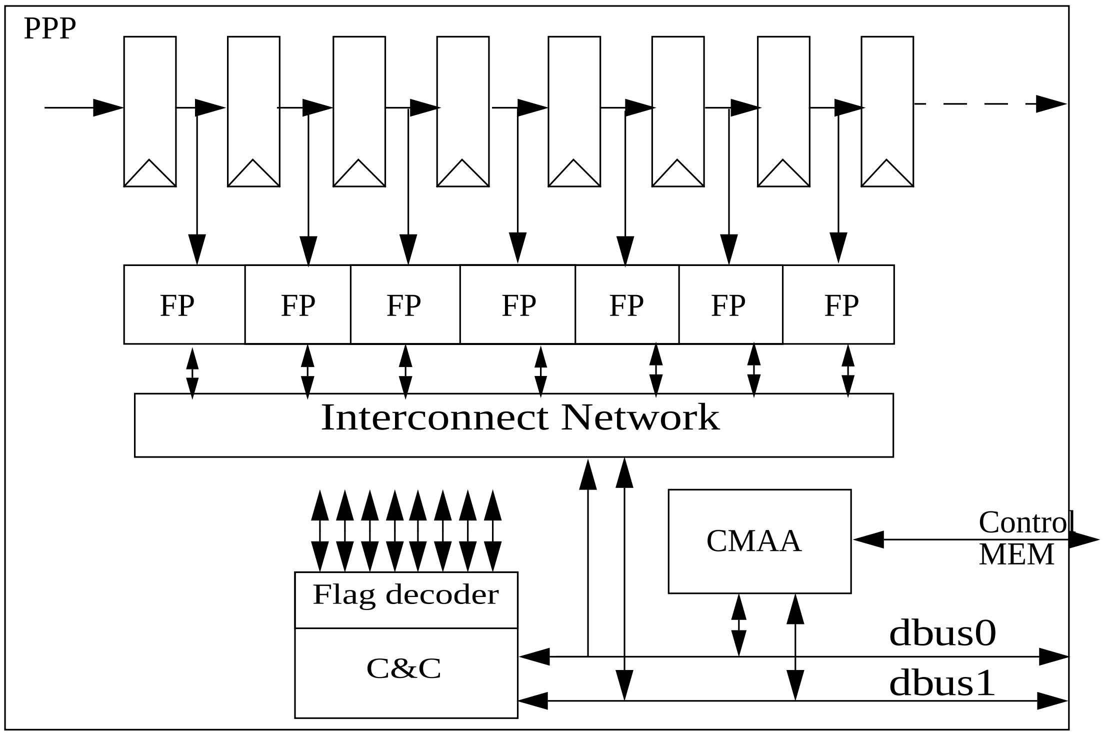

Figure 8.4: The programmable packet processor consists of 4 parts: The Counter and Controller (C&C), the input buffer chain, accelerating functional pages and a Control Memory Access Accelerator (CMAA).

Since the processor operates on streaming data, instead of stored data in memory, decisions on which program flow to execute requires minimal latency. Different protocol configurations uses different program flows. Hence program flow selection is dependent on the type and content of the receiving packet. The C&C includes a special instruction buffer for acceleration of multi-choice conditional jump instructions in order to provide maximum system performance. The payload of received packets of TCP or UDP type will be sent to the host while the payload of control oriented protocols such as ARP, RARP, ICMP, IGMP will be stored in the control memory. The control memory acceleration part is further discussed in chapter 3.

8.2.1 Functional pages

The FPs must operate at wire speed. FPs are configured from the micro controller during set up for a specific set of protocols or a single protocol. Each of the FP are dedicated ASIC with a limited configurability. Together, the micro controller and the C&C supports a high degree of programmability. To better understand the nature of the FP a common set of network protocols has been used. The protocols are TCP, UDP, ICMP, IGMP, IPv4, IPv6, ARP, RARP and Ethernet (Fast E and GigE). In order to support processing of these protocols, the following FPs, have been implemented:

- 1 CRC FP described in [15]
- 2 eXtract And Compare (XAC) FP responsible for checking address numbers and port numbers against the actual host address. Further they are used to extract and compare checksums.
- 2 length counting adders.
- 2 checksum calculation adders
- 1 generic adder

Other possible processing tasks suitable for acceleration in a FP is various types of decoding and decryption algorithms. They are however not used since such algorithms is not included in the selected protocol suite.

