Linköping Studies in Science and Technology

Thesis No. 911

Hardware Architecture for Protocol Processing

Tomas Henriksson

INSTITUTE OF TECHNOLOGY

LINKÖPINGS UNIVERSITET

ISBN: 91-7373-209-5 ISSN: 0280-7971

LIU-TEK-LIC-2001:48
Department of Electrical Engineering
Linköpings universitet, SE-581 83 Linköping, Sweden
Linköping 2001

Abstract

Protocol processing is increasingly important. Through the years the hardware architectures for network equipment have evolved constantly. It is important to make a difference between terminals and routers and the different processing tasks they encounter. It is also important to analyze in detail the functional coverage of a hardware architecture. The maximal supported line speed is also interesting and especially which functionality can be kept at this line speed.

There are some types of hardware architectures that have gained much attention in research and from industry. Among these application specific instruction set computers, RISC with optimized instruction sets and reconfigurable hardware architectures are most often used. Very many network processors have been presented that aim for routers. So far not many protocol processors for terminals have been suggested. In terminals the requirements are different, for example low power consumption is very important for battery powered terminals.

I and my colleagues have proposed a novel way to build a protocol processor for a terminal. The main concept is to use an array of reconfigurable functional pages, which are connected in a deep pipeline. This deep pipeline serial processor is supported by a micro controller for exception handling and configuration tasks. The most performance-critical functional page in an Ethernet TCP/IP environment is the cyclic redundancy check. We allocated and scheduled the cyclic redundancy check in parallel with other functions. After having investigated different solutions we found that our functional page for cyclic redundancy check can manage 10 Gb/s, if a 0.15 micron manufacturing process is used in combination with optimized RTL code and synthesis.

Our architecture allows extensive parallel operation. The functionality is partitioned into the autonomous functional pages, which work in parallel. This reduces control overhead and simplifies the verification process. Low control overhead and extensively parallel computations admit low-power operation. The designed processor handles reception processing on a single packet or frame. It works in parallel with the host processor and significantly reduces the workload on the host processor. The designed processor always operates at line speed and supports up to 10 Gb/s.

Preface

This thesis presents the results of my research during the period August 1999-October 2001. The following four publications are included in the thesis:

- 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
- 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, 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”, Invited submission to a *special issue of IEE Proceedings on Circuits, Devices and Systems* as an extended version of Paper 2. The manuscript is under the process of review at the time of writing.

Other publications, not included in the thesis:

- 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
- Tomas Henriksson, “Hardware Architecture for 802.11b Based H.323 Voice and Image IP Telephony Terminal”, in proceedings of *Swedish System-on-Chip Conference 2001*, Arild, Sweden, March 20-21, 2001

Acknowledgments

First of all I would like to thank my advisor Professor Dake Liu for coming up with good ideas, for always being positive and supportive and for all interesting discussions we have had. I would also like to thank Professor Christer Svensson for making it possible for me to start my research activities and for many interesting discussions.

I greatly acknowledge the hospitality of Professor Ingrid Verbauwhede at University of California at Los Angeles (UCLA) for letting me work in her group for five months during the spring and summer of 2001. There I gained much knowledge on routers and switches and also had the opportunity to meet many fellow researchers and discuss various topics with them.

I thank my fellow Ph. D. students Ulf Nordqvist and Lic. Eng. Henrik Eriksson, and Professor Per-Larsson Edefors, who have coauthored papers with me.

All other members of the divisions of Electronic Devices and Computer Engineering at Linköpings universitet who have contributed to a nice working environment and with technical and administrative support.

SwitchCore Corp. and George Liu, Ericsson Research, have been valuable resources of information about future industry needs and current industry practice.

The thesis work was sponsored by the Swedish Foundation for Strategic Research (SSF) by the Integrated Electronic Systems Programme (INTELECT).

Contents

1

Introduction

1.1 Background

Computer network usage has increased enormously over the last decades. When the Internet gained popularity in the middle of the 1990's, computer networks became everyone's concern and the network capacity was far less than the demand. In the last five years the data traffic has actually more than doubled every year [1.1]. In the second half of the 1990's, the optical transmission equipment and the optical fibers have been developed in order to cope with tens of gigabits/s (Gb/s) on each physical link. With the introduction of dense wavelength division multiplexing (DWDM) the fiber throughput increased further by more than one order of magnitude. In the near future capacities of 3.2 Tb/s (=3200 Gb/s) will be available. The physical limit of the fiber is somewhere around 100 Tb/s, when efficient coding techniques are used.

The electronic equipment that processes the information sent over these optical links has however not kept up with the increase in transmission speeds. This in combination with the need for more advanced network features, such as quality of service, traffic shaping and network security has recently led to the development of many new hardware architectures for network equipment. Some of these architectures are already used in commercial systems, but others are only in the development phase. This thesis discusses the suggested hardware architectures and presents a fundamentally new way of performing protocol processing.

This introductory chapter briefly explains the basics in computer networks and discusses the processing needs in computer network equipment. The next chapter introduces network processor architectures and surveys existing and upcoming hardware architectures. Finally, the novel architecture, developed by me and my colleagues, is presented and the following chapters, including papers 1-4, are introduced.

1.2 Computer Network Basics

Computer networks are complex and many different types exist. To thoroughly describe them all is outside the scope of this thesis, however, computer network knowledge is of importance to understand the content of the thesis. Except for this short introduction to computer networks, the interested reader is encouraged to read the books [1.2] and [1.3] to gain further knowledge.

Computer networks are used to connect small groups of computers, located close to each other, so called local area networks (LANs). There is a need to interconnect these islands of connectivity and several different technologies have been developed. Nowadays two big winners have evolved, for LANs it is Ethernet and for LAN interconnection it is the Internet Protocol (IP). Other protocols keep appearing, such as wireless access protocols. At the same time, new types of equipment, not traditional computers, get connected to computer networks. This makes the computer networks steadily changing. In this thesis focus on Ethernet, IP, and associated protocols is maintained throughout the text.

To easier understand how networks function, it is good to use the ISO/OSI (Industry Standard Organization/Open Systems Interconnection) reference model, which consists of seven layers, see figure 1.1. The reference model was developed 1984 and really describes how protocols should be

Layers and names	Examples
Layer 7: Application Layer	HTTP, SMTP
Layer 6: Presentation Layer	
Layer 5: Session Layer	
Layer 4: Transport Layer	TCP, UDP, ATM
Layer 3: Network Layer	IP, ATM
Layer 2: Data Link Layer	Ethernet, PPP, SONET, SDH
Layer 1: Physical Layer	Ethernet, SONET, SDH, V32bis

Figure 1.1: The 7 layer ISO/OSI reference model

implemented. Most standards do not follow the reference model in detail, but it can be seen as a basis for all computer networks. For the Internet protocols normally layer 5 and 6 are not used.

This layering can be used to describe the difference between LANs and the Internet. LANs only use layer 2 to forward packets to the correct destination, but the Internet uses layer 3 to forward packets to the correct destination. A destination is identified by an address. Addresses on layer 3 are world wide unique, but addresses on layer 2 could in principle be reused in every LAN in the world. For Ethernet however, also layer 2 addresses are world wide unique. This reduces the configuration need when one device is moved from one LAN to another. Protocol layers above layer 3 are only used in the communication peers in the strictly layered model. In reality however, the network infrastructure make use of layer 4 and sometimes also layer 7 information.

A protocol on layer N provides a service for protocols on layer N+1. At the same time it uses services from layer N-1. Protocols on layer 7 provide services for applications and protocols on layer 1 use a physical medium instead of any services from a lower layer protocol to fulfill its operation. Just as the seven layer model, also this nice service provider model only works in the-

ory. In practice, the layers are sometimes not easily distinguishable and even sometimes not wanted. For example ATM, which provides a transport service and thus should be categorized as a layer 4 protocol, is used to transport IP packets which reside in layer 3. Therefore a well bounded layer 4 network, such as an ATM network, can be seen as a layer 2 network by layer 3 protocols, since layer 4 protocols provide similar service as layer 2 protocols, but normally over much wider area. The Internet Protocol can also use its own service, i.e. a layer 3 protocol uses the service of itself in the operation called tunneling. There are many more examples of when the protocol layering does not model the exact behavior of a network, but nonetheless it provides us with a helpful way of thinking about network protocols.

A layer handles protocol data units (PDUs), which are used for communication. A PDU consists of a service data unit (SDU) and some control information, normally organized as a header. The PDU of layer n+1 is the SDU of layer n, i.e. more control information is added in each layer as the original information proceeds towards layer 1 in the sender, see figure 1.2.

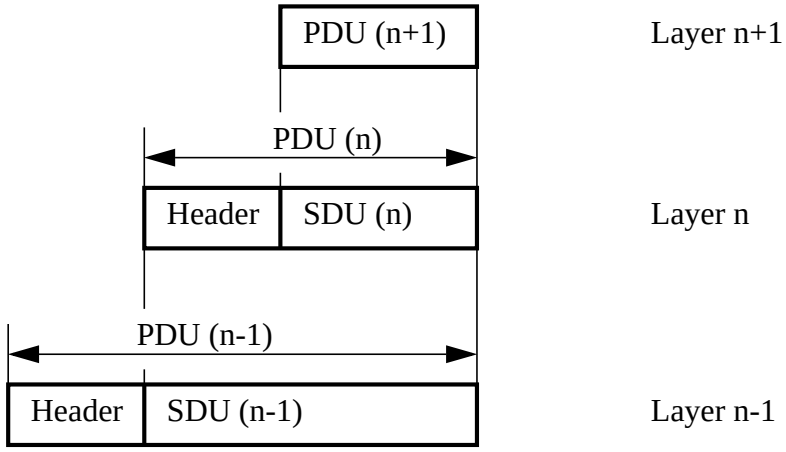


Figure 1.2: PDU passing between protocol layers

1.3 Equipment in Computer Networks

The different types of equipment in computer networks can roughly be categorized into two main classes: terminals and routers. Of course this coarse categorization is not enough when looking at implementation details. In this section, the terminals and their functionality will be described in detail and then routers are explained more briefly, since they are not of the same importance for the hardware architecture presented in this thesis.

1.3.1 Network Terminals

Network terminals are present in many different environments. The traditional network terminal is the desktop or laptop computer, but new types of terminals such as network printers, web cameras, IP phones and various embedded systems appear more and more frequently in our offices and in our homes. This kind of terminals are typically connected to an Ethernet or via a modem connection. Another type of terminal is the handheld battery-powered appliance which can be an advanced mobile phone, a personal digital assistant (PDA), or a digital camera. This type of terminal is connected through a wireless access network. Voice and image telephones will also be connected through a multi service computer network instead of the traditional circuit switched

specialized networks that have been used so far. These will be a very important terminal category in the near future.

Obviously, there are totally different requirements on these totally different types of terminals. The thing that is characteristic for all terminals is that they are end-points is a computer network. That means that when data is transported in the network a terminal is either the source or the destination of the communication. No data passes through the terminal, therefore it does not have a need for routing capability. A general view of a terminal can be seen in figure 1.3.

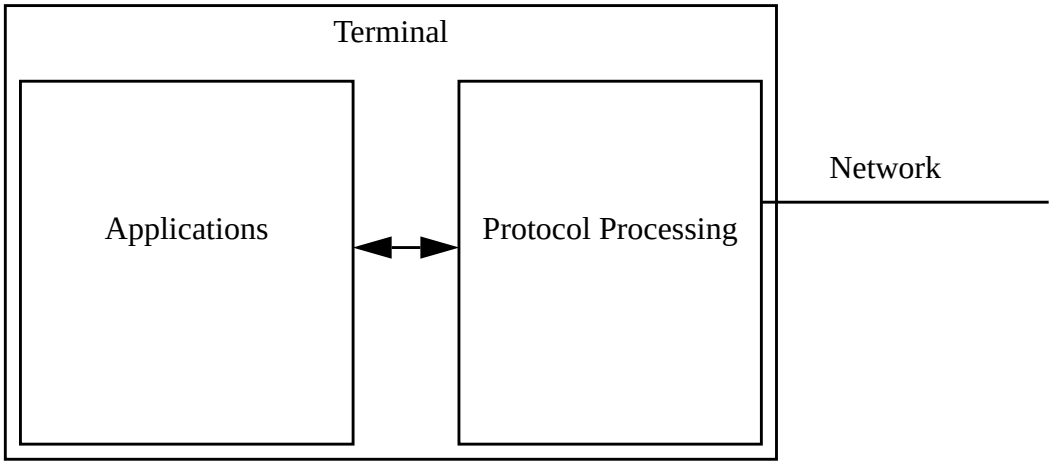


Figure 1.3: A terminal

The terminals have to handle all layers of protocols in the ISO/OSI reference model. Layers 5 through 7 are application oriented and not discussed here. Instead the study starts with the transport layer protocols. These transport layer protocols deal with the end to end communication and many, e.g. TCP, are connection oriented. A connection oriented protocol must build up a connection before any data can be transported and it tears down the connection when the data transfer is completed. A connection oriented transport protocol typically assures correct delivery of data to the destination. This means that the terminal must keep track of many connection state variables, which describe the state of the connection, e.g. how much data has been sent, received, and acknowledged. A terminal must also handle conectionless transport protocols, e.g. UDP, which provides a simple datagram service to the application oriented layers.

Except for the transport layer the network layer and the link layer must also be handled. The network layer deals mainly with routing and the link layer normally with point to point communication. The link layer in the original Ethernet worked as an ether, where every terminal could hear all traffic on the ether, which was a coaxial cable. Nowadays, however, the terminals are most often connected with point to point links. The network layer is by definition used by the network, i.e. the routers, and is of little interest for the terminal. It is used to identify the communication peer and the terminal needs to insert the correct addresses in the network protocol packet header when sending data and check and strip the same header when receiving packets.

The link layer can just as the transport layer be connection oriented, but in many networks this is not the case, e.g. Ethernet. Some wireless protocols are connection oriented and they introduce similar requirements on functionality as TCP. It seems like the development goes towards conectionless link layer protocols even for wireless protocols, since the IEEE 802.11 standards are gaining popularity.

1.3.2 Routers

Routers are fundamentally different from terminals, since their main operation is not to take part in any communication, but rather to forward data to the correct destination. Routers can operate at many different layers in the ISO/OSI reference model, traditionally they only manage protocols up to layer 3, but this is about to change as will be discussed later on.

A router that only works on layer 1, the physical layer, is called a repeater. A repeater simply amplifies the signal carrying the data so that it can travel longer distances. A router that works on layer 2 is normally called a switch. A complete router works on layer 3 and interconnects two or more layer 2 links of different type, which are nowadays normally point to point links. The Internet Protocol (IP) is the network protocol that has become dominant in the world and it is not likely that any other network protocol will take over in the foreseeable future, except from the next generation IP, IPv6.

Although routers main task is to forward data packets, they also act as end points for some communication, the management communication. In a router the rules for how to forward packets are stored in a table. Updating information for this table is among the things that must be communicated to the router. Management protocols normally also allow an operator to collect statistical information about the data traffic that flows through the router.

Generally the router activities can be split into two planes, the data plane and the control or management plane. The data plane consists of the performance demanding and simple task of forwarding packets to the correct link. The control plane consists of all other activities, that are more complex, but do not require so fast handling, although routing table updates may be as frequent as several hundred per second.

Traditionally routers are therefore built up with two different paths for packets. The fast path for packets that shall only be forwarded and the slow path for packets that require more complex handling. The packet stream from each link is split into the fast path and the slow path as early as possible in the processing.

Routers are becoming more complex as it is realized that they can be used for traffic shaping, policing, statistics gathering, security, and quality of service. All of these tasks require the router to process also layer 4 information in the packet header and sometimes even layer 7 information, which typically requires scanning through all data in the packet.

Traffic shaping deals with the task of balancing the traffic that must pass a single point in the network on two or more links that eventually all will lead to that point. Policing is concerned with limiting the traffic from or to a certain terminal or group of terminals, e.g. if a company has only paid for a certain bandwidth, the Internet service provider (ISP) will limit their usage to this bandwidth. Statistics gathering is needed to efficiently be able to use traffic shaping and policing and may also be used by the operator in order to analyze the network performance and resolve possible bottlenecks. Quality of service normally deals with guaranteed bandwidth and guaranteed delay from source to destination with a certain high probability. Thus packets from certain sources or packets that are part of certain connections may get prioritized treatment over packets with only best effort service. For real time voice and video transmission, quality of service is essential. Security can be anything from simple firewall tasks, such as not allowing packets from a specific source to enter a special part of a network, to advanced virus scanning and denial of service attack detection and prevention.

Routers are built in two completely different ways. The traditional architecture has the links connected to line cards. The line cards include almost all functionality of the router. They are connected to a switching backplane, which takes care of the transport of packets from one line card to another. Packets are split into cells, which are passed over the backplane after arbitration has been

used. Each line card needs to have buffers for incoming and outgoing packets, since the backplane normally does not have the capability to store many cells. Each line card can support one or many links. If there are many links these typically share the processing resources on the line card. This type of routers are referred to as big routers in this thesis, and can be seen in figure 1.4. They are

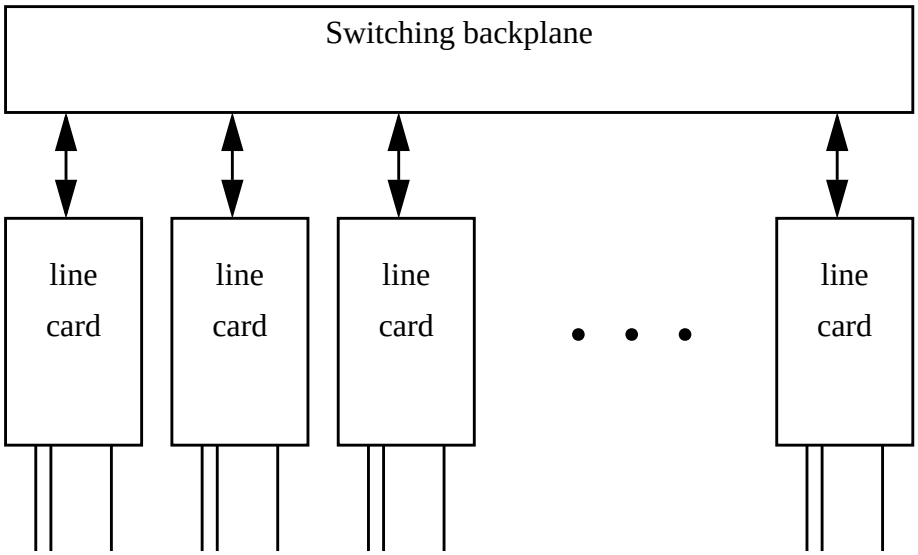


Figure 1.4: A general view of a big router

common in the core and in the distribution parts of the networks.

The other way to build a router is the one chip solution, which tries to include all functionality into one single silicon die. These routers obviously are not as scalable as the line card based, but may allow high transmission speeds on a small number of links. Normally only the fast path is handled by this one chip router and additional components such a slow path processing and memories are needed to complete the router. This type of router is referred to as a small router in this thesis and can be seen in figure 1.5. Here the switching between the links is normally handled by shared memory. All links must be able to write and read packets from the shared memory at line speed, which puts a very high requirement on the memory bandwidth. This type of router is normally used in the access parts of the network. There are of course also mixtures of the two main types of architectures.

1.4 Processing Tasks in Computer Network Equipment

The processing tasks that arise from the protocols used in computer networks will be addressed in this section. They will be the foundation for the discussion on hardware architectures for network processors in the next chapter. Like the previous section also this one is split into one part that discusses network terminals in deep and another section that briefly discusses the routers.

1.4.1 Processing Tasks in Network Terminals

The processing tasks in network terminals are described here, special attention is given to the tasks which have to be performed on incoming packets. We are more interested in the reception of packets than the sending of them because of the unbalanced data load. A terminal receives more data than it transmits. Therefore it is more interesting to optimize the process of packet reception than that of packet transmission.

When a packet arrives at the terminal, first of all the physical layer must be handled. The physical layer includes some kind of coding or modulation or both. The physical layer protocols differ

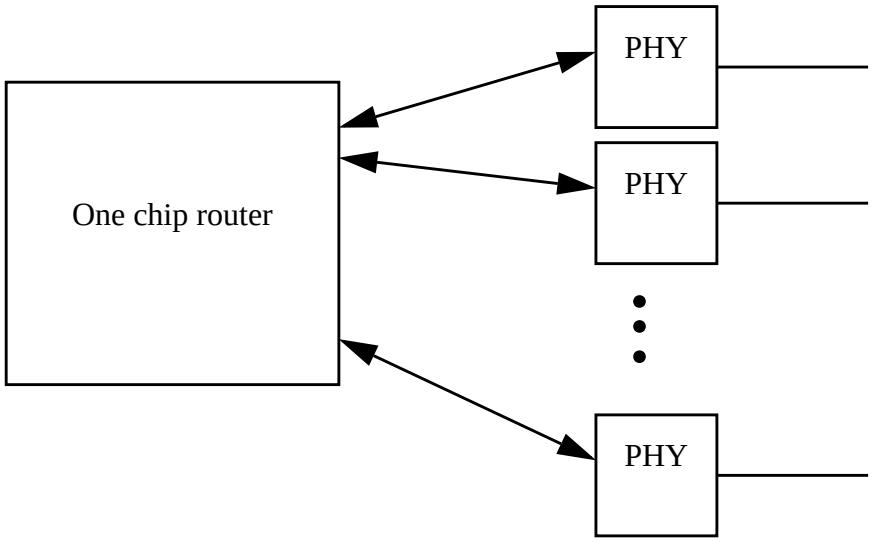


Figure 1.5: A general view of a small router

a lot, depending on if the medium is wire, wireless or fiber. Also distance and bitrate influence the protocol. In this thesis, the concentration is on layer 2-4 and the important observation is that from the physical layer processing a bitstream is outputted. This bitstream is supplied to the data link layer. In the data link layer typical tasks are:

- Check the destination address
- Determine the packet length
- Calculate and check a checksum
- Demultiplex the packet stream, dependent on layer 3 protocol
- Decompress compressed headers and/or data

For connection-oriented data link layer protocols, tasks coping with the connection handling are also required. Those are discussed in the transport layer.

