On Protocol Processing

Ulf Nordqvist
Dept. of Physics and Measurement Technology
Linköping University
Sweden

Abstract— **The topic of this paper is the emerging area of protocol processing. The use of protocol processing is rapidly increasing today when more and more networks for computers and others are coming into use. Emerging high speed networks puts very high demands on the hardware which is going to process the packets in order to get the vital information. The result is that software processes running on a standard CPU will not be able to process the packets at wire-speed. Therefor expensive and power consuming memories has to be used. To avoid this problem protocol processors is used as a hardware platform specialized on processing of these packets and their protocols. The aim of this paper is to give the background too, and an overview of the novel academic research area called protocol processing.**

I. Introductory outline

The aim of this paper is to present an up-to-date overview on the area of protocol processing in general and specifically the protocol processor proposed by the author and his research team. The protocol processor is presented as a solution to the limitations and problems with todays traditional protocol processing. In section 2 the academic research area as well as the industrial area of protocol processing are introduced. The section also contains a presentation of the implementation methods, which traditionally have been used for processing of protocols. The drawbacks and limitations of these implementation methods are also discussed. Finally an overview of the applications, with their respective protocols, that might especially benefit from use of a protocol processor, is presented. In section 3 a relatively novel method for integration of large scale systems into one chip is presented. Moreover the integration of a protocol processor into such a system is motivated and discussed. In section 4 a protocol processor that can meat tomorrows high requirement is presented. The application area and some examples of the central parts of this processor are then presented in section 5.

II. Protocol Processing

As a result of the explosive growth of computer networking and the Internet, network protocol processing is performed in more and more situations. There is also a strong development towards an increased use of networks protocols in cases where other techniques were common earlier, e.g. voice and video. One reason is that network protocols normally can handle a mixture of any kind of traffic.

Today, protocol processing in computer networks is performed either on standard Central Processing Units (CPU) or by one or several Application Specific Integrated Circuits (ASIC). Most processing of protocols are performed in software, running on standard computers with their standard, general purpose CPUs, or on embedded CPU cores. Such implementations are expensive in cost, space and power, because of the lack of dedication in a general CPU. There is also an upper capacity limit, set by the I/O capacity and the maximum instruction rate of the CPU.

The alternative solution is to implement the protocol processing in one or several ASICs. This may give solutions with better cost, power and capacity performance, at the cost of very low flexibility. Even if ASIC solutions are quite common today, the low flexibility becomes less and less acceptable. The development time of the ASIC solution is large and product debugging or upgrading demands a very slow and costly hardware redesign. We think that the solution to these problems is an application specific programmable protocol processor. This solution is also supported by [1].

A. What is new?

Internet traffic is doubling every six months. New network services such as Voice over IP (VoIP) and policy-based Quality of Service (QoS) result in demands for increasing processing performance. At the same time, high speed protocols are becoming more and more common in the computer networks and in a not to distant future the network terminals (NT) will use protocols such as 10 G-Ethernet (10 Gbit/s). Using such a protocol results in a tremendous amount of data, that streams in and out of every network terminal. All data have to be processed before it is used by the application on the receiving terminal. If we then want to use the increased bandwidth provided by the network, the NT has to increase its computational load. Another example of the increased demands on high throughput is the 3G mobile phone application where a baseband processor will have to manage a couple of hundreds Mbit/s. Balance between on one hand demands on increasing speed and complexity (computational demands) and on the other hand power consumption, reusability, die size etc, is then the hot issue for tomorrows protocol processing hardware resources. Today it is clear that the traditional solutions are not capable of meeting those requirements. That is what is new.

B. Traditional Protocol Processing Solutions

Traditionally protocol processing has been performed on a heterogeneous platform with two different appro-

Ulf Nordqvist is a Ph.D. student at the Electronic Devices group, Dept. of Physics and Measurement Technology, Linköping University, Sweden. E-mail: ulfnor@ifm.liu.se

aches to protocol processing.