As mentioned earlier, the FPs are self-contained dedicated ASICs. After configuration the control needed for their operation is very limited. Actually, most of the control signaling can be reduced to only start and stop flags since most control is distributed to the individual FPs.

8.3 Control Memory Access Accelerator

As mentioned earlier the micro controller is responsible for the communication control or 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. Further, acceleration of slow path processing off-loads the micro processor. Hence, a platform including accelerating hardware assist and control interface dedicated for packet recognition and control memory access have been developed. The Control Memory Access Accelerator (CMAA) presented in this article uses 2 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 [16].

8.3.1 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 results is provided to one of the checksum calculating adders which adds it to the recent packets checksum using a 1-complement addition which produces a new checksum. If the length is equal to the total length which means that the hole 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 adress 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 addresses 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 high-lighted in bulleted text. The information stored in the control memory can also be used to calculate the host memory adress. An algoritm for this type of memory address calculation remains to be implemented for the general case even if it is simple for special applica-tions, e.g. VoIP. A general algoritm for in-order data-buffering in the host memory would significantly reduce the host processor interrupts. This type of algoritm would benefit from an accelerated access to the control mem-ory. This issue will not be further discussed in this paper.

8.3.2 Accelerator interface

The CMAA interface to the rest of the PPP and the micro controller is illustrated by figure 8.5 .

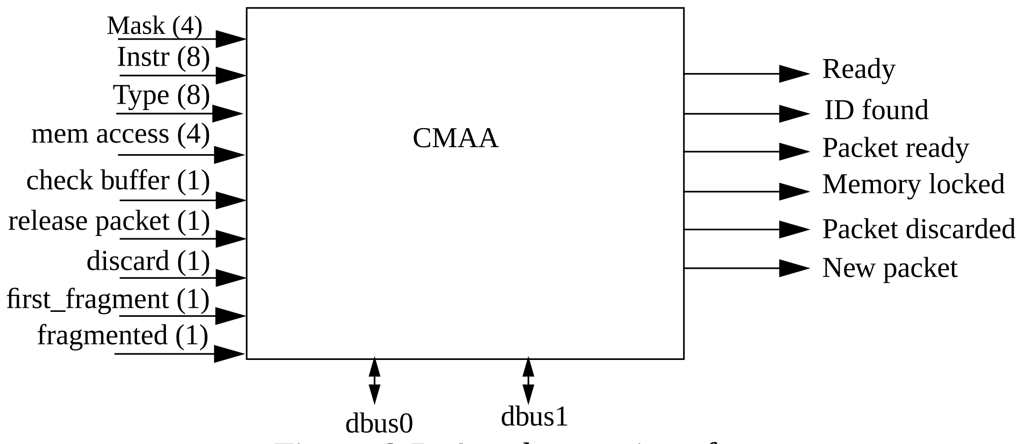

Figure 8.5: Accelerator interface.

Basically the input to the CMAA consists of flags and an instruction generated in the C&C. In table 5 the simple instruction set (6 instructions) is listed.

Table 5: 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 for data transport.