When the layer 2 header has been stripped off, the packet it is sent to the network layer entity. Here only IP and IPv6 are considered, since they are expected to dominate the network layer for a long period of time. In e.g. Ethernet networks an address resolution protocol (ARP) is used to find the corresponding Ethernet address to a given IP address. ARP can also be seen as part of the network layer, although it is often labeled layer 2.5. This is because it provides a service to the IP and thus is layer 2 protocol, but it uses services provided by Ethernet and thus is layer 3. It is typical for control and management protocol that they do not fit in the reference model very well. Similarly internet control message protocol (ICMP) and internet group management protocol (IGMP) could be considered to be on layer 3.5. Normally however they are said to be part of the network layer. When considering all above mentioned protocols to belong to the network layer, the processing task are the following:

- Check ARP operation code
- Update ARP table
- Trigger ARP reply
- Check the IP destination address
- Check IP version
- Calculate and check IP header checksum
- Decrypt encrypted packets
- Reassemble fragments

- Handle time-outs for missing fragments
- Process IP options and IPv6 extension headers
- Demultiplex the packet stream, dependent on layer 4 protocol
- Check ICMP type and code
- Trigger ICMP reply
- Check IGMP version
- Check IGMP type

On layer 4, like IP on layer 3, TCP and UDP have become totally dominant during the last years and there is no reason to believe that any other layer 4 protocol can compete with these in the foreseeable future. They also represent two totally different types of transport protocols, TCP is connection oriented and provides a reliable byte stream between sender and receiver. UDP is conectionless and provides a best effort datagram service between sender and receiver. TCP is typically used for point to point communication of important and delay insensitive data. UDP on the other hand, is used for broadcast, multicast and delay sensitive communication or by applications, that implement the reliability in the application layer. The processing tasks in the transport layer are:

- Calculate and check the checksum
- Check destination port
- Demultiplex the packet stream and deliver payload to application
- Reorder out of order data
- Discard duplicate data
- Trigger an acknowledgment packet
- Check TCP flags
- Manage the connection state
- Update the window size
- Process TCP options

1.4.2 Processing Tasks in Routers

The processing tasks in routers are twofold. All routers can act as terminals for control and management functions, thus they require the same processing as terminals, as well as the processing of the routing and management protocols. However, for terminals it is often important to perform this processing with small memory, low power consumption and so on. For routers it is a small part of the total functionality and most packets that enter a router are not destined for it. Instead they must be forwarded and this subsection will study the processing tasks that are associated with the forwarding of packets.

As for terminals, no attention is given to the physical layer. The data link layer on the other hand can be of different type than for terminals. For terminals most often Ethernet or a wireless local area network (WLAN) are used. For routers, on the other hand, SONET, SDH, and ATM are common, but also Ethernet and other types of access networks. Although ATM handles switching functionality internally, it is seen as a data link layer when IP packets are transported in ATM cells between IP routers. The layer 2 processing is similar to the processing in terminals and therefore not presented again. ATM imposes other processing tasks as well, but they are not discussed here.

The network layer is the most important layer for the router. The traditional router only uses information in the network layer to determine the handling of a packet. In a pure layered network the router should not care at all about the transport layer, but in IP networks the router has to do some operations also on the layer 4 header fields. The classical processing tasks for the network layer and the transport layer are:

- Extract destination address

- Perform longest prefix match on destination address with routing table
- Switch packet to the right output link
- Decrease time to live/hop limit counter
- Process IP options
- Update checksums
- Gather traffic statistics

When more functionality is added to the routers, the processing requirements change. For most of the new functions a more thorough packet classification is needed. It is no longer enough to forward packets based on their destination address. More header fields are used to identify flows and groups of flows. Sometimes only the destination address and the source address are used to classify the packet, but also transport layer protocol, source port and destination port can be used. To enable single time classification for each packet, multiprotocol label switching (MPLS) has been suggested and is likely to be used. The main idea is to classify packets at the first router they encounter, give them a label which identifies the flow they belong to by a 24 bit label. This label can then be used by the other routers in the network to forward the packet. The use of MPLS introduces new processing tasks, such as packet classification, label management and label updates.

Another type of new functionality is scanning through the data content of the packet. This may have two different purposes. First, it can be to detect and discard packets that contain viruses or other not wanted data. Second, it may be to use layer 7 information in the forwarding process. The layer 7 information is not organized in a structured header as layer 3 and layer 4 information is, instead it is in text format, e.g. http and ftp. An example is to have a router, connected to a web server farm, where each server has different content, one may serve pdf-file requests, one may serve a special group of customers that require better and faster service than the public, one may handle secure transactions and so on. Then the router must detect which kind of http request the packet contains in order to forward it to the adequate server. These types of functionality require scanning through the data of the packet, which is a multiple string matching operation.

1.5 Performance Measures

Network equipment can be of very different complexity. Except for the processing that for example a router is capable of, another important issue is how fast transmission speeds it can support. Typically two different classes of tasks are essential, those that are made on all parts of the packet and grow in complexity linearly with the packet length and those which are made once per packet and have the same complexity no matter how much data the packet may contain.

This can give two limitations on the router performance, one is straightforward. That is the throughput of the per bit tasks, which directly limit the link transmission speed. The other comes from the per packet tasks. Here sometimes the smallest possible packets are considered, which guarantees functionality independent of packet length. The smallest packets in most networks are TCP acknowledgments, which consist of 40 bytes on layer 3 and 44 bytes on layer 2 in SONET environment. In some cases however the average packet length is considered, and this requires that a huge buffer is present in order to store incoming packets that have not been processed so far.

Only some standard link transmission speeds are important to support, for the core network these are 2.5 Gb/s, 10 Gb/s, and 40 Gb/s at the time of writing. These transmission speeds correspond to the standards OC-48, OC-192 and OC-768 respectively. OC stands for Optical Carrier and OC-1 corresponds to 51.84 Mb/s. After OC-48 normally the next standard is an increase in transmission speed by a factor of 4. This means that the next step will be OC-3072 which standardizes 160 Gb/s as the new transmission speed. It will, however, take some years until equipment can be built that supports such high speed. Most product announcements during the year

2001 aim at OC-192, i.e. 10 Gb/s, this standard is very appealing to support since it also means that LANs and MANs that use 10 Gigabit Ethernet can be handled with the same processing resources.

Considering the transmission speed 10 Gb/s and the minimum packet of 44 bytes = 352 bits, the network processor has only 35.2 ns to process each packet. Since most network processors are designed by using semicustom ASIC design flow, the clock frequency normally is limited to less than 300 MHz and approximately 10 clock cycles are available for each packet.

So when the maximal speed is supported most often the processing is very limited and typically only includes the basic functionality. When more complex processing is wanted the speed decreases and since only standard speeds are really interesting to support suddenly 4 times as much processing can be used in the next step. When investigating network processors it must be studied in detail exactly which functionality can be maintained at which speed. This thesis will however not evaluate network processors in detail and such information is left out in the next chapter, which surveys many existing network processor architectures, based on the fundamental processing element organization.

References

[1.1] L. Geppert, “The New Chips on the Block”, IEEE Spectrum, January 2001, pp. 66-68

[1.2] A. S. Tanenbaum, “Computer Networks”, Prentice Hall, third edition, 1996

[1.3] W. R. Stevens, “TCP/IP Illustrated, Volume 1 The Protocols”, Addison-Wesley, 1994

2

Network Processors

2.1 Network Processor Architectures

For terminals and especially for routers many new types of hardware architectures have been introduced during the recent years [2.1]. However, specialized programmable architectures for protocol processing have been suggested earlier, two of the earliest publications are [2.2] and [2.3]. This section studies the basic hardware architectures that are used for protocol processing, starting with traditional general purpose CPUs (central processing units) and fixed function ASICs (application specific integrated circuits). Then more advanced architectures are described. These new specialized architectures are often called network processors or protocol processors. They are of the type domain specific processors, which also exist for other applications [2.4]. The purpose of this type of processors is to cover only the functionality of an application domain, but still provide flexibility within this domain. This approach adds flexibility when compared to an ASIC solution and reduces control overhead when compared to a general purpose CPU. Reducing the control overhead leads to less power consumption and higher performance.

Except from the processing unit of networking equipment, also the memory organization and the interconnect system are utterly important. They are however harder to categorize and most systems still use the von Neuman architecture. This means that data is stored in a memory, which is connected to the processing unit via an arbitration based bus. Writing and reading data to and from memory are tasks that require very much time and consume power. It is clearly not the optimal memory architecture for high speed networking equipment. Although changing the memory organization and the interconnect system can reduce the flexibility, to a large extent it only influences the performance, not the functionality, of the system.

Some of the network processors which are discussed in the next section are integrating several components on a single chip. This clearly reduces the flexibility of the system, since restrictions

have been made concerning the interconnect structure between the components of the system. However, it may be the best way to achieve the desired performance since on chip communication normally is faster and more power efficient than off-chip communication.

2.1.1 General Purpose CPU

In many terminals, especially desktop and laptop computers, general purpose CPUs are used for protocol processing. This was also the case in early routers and sometimes occur in embedded systems. The general purpose CPU has, of course, all the functionality needed to handle all kinds of protocols, but does it very inefficiently since there is a huge control overhead for each instruction, since it has to be fetched and decoded. There is also an overhead associated with operations on header fields that are shorter than the CPU word length. The general purpose CPU has problems with real time processing due to interrupt handling, cache hierarchy and operating system and normally needs to buffer the packets several times. The general purpose CPU is most often only used for layer 3 and above processing, since layer 2 tasks often require real time processing.

2.1.2 Fixed Function ASIC

The fixed function ASIC is the total opposite to the general purpose CPU. The ASIC is designed for one protocol only and handles that very efficiently. The big problem is the lack of flexibility, nonetheless, ASICs are very successful for some protocols, e.g Ethernet. The ASIC typically operates on a stream of data and does not suffer from the overhead of buffering data in a random access memory.

2.1.3 Application Specific Instruction Set Computer

Application specific instruction set computers are also called application specific instruction set processors (ASIPs). The idea with an ASIP is to design an instruction set that matches the application. This gives fairly high flexibility and at the same time, crucial operations can have a single instruction and thus be executed efficiently. The ASIP uses a standard von Neuman architecture. To find the correct instruction set, instruction profiling can be used. This means that the algorithms are executed in a simulator and statistics are gathered of how often each operation is used. It must then be decided which operations get specialized instructions and which will require a sequence of instructions. If some sequence of operations is frequently appearing in the execution it may be considered to cluster these operations into one single instruction.

The number of instructions required for a certain task can differ a lot between ASIPs and general purpose CPUs and also between two different ASIPs. Therefore, the traditional measurement of computing power, million instructions per second (MIPS) no longer suffice as performance measurement when comparing network processors.

2.1.4 RISC with Optimized Instruction Set

The RISC (reduced instruction set computer) with optimized instruction set is similar to the ASIP. The big difference is that here a standard RISC architecture is the basis and some instructions are added to this core instruction set. This simplifies the development of the architecture as well as of the supporting tools, such as compilers. However, it may result in some implemented functionality that is unnecessary for the network processing domain. This approach can be seen as a mixture of the general purpose CPU and the ASIP.

The added instructions can be implemented in some different ways:

- *Instruction acceleration* means that no new instructions have been added, just that the instructions most frequently used are efficiently implemented.

- *Data path extension* means that the data path has been extended with new blocks that handle the added instructions. Still the execution is controlled in the same way as for the original RISC. The single instruction multiple data (SIMD) type of processors fall into this category.
- *Slave processor extension* means that the added instructions are executed in a slave processor, that makes the modified RISC become a multiple instruction multiple data (MIMD) machine. That is, the new instructions can execute in parallel with the original RISC instructions and the control path of the processor must be redesigned.

2.1.5 Reprogrammable Hardware Architectures

Reprogrammable hardware architectures have successfully been used for many types of applications. The big advantage over traditional processors is that they allow much more parallel processing, just like an ASIC. Reprogrammable architectures also have the flexibility that an ASIC lacks. Two types of general low grained reprogrammable architectures are available, programmable logic devices (PLDs), which realize sum of products functionality and field programmable gate arrays (FPGAs), which have small lookup tables to allow any logic function. PLDs are suitable for implementing finite state machines (FSMs), which are normally used in the control path of an architecture and FPGAs are suitable for implementing the data path.

Reprogrammable architectures can also be more application specific. They would then be coarse grained in the sense that the reconfigurability is not on bit level, but on word level [2.5]. This type of architectures sacrifice some flexibility in order to achieve better performance and lower power consumption than PLDs and FPGAs.

2.2 Network Processor Survey

In this section several network processors are discussed and categorized in two dimensions. First it is discussed where in a system they fit in and then their internal architecture is analyzed. The network processors are of very different nature, some are commercial products, others are research projects and yet some may only be recently announced and therefore not much information is available.

This section will mostly discuss the internal architecture of the processing elements, but as mentioned in the previous section, the communication architecture and the memory organization are of at least as big importance. For some of the architectures which will be presented it is clear that concern has been taken to these issues and then it is mentioned in the text.

Apart from the hardware architecture of a network processor also the programming support is very important. However, the programming environment, consisting of high level languages, compilers, debuggers etc. is outside the scope of this thesis. Nonetheless it should be noticed that some network processors have compilers for C or C++, others have support for some special network programming language and some have to be programmed using low level microcode. Which is the best is not clear and all approaches are likely to coexist for the next few years.

Table 2.1 summarizes the important characteristics of the discussed network processors.

Table 2.1: Network Processor Comparison Summary

Processor	System Aspects	Applications	Architecture
Motorola C-5	Router	Layer 2-7	RISC + ASIP
Intel IXP 1200	Router	Layer 2-7	RISC + ASIP
Agere FPP, RSP, ASI	Line Card	Data Path	ASIP + ASIC

Table 2.1: Network Processor Comparison Summary

Processor	System Aspects	Applications	Architecture
Agere NP10 and TM10	Line Card	Data Path	ASIP + ASIC
AMCC nP7120	Line Card	Layer 3	RISC + ASIC
AMCC nP7250	Line Card	Layer 3	
AMCC nP3400	Router	Layer 3-7	RISC
IBM PowerNP 4GS3	Router	Layer 2-4	RISC + ASIP + ASIC
Silicon Access iAP	Line Card	Address Lookup	ASIC + ALU
SwitchCore CXE-16	Router	Layer 2-4	ASIC
Coresma 6001	Terminal	Layer 2-3	RISC
PMC-Sierra PM7388	Line Card	Layer 2	ASIC
PMC-Sierra PM2329 ClassiPI	Line Card	Classification	ASIP + ASIC
Broadcom BCM 5680	Router	Layer 2-7	ASIP + ASIC
Broadcom BCM 5632	Router	Layer 2	ASIC
Broadcom BCM 1250	Router	Layer 4-7	General-purpose CPU
Solidum Systems PAX 1100	Line Card	Classification	Reconfigurable
Sitera (Vitesse) IQ 2000	Router	Layer 2-7	ASIP + ASIC
Xelerated X40	Line Card	Layer 2-4	ASIP
Lantronix DSTini	Terminal	Layer 2-4	General-purpose CPU + ASIC
Clearwater Networks CNP810SP	Router	Control Path	RISC
ClearSpeed Platform	Line Card	Classification + forwarding	SIMD
Lexra LX8000	Line Card	Layer 3-7	RISC
STM Network Processors			ASIP + ASIC
Infineon + Dresden University	Router/Terminal	Layer 2-4	RISC

Table 2.1: Network Processor Comparison Summary

Processor	System Aspects	Applications	Architecture
Tampere University TTA	Router	ATM switching	Reconfigurable
UC Berkely platform for wireless	Terminal	Layer 1-4	Reconfigurable
PRO^3	Router/Terminal	Layer 3-4	Reconfigurable + RISC
KTH Protocol Processor	Router	IP forwarding	RISC
UCLA Packet Decoder	Router	Packet decoding	ASIP

2.2.1 Motorola C-5 Network Processor

The C5NP can handle cell and packet processing, table lookup processing, and queue management functions, which makes it suitable for routers. It could with some supporting circuits such as memory be the heart of a small router, but it could also reside on a line card in a traditional router.

The C5NP is a multiprocessor chip, which has 16 channel processors (CP) and 5 supporting units for table lookup, queue management, buffer management, switching interface and control, see figure 2.1. The supporting processors are most likely of ASIC type, except for the executive processor for control, which is a standard RISC core. The CPs consist of 3 parts, one RISC core for internal control and two serial data processors (SDPs), one for transmit and one for receive, see figure 2.2. The SDPs are special processors, that run microcode and thus can be categorized as application specific instruction set processors. [2.6]

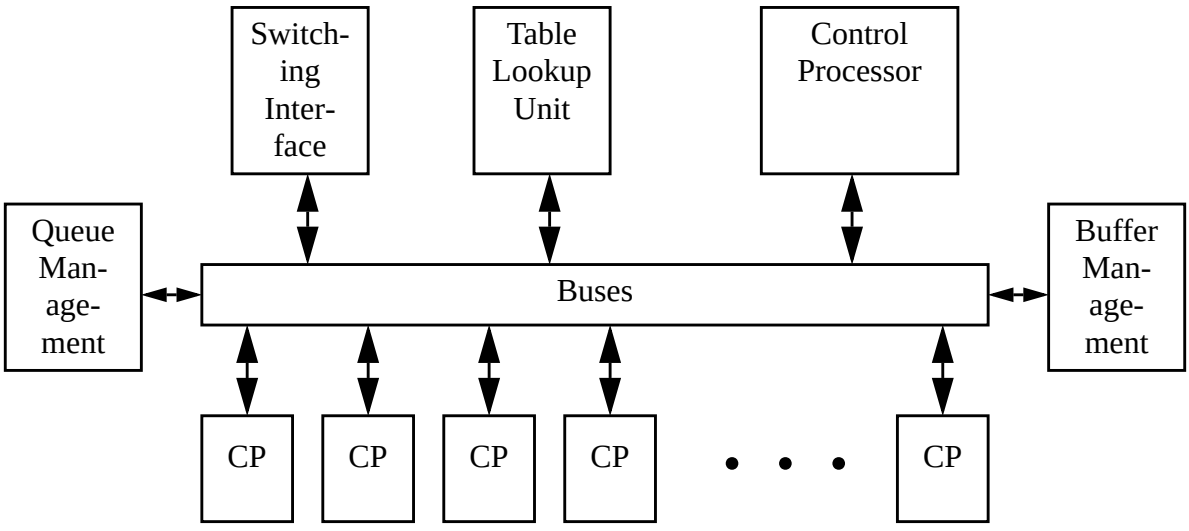


Figure 2.1: C5NP Architecture Overview

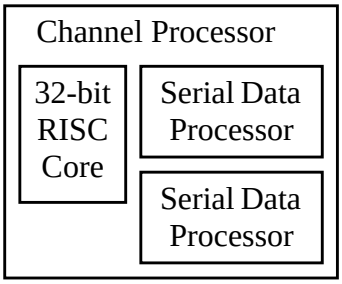


Figure 2.2: C5NP Channel Processor Organization

2.2.2 Intel IXP 1200 Network Processor

The IXP 1200 can just as the C5NP act as the heart in a small router, but requires some more surrounding circuits.

The IXP 1200 consists of a StrongARM RISC core and 6 microengines. The microengines are application specific instruction set processors and just like the SDPs, they run special microcode. [2.7]

2.2.3 Agere FPP, RSP and ASI

The Agere three chip solution is aimed for line cards in a big router. The three chips can perform classification, policing, traffic management, quality of service, traffic shaping and packet modification functions. The fast pattern processor (FPP) and the routing switch processor (RSP) are in the fast path of the router, but the agere system interface (ASI) is performing tasks both in the fast path and in the slow path, see figure 2.3. Together the three chips manage all fast path tasks for a line card, but for the slow path there is a need for support, typically a RISC processor. The ASI only provides an interface for such a processor.

Internally the fast pattern processor (FPP) is built up of several blocks, that work in parallel. The input framer splits the data stream into 64 byte blocks. The rest of the processor works on these blocks. Totally the FPP can be considered as an ASIP with supporting fixed function circuitry on-chip. The RSP is internally built up of three very long instruction word processors (VLIW) which are ASIPs. Each of these three is dedicated to its function, traffic management, traffic shaping, and stream editing respectively. There is also some fixed function circuitry on-chip in the RSP. The ASI is an ASIC for configuration, interfacing to a microprocessor bus and for some support to the FPP. [2.8]

2.2.4 Agere NP10 and TM10

Agere NP10 and TM10 are the successors to the FPP, RSP, and ASI. They are intended to handle the full functionality on a line card.

The internal architecture is not revealed, but they are programmable to some extent so most probably they are again a combination of ASIPs and fixed function circuitry on-chip. [2.8]

2.2.5 AMCC nP7120

The nP7120 is a network processor for a line card in a big router. It handles packet processing, including packet classification and manipulation. It is extendable since it has interfaces for external units such as search engines.