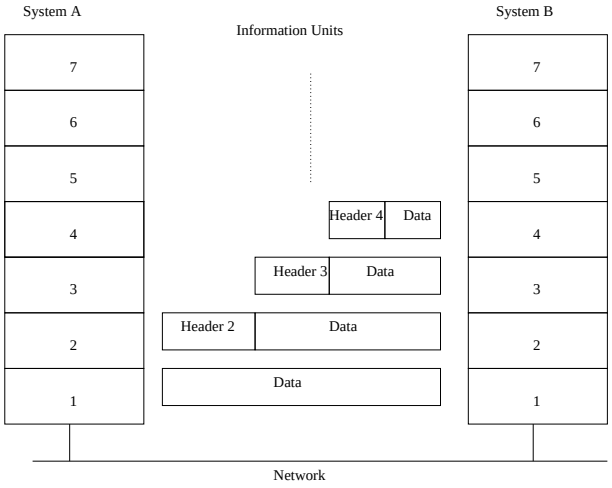

Fig. 1. The information exchange process is defined in the OSI standard. As illustrated by the figure there are a lot of headers transfered in a computer network. Each layer communicates with the same layer on the communicating machine.

• The lower two layers, called Physical and Data Link Layer in the OSI protocol stack [2], have traditionally been processed by dedicated hardware (i.e. one or more ASICs). The layered information exchange process standardized by the OSI standard, is illustrated in figure 1. The main advantage using ASICs as protocol processing unit is the performance. The problems with ASIC solutions are today obvious. First of all the ASIC lacks configurability, meaning that it is impossible to change the functionality of the ASIC without replacing the hardware. The result is then that the user has to buy a new network card every time the network protocols are updated or new protocols are being adapted. That costs a lot of money and time. On the other hand, using ASICs is the only way of designing the high speed backbone network that customers today requires. In a few years time the same computational load will be required also for NTs. For network node components such as routers, bridges etc the configurability needs are not as crucial as it is for NTs. The reason to why it is still possible to implement everything using ASICs in the backbone networks is simple. The customers to such equipment are not very cost sensitive. On the other hand, the time-to-market demands from the networking industry are becoming very hard to meet. This means that also for the network node components, novel implementation methods have to be adapted. The long time-to-market and the short time-on-market are probably the most serious drawbacks with the ASIC approach.

• The upper layers (3 and above in the OSI stack) have traditionally been processed in software, running on the host machine where the source or destination application also runs. The primary advantage using a general CPU as protocol processor is the CPUs infinite SW resources. Upgrading and modification of the processor can then simply be done by downloading a new program. This extends the time-on-market which is very important. On the other hand, the CPU that processes the upper layers cannot process them at the speed of the emerging network protocols, due to its general nature. The situation is getting worse and worse with the increasing amount of communication that has to be handled by each network terminal. When the next generation high speed networks is coming into use, todays very general CPUs have to give up. At those speeds buffers would get filled very quickly if the packets where not processed at the same rate as they arrive. Since the CPU cannot handle these speeds even for the processing of the layer 3 and above, the conclusion must be that a bigger part of the processing tasks has to be handled by dedicated hardware.

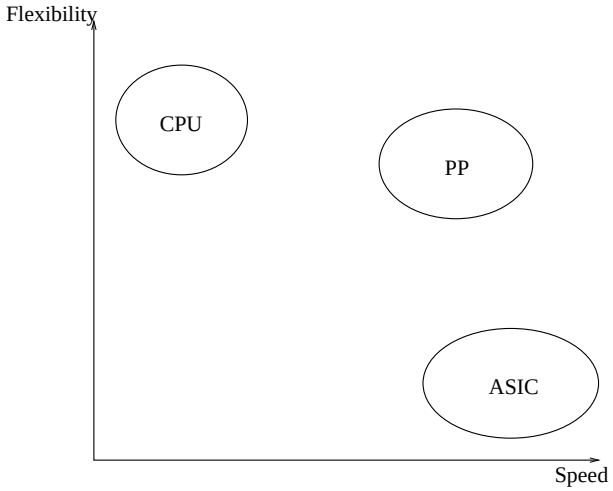

Fig. 2. The Protocol Processor tries to combine the flexibility of a CPU and the speed of an ASIC in the same hardware.

