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.

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.

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

General architecture proposal

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

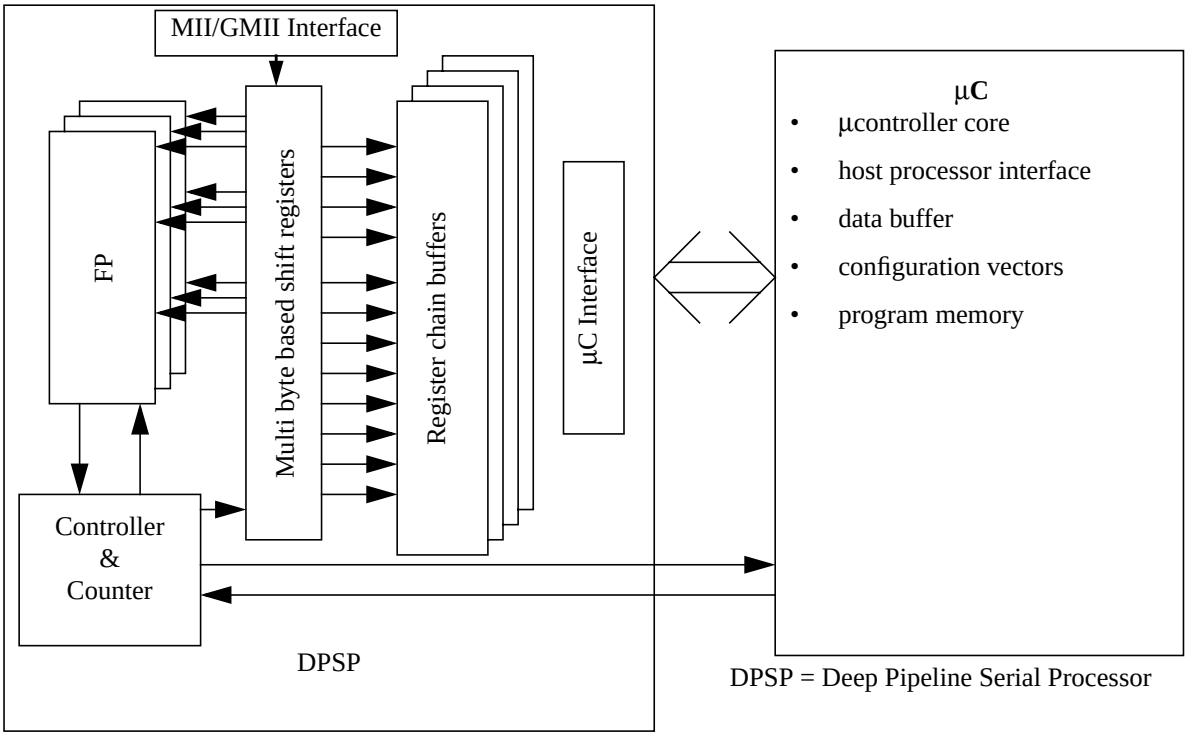

Figure 1:Overview of the architecture, the FPs perform the actual protocol processing

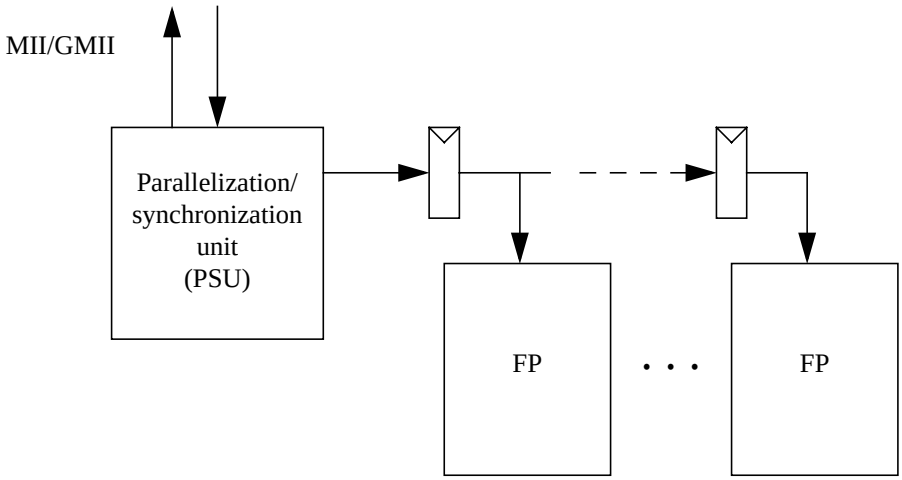

Figure 2:Data is synchronized and parallelized, thereafter one pipeline register is situated between every FP to decrease the fan-out requirements

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.