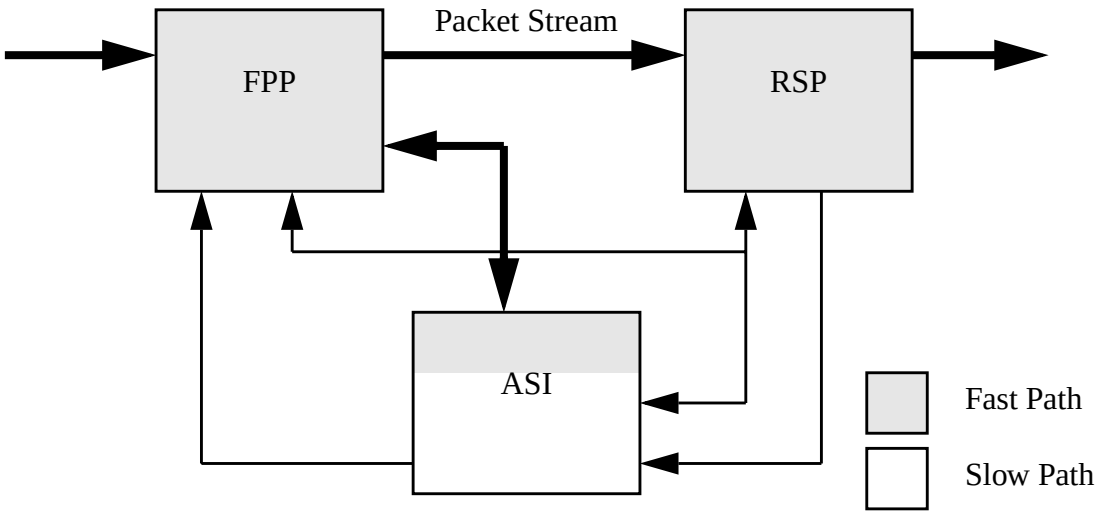


Figure 2.3: Agere FPP, RSP and ASI Overview

The nP7120 consists of two processor cores which are optimized for network processing and therefore resides in the RISC with optimized instruction set category. Accompanying these processor cores are also some on chip fixed function blocks. [2.9]

2.2.6 AMCC nP7250

The nP7250 is just like the nP7120 a network processor for a line card in a big router. The processing it can handle is also similar, but nP7250 can cope with higher data rates than nP7120.

Internally the nP7250 consists of two network processor cores, a packet transformation engine and a policy engine. It has not been revealed of which type the internal units are. [2.9]

2.2.7 AMCC nP3400

The nP3400 is a single chip router. It can handle layer 3-7 policy based switching and two nP3400 can be combined to allow more ports. One nP3400 handles 24 Fast Ethernet ports and 2 Gigabit Ethernet ports.

Internally the packet processing is handled by multiple processor cores and other modules for e.g. switching, queue management, scheduling, statistics gathering, and classification. The processor cores are most likely of the same type as the ones in nP7120, i.e. in the category of RISC with optimized instruction set. [2.9]

2.2.8 EZChip NP-1

NP-1 is a network processor, which is aimed for line cards in a big router. It can be used for e.g. flow-based traffic policing and URL switching. Thus it performs protocol processing up to layer 7.

NP-1 is built up of an array of processor cores. Each of the processors is optimized for it specific task, there are processors for parsing, searching, resolving and modification. Each is programmable and has an application specific instruction set. Beside these four processor cores there is also some fixed function circuitry on chip. [2.10]

2.2.9 IBM PowerNP 4GS3

The PowerNP can function as a single chip router, or be part of a bigger router. Two PowerNPs can be directly connected to form a medium sized router or the PowerNP can be placed on a line card in a big router. The PowerNP can handle layer 2-4 protocol processing. The PowerNP handles both fast path processing and slow path processing.

Internally the PowerNP is constructed as an embedded processor complex (EPC) surrounded by fixed function medium access control units (MACs) and queue management and scheduling units. The EPC consists of one embedded PowerPC processor and 8 dyadic protocol processor units (DPPUs). Each DPPU has two core language processors (CLPs) and 9 coprocessors. The CLPs execute picocode and can thereby be categorized as application specific instruction set processors. The coprocessors accelerate special functionality, such as checksum calculation and tree search. The coprocessors are of fixed function circuitry type and do not execute any sequential code. [2.11]

2.2.10 Silicon Access iFlow

The iFlow family of network processors has many members, which can be combined into various combinations and can thus fit in many different places in the network equipment. Most appealing, however, is to combine some of the chips to build a line card for a big router.

Information is only available on the iFlow address processor (iAP), which manages the routing table and lookup requests. This is done in a pipelined fashion and the iAP belongs to the fixed function ASIC category, since it does not execute any sequential program, nor is it reconfigurable enough to be a reconfigurable hardware architecture. Silicon Access calls it a coprocessor. On chip there is also an arithmetic logic unit (ALU), which operates in parallel with the pipelined lookup. The ALU can be used for field modification or statistics gathering. [2.12]

2.2.11 SwitchCore CXE-16

The CXE-16 is a single-chip router for layer 2-4 switching. It provides packet decoding, queue and buffer management, classification, and prioritization. Even the packet buffer is on chip.

The CXE-16 is built up of fixed function blocks, with limited reconfigurability. Thus it belongs to the fixed function ASIC category. [2.13]

2.2.12 Coresma 6001

The Coresma 6001 is a host-based universal protocol processor. It can handle many protocols on layer 2 and layer 3. It is intended for desktop computers and supports a PCI interface in order to be able to connect to the main computer bus.

The Coresma 6001 consists of 4 RISC processors, two work in the transmit path and the other two in the receive path, see figure 2.4. There are also fixed function units to support e.g. CRC (cyclic redundancy check) and DES (data encryption standard). There is no information on any optimization of the RISC cores, so the Coresma 6001 is considered to belong to the general-purpose processor category, although the interconnection and the special units are dedicated for the applications. [2.14]

2.2.13 PMC-Sierra PM7388

The PMC-Sierra PM7388 is a frame engine a data link manager and handles protocols on layer 2. It is typically situated on a line card in a big router. It can handle Frame relay, HDLC, and PPP for several connections.

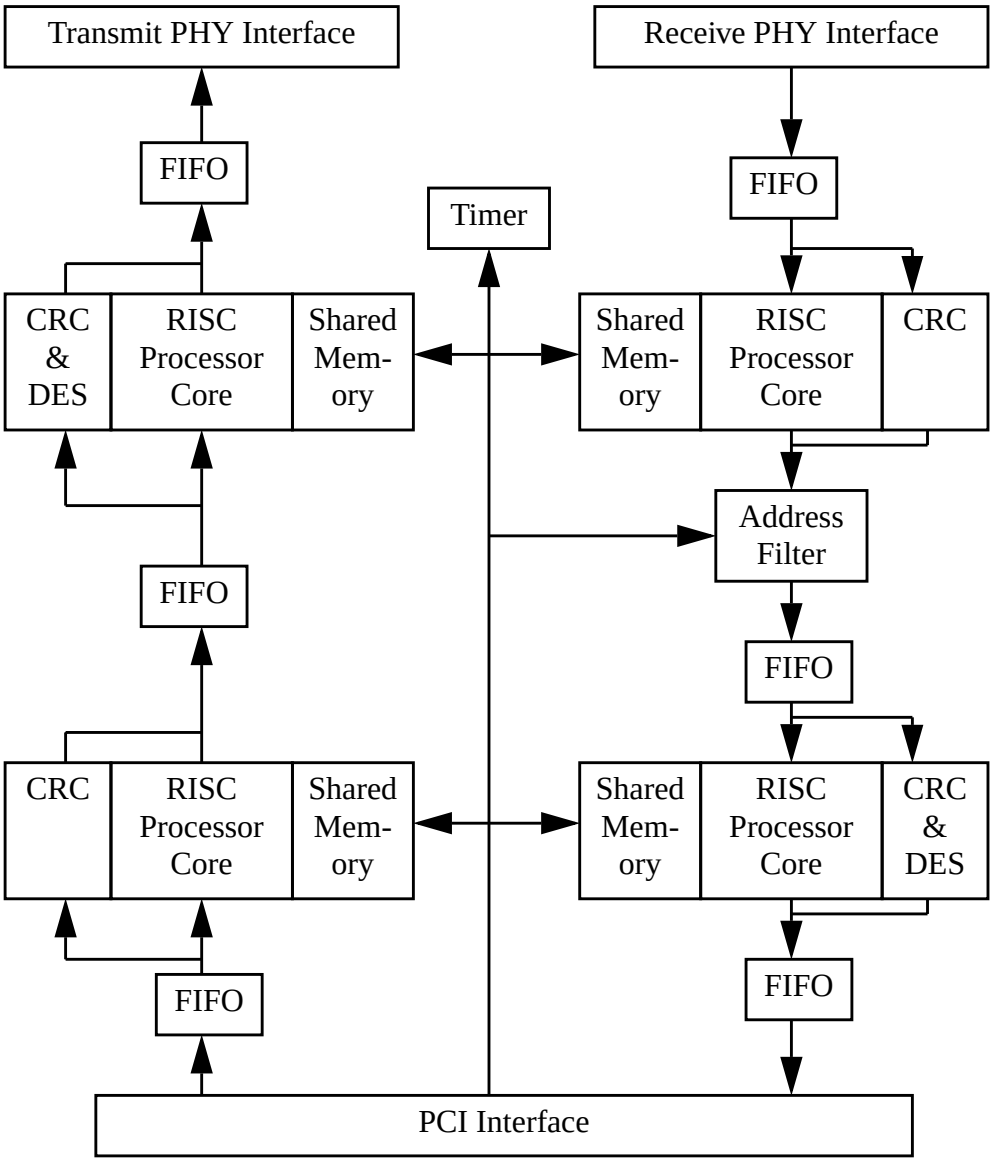


Figure 2.4: Coresma 6001 Overview

The PM7388 can handle only three different protocols and does not have any programming possibilities so it belongs to the category of fixed function ASICs. Internally a fixed interconnection of functional blocks manages the data flow. [2.15]

2.2.14 PMC-Sierra PM2329 ClassiPI

The ClassiPI chip solves the packet classification task and typically resides on a line card in a big router. It is intended to work in cooperation with a network processor and is capable of handling rules on layer 2, 3, 4, and 7. One ClassiPI chip can hold 16K rules, but can be cascaded with up to 7 other chips in order to increase the capacity.

The main component of the ClassiPI is the classification engine (CE), which performs the actual rule matching. Other important blocks are the field extraction engine (FEE), which performs fixed and configurable header field extraction, and the results FIFO, which stores results until the cooperating network processor takes care of them. There is also a control/sequencer, which controls the other units. The control/sequencer is running a sequential program, but the other units are

fixed function blocks, with only some configuration possibility. The ClassiPI must therefore be considered as a mixture of an ASIP and a fixed function ASIC. [2.16]

2.2.15 Broadcom BCM 5680

The BCM 5680 is a single chip router. It can handle Ethernet forwarding and IP switching and also filtering tasks up to layer 7. It works with Ethernet interfaces up to 1000 Mb/s. Totally 8 ports are supported.

The major blocks, except for the MACs, are the packet buffer, the non-blockable switch, and the ARL tables. The filter engine is programmable and thus the BCM 5680 consists of both an ASIP and fixed function blocks. [2.17]

2.2.16 Broadcom BCM 5632

The BCM 5632 is a single chip router, capable of handling Ethernet switching. It supports one 10 Gigabit Ethernet port and twelve 1 Gigabit Ethernet ports.

The BCM 5632 is built up from fixed function blocks for buffer management, queue handling, port managing and payload buffering. It is a typical fixed function ASIC with a clear application specification. [2.18]

2.2.17 Broadcom BCM 1250

The BCM 1250 is a multi-processor aimed for many kinds of routers and switches. It can reside on a line card in a big router, but also be the heart of a small router. It can perform deep packet lookup and layer 4-7 processing.

The BCM 1250 consists of two 64-bit MIPS processor cores, some high-speed interfaces, memories and buses to connect it all together. This makes the BCM 1250 reside in the general purpose computer category. [2.19]

2.2.18 Solidum Systems PAX 1100

The PAX 1100 is a classification engine, which works in cooperation with another network processor. It is typically situated on a line card in a big router. The functionality includes classification on layers up to layer 7.

The PAX 1100 is built around programmable state machines. This allows programming of the classification details, but the general functionality is fixed so the PAX 1100 belongs to the reconfigurable hardware architecture category. [2.20]

2.2.19 Sitera (Vitesse) IQ 2000

The IQ 2000 network processor supports deep packet processing and could be part of a small router or could also be situated on a line card in a big router. It can handle processing on all protocol layers up to the applications.

Internally 4 processor cores work together with coprocessors to manage the functionality. There is also a special interconnect system on chip. The processor cores are of application specific instruction set type, but the coprocessors handle fixed functions and thus resides in the category of fixed function ASICs. [2.21]

2.2.20 Xelerated X40

The X40 Packet Processor is aimed for line cards in big routers. It can handle classification, forwarding, framing and deframing.

The X40 is constructed as a programmable pipeline. The core block is a packet instruction set computer (PISC) which belongs to the group of ASIPs. One or more PISCs in combination with a classification block form a classify-action unit. The whole architecture consists of several of these classify-action units, which each perform one task in the packet processing pipeline. [2.22]

2.2.21 Lantronix DSTni

The Lantronix DSTni is an architecture for terminals. It is typically used for terminals which have less processing power than desktop or laptop computers. It can handle Ethernet and TCP/IP processing.

The DSTni is built up from several cores on chip, for the network connection a fixed function Ethernet MAC and an enhanced 80186 processor are essential. All layer 3 and 4 processing is performed in the 80186 microprocessor core. [2.23]

2.2.22 Clearwater Networks CNP810SP

CNP810SP is a network services processor that can be used on line cards for exception processing, i.e. handling parts of the slow path. It can also support the control processor in a big router and then be situated on a service card.

The CNP810SP is constructed as a simultaneous multi-thread core which runs MIPS code. 8 instruction queues can execute in parallel on 10 functional units with a separate set of registers. The CNP810SP can also execute a small number of additional instructions, so called network processing extensions. Thus it can be categorized as a RISC with optimized instruction set from the programming point of view, but the architecture is different from a traditional RISC core. [2.24]

2.2.23 ClearSpeed Platform

The ClearSpeed platform is intended for line cards in big routers. It can handle advanced classification and forwarding and is therefore part of the fast path on the line card.

The internal structure consists of a number of multi-threaded array processors (MTAP), which are connected through an on-chip network called ClearConnect. Each MTAP can have up to 8 processing element (PE) blocks and each PE block contains up to 256 PEs. There is also a central sequence controller and a generic unit controller in each MTAP. The MTAP is a single instruction multiple data (SIMD) machine since all PEs get the same instruction sequence. The PE is an 8-bit ALU with register file, local memory, and interconnect interfaces. It is hard to categorize the ClearSpeed platform to be any of the types of architectures discussed in the previous section. It is clearly a highly parallel computing engine, where not much is revealed about the internal PE instruction set. [2.25]

2.2.24 Lexra LX8000

The Lexra network processor core LX8000 is aimed at line cards in big routers. It can also be used for load balancing web server switches and voice over IP (VoIP) gateways. So it can handle processing up to layer 7.

Internally the LX8000 is a RISC core that executes the MIPS instruction set, but also has specific instructions for supporting multithreading and bit field manipulation. It clearly fits in the category of RISC with optimized instruction set. [2.26]

2.2.25 Entridia Forte

Forte is a network processor for packet classification and forwarding. It fits on a line card in a big router. It can handle policing, traffic shaping and packet modification.

The Forte chip is a configurable hardware architecture with fixed functionality. The configurability is limited to choosing which functionality to include in the packet processing. Interfaces are present for external packet storage as well as for external forwarding table and forwarding table management. The circuit belongs to the category of configurable architectures and provides a coarse grain configurability. [2.27]

2.2.26 STM Network Processor Development

The STM network processors are based on Octagon, an on chip communication network, that allows multiple data transfers concurrently. The actual processing is performed in nano processors and fixed function blocks. [2.34]

2.2.27 Infineon and Dresden University

This processor is developed for various network applications. It can be used both in terminals and in routers. Focus is on layers 2-4, where header fields do not match word boundaries. The processor has been used for xDSL protocol processing as an example.

The architecture is based on an ARM RISC core, but has significant modifications. Among these are the data movement instructions and the bit position addressing the most important. Bit position addressing means that an instruction specifies not only which registers to operate at, but also the bit width and the starting positions in each register. The processor is still a RISC with optimized instruction set. [2.28]

2.2.28 Tampere University TTA for Protocol Processing

The transport triggered architecture (TTA) presents a totally new view on how to design a processor. Originally the concept was developed for other types of applications than network processing. The main concept is to have only one instruction, which is move. The data can then be transported to the functional units (FUs) of the processor and as soon as input data is available at an FU, it will process it and produce an output. In this way, the program controls data movement rather than data processing. The FUs can be fixed function blocks or to some extent configurable.

The TTA was used for network processing in the TACO protocol processor, which was exemplified with an ATM processor, see figure 2.5. The key design tasks are to decide on which FUs to include in the processor and how to design the interconnections. If a bus system is used it must be decided how many buses there are and also if all FUs have to be connected to all buses. Since the sequential program controls the data movement and only indirectly the processing, this processor is considered as a reconfigurable hardware architecture. [2.29]

2.2.29 UC Berkeley Reconfigurable Platform for Wireless

This project aims at designing protocol processing hardware for wireless terminals. The main goal is to achieve very low power consumption. The functionality should include layers 1-4 for protocols such as 802.11, Bluetooth, and HomeRF. This hardware architecture should also be possible to use in sensor networks, where the power consumption requirements are even harder.

The way to achieve low power is to use reconfigurable hardware. The observation is made the FPGAs are good for data path operations and PLDs are better for control intensive operations, such as FSM implementation. Therefore all processing tasks are categorized as either data intensive or control intensive and then implemented in the corresponding part of the mixed FPGA and PLD reconfigurable platform. The architecture undoubtedly belongs to the reconfigurable hardware architectures category. [2.30]

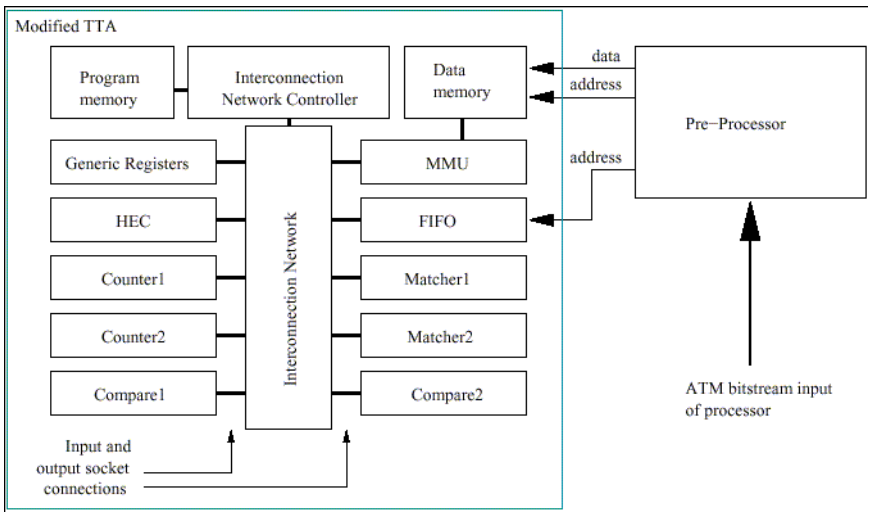


Figure 2.5: TACO Protocol Processor in ATM configuration (courtesy S. Virtanen)

2.2.30 EU Protocol Processor Project

The protocol processor project (PRO^3) aims at developing a protocol processor for high speed applications. It could be used in terminals as well as in routers. In a terminal it would handle protocols on layer 3 and 4 as well as an interface to the application layer.

The PRO^3 consists of two major parts, an embedded RISC core and a reconfigurable pipelined module (RPM), see figure 2.6. The RPM is intended to handle the small number of functions, that corresponds to the big portion of the total workload. Other control related tasks are to be handled by the RISC core. The RPM can be implemented as fixed function blocks, with some configurability or in an FPGA. This architecture obviously resides in the reprogrammable hardware architecture group. [2.31]

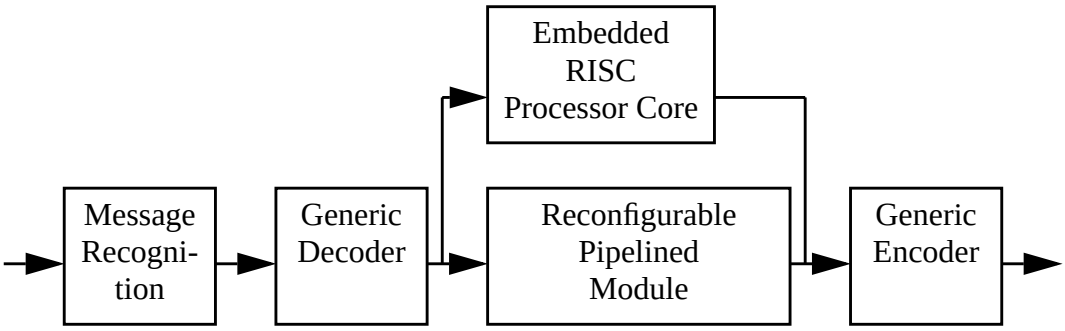


Figure 2.6: PRO^3 Functional Architecture

2.2.31 KTH Protocol Processor

In this project the focus is on the control path of a protocol processor. The intended applications include IP forwarding, which requires many if-then-else statements when implemented purely in software. Thus the processor should support the fast execution of branch instructions.

The key architectural feature is the instruction memory, which contains special fields associated with each instruction for fast branching. The traditional part of the instruction controls an ALU and a register file. Thus it can be categorized as a RISC with optimized instruction set, where the added instruction part is executed in parallel with the original instruction in a slave processor unit. [2.32]

2.2.32 UCLA Packet Decoder

The UCLA project tries to develop a new design methodology for domain specific processors and uses network processors as the first application. The packet processor that has been developed so far handles packet decoding on layer 2, 3, and 4. It is intended as a part of a one chip router, but could also be used on a line card in a big router or in a terminal.

The packet decoder consists of three data paths and one control path. There is one data path on each protocol layer, but just one central controller. The packet decoder has been developed by instruction profiling and the construction of a new instruction set. Therefore it is an ASIP. It is however different from traditional processors since it operates on a data stream instead of on data stored in memory. [2.33]

2.3 Network Processor Survey Conclusion

The conclusion of the survey is that many different architectures have been introduced. They all try to solve different tasks within the protocol processing domain. Some architectures are aimed at high-speed operation of a small number of tasks. Other network processors aim at full programmability and thereby should be able to handle all possible processing tasks.