8.3.3 Data path

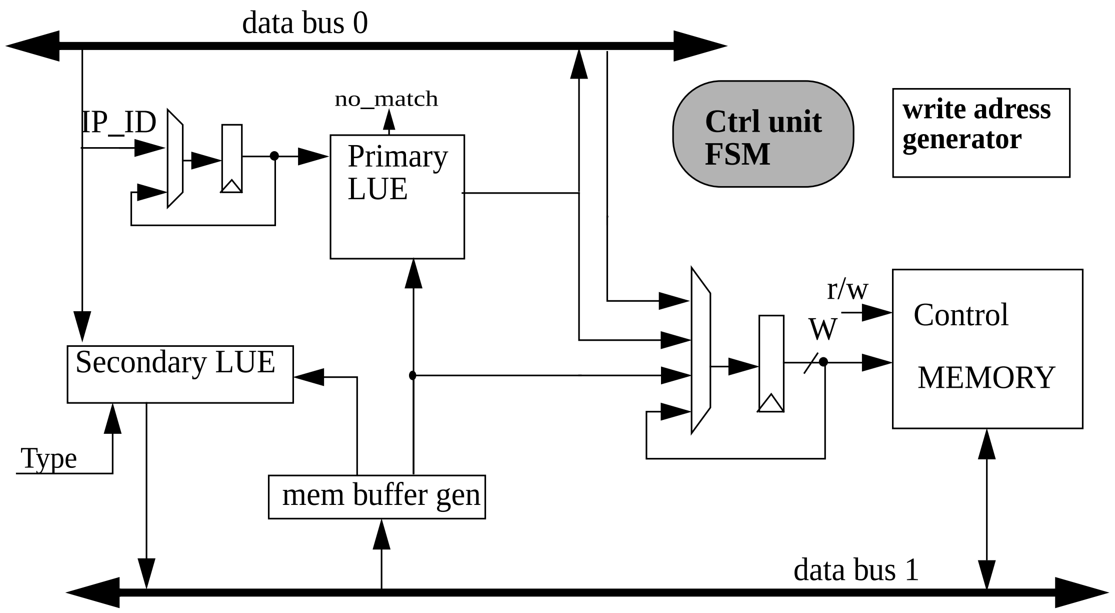

Figure 8.6: CMAA architecture. An accelerating hardware architecture for control memory access in the protocol processor. Based on traditional packet classification techniques it support low latency access to stored connection variables in the control memory

An overview of the CMAA architecture is illustrated in figure 8.6 . The CMAA data path includes two LUE, a buffer pointer generator, and a simple memory access selector. The Primary LUE (PLUE) only includes one

CAM which has 16 bit wide entries, has M entries and the result memory is W bits wide. The purpose of this unit is to check if we already have received an 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 exist 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 8.6 . The SLUE is a classification engine including 6 CAMs and its purpose is to check for valid connections. The two data buses is 32 bit wide. The mem buffer gen generates new buffer addresses for both packet buffers and connection buffers. The adress 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 an huge impact on the performance of the CMAA. This issue is further discussed in section 8.3.7.

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.

8.3.4 Control procedure

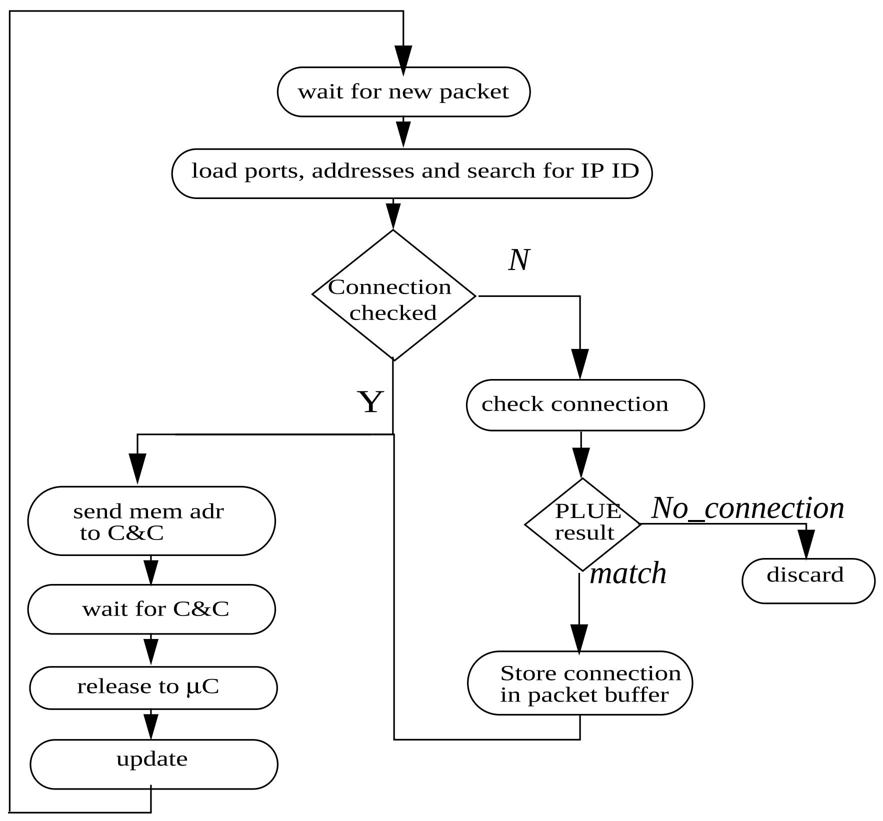

Figure 8.7: Control handling procedure within the CMAA.