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

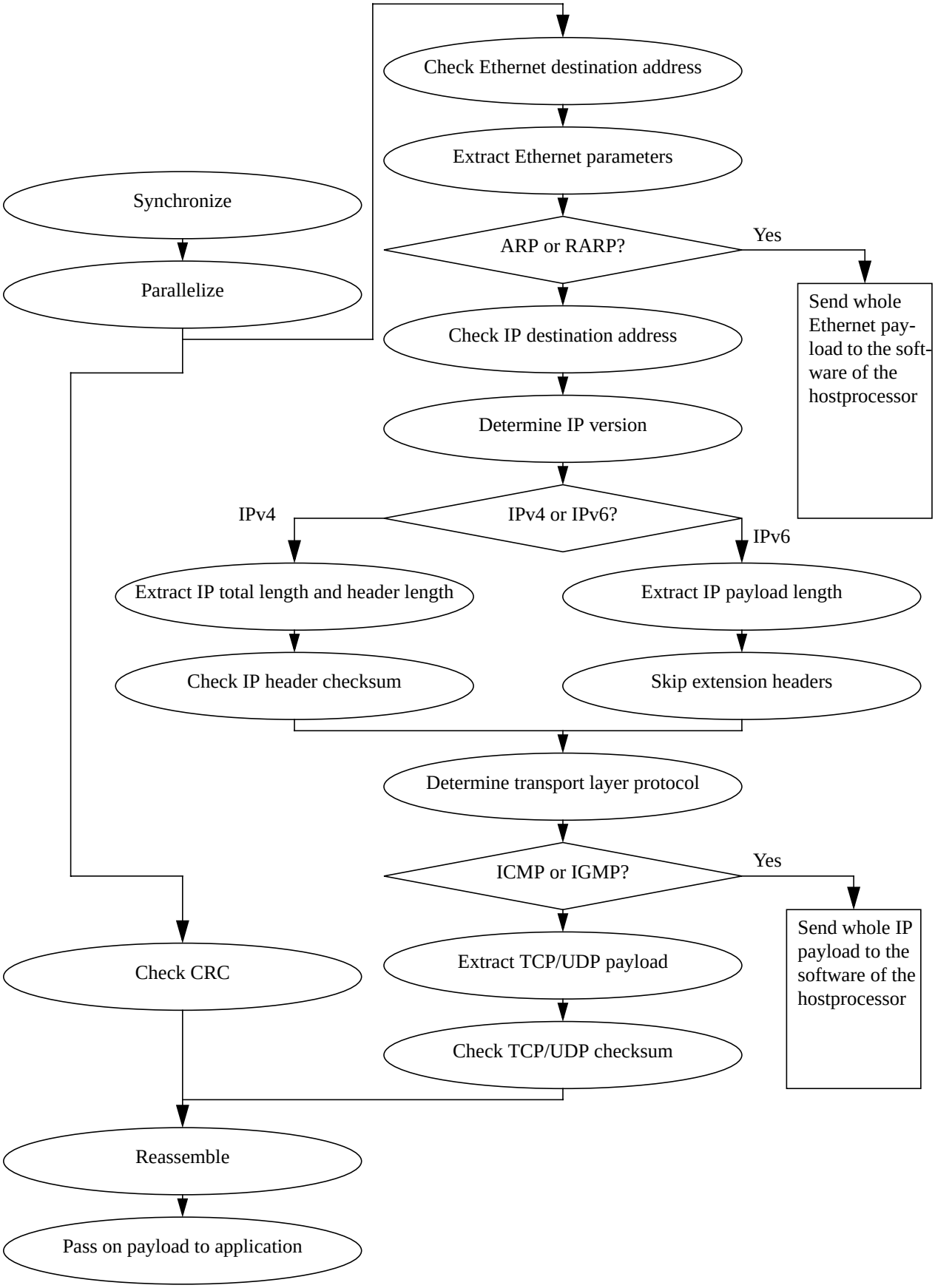

Figure 3:Flowchart that illustrates the operation

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.

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.