Most efforts have focused on high-speed routers, for which performance is much more important than low-power operation. Most network processors are aiming for the fast path in the routers, but some cover parts of the slow path as well. There is a general disagreement on how much functionality to include on one chip and how much freedom to leave for the system designer.

In the next chapter I introduce yet another architecture, which has been developed in our research team. It belongs to the reconfigurable architecture category and of the architectures described here, it has most in common with the PRO^3 architecture. In the end of the next chapter our architecture is compared to the programmable architectures from several different points of view.

References

[2.1] B. Hubbs, “A Survey of Highly Integrated Ethernet DataComm”, IEEE Aerospace Conference 1998, vol. 4, pp. 489-498

[2.2] H. Ichikawa, H. Yamada, T. Akaike, S. Kanno, M. Aoki, “Protocol Control VLSI for Broadband Packet Communications”, Globecom ‘88, pp. 1494-1498

[2.3] A. S. Krishnakumar, W. C. Fischer, K. Sabnani, “The Programmable Protocol VLSI Engine (PROVE)”, Supercomm/ICC ‘92, pp. 459-465

[2.4] R. Ernst, “Embedded System Architectures”, in A. A. Jerraya, J. Mermet, “System-Level Synthesis”, Kluwer 1999, pp. 1-43

[2.5] P. Schaumont, I. Verbauwhede, K. Keutzer, M. Sarrafzadeh, “A Quick Safari through the Reconfiguration Jungle”, DAC 2001, pp. 172-177

[2.6] http://www.motorola.com/SPS/RISC/smartnetworks/products/netproc/C5NP_M994862703126.htm

[2.7] http://developer.intel.com/design/network/products/npfamily/ixp1200.htm

[2.8] http://www.agere.com/netcom/nps/docs.html

[2.9] http://www.mmcnetworks.com/Solutions/nP7120.asp

[2.10] http://www.ezchip.com/html/in_prod.html

[2.11] http://www-3.ibm.com/chips/techlib/techlib.nsf/products/IBM_PowerNP_NP4GS3

[2.12] http://www.siliconaccess.com/products/

[2.13] http://www.switchcore.com/frames.php?left=/left_menu.php&hilite=9&menu=2&content=/products/cxe-16.shtml

[2.14] http://www.coresma.com/prod_6001.html

[2.15] http://www.pmc-sierra.com/products/details/pm7388/

[2.16] S. Iyer, R. R. Kompella, A. Shelat, “ClassiPI: An Architecture for Fast and Flexible Packet Classification”, IEEE Network, March/April 2001, pp. 33-41

[2.17] http://www.broadcom.com/products/5680.html

[2.18] http://www.broadcom.com/products/5632.html

[2.19] http://www.broadcom.com/products/1250.html

[2.20] http://www.solidum.com/products/1100.cfm

[2.21] http://206.216.176.9/products/categories.cfm?family_id=5&category_id=16

[2.22] http://www.xelerated.com/page.asp?page=products

[2.23] http://www.lantronix.com/products/eds/dstni/index.html

[2.24] http://clearwaternetworks.com/products.html

[2.25] http://www.clearspeed.com/pdf/epf_presentation.pdf

[2.26] http://www.lexra.com/products.html

[2.27] http://www.entridia.com/products/forte.html

[2.28] X. Nie, L. Gazsi, F. Engel, G. Fettweis, “A New Network Processor Architecture for High-Speed Communications”, SIPS’99, pp. 548-557

[2.29] S. Virtanen, J. Lilius, T. Westerlund, “A processor architecture for the TACO protocol processor”, IEEE NORCHIP 2000, pp. 204-211

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

[2.31] 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

[2.32] Y. Ma, A. Jantsch, H. Tenhunen, “A programmable protocol processor architecture for high speed internet protocol processing”, IEEE NORCHIP 2000, pp. 212-216

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

[2.34] P. G. Paulin, F. Karim, P. Bromley, “Network Processors: A Perspective on Market Requirements, Processor Architectures and Embedded S/W Tools”, IEEE DATE 2001, pp. 420-426

3

Novel Architecture

3.1 Novel Network Processor Architecture Introduction

In this chapter, the main contribution of this thesis is introduced. It is the result of a research project from my research team, which started in 1999. I and my colleagues have developed a novel architecture for protocol processing. In this chapter, an overview of the functionality and the architecture is given. Then, in the next 3 chapters material published at conferences is presented and finally the last chapter consists of a submission to a journal.

We call our architecture the general-purpose protocol processor (GPPP). The architecture focuses on terminal protocol processing. Especially it deals with processing of tasks that only concern reception of one packet or frame. For tasks that involve several packets, the GPPP only provides supporting functions, such as extraction of information and information preprocessing.

The GPPP consists of configurable units, called functional pages (FPs). It is a data stream processor, i.e. it operates on a stream of data rather than on data stored in a memory. There is no sequential program executing in the GPPP, instead the control is hierarchical, and supported by a micro processor [3.3]. An overview can be seen in figure [3.1].

3.2 Functional Coverage

The GPPP is an architecture template. Starting from this general template many different instantiations can be made. They all have the same general structure, but may differ regarding which FPs are included. The thing that is common to all of them, regarding the functional coverage is that they are handling tasks, which can be completed on one single frame or packet. For protocol processing tasks that include a sequence of packets or both receiving and sending packets, the

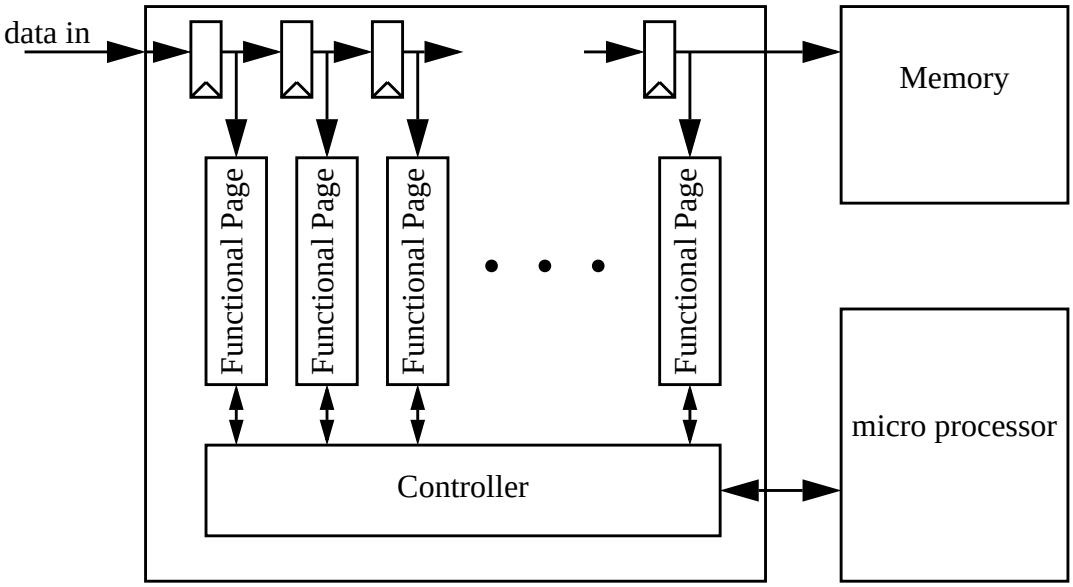


Figure 3.1: Overview of the GPPP

GPPP needs support from a unit that handles the interpacket behavior. The GPPP however can be used as an assistant.

The GPPP is mainly concerned with the reception of packets, but a similar architecture could be used for sending packets. The exact requirements on such a processor have not been investigated.

As described in the first chapter, protocols are layered and each layer adds information to the original packet. Normally this information is added in the form of a header, preceding the payload data. Eventually this leads to a packet looking like the one in figure 3.2. This is the kind of packet that the GPPP will receive and decode. The layer 2 header will be received first, then the layer 3 header, and so on. The GPPP can perform processing on all layers simultaneously, so called integrated layer processing.

Payload	Layer 5-7 header	Layer 4 header	Layer 3 header	Layer 2 header

Figure 3.2: A general packet that is received by the GPPP

The GPPP works in parallel with the host processor of the terminal. In traditional terminals such as desktop and laptop computers, the protocol processing of layers 3 and above is handled in the host processor. The GPPP will reduce the workload on the host processor by managing the protocol processing for received packets.

3.3 Mapping of the Functionality to the Architecture

For each protocol layer some tasks need to be executed. Dependent on how complex the tasks are one or more FPs are used for each task, see figure 3.3. Each FP is constructed as a coarse-

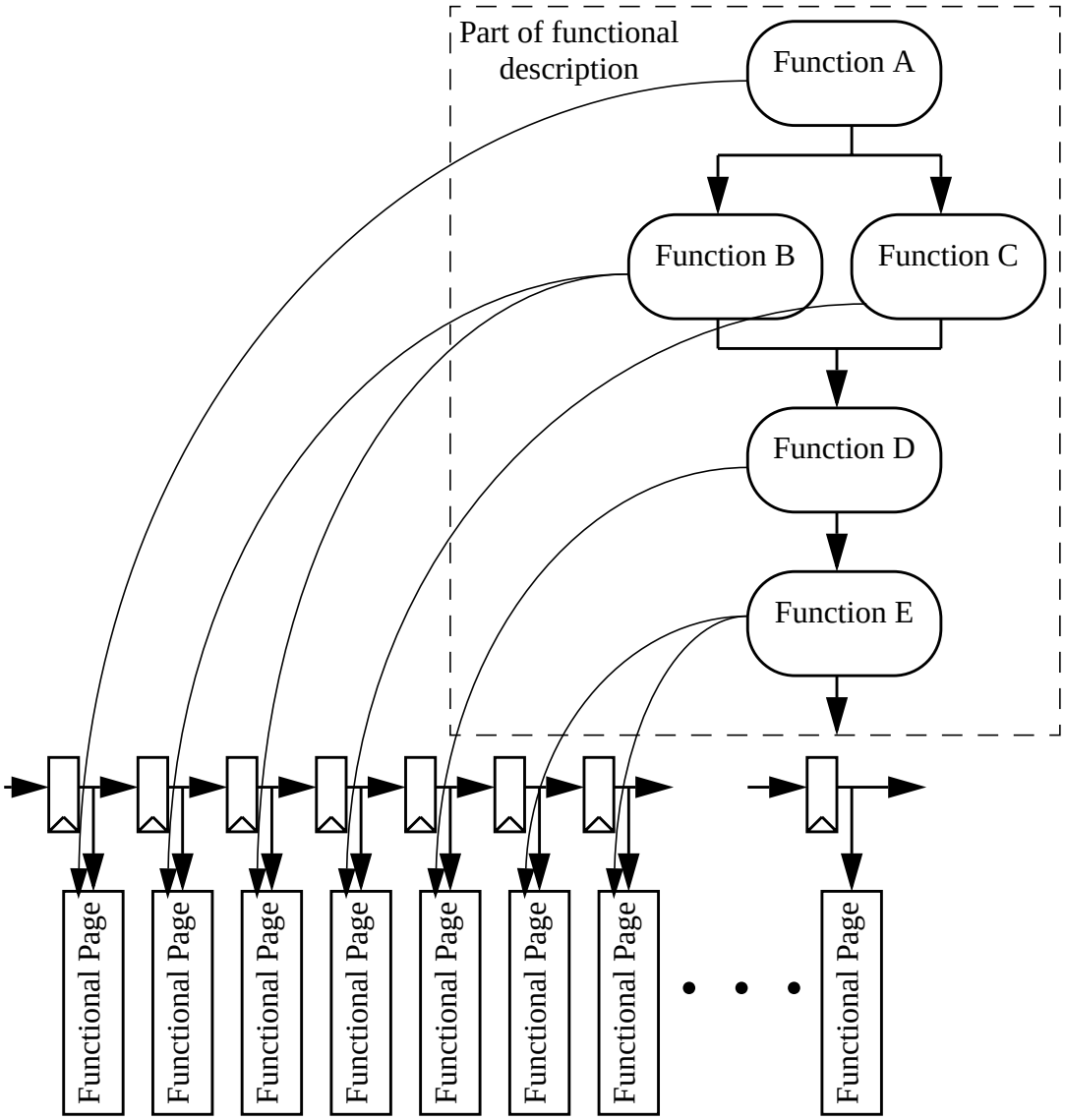


Figure 3.3: Example of mapping of the functionality

grained reconfigurable unit. This means that the reconfiguration is possible on word level instead of on bit level as in e.g. FPGAs. The functionality of a FP is thus predefined to a certain extent, all FPs are different and specialized to a group of tasks.

Each FP is self-contained, i.e. it handles its task without the need of any support. It solely requires a start signal and once it has completed its task, which may take several clock cycles it reports the result.

The packet is received in words, which may be of arbitrary length. For the instances of the GPPP that have been implemented so far, the word length has always been 32 bits. The words go through a pipeline in their way through the GPPP. After each pipeline register the data may be distributed to one or more FPs. In this way, each FP only needs some of the words for the completion of its task. The important aspect here is to notice that there will be several clock cycles when the

FP does not receive any new data. These clock cycles are used to complete the task or if not needed the FP goes into sleep mode.

Together the FPs execute all tasks that are required for the processing when receiving a packet. They need however to be coordinated. This is handled by the controller. The controller takes care of the firing of the FPs and makes sure that the scheduling is correct. It also collects and combines the results of the FPs in order to decide how to proceed with the packet. This decision is handed over to the supporting micro controller, which can take care of communication with the application, handle interpacket tasks and memory management.

3.4 GPPP Configuration

The configuration of the GPPP is done at three levels [3.2]:

- FP selection
- FP configuration
- FP firing

Each level has different possibilities and restrictions. They will each be described in the next subsections.

3.4.1 FP Selection

There are a number of FPs designed. More can be added when a need appears. Since each FP has a configurable functionality it can be used in several protocols, which include similar tasks. The combination of the tasks for a certain protocol stack requires a certain minimum set of FPs to be able to execute all tasks. For an instance of the GPPP to be able to handle this protocol stack, at least this set of FPs must be included in the processor.

By choosing such a minimum set the processor has limited flexibility. This can easily be improved by adding more FPs. For each protocol stack that should be covered by an instance of the GPPP the minimum set of FPs is extracted. Then the union of all these sets is derived to form the required FPs for the processor. By adding FPs in this way a well described flexibility has been introduced to the instance of the GPPP.

3.4.2 FP Configuration

Each FP has internal configuration possibilities. Some FPs have less need than other for flexibility internally. For those the configuration may include some control signals for a data path. Other FPs require more flexibility, which can be implemented as a small micro controller which runs micro code inside the FP. In this case the micro code is the configuration data.

A concrete example of an FP and its configurability is a CRC (cyclic redundancy check) calculation unit, which has a data intensive task of calculating the CRC of all data in the packet. The configurability resides in the CRC polynomial and possibly the polynomial length.

The FP flexibility is designed into the FP before it is known exactly which protocols it will be used for. Thereby the flexibility is not driven by the application, but rather by the designer, who has to make the choice of how much flexibility to include in an FP. This leads to flexibility which may not be required by the final processor. However, it could also improve the lifetime of a processor, by providing the possibility to change protocols in a not anticipated way.

3.4.3 FP Firing

FP firing is concerned by the fact, that some protocol headers are not always of the same size. This leads to that header fields of higher layer headers may be received at different latencies from

the start of the packet for consecutive packets in the same network. Therefore, the FPs concerned with those header fields must be fired at different times for different packets.

The controller is the unit that conducts the whole operation of the GPPP and fires the FPs. To be able to know when to fire which FP all FPs that complete their tasks or part of their tasks report this to the controller. With the use of this information, the controller can fire each FP at the correct instant of time.

The FP firing is the only GPPP configuration which is data dependent, the FP selection and the FP configuration are made before execution.

The controller is configured to do this firing depending on which protocol stack to process. The configuration of the controller as well as of the FPs is managed by the supporting micro processor.

3.5 GPPP Benefits

The GPPP is a novel architecture for protocol processing and has some big advantages over traditional architectures, but also some drawbacks. The drawbacks could be summarized by one word, support. The GPPP requires support by for example a micro processor, which has to handle the tasks that the GPPP is not capable of.

The benefits are more, but different depending on what is compared to. If the alternative is an ASIC the advantage of the GPPP is mainly flexibility. It is more adequate to compare to a traditional processor, running a sequential program, since in the domain of protocol processing of reception of packets the GPPP offers similar flexibility.

3.5.1 Performance

The performance of the GPPP is definitely higher, since it operates at line speed and each clock cycle one new word of packet data is taken care of. The traditional processor loses a lot of valuable clock cycles just when fetching data from the memory.

The combination of the GPPP with the host processor results in less computing load on the host processor. A detailed example of this is given in section 3.8.

3.5.2 Power Consumption

The GPPP has a very small control overhead, almost all activity in the processor is useful for the operation. FPs that are not needed at the moment can be kept in sleep mode, which further reduces the power consumption. The data is never stored in memory until the processing is finished and the payload can be stored in a memory area allocated by the correct application.

In traditional processors, the whole packet is stored in memory and read and written to memory more than once after that. The control overhead is huge in a traditional processor, consisting of fetching and decoding instructions. There is also a computational overhead when the operation uses a shorter word length than the processor.

3.5.3 Functional Verification

The functional verification is becoming an increasingly big part of the design of digital systems. Normally the verification of the hardware is considered tough, but verifying software of a complex system is also tedious. Especially if one processor executes several functions, which are timing critical. Then the interrupt response times get very important and hard to foresee.

The GPPP simplifies verification of the hardware by splitting it into well partitioned FPs, which can be verified one at a time. The infrastructure which connects the FPs to the data pipeline and the controller can also be verified stand-alone. The configuration vectors for the FPs are of a lot

smaller complexity than the program for executing the same task in software on a traditional processor.

3.5.4 Silicon Area

Although the silicon area of a circuit is no longer of the same importance as some years ago, it is still expensive to produce big chips. The GPPP is intended to be integrated in a system-on-chip, which may contain several other processors, memory, analog circuits, etc. The yield when manufacturing such large chips is dependent of the chip area, so it is important to reduce the size of all components.

The GPPP achieves a small silicon area since the overhead is very low. Almost all transistors contribute to the actual computations that are to be made. In a traditional processor the instruction decoding and the cache memory management take up a big part of the area.

3.5.5 System Perspective

When considering a terminal as the system, the GPPP will improve the overall system performance. This is done by moving the protocol processing functionality from the host processor to the GPPP. Thereby the workload on the host processor is reduced and it can more efficiently serve the applications, or for power sensitive terminals the host processor can run at a slower frequency and thus save power. The power consumption in the GPPP is much less than the saved power in the host processor due to the less overhead, as described earlier.

The GPPP could also be used as the packet decoder in a router. In a small router, which tries to integrate all functionality onto one chip the small area as well as the low power consumption is important. In a big router which handles a few links on a line card, the most important feature is the performance, which is kept even at complex processing. This is the case since more complex processing in the GPPP means more FPs, which all work in parallel. In a processor that runs sequential code, more complex processing means more instructions and thus longer execution time per packet. This means that more complex processing reduces the performance for a traditional processor, but not for the GPPP.

3.6 Functional Page Specification

As mentioned earlier, an FP may have any internal micro architecture. It could be a small micro controller, running some short sequence of micro code instructions, a data path with configurable control signals, a configurable finite state machine (FSM) or a combination of the three.

For different tasks different micro architectures are most suitable. It is up to the designer to chose the appropriate micro architecture when implementing the FP. The only thing that is somewhat fixed are the interfaces. There are three interfaces to each FP, see figure 3.4.

3.6.1 Interface to the Data Pipeline

The interface to the data pipeline is simple. The data pipeline lets the incoming data pass through the pipe word by word. The FP has to have an input that can receive the data from the pipeline. Normally this input should be of the same width as the word length of the pipeline. It could, however, happen that an FP only uses a header field, that is part of a word and then the input could consist of only those bits. This reduces the flexibility, since if the protocol changes so that the header field will reside in another part of the data word, the interface must be redesigned.

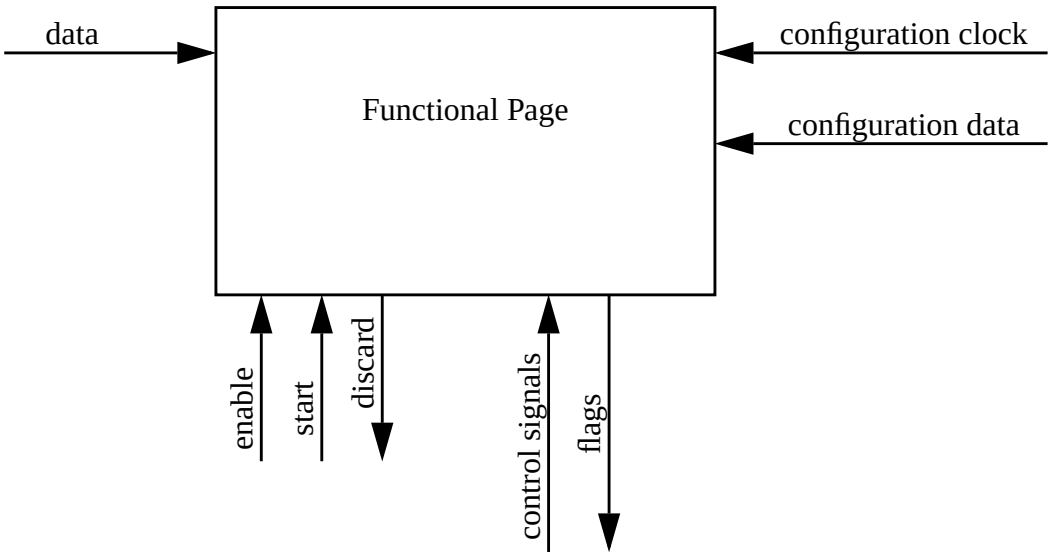


Figure 3.4: Functional Page Interfaces

3.6.2 Interface to the Controller

The interface to the controller consists of two parts. One part is fixed and has to be implemented by all FPs. The other part is variable and may be different for every FP. The fixed part consists of a start input, which is the signal for firing the FP, an enable input, which can put the FP in sleep mode by going inactive, and a discard output, which is a signal from the FP that tells the controller to discard the packet.