The normal packet reception procedure of operation in the CMAA, is illustrated by figure 8.7 . 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 Identification 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, e.i. 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 adress pointer is provided to the data bus 1. Next clock cycle, the data bus 1 value will be stored at the packet buffer adress 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.

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 adress for new entries to the two LUE. This is only done if a write operation has been performed. During the write adress 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.

8.3.5 Control memory organization

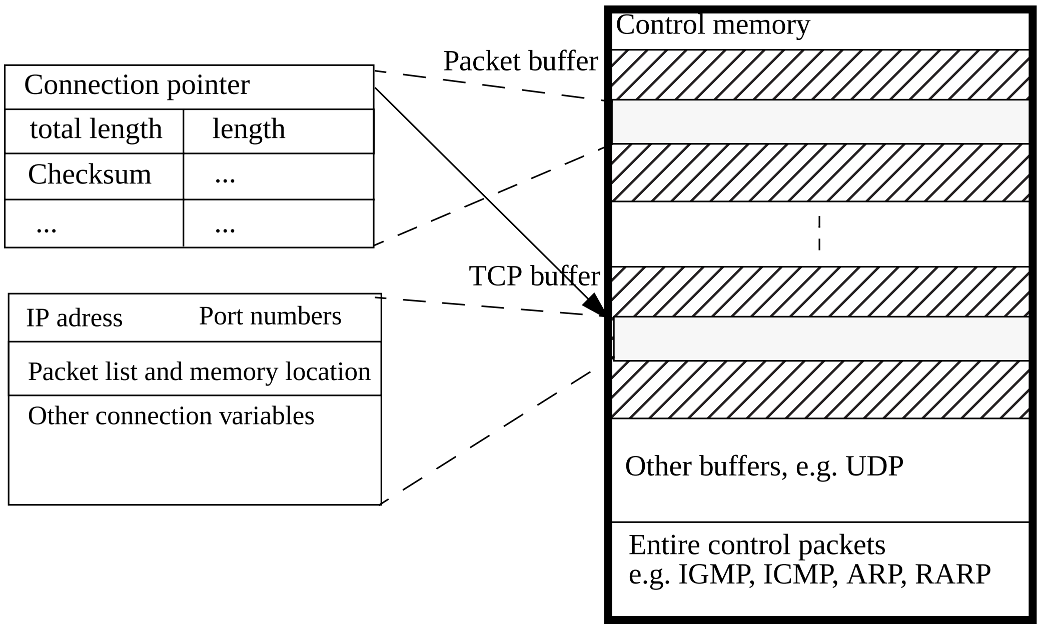

Figure 8.8: Memory organization in the control memory.

The control memory is organized according to figure 8.8 . As illustrated the control memory consists of a number of different buffers storing inter-packet information. Further the memory include all the control oriented packets that is going to be processed in the micro controller software. Since these protocols is completely processed by the micro controller, also the payload of these packets is stored in the control memory. For TCP and UDP type of packets only preprocessed header information 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.