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 control-

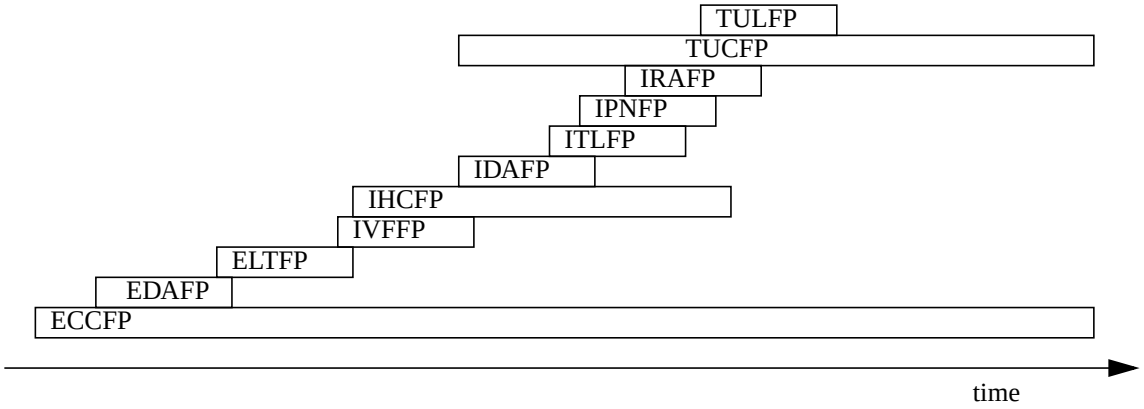

Figure 4:Principle scheduling of FPs for an TCP on top of IPv4 example. The boxes show when the FPs are active. Job abbreviations are specified later in this paper.

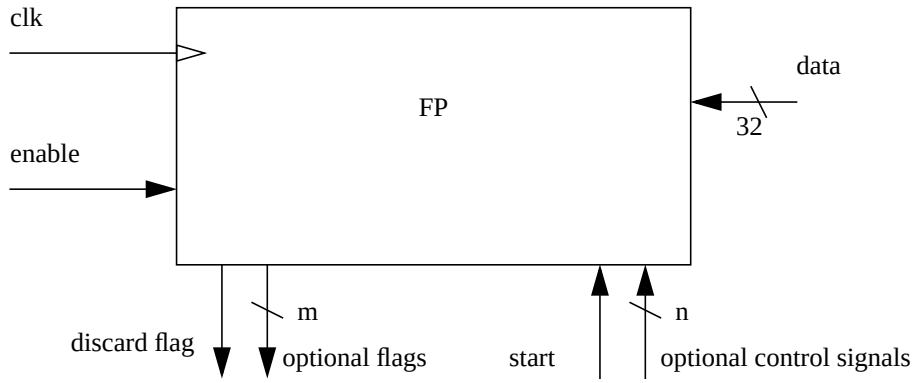

Figure 5:General interface of a FP

ler unit except the data and the clk. Below each FP is explained in somewhat more detail.

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.

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 multi-cast address. If the frame is not addressed to this network terminal a discard flag is sent to the C&C.

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.

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.

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.

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.

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.

IP total length extraction FP (ITLFP)

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

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.

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.

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.

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.

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.

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.

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.

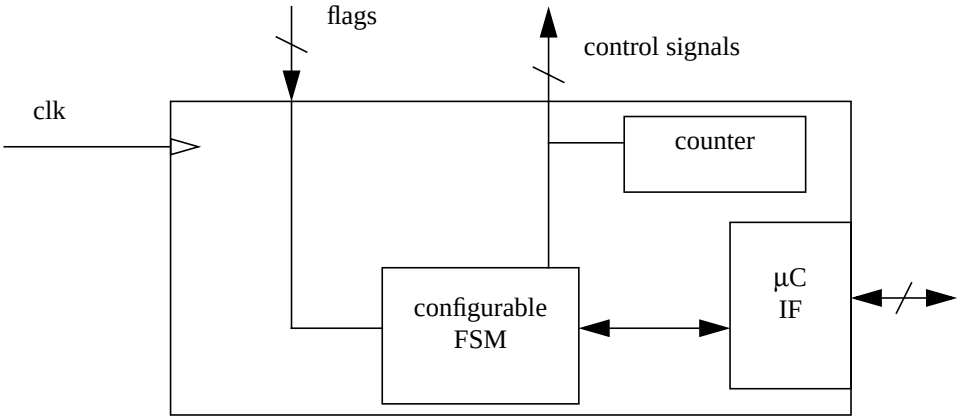

Figure 6:Controller and counter unit overview

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.

Conclusions

By using the proposed architecture and control, a configurable GPPP is accomplished. The configuration-based architecture makes hardware reuse and wide functional covery 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

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

[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

[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

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

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

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