The variable part consists of an arbitrary number of control signals, which are inputs to the FP and likewise an arbitrary number of flags, which are outputs form the FP. The control signals and flags support the FP operation and may be used for inter FP communication since the controller can forward the flags from one FP as control signals to another FP.

3.6.3 Interface to the Configuration

Before the normal operation of the GPPP can start it must be configured. The configuration interface is fixed for all FPs. It consists of two inputs, the configuration clock and the configuration data. The scan chain of flip-flops that store the configuration vector is clocked by the configuration clock and when the GPPP is operational the configuration clock is shut off. Thereby the configuration bits are not changing when the GPPP is operating.

3.7 Functional Page Examples

I have implemented a VHDL model of the GPPP framework, including the data pipeline and the controller. I have also implemented one instance of the GPPP, that handles Ethernet and IP processing and parts of the UDP and TCP protocols.

In this section somewhat detailed descriptions of three functional pages are given. First the CRC functional page, then the functional page for TCP and UDP checksum calculation and finally a functional page for IP destination address checking.

3.7.1 CRC Functional Page

The cyclic redundancy check (CRC) calculation is the performance limiting factor for the GPPP for Ethernet, IP and TCP. Therefore it is important to have an efficient implementation of the CRC. To achieve this I have studied several previous CRC implementations, some are listed in Paper 3. Then the best realization for hardware implementation was selected, which is based directly on Galois fields.

The CRC specification uses a bit sequential implementation as its reference. It is very hard to improve the performance of the CRC without using parallel techniques. The general structure of a parallel implementation can be seen in figure 3.5. The register and the output are of width 32 bits

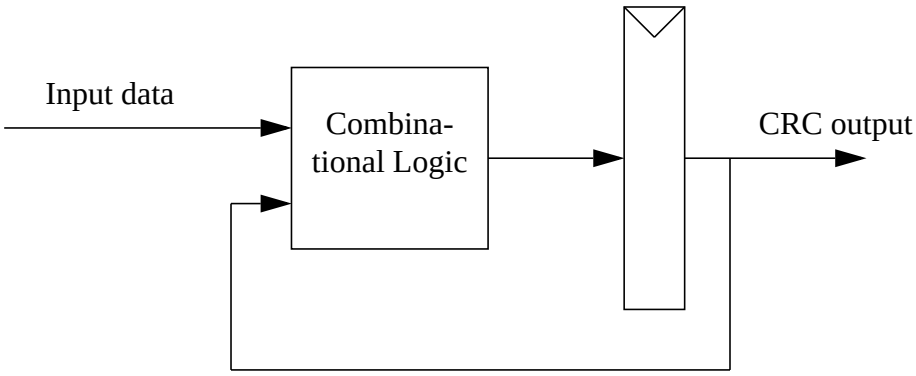


Figure 3.5: Parallel CRC implementation

for Ethernet. The CRC used for Ethernet is called CRC-32 and is by the way the same that is used for ATM. I chose to implement the CRC with an 8 bit parallel input, since that is the minimum data unit in TCP and UDP communication.

The realization I did and most other are based on XOR functions, since that is the basic function in the CRC specification. In standard cell libraries XOR gates are not very well implemented and the result is that after synthesis the gate-level netlist contained no XOR gates at all. Instead the CRC functionality was realized with NAND gates, NOR gates and inverters. This is not an optimal implementation, but estimations predict that it can nonetheless achieve 10 Gb/s in a 0.15 micron process technology [3.4].

By making a full custom layout we succeeded to achieve more than 5 Gb/s throughput in a 0.35 micron process technology, so there is a potential to make even faster designs with the use of optimized XOR gates.

By introducing more complex control logic it is possible to use a higher degree of parallelism and also take care of the special cases that may arise in the beginning and the end of the calculations. These special cases arise if the packet length is not divisible by the CRC input width. I have made a design with 32 bit wide input. There is of course an area penalty associated with this change, but since 4 times as many bits can be handled every clock cycle, the throughput can be increased.

3.7.2 Internet Checksum Functional Page

The checksum calculation in TCP and UDP is traditionally done in the host processor within the operating system kernel, before the payload data is handed over to the application. The GPPP introduces the possibility to handle the time and power consuming calculation in an FP. We have designed such an FP, that handles all computations and also keeps track of partially calculated checksums, that belong to fragmented IP packets.

This FP is briefly described in paper 4. The functionality is straight forward for non-fragmented packets. For those the packet is split into 16 bit words, which are added by one's complement addition. To work well in the 32 bit instance of the GPPP, this FP first adds the first and the second part of the current data word and then adds that sum to the accumulated sum, see figure 3.6.

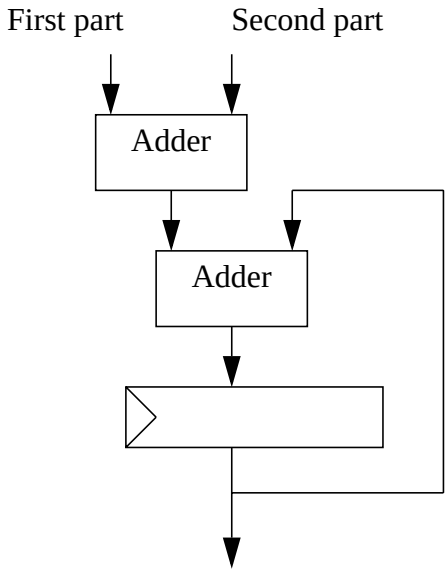


Figure 3.6: Internet Checksum Calculation

The calculation unit also has to manage the pseudo header form the IP header, that must be included in the calculation. This pseudo header differs for IPv4 and IPv6 since the address format changes between the versions of the Internet protocol.

The computation gets a lot more complicated when fragmented packets are considered. The FP thus needs to keep track of a lot of state variables, such as which fragments have been received, what is the current value of the checksum and so on. In cooperation with a reassembly supporting FP it also manages time-outs, which have to occur if not all fragments of a packet are received within a certain time limit.

The data path of our FP supports more than 11 Gb/s throughput when implemented in a 0.18 micron process technology [3.5].

3.7.3 IP Destination Address Functional Page

The IP destination address FP (IDAFP) is exemplifying a broader type of FP, the extract and compare type. These FPs extract a certain part of the header and compares it to some pre-configured values. For the IP destination address FP this particular field is the internet address that specifies the packet destination.

There can actually be several values that are acceptable as destination address. The first we normally come to think of is the IP address of the terminal, but also packets sent broadcast and on

multicast may be accepted, dependent on the configuration of the terminal. The IDAFP thus needs to perform many comparisons. I have chosen to implement these comparisons in parallel. This means that the result of the IDAFP will be ready one clock cycle after the IP destination address field has reached it. It is important to produce this result as soon as possible, since if the packet is not meant for the particular terminal where the GPPP is operating all functions should be shut down in order to save power.

The comparison could also have been done sequentially, in order to save chip area. Only the comparators would have been saved and some additional control logic would have been necessary. The memory to store acceptable addresses would still have been required and thereby the constraint on how many addresses to support would have remained the same. I have chosen to support 8 addresses in my implementation, one for the address of the terminal, and the remaining 7 for broadcast and multicast addresses.

The sequential comparison would imply that the IDAFP would need more clock cycles to finish and produce its result. It would however not reduce the performance of the GPPP, since all other FPs can operate in parallel with the IDAFP and the number of clock cycles needed would still be small enough in order to finish before the next packet header could reach the GPPP.

3.8 Detailed Performance Comparison Example

To get a feeling for how the performance of the GPPP differs from the performance of a generic RISC processor, which is executing a sequential program, I here present an example comparison. To be able to do this I have defined some Pμops (protocol micro operations) for generic execution of protocol processing functions. The functions described with these Pμops can easily be mapped to an actual instruction set architecture (ISA).

The example I have chosen here is the IP destination address checking. The FP for this function was described in the previous section and there it was concluded that the operation will finish in one clock cycle. Here, first of all the Pμops are briefly explained, then the implementation of the destination address checking algorithm is given and finally a performance comparison is provided.

3.8.1 Protocol Micro Operations

For general protocol processing several micro operations are necessary, but for the destination address checking 6 Pμops suffice to achieve an efficient implementation. These are:

- MOV, move data between registers and memory
- ADD, addition
- SUB, subtraction
- JMPZ, jump if zero
- JMPN, jump if negative
- JMP, unconditional jump

The micro operations ADD and SUB use three operand notation, where the operands must be registers or absolute values, e.g. ADD Ra, Rb, Rc means Rc = Ra + Rb. MOV uses two operand notation, where the operands can be registers, direct memory addresses, or indirect memory addresses. For example MOV ip_da, Ra means Ra = ip_da, where ip_da specifies a memory address where the value of the variable ip_da is stored. MOV (Rd), Rb means Rb = the value stored at the location pointed to by Rd and is an example of the indirect memory address. JMPZ and JMPN are conditional jumps which use the result of the last operation in order to evaluate if they should jump or not. They only take one operand, which is a label to the code line to which

they can jump. JMP is unconditional jump and also take one operand, which is a label to the code line to which it jumps.

3.8.2 Implementation of Destination Address Checking

The implementation of the destination address checking depends on the under laying data structure. To make this example as simple as possible I assume that all acceptable addresses are stored in an array, `addr_array`. This includes broadcast and multicast addresses. The number of valid entries in this array is also stored, in the variable `valid_addr_cnt`. The received destination address in stored in the variable `ip_da`.

The micro code can be seen in figure 3.7.

```
MOV ip_da, Ra                          (Ra = ip_da)
MOV addr_array, Rd                     (Rd = addr_array)
MOV valid_addr_cnt, Rb                 (Rb = valid_addr_cnt)
ADD Rb, Rd, Rc                         (Rc = Rb + Rd)
loop:
MOV (Rd), Re                           (Re = (Rd))
SUB Ra, Re, Re                         (Re = Ra - Re)
JMPZ match:
ADD Rd, #1, Rd                         (Rd = Rd + 1)
SUB Rd, Rc, Re                         (Re = Rd - Rc)
JMPN loop:
no_match:
/* Discard packet */
JMP next_packet:
match:
/* Continue processing */
JMP next_packet:
```

Figure 3.7: Pμops code for destination address checking

3.8.3 Performance Comparison

The code in figure 3.7 requires 4 + 8 * 6 = 52 instructions in the worst case, when the total number of addresses is limited to 8. 16 of these instructions are conditional jumps, which may cause control hazards and further increase the number of required clock cycles. The IDAFP of the GPPP takes care of the whole destination address checking function and thus the host processor does not have to spend any clock cycles when the GPPP is used.

From the first chapter we know that only around 10 instructions are available per minimum sized packet in a 10 Gb/s environment. If the host processor should take care of the whole protocol processing it would be overloaded with just that task and no processing power would be left

for the applications. This holds true even if it is considered that all packets are not of minimum size and that there is normally not a constant packet stream sent to one terminal.

This destination address checking function is typical for the broad category of extraction and comparison functions. Other functions, such as checksum computation, load the host processor even harder.

3.9 Further Details

The next four chapters contain one paper each. The first three papers have been presented at conferences and the fourth is an invited submission to a journal. All four papers concern the same protocol processing architecture, which was described here, but deal with different aspects of it. The first paper, which is an abstract from a poster presentation introduces the protocol processor architecture and discusses the overall aspects. The second paper gives a detailed specification on a particular instance of the protocol processor architecture, which is intended for Ethernet, IP and TCP processing in a network terminal. The third paper gives detailed implementation information on one functional page in the processor, that is the functional page for cyclic redundancy check (CRC). The fourth paper is an invited extension of the second paper. This paper introduces new results on TCP checksum calculation as well as a more thorough discussion on the architecture.

3.10 Conclusion

I and my colleagues have designed a new architecture for protocol processing, called GPPP. The GPPP is aimed for terminals, but could also be used in routers. The GPPP handles reception processing on a single packet or frame. It works is parallel with the host processor and reduces the workload on the host processor significantly. The GPPP architecture consists of several functional pages, which operate in parallel. Therefore a GPPP instance for Ethernet and TCP/IP can handle line speed operation up to 10 Gb/s. The most performance critical part is the CRC calculation.

References

[3.1] D. Liu, U. Nordqvist, C. Svensson, “Configuration-Based Architecture for High Speed and General-Purpose Protocol Processing”, SIPS’99, pp. 540-547

[3.2] 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

[3.3] 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

[3.4] 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

[3.5] 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

4

Paper 1

Configurable Port Processor Increases Flexibility in the Protocol Processing Area

Tomas Henriksson, Ulf Nordqvist, and Dake Liu

Dept. of Physics and Measurement Technology, Linköping University, SE-581 83 Linköping, Sweden

Phone: +46-13-28{8956, 5816, 1256}, Email: {tomhe, ulfnor, dake}@ifm.liu.se

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

Configurable Port Processor Increases Flexibility in the Protocol Processing Area

Tomas Henriksson, Ulf Nordqvist, and Dake Liu

Dept. of Physics and Measurement Technology, Linköping University, SE-581 83 Linköping, Sweden

Phone: +46-13-28{8956, 5816, 1256}, Email: {tomhe, ulfnor, dake}@ifm.liu.se

Abstract

The limitation in networking is no longer only the physical transmission media but also the end equipment, which has to process the protocol control fields. In most end terminals this processing has been performed by the main processor, but different types of co-processor have lately appeared to relieve it from this task. These co-processors have high power consumption since they are based on a RISC core. Instead ASIC:s can be used, but they lack flexibility and are specific for only one single protocol. It is clear that a new approach is needed.

A new type of architecture for protocol processing has been specified. Our configurable port processor for protocol processing (CPP) could be situated in an end terminal or as part of a switch or router. The processor works with a non-von Neuman architecture to reduce the power consumption and keeps the flexibility by being heavily software reconfigurable. The configurations can be generated from a description language, for example SDL or C++. The CPP has the performance and power consumption similar to an ASIC synthesized by a protocol synthesizer, but has the observability and flexibility within the protocol processing area of a normal von Neuman processor. This makes the CPP suitable for System-on-Chip integration.

The CPP consists of two parts, see figure 1, a deep pipeline serial processor (DPSP) and a micro controller (µC). The protocol processing takes place in the DPSP. The µC only handles configuration and some interface functions, it never touches the main data stream. The DPSP is based on configurable functional pages (FP), which each take care of one small task such as checksum calculation or field extractions.

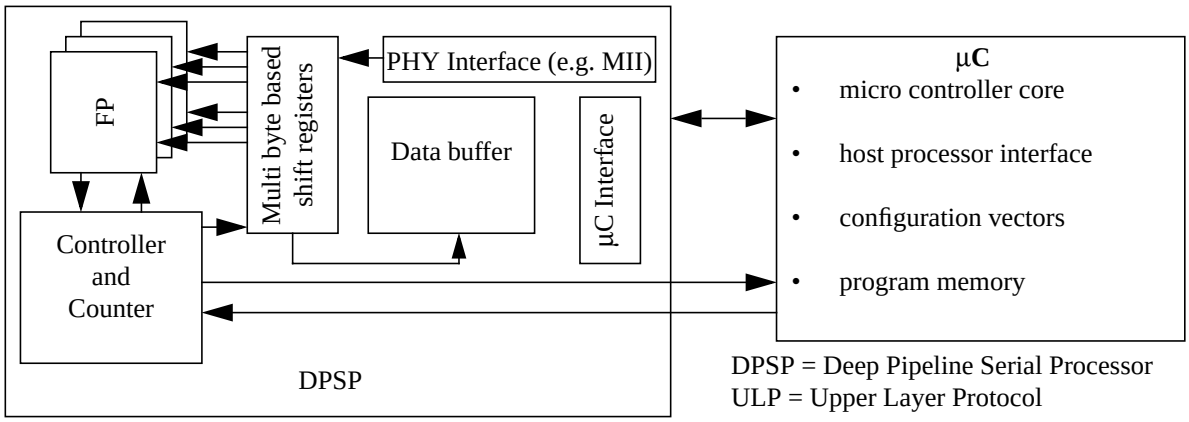


Figure 4.1: Overview of the architecture, the FP:s perform the actual protocol processing

Data is received via the PHY Interface and is then synchronized and parallelized and passed on to the multi byte based shift register. Data is then pipelined and each FP collects the protocol fields that it needs to perform its task. The FP:s are activated on control signals from the controller and counter (CC). The FP:s report to the CC by sending flags. The CC activates and shuts down

FP:s according to the dataflow and configurations. In particular, when the CC receives a flag that indicates that the packet must be discarded the CC shuts down all FP:s to save power.

The CC and the FP:s are software reconfigurable. Software reconfiguration is controlled by the μC. Configuration can also be performed in the design phase. If the CPP is placed in a System-on-Chip where only specific protocols are used, the CPP can be optimized for that particular environment.

The CPP can perform protocol processing on layer 2, 3, and 4 of the ISO-OSI reference model (e.g. Ethernet MAC, IP, and TCP) at the same time. Since all three protocol layers are processed simultaneously the latency can be kept very low and the memory access can be minimum.

5

Paper 2

Specification of a configurable General-Purpose Protocol Processor

Tomas Henriksson, Ulf Nordqvist and Dake Liu

Dept. of Physics and Measurement Technology, Linköping University, SE-581 83 Linköping, Sweden

Phone: +46-13-28{8956, 5816, 1256}, Email: {tomhe, ulfnor, dake}@ifm.liu.se

In proceedings of Second International Symposium on Communication systems, Networks and Digital Signal Processing, Bournemouth, UK, July 19-20,2000, pp. 284-289

Specification of a configurable General-Purpose Protocol Processor

Tomas Henriksson, Ulf Nordqvist and Dake Liu

Dept. of Physics and Measurement Technology, Linköping University, SE-581 83 Linköping, Sweden

Phone: +46-13-28{8956, 5816, 1256}, Email: {tomhe, ulfnor, dake}@ifm.liu.se

Abstract

A general-purpose protocol processor is specified with a dedicated architecture for protocol processing. This paper defines a functional coverage, analyses the control requirements, specifies functional pages and a controller unit. The general-purpose protocol processor is aimed for network terminals, therefore routing is not completely supported. However it should be possible to use it as part of a router with some minor modifications. The general-purpose protocol processor is partitioned into two parts, a configurable stand alone part and a program based microcontroller. The configurable part performs the protocol processing without any running program. The processor does not execute any cycle based program, instead execution is controlled by configuration vectors and control vectors. The microcontroller assists with the interface to the host processor and handles the configuration. It is concluded that by partitioning the control into three levels, the architecture is flexible and verification is simplified. The proposed architecture also has higher performance and lower power dissipation than other solutions.

5.1 Introduction

Networking is developing very fast and more and more protocols are emerging for different applications. Higher processing performances are requested by the applications. Two kinds of protocol processors are available on the market, one is the single protocol ASIC without flexibility, the other is the general purpose processor with limited performance. It is clear that a new type of architecture for protocol processing is needed to reach the real-time processing speed for Gigabit/s or higher speeds with enough flexibility [1], [2], [3].

The aim of this paper is to specify a protocol processor which will lead to the implementation of a prototype. The architecture is compared to conventional solutions to clarify the value of this type of architecture.

5.2 Functional Coverage

To cover both the compatibility and flexibility the architecture will include the most frequently used protocols. So that the architecture can be simple and still flexible. It means there are no problems to later include more protocols. This work is concentrated on different types of Ethernet, with IP/TCP-UDP [4] on top. The general-purpose protocol processor (GPPP) receives frames and processes them at real time speed, but it does not create and send frames at the same speed. The interface to the physical layer is the MII/GMII [5] and the interface to a host processor is in the middle of the TCP-UDP layer. As a platform for protocol processing the GPPP performs all Ethernet processing, all IP processing and TCP-UDP processing for terminals.

To cover the protocols IP/TCP-UDP also ARP, RARP, ICMP and IGMP have to be managed. Packets of these control oriented protocols are not that common and there is no need to design specialized hardware for them. Instead the functions can be performed in software in the host pro-

cessor with a relatively small total overhead. These kinds of packets are only recognized and then passed on to the host processor.