In the last couple of years, the first generation of protocol processors has been presented by the industry. One of those processors was presented in [3]. They normally consist of some specialized hardware for the physical layer and a couple of very powerful General Purpose CPUs which process the upper layers. All are then integrated on the same chip and therefor they can together process several Mpackets/s. Such solutions are mostly aiming for the backbone applications and especially routers. These solutions have clarified the need for novel solutions for protocol processing also in NTs. Another very important issue, that the industry solutions have pointed out, is the programming environment. It is absolutely necessary to have a high level compiler included in the system design so that the user does not need to program the PP using micro-code.

Conclusion: The key issue for successful protocol processing in a NT is to be able to place all the toughest and processing intensive jobs on dedicated hardware. The reason is that dedicated hardware will always have superior cost, power and performance characteristics according to figure 2.

C. Application and Protocol examples

More and more network communication becomes packet-based. One example is Voice over IP (VoIP) which basically is ordinarily phone calls using the Internet to communicate via IP packets. Another emerging application area is residential Local Area Networks (LAN) using wireless communication. The protocols used for those applications could be Bluetooth, WLAN or Hiperlan [4]. In practise any application including digital communication, packet based or not, can be efficiently processed by dedicated hardware in the form of a protocol processor. Of course the traditional network protocols such as TCP, IP, UDP, RARP, ARP, IEEE 802.X [5] as well as ATM is interesting to include in the protocol coverage of a protocol processor, due to their widespread use. The core issue when specifying and designing a protocol processor is to find the most useful set of protocols that can run on the same hardware.

III. System On a Chip

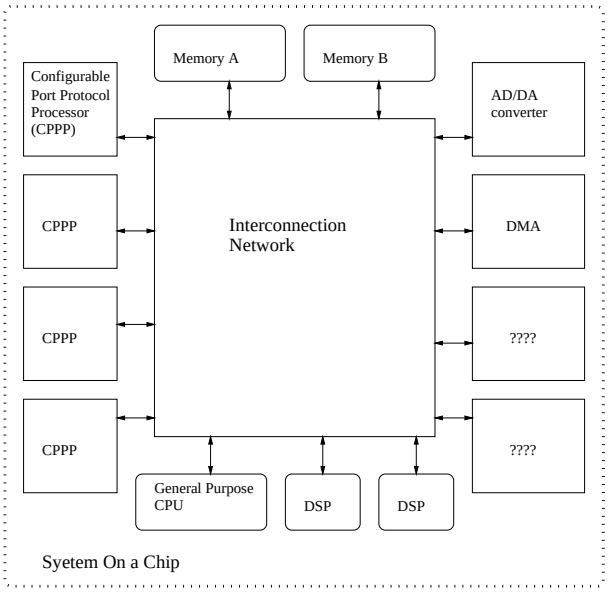

Fig. 3. A System on a chip (SOC) can include any kind of hardware and software on the same chip.

With the scaling, or down-sizing, of the transistors in modern manufacturing processes it is possible to put more and more transistors into a chip. The well known Moores Law says that the number of transistors per area unit doubles every 18 month. This gives a system designer a lot of opportunities. It is today possible to put CPU, memories, a couple of DSPs (for graphic, audio, protocol processing etc), analog parts and various kinds of custom-made hardware blocks (a.k.a. Intellectual Properties (IP)) into one single chip. One of these so called System On a Chip (SoC) is shown in figure 3. By keeping the communication on-chip, the memory bandwidth can be very high while the power consumption from the drivers of the I/O buffers, is deleted. By including analog parts on-chip, the sensitivity to noise is decreased. The use of a CPU on-chip gives the solution flexibility through its programmability. All these benefits is there today so there is no question about that SoC is the solution for the next generation of protocol processors.