8.3.6 Look-up Engine architectures

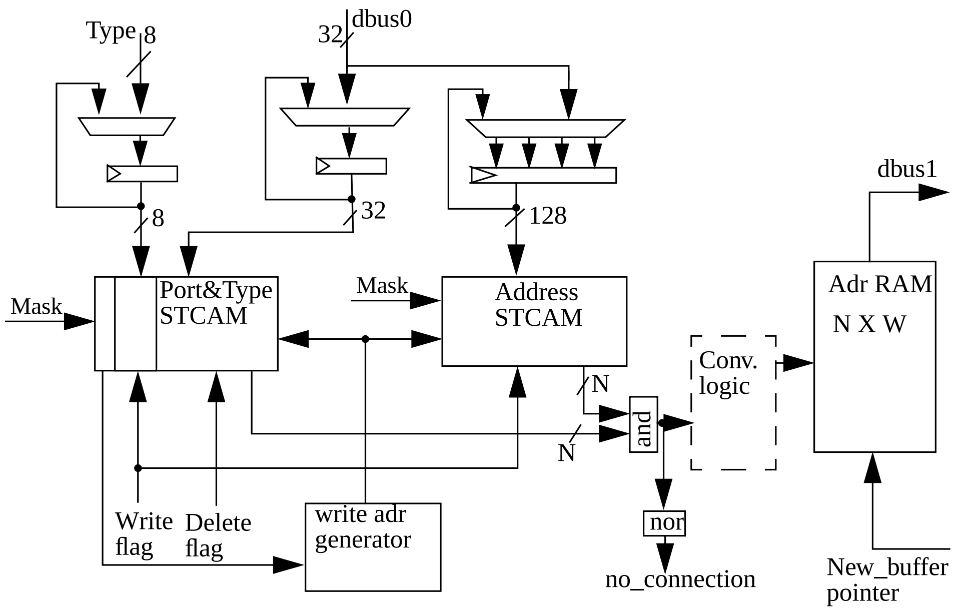

Figure 8.9: Secondary Look-Up Engine (SLUE) architecture. Note that the conversion logic that converts the matching vector to a result memory adress can be eliminated if the matching vector is used directly as word lines in the memory. This however require that the RAM must be implemented in the same manufacturing process.

The SLUE consists of 6 CAMs as illustrated by figure 8.9 . 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, e.i. the control memory address for the received packet.

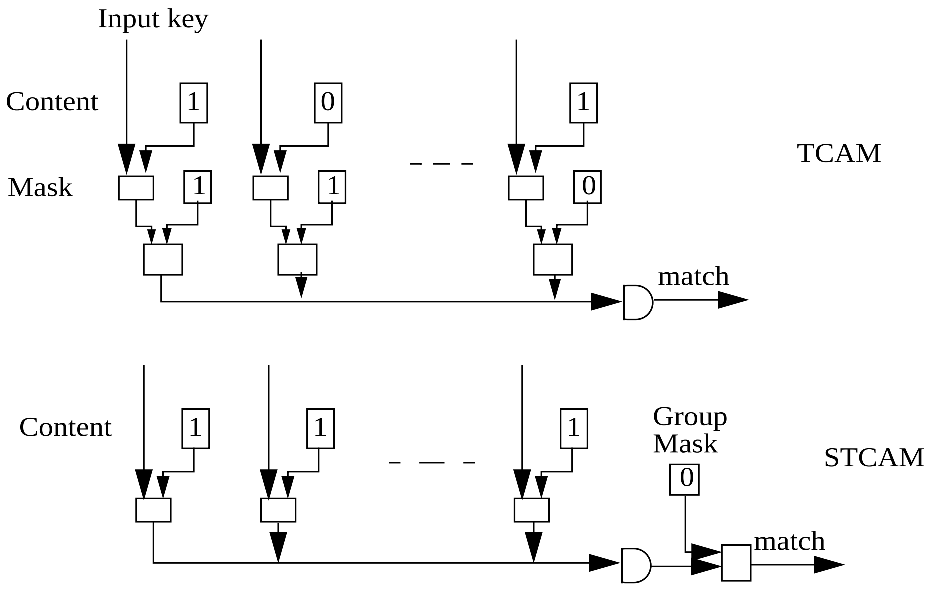

Figure 8.10: Simplified TCAM principle.

The 7 different CAMs we propose to be used in the CMAA architecture will have an huge impact on the performance, size and power figures of the entire design. Therefor they require a thorough investigation and optimization procedure, in order to obtain the optimal system performance. Even if the optimization of these CAMs not is in the scope of this paper, some characteristics and requirements on the CAMs can be noted. First of all we propose that CAMs should be instead of TCAMs ([19] and [20]). This reduces the cell size and power dissipation. The primary LUE is a standard CAM memory 16 bit content and M entries. The result memory is M times the length of the control memory adress W.

In order to provide flexibility for different protocols we use a concept we call Simplified TCAM (STCAM) illustrated by figure 8.10 in the secondary LUE. Instead of using ternary bit comparisons as in TCAMs we only provides an wildcard function to the entire CAM. In figure 8.11 there is an illustration showing how the secondary LUE uses the STCAM principle.