5.3 General architecture proposal

The proposed architecture is shown in figure 1. The GPPP consists of two parts, a deep pipeline

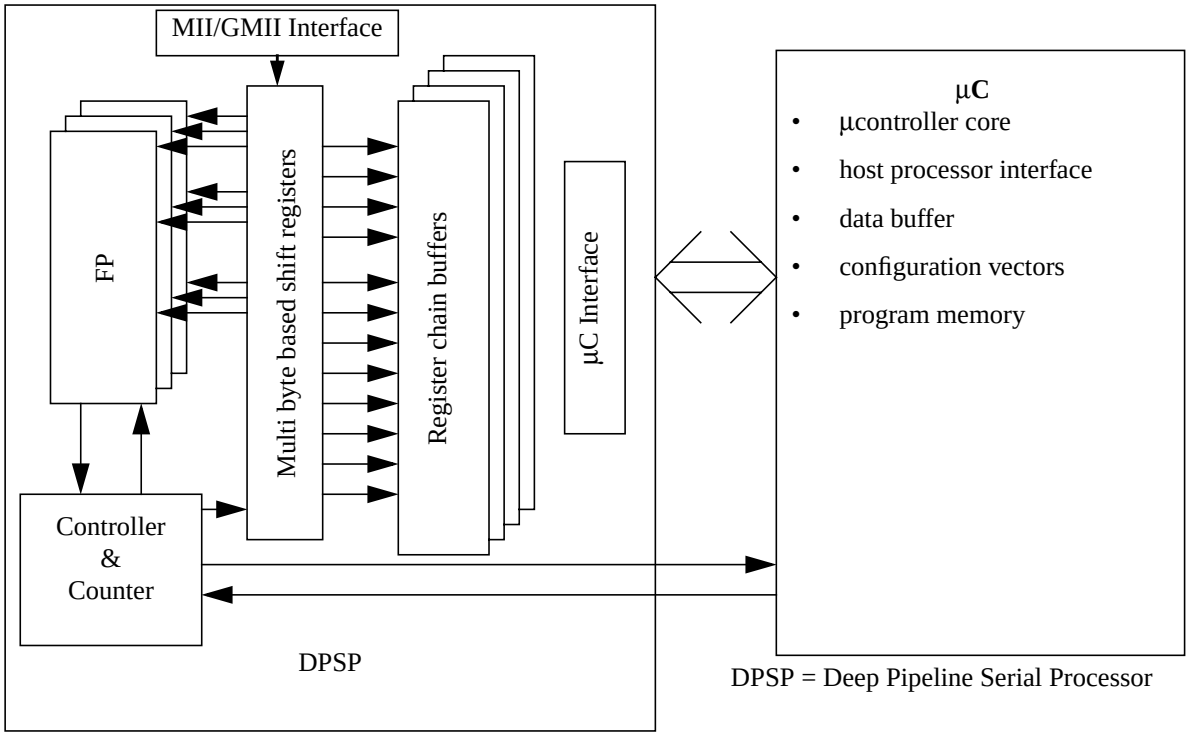


Figure 5.1: Overview of the architecture, the FPs perform the actual protocol processing

serial processor (DPSP) and a microcontroller (µC). The DPSP is based on software reconfigurable functional pages (FP) as well as a software reconfigurable controller and counter unit (C&C). The DPSP does not perform a cycle based program execution. One instruction word is a configuration vector for the complete execution of a FP. The data in the current data packet selects the next control vector so this is a data driven control process. It offers high performance and low-power operation combined with high flexibility within the protocol processing area.

The µC is used to configure the DPSP and to interface to the host processor. The DPSP runs stand alone after initial configuration.

The actual processing is performed in software reconfigurable FPs [1]. Each of these FPs has its own specific task. The FPs are fed with data from a parallelization/synchronization unit (PSU), see figure 2. Data is pipelined and the FPs will produce results at different times. To evaluate the results and take care of extracted values the C&C supports the FPs. The FPs that are needed are specified in a later section.

Each FP is autonomous as it performs an operation after configuration on a given start signal. The start signal is generated from the PSU and is given to the FPs by the C&C. Communication directly between the FPs is avoided, all FPs are controlled by the C&C and send flags to the C&C when they have something to report. In this way the verification of the FPs is greatly simplified and the architecture is more flexible. Every FP is controlled by a counter when to be active. Since three layers of protocols are being processed at the same time, FPs cannot be reused on different layers.

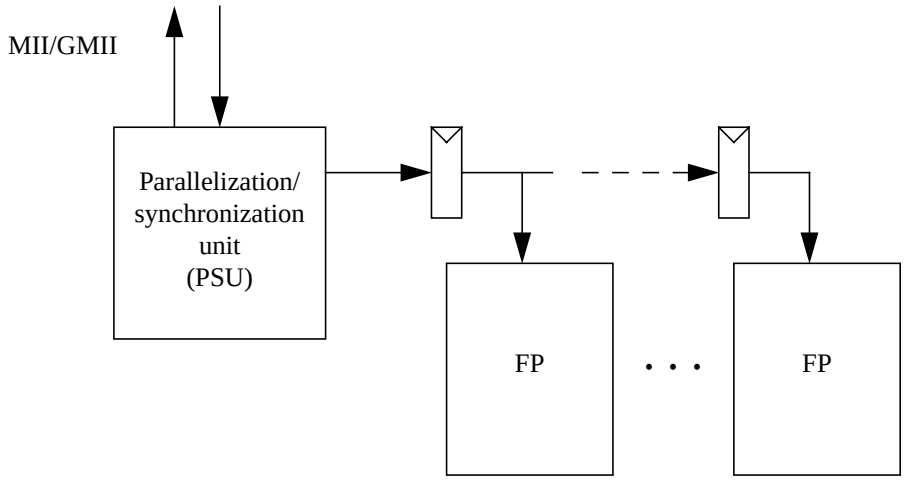


Figure 5.2: Data is synchronized and parallelized, thereafter one pipeline register is situated between every FP to decrease the fan-out requirements

5.4 Control Requirements

Protocol processing is a control intensive operation with different processing tasks and heavy data dependency, see figure 3. The control can be divided into two different types:

- configuration of the DPSP depending on the protocols used in the network
- control of the DPSP depending on the received data

5.4.1 Layer transparent control

When a frame/packet for some reason has to be discarded, all FPs should be shut down to save power and the GPPP should wait in idle mode until the next frame arrives. This calls for enable control of each FP.

5.4.2 Peripheral control

The payload has to be delivered to the software of the host processor in some way. The peripheral control consists of the payload delivery control and memory allocation assistance.

5.5 Specification of the Functional Pages

Figure 3 shows job allocation and order scheduling. As can be seen the Ethernet checksum calculation FP (ECCFP) is active at the same time as the other FPs. Since the data is pipelined the concurrency is dependent of how the FPs are placed along the pipeline. An example of the scheduling is shown in figure 4. FPs will be placed as to get shortest pipeline and scheduling. The interface to the FPs can be seen in figure 5. All signals and flags connect to the controller unit except the data and the clk. Below each FP is explained in somewhat more detail.

5.5.1 Ethernet checksum calculation FP (ECCFP)

The ECCFP receives a start signal and then performs CRC-32 calculation on all data passing through. In the end of the frame the FP will receive a frame end signal and compare the calculated value to the received frame check sequence. On non equality a discard flag is sent to the C&C.

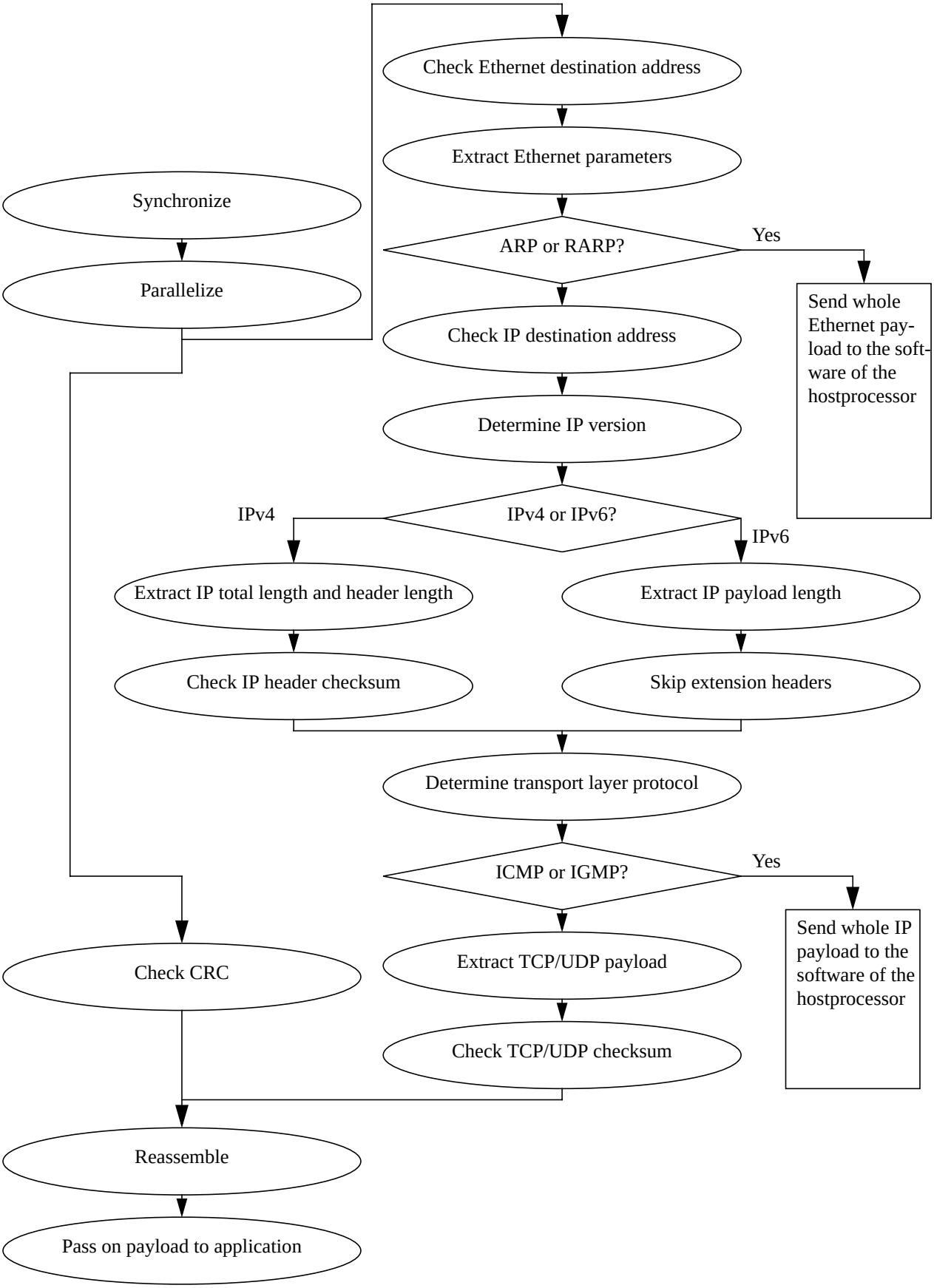


Figure 5.3: Flowchart that illustrates the operation

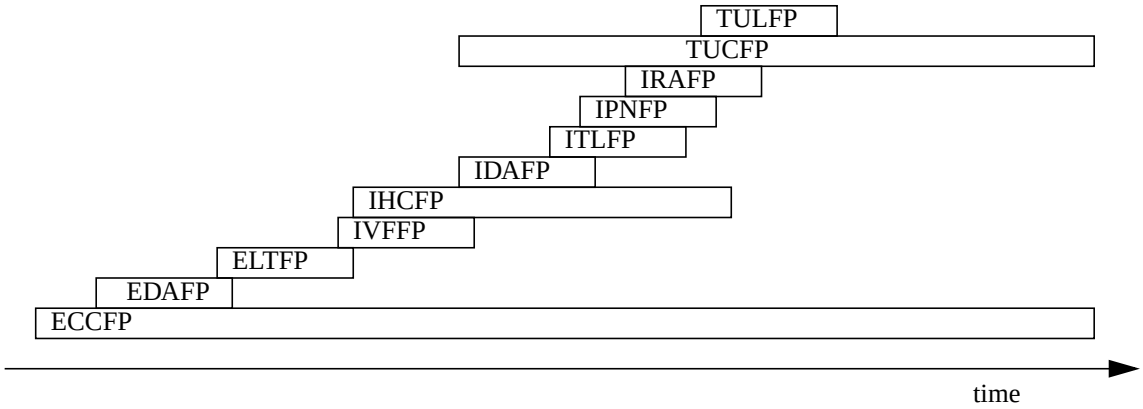


Figure 5.4: Principle scheduling of FPs for an TCP on top of IPv4 example. The boxes show when the FPs are active. Job abbreviations are specified later in this paper.

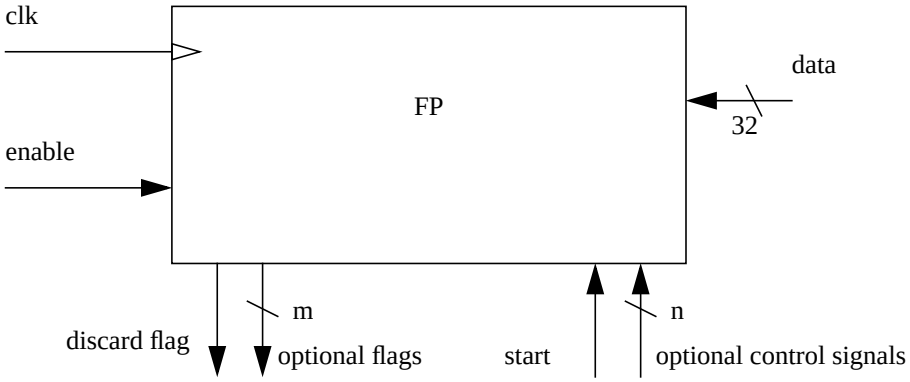


Figure 5.5: General interface of a FP

5.5.2 Ethernet destination address extraction and comparison FP (EDAFP)

The EDAFP is configured with the address of the terminal were the GPPP is situated. The FP receives a start signal and extracts and compares the received address to the configured one and checks if the extracted address is a multicast address. If the frame is not addressed to this network terminal a discard flag is sent to the C&C.

5.5.3 Ethernet length/ethertype field extraction FP (ELTFP)

The ELTFP extracts the length/ethertype field. If an ethertype is given, the length is expected from the ITLFP. The value is distributed by the C&C. A counter keeps track on how much data that has been received. When the counter reaches the length value a frame end flag is sent. This FP also gives the ethertype value to the C&C so that special jobs, like ARP and RARP, can be handled correctly.

5.5.4 IP header checksum calculation FP (IHCFP)

The IHCFP is active if the IP version field is IPv4. It then calculates the checksum by performing 16-bit one's complement addition of the header fields and makes sure the result is 0. If not a discard flag is sent to the C&C.

5.5.5 IP version field extraction FP (IVFFP)

The IVFFP extracts the IP version field and sends a flag to the C&C telling which version of IP is used.

5.5.6 IP destination address extraction and comparison FP (IDAFP)

The IDAFP is configured with the terminal address for the application. The FP receives a start signal and IP version information and extracts and compares the received address to the configured one, it also checks if the extracted address is a multicast address. If it is an unrecognized address a discard flag is sent to the C&C.

5.5.7 IP header length extraction FP (IHLFP)

The IHLFP sends a flag when the IP header has been received. In IPv4 the IHL field specifies the length. In IPv6 the header is always 40 bytes plus optional extension headers. The extension headers, except fragmentation, in IPv6 are not processed, since they concern routers and management protocols.

5.5.8 IP total length extraction FP (ITLFP)

The ITLFP extracts the length field to send the length value to the ELTFP.

5.5.9 IP protocol/next header extraction FP (IPNFP)

The IPNFP extracts the protocol field from the IP header and sends a flag to the TCP-UDP FPs to tell if the present packet is TCP or UDP. If there exist extension headers in IPv6 packets these are skipped and the extension header length field is used to find out when the next header starts. This is done until a known header type is received. Known headers are TCP, UDP, ICMP, IGMP and ICMPv6.

5.5.10 IP reassembly FP (IRAFP)

The IRAFP extracts the fragment fields from IPv4 header and searches for a fragment extension header in IPv6. If fragmentation is present this FP manages payload data to be stored in memory on the right place and controls the TUCFP to process the right data. To assist the IRAFP memory tables and timers for reassembly are present.

5.5.11 TCP-UDP checksum calculation FP (TUCFP)

The TUCFP calculates the checksum by performing 16-bit one's complement addition of the whole packet, including some IP header fields. If the result is non zero a discard flag is sent to the C&C. Multiple accumulators are used to calculate checksums of multiple packets, since fragments of them may arrive nestled.

5.5.12 TCP-UDP packet length counter FP (TULFP)

The TULFP extracts the length value and provides this to the software of the host processor. The length is also needed for reassembly and checksum calculation.

5.6 The Controller and Counter Unit

Figure 6 shows the general structure of the C&C. The C&C has to manage high-level control only, since FP specific control is handled within each FP. It receives flags from the FPs and schedules the pipeline delay it sends control signals to the FPs. The controller unit is based on a configurable finite state machine (FSM), which controls the discarding or delivery of packets depending on the flags it receives from the FPs. When a flag, that tells the C&C to discard a packet is received, all activities are switched off except for the PSU, which looks for the next frame.

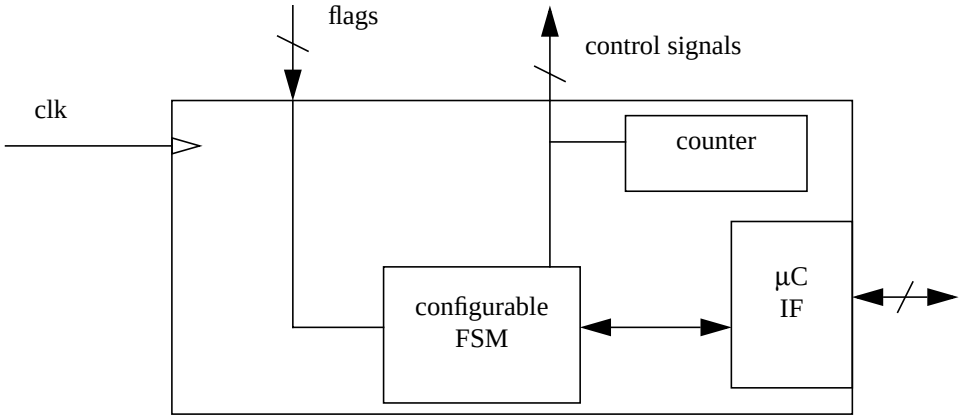


Figure 5.6: Controller and counter unit overview

If a packet is received without any problem, the C&C notifies the µC and tells it where in memory the packet can be found. The C&C also manages memory allocation and storing of payload with help from the µC.

5.7 Discussion

The proposed architecture uses extensive parallelism and configurable control to cut down the hardware redundancy and so the power- and time-consuming characteristics of a programmable processor. The critical path to the real-time speed limit has been found in the ECCFP. If a data width of 32 bits is used there should be no problem to deframe Gigabit Ethernet data and perform the CRC check using a 31.25 MHz clock. In a conventional solution, the CRC check is performed in the MAC controller, but all network- and transport-layer processing is performed by the host processor. The GPPP relieves the host processor from this burden which is of great importance as the speed increases. Other dedicated processors, but still program based, can solve the same tasks but suffer from much higher power dissipation than the GPPP. Also other dedicated protocol processing solutions normally make use of a layer-based pipelining technique [6], which introduces latency. This latency is eliminated in the GPPP since all layers are processed concurrently.

5.8 Conclusions

By using the proposed architecture and control, a configurable GPPP is accomplished. The configuration-based architecture makes hardware reuse and wide functional coverage possible and moves unnecessary hardware design to a compiler. The control is partitioned into three different parts, FP internal control, the C&C and the µC. This partition simplifies verification and increases the flexibility and supports future changes in the protocols. The proposed architecture has higher performance and lower power dissipation than its competitors.

The project is under functional implementation phase, approaching to the delivery of the payload to the host processor. Studies are also being made concerning problems occurring when not buffering the whole Ethernet frame and how to solve the reassembly of IP packets in hardware.

Acknowledgments

This study was supported by the Intelect program of Swedish Foundation for Strategic Research (SSF). Authors would like to thank Dr. George Liu, Ericsson Research, for interesting discussions.

Appendix: List of Jobs

Ethernet/802.3 CRC check, Ethernet/802.3 destination address check, Ethernet/802.3 payload protocol determination, IP version determination, ARP/RARP recognition, IPv4/IPv6 destination address check, IPv4 header checksum check, IP reassembly support, IP payload protocol determination, TCP packet length determination, and TCP/UDP checksum check.

References

[5.1] Configuration-based architecture for high speed and general-purpose protocol processing, Dake Liu, Ulf Nordqvist and Christer Svensson, proceedings of SIPS'99, Taiwan

[5.2] Scalable Protocol Engine for High-Bandwidth Communications, Chirstos J. Gerorgiou and Chung-Sheng Li, IEEE Int. Conf. on Communications, 1997. ICC'97 Montreal, Towards the Knowledge Millennium. pp.1121-1126 vol.2 1997

[5.3] A Design Methodology for Protocol Processors, Michael Yang and Ahmed Tantawy, Proceedings of the Fifth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems, 1995, pp.376-381

[5.4] Computer Networks, 3rd Ed., Andrew S. Tanenbaum, Prentice Hall PTR, ISBN 0-13-349945-6, 1996

[5.5] Gigabit Ethernet, Jayant Kadambi et al, Prentice Hall PTR, ISBN 0-13-913286-4, 1998

[5.6] The Parallel Protocol Engine, Matthias Kaiserswerth, IEEE/ACM Transactions on Networking, vol.1 No.6 December 1993 pp.650-663

6

Paper 3

VLSI Implementation of CRC-32 for 10 Gigabit Ethernet

Tomas Henriksson, Henrik Eriksson, Ulf Nordqvist, Per Larsson-Edefors, and Dake Liu
Department of Physics and Measurement Technology
Linköping University, SE-581 83 Linköping, Sweden
Phone: +46-13-28{8956, 2483, 8965, 1224, 1256}, Fax: +46-13-132285
E-mail: {tomhe, hener, ulfnor, perla, dake}@ifm.liu.se

In proceedings of The 8th IEEE International Conference on Electronics, Circuits and Systems, Malta, September 2-5 2001, vol. III, pp. 1215-1218

VLSI Implementation of CRC-32 for 10 Gigabit Ethernet

Tomas Henriksson, Henrik Eriksson, Ulf Nordqvist, Per Larsson-Edefors, and Dake Liu
Department of Physics and Measurement Technology
Linköping University, SE-581 83 Linköping, Sweden
Phone: +46-13-28{8956, 2483, 8965, 1224, 1256}, Fax: +46-13-132285
E-mail: {tomhe, hener, ulfnor, perla, dake}@ifm.liu.se

Abstract

For 10 Gigabit Ethernet a CRC-32 generation is essential and timing critical. Many efficient software algorithms have been proposed for CRC generation. In this work we use an algorithm based on the properties of Galois fields, which gives very efficient hardware. The CRC generator has been implemented and simulated in both standard cells and a full-custom design technique. In standard cells from the UMC 0.18 micron library a throughput of 8.7 Gb/s has been achieved. In the full-custom design for AMS 0.35 micron process we have achieved a throughput of 5.0 Gb/s. The conclusion, based on extrapolation of device characteristics, is that CRC-32 generation for 10 Gb/s can be designed with standard cells in a 0.15 micron process technology, or using full-custom design techniques in a 0.18 micron process technology.