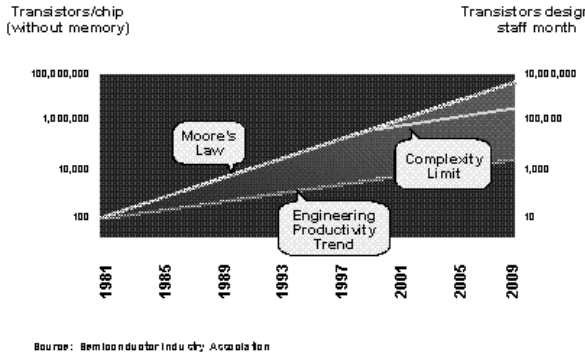

Fig. 4. Moores law is a well known description of the scaling in CMOS circuits. The complexity in VLSI circuits in terms of number of transistors that can be integrated on a chip increases at the rate of 58% per annum compounded. Whereas design productivity increases at 21% per annum. This gap is called the design crisis and is crucial to close in order to reach the optimal design.

Time-to-market is today one very important parameter if a system is going to be economically successful. Further it is not possible to keep up with Moores law and take advantages of the increased integration possibilities if not old designs are reused. Thats why the design flow using IP reusability is growing very fast. Reusing the IP normally means that an old IP (e.g. a processor, DSP or some other design) is either bought from an external designer or picked from one of the designers old project, and then integrated into a new design together with new logic. This way, redesign of a hole new chip can be avoided. Reusing IPs also reduces the verification time (cost) and increases the chance for a successful first design since the IP is already verified.
In order to be able to reuse an IP and use it in a new SoC, it is necessary that the IP is flexible and have a well defined interface to the rest of the SoC. Since the verification time can be as high as 3/4 of the total time spent on the project, reuse is very important for the cost efficiency of the designs [6].

IV. A Configurable Protocol Processor

Our protocol processor is targeted for use in a network terminal. To begin with the project has been focused on downstream processing, i.e. on the receiving terminal. To provide both compatibility and flexibility the architecture must be able to handle at least the most frequently used protocols. There are no problems to later in the project, include more protocols. In order to develop an optimal protocol processor it is necessary to find out which protocols are suitable for execution on such.
According to figure 5 it is possible to divide the tasks that should be performed in the NT into two groups. Our protocol processor targets on the deterministic jobs which

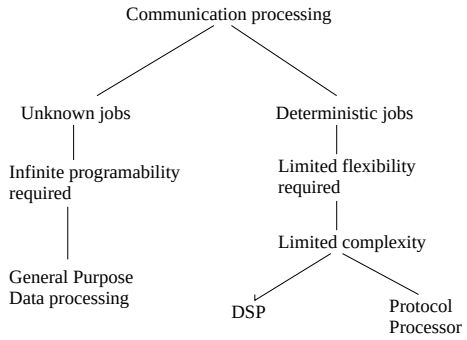

Fig. 5. Some jobs is requires infinite programmability which makes them unsuitable for protocol processors.

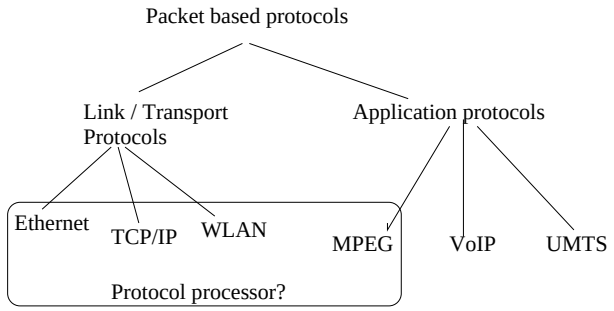

Fig. 6. The protocol coverage of our processor is not finally specified.

require an limited flexibility and programmability. Some of the tasks which are deterministic are still not suitable for our processor. They can instead be left to other dedicated hardware in the system, e.g. to a Digital Signal Processor (DSP).

This paper has shown that the main reasons for using a protocol processor instead of a CPU are:

- High performance (Throughput)
- Low power by reduction of memory access or data processing.