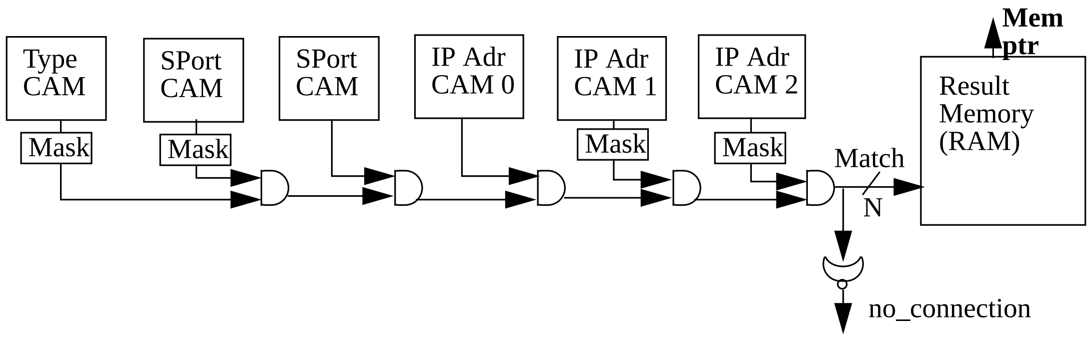

Figure 8.11: The two different STCAM in the SLUE each consists of three ordinary CAMs and some masking functions. Each of the CAMs uses N entries.

The mask input enables a wildcard functionality for different fields when recognizing an incoming packet according to table 6. 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 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 6: 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	*

Table 6: Configurations using masking for different packet types and applications.

Protocol examples	Type	Source Port	Destination Port	Source Address	Destination Adr
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 that 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 is 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 8.11 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.

8.3.7 CAM implementation issues

The total size of the 7 CAMs and there result memories will be a major part of the system 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 mm^2 each.

$$A_{CAM} = (16 \times M + (128 + 40) \times N) \times 350 + (M + N) \times W \times 180 \qquad \mu m^2 \tag{EQ 3}$$

As an example M=16, N=64 and W=20 can be considered. The chip-area for the two LUE 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 analyze 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 LUE:s to be pipelined, e.i. 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 shows that the small PLUE will not require more than 2 clock cycles to complete one search, e.i. 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 3 clock cycles. The design is simply to large and hence the P&R problem to complex. Therefor 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 is 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, e.i. 0.13 micron. The resulting latency for CMAA operations is further discussed in section 8.3.8.

The latency, critical path, power consumption in the LUE is 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.

8.3.8 Latency

The proposed architecture for access of the control memory, reduce 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 7 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.

Table 7: 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

8.3.9 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.

8.3.10 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 need 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 8.7 .

8.4 Conclusions

A novel architecture for acceleration of control memory access in a protocol processor for network terminals was presented. 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.

8.5 Further work

In order to complete the specification of the protocol processor three main research areas remains. The first one is to specify the interface between the μC and the host system and its DMA. Secondly the counter and controller unit is not finally implemented and programmed. The third issue regards the configuration method of the protocol processor. What does the programming and re-configuration interface look like from the μC?

8.6 Acknowledgments

This research was funded by ECSEL graduate research school and TFF the Swedish Scientific Research Foundation.

References

[9] 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

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

[11] 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

[12] A. S. Tannenbaum, “Computer Networks”, 3nd Edition, Prentice Hall PRT, ISBN 0-13-349945-6, 1996

[13] “Building Next Generation Network Processors”, White paper, Sep 1999, Agere Inc., http://www.agere.com/support/non-nda/docs/Building.pdf

[14] D. Husak, “Network Processors: A Definition and Comparison”, White paper, C-PORT, http://www.cportcorp.com/solutions/docs/netprocessor_wp5-00.pdf

[15] U. Nordqvist, T. Henriksson, D. Liu, “CRC Generation for Protocol Processing”, NORCHIP 2000, Turku, Finland

[16] McAuley A. et al., “Fast Routing Table Lookup Using CAMs”, IEEE INFOCOM ‘93, March 1993

[17] P. Steenkiste, “A Systematic Approach to Host Interface Design for High-Speed Networks”, IEEE Computer, v.27 n.3, pp.47-57, March 1994

[18] M. B. Abbott, L. L. Peterson, “Increasing network throughput by integrating protocol layers”, IEEE/ACM Transactions on Networking, vol. 1, pp 600-10, 1993

[19] 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

[20] 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