6.1 Introduction

Digital communication is becoming more and more important and higher bit rates are constantly required. To manage this, not only the optical fibers have to work at higher speeds, but also the electronic equipment at the ends of these fibers. One thing that has been proven hard to speed up is the checksum calculation and especially the type cyclic redundancy check (CRC), which is used for example in the Ethernet and ATM protocols.

In the proposal for the new 10 Gb/s Ethernet standard it is specified that CRC will still be used and also that the minimum data unit is a symbol of 8 bits [6.1]. It is highly beneficial to perform the CRC calculation at wire speed, since buffering of all received data would imply high power consumption and introduce an unnecessary delay [6.2].

The aim of this paper is to show that it is possible to perform the 32-bit CRC used in Ethernet (CRC-32) at the speed of 10 Gb/s with process technologies available today. We look at two different designs and discuss their performance based on our simulations. Since we do not have access to cutting edge process technologies, we base our conclusions on extrapolation of device characteristics [6.9].

6.2 Mode of Operation

All modern high-speed implementations of CRC make use of parallelism. This gives an obvious advantage over the original bit-serial implementation, since the required clock frequency can be reduced with a factor corresponding to the level of parallelism. In previous work, we have investigated some different architectures for parallel CRC generation and concluded that the fixed logic implementation is somewhat faster than a look-up-table based architecture [6.3].

The higher the parallelism, the lower the clock frequency can be used. Hobson and Cheung [6.4] have proposed a 32-bit parallel CRC-32 engine which could reach 5 Gb/s. For Ethernet however, it is desirable to have only 8-bit parallelism, since the minimum symbol in an Ethernet packet is 8 bits. This is to avoid complicated handling of initial and ending 8-bit symbols.

Glaise and Jacquart [6.5] have shown that the CRC-32 can be calculated efficiently by using the properties of Galois fields. Several other optimizations to the CRC algorithm have been made [6.6], but many of them are only suitable for software and actually make a pure hardware solution more complicated.

The general architecture of the CRC generator is shown in figure 6.1. The middle register is the

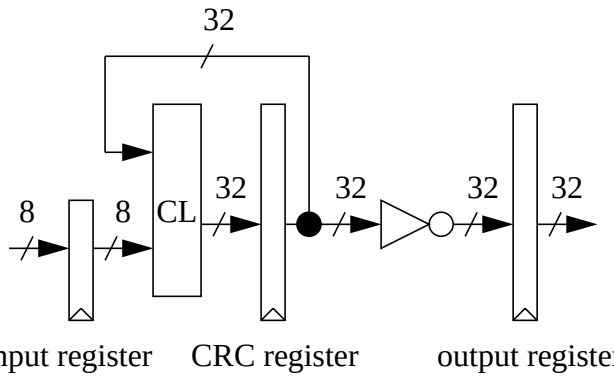


Figure 6.1: General architecture of the CRC generator

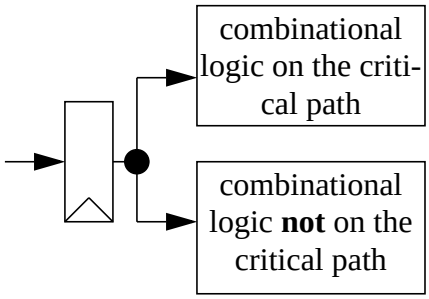
CRC register. Since we have chosen the CRC-32 algorithm of Glaise & Jacquart, the CRC register must be preset not to all 1's, as in most software implementations, but to the vector, in hex, "46AF6449" [6.5]. To achieve the correct CRC value, 4 bytes with all 0's must also be supplied after the actual data. The output must be inverted, which is taken care of before the output register. The input and output registers do not have set or reset, but the register bits in the CRC register have set or reset according to the vector mentioned above. It is assumed that the set, reset, and clock signals can be generated to suit this architecture. That implies for example, that the clock signal is constantly "0" when set and reset are activated.

6.3 Implementation Considerations

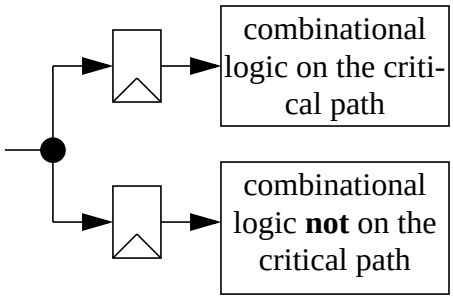
The CRC generator has been implemented in two different ways. The target technology for a full-custom implementation was the 3 metal layer AMS 0.35 micron process and the target technology for a synthesized standard cell implementation was the 6 metal layer UMC 0.18 micron process.

6.3.1 Standard Cell Implementation

A key to achieve high speed is to have a flip-flop with low delay from clock to output and a high driving capability. The setup time should also be short. When doing a standard cell implementation, some flip-flops should be parallelized. This pertains to the flip-flops, that drive critical paths. The fan-out is shared on two flip-flops, so one is driving only the gate in the critical path and the other one is driving all other gates, that need the same input. By doing so the delay from the flip-flop to the first gate in the critical path is minimized. An example of this can be seen in figure 6.2. The input to the flip-flops is not in the critical path, so loading the last gate with two parallel flip-flops does not impact the maximum clock frequency for the design. Unfortunately the synthesis tool could not handle the parallelization of flip-flops, therefore it had to be done manually in the RTL code.



Non-parallelized flip-flop



Parallelized flip-flop

Figure 6.2: Parallelization of flip-flops

The CRC generator has been described in Mentor Graphics Renoir and VHDL. Synthesis was performed by Cadence Buildgates and timing-driven place&route was made by Cadence Silicon Ensemble (SE).

The standard cell implementation consists of totally 432 cells, which require a chip area of 11111 μm^2.

6.3.2 Full-Custom Implementation

By using the equations of Glaise and Jacquart, we can limit the maximum number of inputs to any of the bits in the CRC register to 8. This leaves us with the logic depth of 3 if we exclusively use 2 input XOR gates for the combinational logic. In the full-custom implementation we use the XOR gate proposed by Wang et. al. [6.7], which also was used by Hobson and Cheung.

To make the design work at the required high frequency, sizing of the logic gates is needed due to the high fan-out of some of the gates. Load balancing and logic sharing are also two subjects for optimization. We have generally tried to share as much logic as possible to reduce the fan-out of the CRC register.

In the full-custom design we have not used flip-flops for the CRC register. Instead we have used two stages of latches, with XOR gates and inverters merged into the latches, an example is shown in figure 6.3. The inverter in the latch is used as the last stage of the XOR gate. By doing this we reduce the logic depth and hence the total delay. The latches are the simplest possible, an inverter followed by a transmission gate, which have proven to be fast in [6.8].

The XOR gates have one input, that is slightly faster than the other. In the critical path we have throughout the design used the faster input.

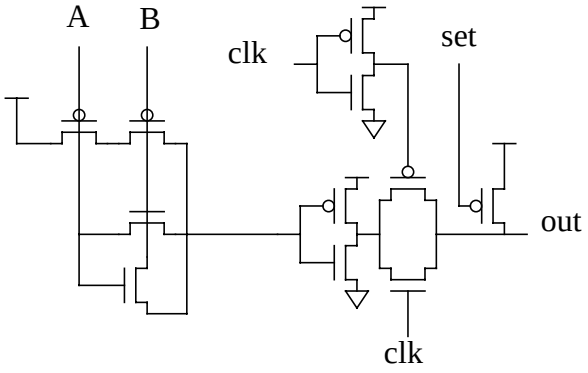


Figure 6.3: XOR merged into positive latch, with active low set signal

The full-custom implementation was designed in Mentor Graphics Layout Editor. The layout is shown in figure 6.4. Totally 908 transistors have been used and the largest transistors have W_p=17 µm and W_n=9 µm respectively.

6.4 Simulation Results and Static Timing Analysis

6.4.1 Standard Cell Implementation

For the standard cell implementation, the maximum clock frequency was identified by using the static timing analysis (STA) in Buildgates. When doing this, parasitic wire capacitances extracted from the layout by Silicon Ensemble were used.

A maximum clock frequency of 1.09 GHz was achieved. That corresponds to 8.7 Gb/s of processed data.

6.4.2 Full-Custom Implementation

The complete design has been simulated in switch-level mode in Mentor Graphics LSim simulator to verify the functional correctness.

Parasitic wire capacitances from the complete layout were extracted and the critical path was manually identified to be the structure, which can be seen in figure 6.5. When the simulation was performed, the critical path was separated from the layout and the adequate parasitic capacitances were used. This was done in order to be able to easily apply the worst-case input vector. The critical path was simulated with HSpice™ using level 49 with mean parameters.

The critical path simulation gives a worst-case delay of 1.60 ns, which corresponds to a maximum clock frequency of 625 MHz, and thus a throughput of 5 Gb/s is achieved.

6.5 Discussion and Extrapolation of Results

With 8-bit parallelism and 10 Gb/s requirement, obviously a clock frequency of 1.25 GHz is needed, i.e. a clock period of 800 ps. However, this high speed cannot be expected from the 0.35 micron process technology, nor could it be achieved with the synthesized standard cell implementation in a 0.18 micron process.

The synthesized standard cell implementation technique is normally preferred in industry and therefore it is most important to be able to use that technique in order to achieve 10 Gb/s through-

Figure 6.4: Layout of the CRC Generator in the full-custom implementation (width 150 μm, height 720 μm). Input and output registers are not included in the full-custom design.

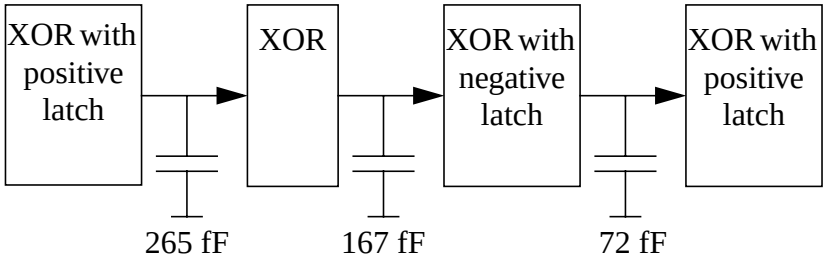


Figure 6.5: Critical path in the full-custom implementation. The loads specify the total output load, including output capacitance of the driving stage, wire capacitance, and input capacitance of the driven gates.

put. According to the SIA roadmap [6.9] the gate delay reduces with a factor of 1.27 when going from 0.18 micron to 0.15 micron technologies. We achieved 8.7 Gb/s in a 0.18 micron process and thus, using the scaling factor, approximately 11 Gb/s could be achieved with a 0.15 micron process. However, the scaling factor 1.27 only considers gate delay reduction. It does not say anything of what will happen with delays due to interconnects. The CRC generator is a fairly small logic block and gate delays dominate over delays due to interconnects. That means that we have strong indications that at least 10 Gb/s will be achieved in a 0.15 micron process.

If a full-custom implementation is made, it is possible to achieve a throughput of more than 10 Gb/s with a 0.18 micron technology, since the scaling factor when going from 0.35 micron to 0.18 micron technologies is more than 2.

6.6 Conclusion

We conclude that it is possible to handle CRC-32 calculation for data streams up to 10 Gb/s by using a standard cell synthesized design. The requirement is to use a 0.15 micron process technology.

A full-custom design can reach a throughput of more than 10 Gb/s in a 0.18 micron process technology. This requires that a high-performance XOR gate is used in combination with fast latches and that a compact layout is made.

Acknowledgment

The authors would like to acknowledge SwitchCore for interesting discussions. This study was supported by the Swedish Foundation for Strategic Research (SSF), the Swedish Research Council for Engineering Sciences (TFR) under contract 314-97-289, and the Center for Industrial Information Technology at Linköping Institute of Technology (CENIIT).

References

[6.1] H. Frazier, “10Gig MII update”, on *the www*, http://grouper.ieee.org/groups/802/3/10G_study/public/nov99/index.html

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

[6.3] U. Nordqvist, T. Henriksson, and D. Liu, “CRC Generation for Protocol Processing”, *Norchip 2000*, Turku, Finland, pp. 288-293.

[6.4] R. F. Hobson and K. L. Cheung, “A High-Performance CMOS 32-Bit Parallel CRC Engine”, *IEEE Journal of Solid-State Circuits*, vol. 34, no. 2, February 1999, pp. 233-235.

[6.5] R. J. Glaise and X. Jacquart, “FAST CRC CALCULATION”, *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, Cambridge, MA, USA, 1993, pp. 602-605.

[6.6] R. N. Williams, “A Painless Guide to CRC Error Detection Algorithms”, on *the www*, http://www.ross.net/crc/ version 3.00, 19 Aug. 1993.

[6.7] J.-M. Wang, S.-C. Fang, and W.-S. Feng, “New efficient designs for XOR and XNOR functions on the transistor level”, *IEEE Journal of Solid-State Circuits*, vol. 29, July 1994, pp. 780-786.

[6.8] C. Svensson and J. Yuan, “Latches and Flip-flops for Low Power Systems”, *Low-Power CMOS Design*, New York: IEEE Press, 1998, part IV, pp. 233-238.

[6.9] Semiconductor Industry Association, *International Technology Roadmap for Semiconductors: 1999 edition*, Austin, TX:International SEMATECH, 1999.

7

Paper 4

Specification of a configurable General-Purpose Protocol Processor

Tomas Henriksson, Ulf Nordqvist, and Dake Liu
Dept. of Electrical Engineering, Linköping University, SE-581 83 Linköping, Sweden
Phone: +46-13-28{8956, 5816, 1256}, Email: {tomhe, ulfnor, dake}@isy.liu.se

Invited submission to a special issue of IEE Proceedings on Circuits, Devices and Systems as an extended version of Paper 2. The manuscript is under the process of review at the time of writing.

Specification of a configurable General-Purpose Protocol Processor

Tomas Henriksson, Ulf Nordqvist, and Dake Liu
Dept. of Electrical Engineering, Linköping University, SE-581 83 Linköping, Sweden
Phone: +46-13-28{8956, 5816, 1256}, Email: {tomhe, ulfnor, dake}@isy.liu.se

Abstract

A general-purpose protocol processor is specified with a dedicated architecture for protocol processing. This paper defines a functional coverage, analyses the control requirements, specifies functional pages and a controller unit. The general-purpose protocol processor is aimed for network terminals, therefore routing is not completely supported. However it should be possible to use it as part of a router with some minor modifications. The general-purpose protocol processor is partitioned into two parts, a configurable stand alone part and a program based microcontroller. The configurable part performs the protocol processing without any running program. The processor does not execute any cycle based program, instead execution is controlled by configuration vectors and control vectors. The microcontroller assists with the interface to the host processor and handles the configuration. It is concluded that by partitioning the control into three levels, the architecture is flexible and verification is simplified. The proposed architecture also has higher performance and lower power dissipation than other solutions.

7.1 Introduction

Computer networks are developing very fast and more and more protocols are emerging for different applications. At the same time as the protocols are becoming more complex the transmission speeds also increase. This leads to a situation where the electronic equipment has a hard task to keep up with the speeds of the optical links and the complexity of the new protocol standards. Many companies and universities are working on new types of architectures for switches and routers.

For network terminals the scene is different, very few projects aim at dealing with the terminal, which is connected to a high speed network. The processing in the terminal also includes the data transformation from high bandwidth network to low bandwidth applications. For terminals, two kinds of protocol processors are available on the market, one is the single protocol ASIC without flexibility, the other is the general purpose processor with limited performance. It is clear that a new type of architecture for protocol processing in network terminals is needed to reach the real-time processing speed for Gigabit/s or higher speeds with enough flexibility [7.1], [7.2], [7.3].

The aim of this paper is to extend the specification of a protocol processor [7.4] which is based on configurable functional pages. The critical parts of the processor have been implemented and simulated. The results of these simulations are provided and the architecture is compared to conventional solutions to clarify the value of this type of architecture.

The rest of this paper is divided into 9 sections. In section 2 the functional coverage of the protocol processor is discussed. In section 3 the general architecture is defined and then the control requirements are discussed in section 4. Section 5 specifies the key components in the architecture and sections 6 and 7 present the results of implementations of the two most critical parts for high speed operation. In section 8 the hierarchical control architecture is described. Section 9 discusses

the protocol processors advantages and disadvantages and finally some conclusions are drawn in section 10.

7.2 Functional Coverage

To cover both the compatibility and flexibility the architecture will include the most frequently used protocols. So that the architecture can be simple and still flexible. It means there are no problems to later include more protocols. This work is concentrated on different types of Ethernet, with IP/TCP-UDP [7.5] on top. This can be seen as an instance of the general-purpose protocol processor (GPPP). The same main architecture with other functional pages (FPs) can be used for totally different protocol stacks. Deciding which FPs to include is the first level of configuration and has to be done before manufacturing the GPPP. The GPPP receives frames and processes them at real time speed, but it does not create and send frames at the same speed. The interface to the physical layer is the MII/GMII [7.6] and the interface to a host processor is in the middle of the TCP-UDP layer. As a platform for protocol processing the GPPP performs all Ethernet processing, all IP processing and TCP-UDP processing for terminals.

To cover the protocols IP/TCP-UDP also ARP, RARP, ICMP and IGMP have to be managed. Packets of these control oriented protocols are not that common and there is no need to design specialized hardware for them. Instead the functions can be performed in software in the host processor with a relatively small total overhead. These kinds of packets are only recognized and then passed on to the host processor. The complete list of jobs can be found in appendix .

7.3 General architecture proposal

The proposed architecture is shown in figure 7.1. The GPPP consists of two parts, a deep pipe-

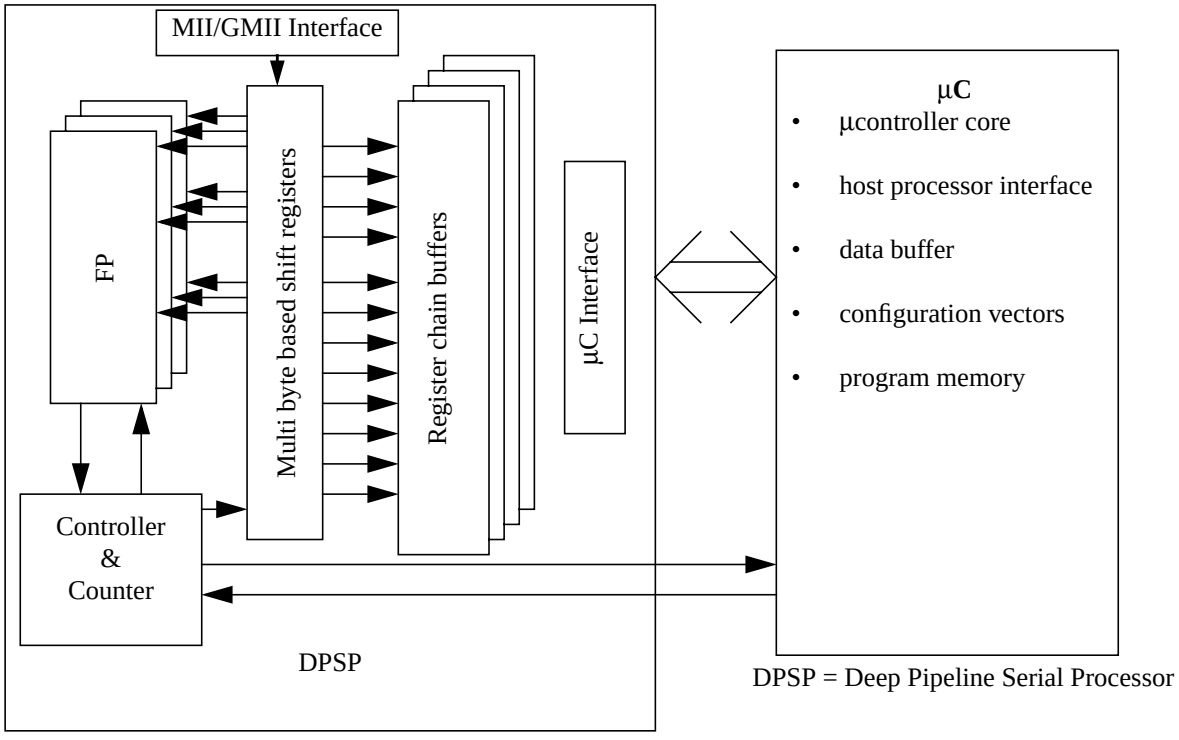


Figure 7.1: Overview of the architecture, the FPs perform the actual protocol processing

line serial processor (DPSP) and a microcontroller (µC). The DPSP is based on software reconfigurable functional pages as well as a software reconfigurable controller and counter unit (C&C). The DPSP does not perform a cycle based program execution. One instruction word is a configu-

ration vector for the complete execution of a FP. The data in the current data packet selects the next control vector so this is a data driven control process. It offers high performance and low-power operation combined with high flexibility within the protocol processing area.

The μC is used to configure the DPSP and to interface to the host processor. The DPSP runs stand alone after initial configuration.

The actual processing is performed in software reconfigurable FPs [7.1]. Each of these FPs has its own specific task. The FPs are fed with data from a parallelization/synchronization unit (PSU), see figure 7.2. Data is pipelined and the FPs will produce results at different times. To evaluate the

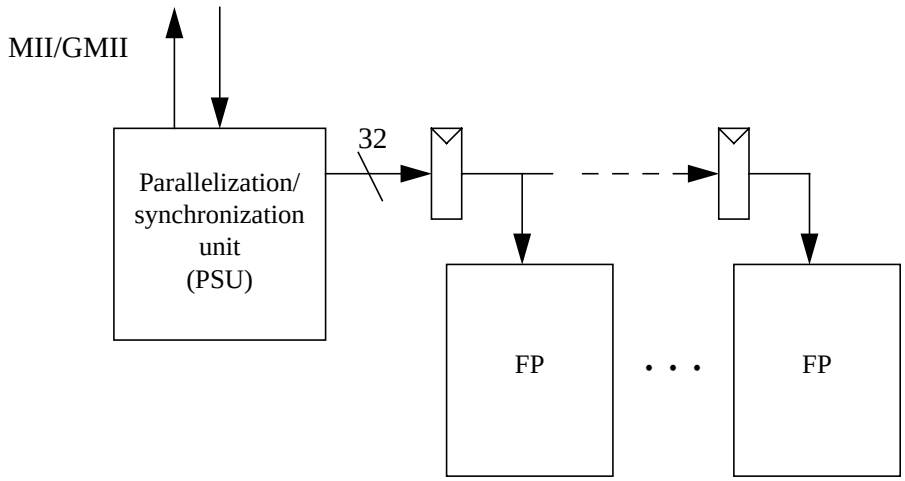


Figure 7.2: Data is synchronized and parallelized, thereafter one pipeline register is situated between every FP to decrease the fan-out requirements

results and take care of extracted values the C&C supports the FPs. The FPs that are needed are specified in a later section.