This means that protocols that do not require high speed on-the-fly processing or are too complex for such, are not very suitable for protocol processing and might therefor be processed on another processing resource. The power reduction in the protocol processor compared to an implementation using a general CPU, is very important for protocol processing in portable devices (i.e. mobile phones etc). All this has to be considered when deciding the application coverage of the protocol processor, e.i deciding which protocols to include in the PP and which should be handled by other processing resources.

Figure 6 describes the different protocols and their suitability for processing our protocol processor. The application protocols are normally not very suitable for processing on a protocol processor. They should instead be processed by for ex Digital Signal Processors (DSP) or in SW running on the platform. On the other hand some application protocols as for example MPEG can be parsed by a PP so that different data streams (Video or Sound) can be ported to different buffers before the hand over to the platform is done. This would relax the requirements on the platform. In this project the specification of the application/protocol coverage is yet to be finally decided.

A. Architecture

Our protocol processor is a non-Von Neuman solution, named Configurable Protocol Processor (CPP). The CPP consists of two parts. The first part, named Deep Pipeline Serial Processor (DPSP), processes the data in a high speed way and is based on a 32 bit wide data path. It is in the DPSP the actual protocol processing takes part. The DPSP is configurable using a common micro-controller (μC). The μC also configures the interface between the DPSP and the application. The μC only controls macro jobs, i.e. configuring the protocol processor for the current protocols and other network parameters while the DPSP after parsing of an incoming message configures itself for the specific protocol in run-time.

The interface to the physical layer is the MII/GMII [2] and the interface to the application is in the middle of the TCP-UDP layer. A system description of the protocol processor can be seen in figure 7. The DPSP is based on Functional Pages (FP) that each manage one small task of the protocol process. Examples of such tasks are checksum calculation and header field extraction. The DPSP data flow is illustrated in figure 8.

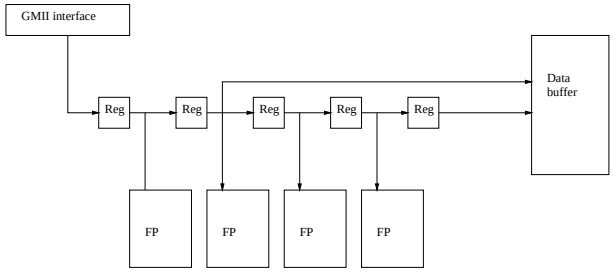

Fig. 8. The data flow in the protocol processor.

The normal protocol processing procedure of the DPSP is as follows:

- Data comes from the MII/GMII interface [2] and is then pipelined in a multi byte shift register.
- Protocol header fields are then provided to concerned FP:s.
- The FP:s are then activated on control signals from the Counter and Controller (CC).
- The FP:s reports back to the CC sending flags.
- The CC shuts down FP:s according to the flags and the configuration.

The flexibility of the CPP is essential in order to manage all possible protocols that exist in the network environment. The CPP is configurable according to the procedure illustrated by figure 9.

The CPP is intended to work as a port processor in a SoC. Port processor means that the CPP is responsible for the protocol processing of one port (physical port, not a TCP port). The CPP then processes incoming packets before the data is passed on to the application, normally via

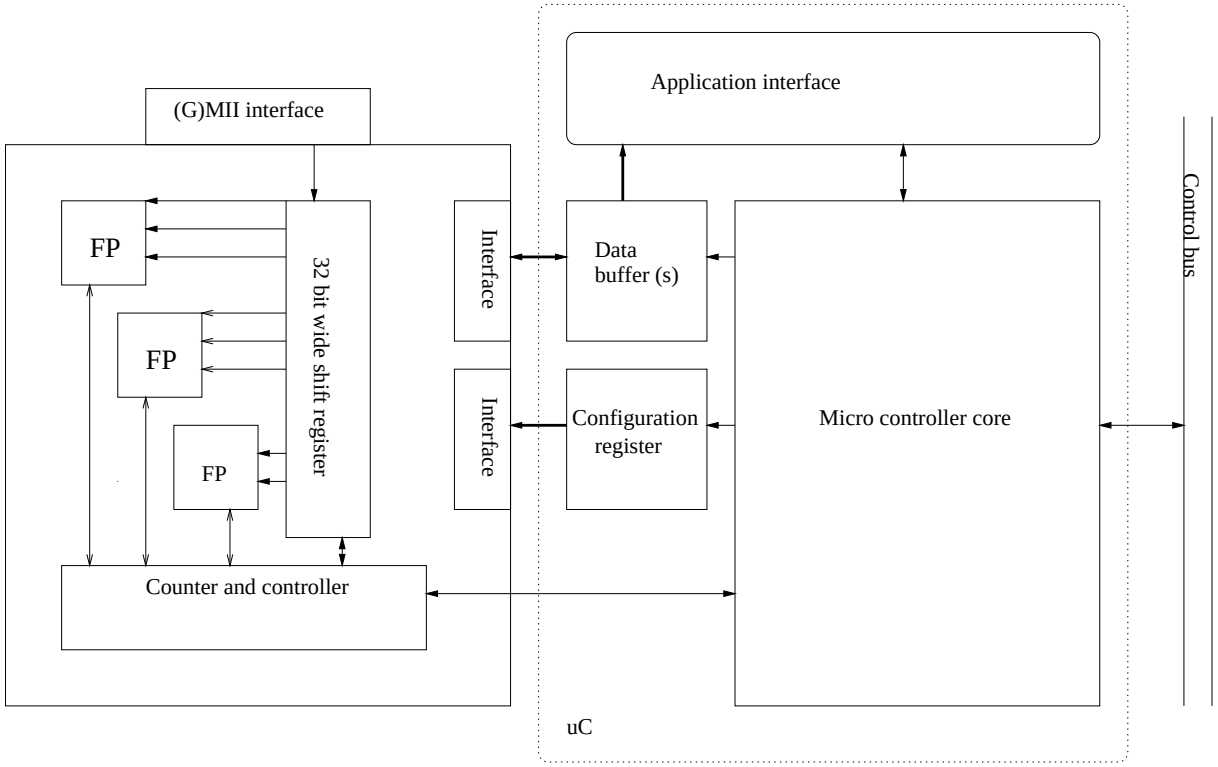

Fig. 7. The system block diagram for the protocol processor.

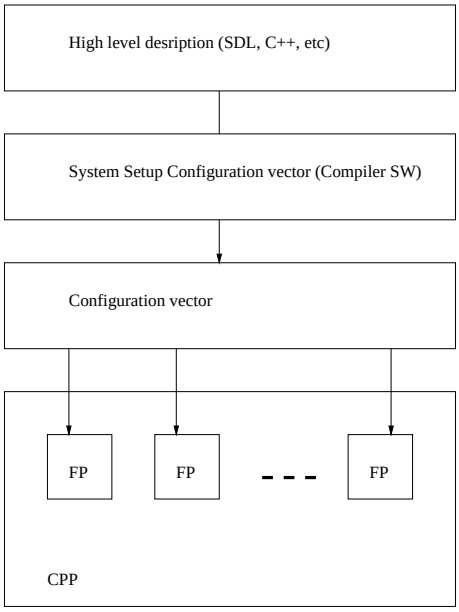

Fig. 9. The system setup configuration procedure for the protocol processor.

memory buffers as illustrated by figure 10. If there are any protocol processing tasks not suitable for the CPP to process, then the packets data and some selected information from the header is sent to the platform which processes the jobs. The programmable platform in the SoC is responsible for the tasks not suitable for the domain specific CPP. This way the infinite programmability is provided.

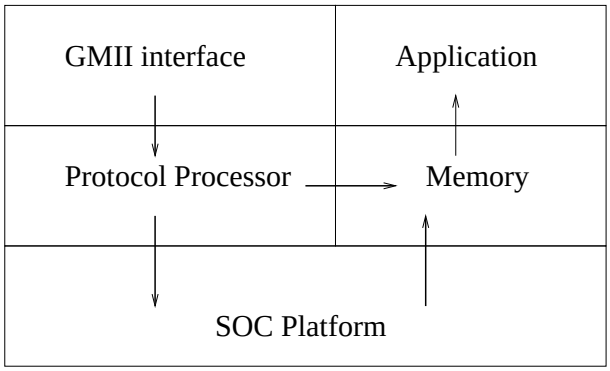

Fig. 10. Platform based processing on the receiver end. The data/information is passed between the different parts according to the arrows when a packet is received.