Each FP is autonomous as it performs an operation after configuration on a given start signal. The start signal is generated from the PSU and is given to the FPs by the C&C. Communication directly between the FPs is mostly avoided, all FPs are controlled by the C&C and send flags to the C&C when they have something to report. In this way the verification of the FPs is greatly simplified and the architecture is more flexible. Every FP is controlled by a counter when to be active. Since three layers of protocols are being processed at the same time, FPs cannot be reused on different layers.

7.4 Control Requirements

Protocol processing is a control intensive operation with different processing tasks and heavy data dependency, see figure 7.3. The control of the FPs can be divided into two different types:

- configuration of the DPSP depending on the protocols used in the network
- control of the DPSP depending on the received data

The first part, the configuration, is handled by the μC, and has to be finished before the packets arrive at the terminal. The second part is conducted by the C&C on a high level and by the FPs internally on a low level. When the packet arrives at the terminal the control signals are dynamically decided and different for each arriving packet, dependent of the contents of that packet.

7.4.1 Layer transparent and dependent control

When a frame/packet for some reason has to be discarded, all FPs should be shut down to save power and the GPPP should wait in idle mode until the next frame arrives. This calls for enable

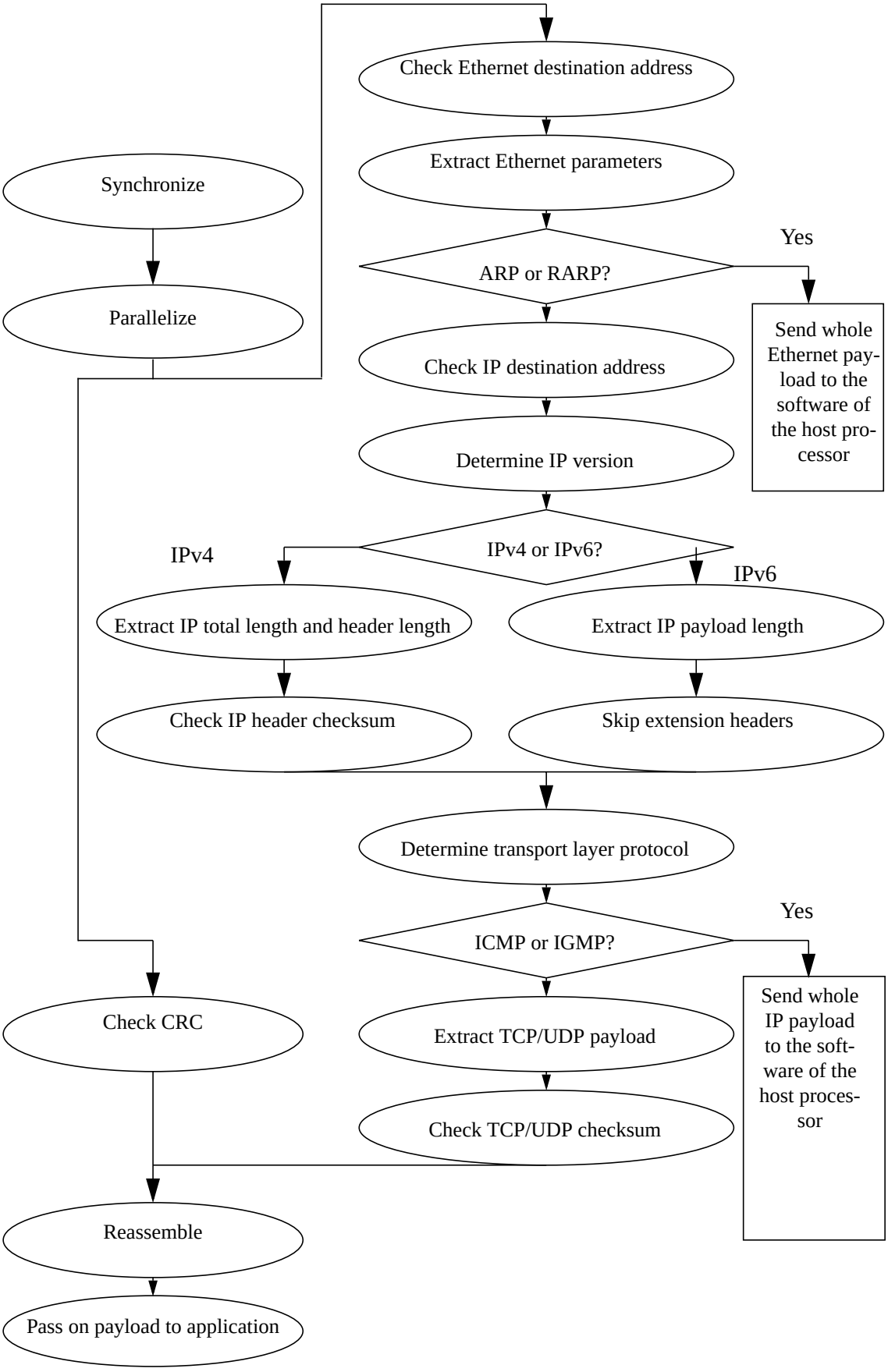


Figure 7.3: Flowchart that illustrates the operation

control of each FP. This kind of control, that applies to all layers in the protocol stack is called layer transparent control and affects the whole GPPP. In the case of, for example, an ICMP packet, the FPs that deal with TCP and UDP tasks can be disabled, but not the rest of the GPPP, this would be the opposite case, the so called layer dependent control. So each FP will be assigned to a set and each such set will be associated with the processing of a certain layer in the protocol stack.

7.4.2 Peripheral control

The payload has to be delivered to the application software of the host processor in some way. This is taken care of by the peripheral control. The peripheral control consists of the payload delivery control and memory allocation assistance. These are both implemented in the μC.

7.5 Specification of the Functional Pages

Figure 7.3 shows job allocation and order scheduling. As can be seen the Ethernet checksum calculation FP (ECCFP) is active at the same time as the other FPs. Since the data is pipelined the concurrency is dependent of how the FPs are placed along the pipeline. An example of the scheduling is shown in figure 7.4. FPs will be placed as to get shortest pipeline and scheduling. The

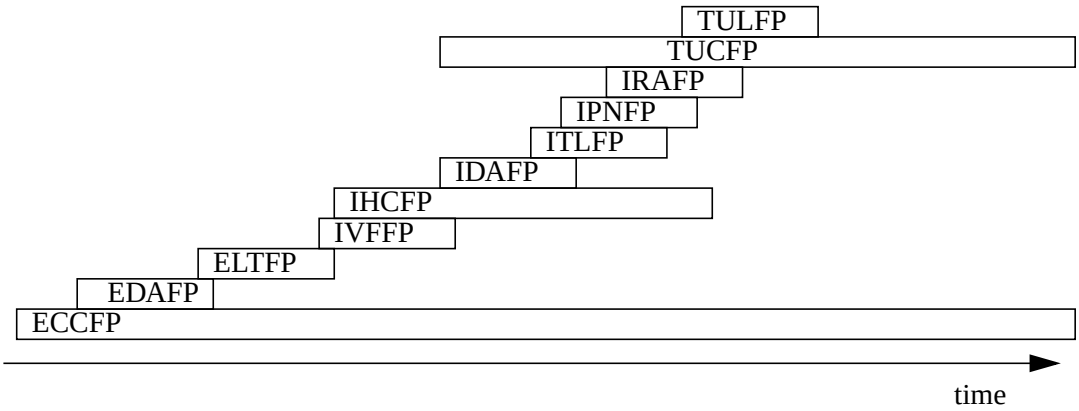


Figure 7.4: Principle scheduling of FPs for an TCP on top of IPv4 example. The boxes show when the FPs are active. Job abbreviations are specified later in this paper.

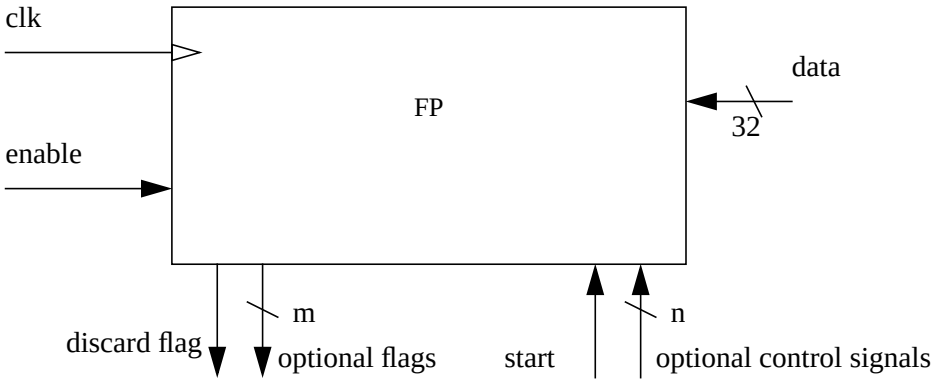


Figure 7.5: General interface of a FP

interface to the FPs can be seen in figure 7.5. All signals and flags connect to the controller unit except the data and the clk. Below each FP is explained in somewhat more detail.

7.5.1 Ethernet checksum calculation FP (ECCFP)

The ECCFP receives a start signal and then performs CRC-32 calculation on all data passing through. In the end of the frame the FP will receive a frame end signal and compare the calculated value to the received frame check sequence. On non equality a discard flag is sent to the C&C.

7.5.2 Ethernet destination address extraction and comparison FP (EDAFP)

The EDAFP is configured with the address of the terminal were the GPPP is situated. The FP receives a start signal and extracts and compares the received address to the configured one and checks if the extracted address is a multicast address. If the frame is not addressed to this network terminal a discard flag is sent to the C&C.

7.5.3 Ethernet length/ethertype field extraction FP (ELTFP)

The ELTFP extracts the length/ethertype field. If an ethertype is given, the length is expected from the ITLFP. The value is distributed by the C&C. A counter keeps track on how much data that has been received. When the counter reaches the length value a frame end flag is sent. This FP also gives the ethertype value to the C&C so that special jobs, like ARP and RARP, can be handled correctly.

7.5.4 IP header checksum calculation FP (IHCFP)

The IHCFP is active if the IP version field is IPv4. It then calculates the checksum by performing 16-bit one's complement addition of the header fields and makes sure the result is 0. If not a discard flag is sent to the C&C.

7.5.5 IP version field extraction FP (IVFFP)

The IVFFP extracts the IP version field and sends a flag to the C&C telling which version of IP is used.

7.5.6 IP destination address extraction and comparison FP (IDAFP)

The IDAFP is configured with the terminal address for the application. The FP receives a start signal and IP version information and extracts and compares the received address to the configured one, it also checks if the extracted address is a multicast address. If it is an unrecognized address a discard flag is sent to the C&C.

7.5.7 IP header length extraction FP (IHLFP)

The IHLFP sends a flag when the IP header has been received. In IPv4 the IHL field specifies the length. In IPv6 the header is always 40 bytes plus optional extension headers. The extension headers, except fragmentation, in IPv6 are not processed, since they concern routers and management protocols.

7.5.8 IP total length extraction FP (ITLFP)

The ITLFP extracts the length field to send the length value to the ELTFP.

7.5.9 IP protocol/next header extraction FP (IPNFP)

The IPNFP extracts the protocol field from the IP header and sends a flag to the TCP-UDP FPs to tell if the present packet is TCP or UDP. If there exist extension headers in IPv6 packets these are skipped and the extension header length field is used to find out when the next header starts. This is done until a known header type is received. Known headers are TCP, UDP, ICMP, IGMP and ICMPv6.

7.5.10 IP reassembly FP (IRAFP)

The IRAFP extracts the fragment fields from IPv4 header and searches for a fragment extension header in IPv6. If fragmentation is present this FP manages payload data to be stored in memory on the right place and controls the TUCFP to process the right data. To assist the IRAFP memory tables and timers for reassembly are present.

7.5.11 TCP-UDP checksum calculation FP (TUCFP)

The TUCFP calculates the checksum by performing 16-bit one's complement addition of the whole packet, including some IP header fields. If the result is non zero a discard flag is sent to the C&C. Multiple back-up accumulator registers are used in order to be able to calculate checksums of multiple packets, since fragments of them may arrive nestled.

7.5.12 TCP-UDP packet length counter FP (TULFP)

The TULFP extracts the length value and provides this to the software of the host processor. The length is also needed for reassembly and checksum calculation.

7.6 The Ethernet Checksum Calculation Functional Page

This section provides a more detailed description of the ECCFP and describes the issues with an high-speed implementation. The ECCFP manages the cyclic redundancy check (CRC) of the Ethernet frame. The CRC is computed on the whole Ethernet frame and after that the frame check sequence (FCS) has been received, the result must be all zeros. The FCS has been calculated by the sender and is the remainder of the data divided by the CRC polynomial.

Various types of CRC implementations have been investigated [7.9], but for this instance of the GPPP only one type of CRC, with fixed polynomial, is needed. The fastest way to do this is to use a parallel implementation. The wider words that are used, the lower the clock frequency can be, but the complexity also grows with the word width. However, the complexity and thereby the critical path delay does not grow as fast as the word width increases so in general an architecture which calculates more bits in parallel will support higher throughput [7.10]. It has to be considered that data in Ethernet frames can be of any number of bytes and if more than 8 bits are computed in parallel the initial and final words may not be of the same width. We use a 32-bit parallel implementation, which is a good trade-off between complexity and performance. Like in [7.11] the register elements that are starting points of critical paths have been duplicated. This is done in a way so that the driving strength can be adjusted to minimize the delay. Combinational logic in the most critical paths has been pushed over the clock cycle boundary defined by the registers to a path with less latency. Before the output this has to be compensated for. In a 0.18 micron process technology implementation, the static timing analysis (STA) provide the result of 10.53 Gbit/s throughput operation for the ECCFP. The STA was done with extracted parasitics form the layout.

7.7 The TCP-UDP Checksum Calculation Functional Page

The other FP that possibly can limit the performance of the GPPP is the TUCFP. This FP has a relatively straight forward task for most packets. When the IP header arrives, the pseudo header must be extracted and the 1's complement addition is used to add up 16-bit words of the pseudo header and the IP payload.

The first observation, that complicates the operation a little bit, is that the required upper layer payload length is not present in the TCP header, so it must be calculated as the IP total length - IP header length. The IP header length is specified in units of 4 bytes, so it must first be left shifted two bit positions.

The second observation, that complicates the operation a whole lot more, is that fragmented IP packets must be dealt with. The fundamental problem is that the GPPP does layer 2-4 processing at each fragment when it arrives, i.e. some layer 4 tasks are performed before the IP packets are reassembled. For the TCP UDP checksum computation this means that the checksums for each fragment is calculated individually and when more fragments arrive they are combined with the already existing partial checksum. It has to be taken care of that the pseudo header is included exactly once in the computation and duplicated fragments must be discarded before computed. There is also a need for a time-out if all fragments do not arrive within a certain time limit. The TUCFP makes use of the IRAFP for some of this functionality.

The TUCFP has been implemented in VHDL and the result is that the operation in an 0.18 micron process technology can support transmission speeds of up to 11.55 Gbit/s [7.8]. The STA was done in the same manner as for the ECCFP.

7.8 The Controller and Counter Unit

Figure 7.6 shows the general structure of the C&C. The C&C has to manage high-level control

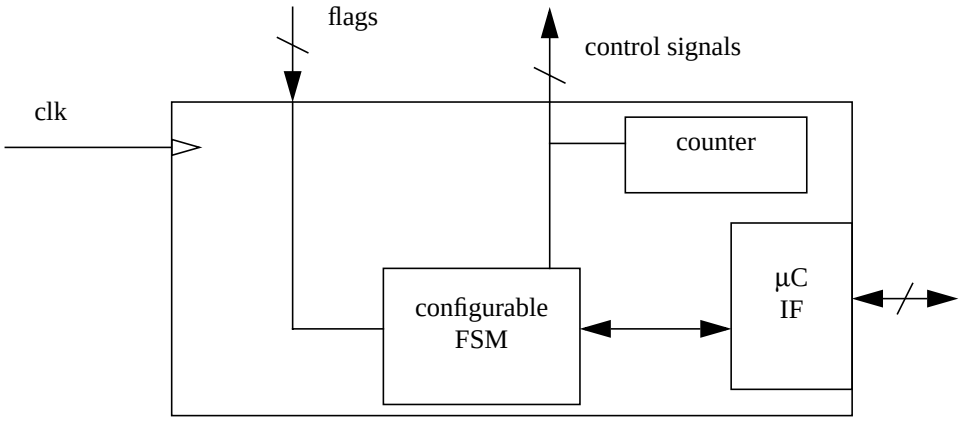


Figure 7.6: Controller and counter unit overview

only, since FP specific control is handled within each FP. It receives flags from the FPs, schedules the pipeline delay, and sends control signals to the FPs. The controller unit is based on a configurable finite state machine (FSM), which controls the discarding or delivery of packets depending on the flags it receives from the FPs. When a flag, that tells the C&C to discard a packet is received, all activities are switched off except for the PSU, which looks for the next frame.

If a packet is received without any problem, the C&C notifies the µC and tells it where in memory the packet can be found. The C&C also manages memory allocation and storing of payload with help from the µC.

7.9 Discussion

The proposed architecture uses extensive parallelism and configurable control to cut down the hardware redundancy and so the power- and time-consuming characteristics of a programmable processor. The critical path to the real-time speed limit has been found in the ECCFP. If a data width of 32 bits is used it is possible to support 10 Gigabit Ethernet. In a conventional solution, the CRC check is performed in the MAC controller, but all network- and transport-layer processing is performed by the host processor. The GPPP relieves the host processor from this burden which is of great importance as the network transmission speed increases. Other dedicated processors, but still program based, can solve the same tasks but suffer from much higher power dis-

sipation than the GPPP. Also other dedicated protocol processing solutions normally make use of a layer-based pipelining technique [7.7], which introduces latency. This latency is eliminated in the GPPP since all layers are processed concurrently.

7.10 Conclusions

By using the proposed architecture and control, a configurable GPPP is accomplished. The configuration-based architecture makes hardware reuse and wide functional coverage possible and moves unnecessary hardware design to a compiler. The control is partitioned into three different parts, FP internal control, the C&C and the μC. This partition simplifies verification and increases the flexibility and supports future changes in the protocols. The proposed architecture has higher performance and lower power dissipation than its competitors.

The project is under functional implementation phase, approaching to the delivery of the payload to the host processor. Studies are also being made concerning problems occurring when not buffering the whole Ethernet frame and how to solve the reassembly of IP packets in hardware.

Acknowledgments

This study was supported by the Intelect program of Swedish Foundation for Strategic Research (SSF). Authors would like to thank Dr. George Liu, Ericsson Research, for interesting discussions.

References

[7.1] LIU, D., NORDQVIST, U., SVENSSON, C.: ‘Configuration-based architecture for high speed and general-purpose protocol processing’, SIPS’99, Taiwan, pp. 540-547

[7.2] GERORGIOU, C. J., LI, C.-S.: ‘Scalable Protocol Engine for High-Bandwidth Communications’, IEEE Int. Conf. on Communications, 1997, Montreal, Towards the Knowledge Millennium. pp.1121-1126 vol.2 1997

[7.3] YANG, M., TANTAWY, A.: ‘A Design Methodology for Protocol Processors’, Fifth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems, 1995, pp.376-381

[7.4] HENRIKSSON, T., NORDQVIST, U., LIU, D.: ‘Specification of a configurable General-Purpose Protocol Processor’, CSNDSP 2000, Bournemouth, U.K., pp. 284-289

[7.5] TANENBAUM, A. S.: ‘Computer Networks’, 3rd Ed., Prentice Hall PTR, ISBN 0-13-349945-6, 1996

[7.6] KADAMBI, J., CRAYFORD, I., KALKUNTE, M.: ‘Gigabit Ethernet’, Prentice Hall PTR, ISBN 0-13-913286-4, 1998

[7.7] KAISERWERTH, M.: ‘The Parallel Protocol Engine’, IEEE/ACM Transactions on Networking, vol.1 No.6 December 1993 pp. 650-663

[7.8] 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

[7.9] NORDQVIST, U., HENRIKSSON, T., LIU, D.: ‘CRC Generation for Protocol Processing’, Norchip 2000, Turku, Finland, pp. 288-293

[7.10] PEI, T.-B., ZUKOWSKI, C.: ‘High-speed parallel CRC circuits in VLSI’, IEEE Transactions on Communications, vol. 40 Issue 4 April 1992, pp. 653-657

[7.11] HENRIKSSON, T., ERIKSSON, H., NORDQVIST, U., LARSSON-EDEFORS, P., LIU, D.: ‘VLSI Implementation of CRC-32 for 10 Gigabit Ethernet’, ICECS 2001, Malta, pp. 1215-1218

Appendix: List of Jobs

Ethernet/802.3 CRC check, Ethernet/802.3 destination address check, Ethernet/802.3 payload protocol determination, IP version determination, ARP/RARP recognition, IPv4/IPv6 destination address check, IPv4 header checksum check, IP reassembly support, IP payload protocol determination, TCP packet length determination, and TCP/UDP checksum check.