V. Functional Pages

A. Research Design flow

We can for research purpose develop each FP as if there was some sort of gray layer between the FP:s and the rest of the CPP. This is illustrated by figure 11. This gray layer assumption allows the designer to investigate and develop dedicated hardware for some small protocol processing tasks, without having a clear interface against the application layer specified in advance. It also means that we can assume that the data is provided to the specific functional page at the right time by the controller, even if the controller is not finally specified and implemented yet.

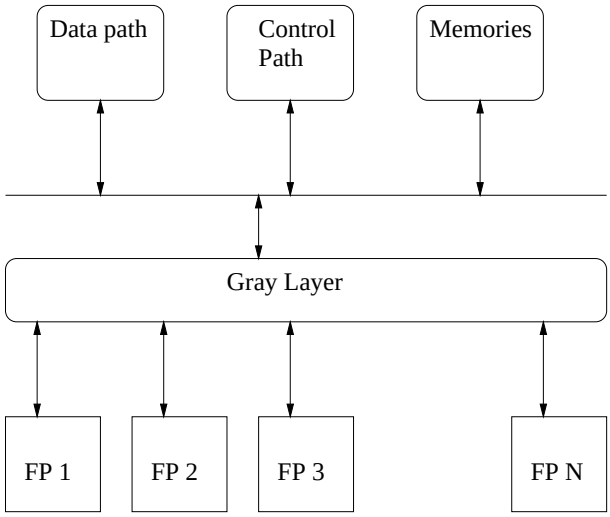

Fig. 11. The development of Functional Pages is stand alone from the overall processor structure. This means that memory management, I/O handling, data scheduling etc is not considered while designing the FP.

B. Some typical tasks for the functional pages

The main applications for our protocol processor is different types of Ethernet, with IP/TCP-UDP on top. These protocols are processed by different FPs. To cover the protocols IP/TCP-UDP, also ARP, RARP, ICMP and IGMP have to be managed. Packets of the later, control oriented protocols, are not that common and there is no need to design specialized hardware for them. Instead of processing them in FPs the functions can be performed in software on the platform with a relatively small total overhead.

Following is a list of typical jobs that is processed by our protocol processor. This list was first published in [7].

- Ethernet checksum calculation
- Ethernet destination address extraction and comparison
- Ethernet length/ethertype field extraction
- IP header checksum calculation
- IP version field extraction
- IP destination address extraction and comparison
- IP header length extraction
- IP total length extraction
- IP protocol/next header extraction
- IP reassembly
- TCP-UDP checksum calculation
- TCP-UDP packet length counter

C. CRC

The most computational demanding task for the CPP when processing TCP/IP over Ethernet or Wireless, is without any doubts the Link layer checksum calculation [8]. Therefor dedicated hardware for this task has been designed as a FP. The task of the FP is to check if the transmission of the packet was erroneous. This checksum calculation used, is known as Cyclic Redundancy Check (CRC). The CRC encoding/decoding procedure is described by equation 1.

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

V(x) is the n bit long data word transmitted and it consists of the original data and U(x) followed by a codeword S(x) called the CRC-sum. S(x) is computed according to equation 2.

$$x^{n-k}U(x) = a(x)g(x) + S(x) \tag{2}$$

S(x) is by other words the remainder resulting from a division of the data stream and a generator polynomial g(x). The actual coding-procedure is the same on both the receiving and transmitting end of the line as can be seen in figure 12. The CRC FP has been implemented using VHDL

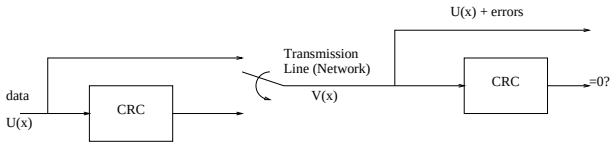

Fig. 12. The receiving NT perform a CRC-check on the incoming message and if the result is zero, the transmission was error free.

and it can manage more then 1 Gbit/s and it supports both Ethernet standards as well as HiperLAN. The layout is shown in figure 13.

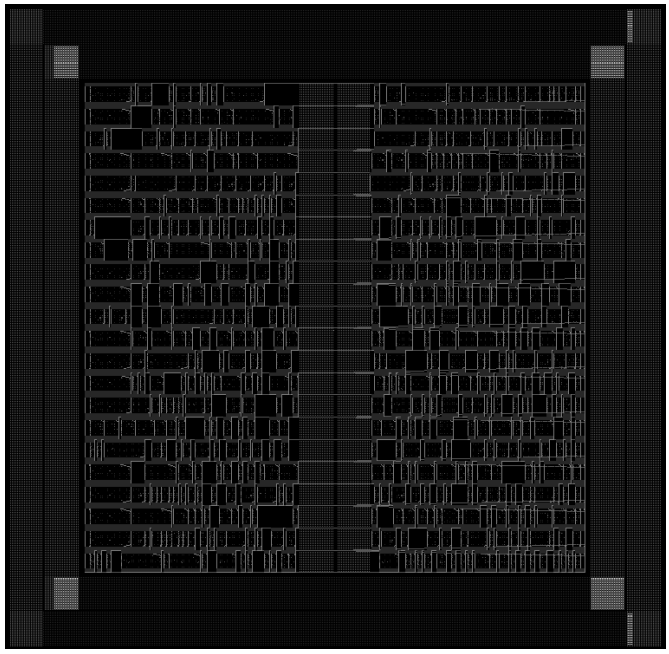

Fig. 13. The configurable CRC implementation combines the requirements on performance, low power consumption and configurability. The technology used is AMS 0.35 μm. The size of the design is 0.052 square mm.

VI. Future work

Memory access is very time and power consuming. The solution for reduction of these drawbacks is to use on-the-

fly processing. On the other hand not all protocols are suitable for streaming based processing due to complexity. By other means a thorough investigating of the protocol coverage has to be done in order to find out which protocols are suitable for processing on-the-fly and which are suitable for general purpose processing. Moreover it has to be clarified how the scheduling of the on-the-fly processing tasks are influenced by the use of non-streaming based processing of other jobs in the protocol.
It is very important to set up a simulation environment for the project in order to verify the functionality of the CPP using real network data.
The final goal of the project is to generate a synthesizable RTL description of a complete CPP.

VII. Acknowledgments

This protocol processor research project is sponsored by ECSEL graduate school in Linköping, Sweden.

References

[1] A. Jantsch, J. Oberg, and A. Hemani, "Is there a nich for a general protocol processor core?," in *Proc of the NORCHIP*, 1998.
[2] Jayant Kadambi, *Gigabit Ethernet*, Prentice Hall PRT, 1998.
[3] Dan Dobberpuhl, "Network protocol processing using high performance cpus in an integrated soc platform," in *Proc of the 18th NORCHIP conference*, 2000.
[4] ETSI TS 101 493 - 1/2, *Technical specification of BRAN and Hiperlan-2*, 2000.
[5] Andrew S. Tannenbaum, *Computer Networks*, Prentice Hall PRT, 1996.
[6] G. Pulini and D. Hulance, "Accelerated verification of soc designs containing ip cores," in *Proc of the Int. workshop on IP-Based Synthesis and SoC Design*, 2000.
[7] T. Henriksson, U. Nordqvist, and D. Liu, "Specification of a configurable general-purpose protocol processor," in *Proc of the CSNDSP, Bournemouth, UK*, 2000.
[8] U. Nordqvist, T. Henriksson, and D. Liu, "Crc generation for protocol processing," in *Proc of the 18th NORCHIP conference*, 2000.