Specification of a configurable general-purpose protocol processor

T. Henriksson, U. Nordqvist and D. Liu

Abstract: A general-purpose protocol processor is specified with a dedicated architecture for protocol processing. The paper defines a functional coverage, analyses the control requirements, and specifies functional pages and a controller unit. The general-purpose protocol processor is for network terminals, and 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.

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, and 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 being the single protocol ASIC (application specific integrated circuit) without flexibility and the other 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 [1–3].

The aim of this paper is to extend the specification of a protocol processor [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.

IEE Proceedings online no. 20020443

DOI: 10.1049/ip-cds:20020443

Published online 18 July 2002. Paper first received 14th September 2001 and in revised form 5th April 2002

The authors are with the Department of Electrical Engineering, Linköping University, SE-581 83, Linköping, Sweden

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 (Internet protocol/transmission control protocol-user datagram protocol) [5] on top. This can be seen as an instance of the general-purpose protocol processor (GPPP); see Fig. 1. 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 manufaturing 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 (media independent interface/gigabit media independent interface) [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 (address resolution protocol), RARP (reverse address resolution protocol), ICMP (internet control message protocol) and IGMP (internet group management protocol) have to be managed. Packets of these control oriented protocols are not that common and there is no need to design specialised 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 recognised and then passed on to the host processor. The complete list of jobs can be found in the Appendix.

3 General architecture proposal

The proposed architecture is shown in Fig. 1. The GPPP consists of two parts: a deep pipeline serial processor

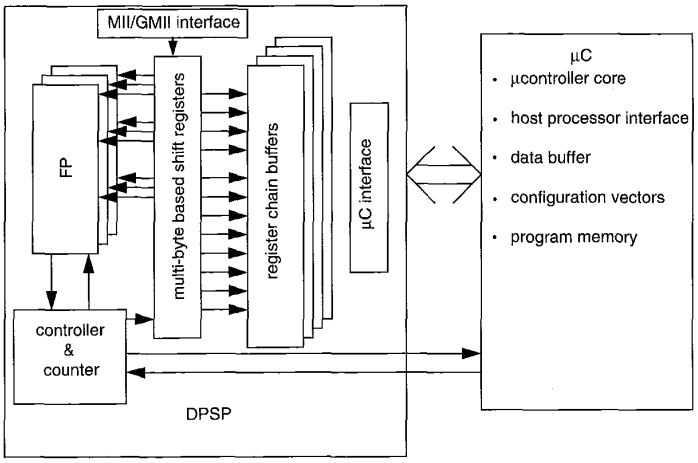

Fig. 1 *Overview of architecture*
The FPs perform the actual protocol processing
DPSP = deep pipeline signal processor

(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 configuration vector for the complete execution of an 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 parallelisation/synchronisation unit (PSU); see Fig. 2. Data are 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 mostly avoided; all FPs are controlled by the C&C and send flags

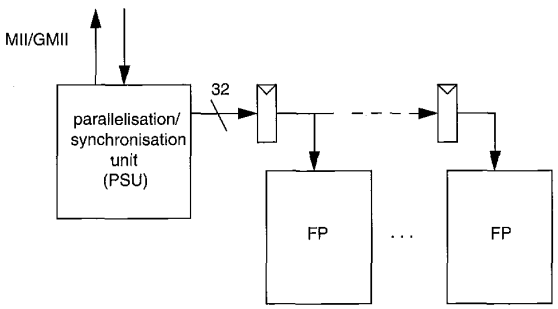

Fig. 2 *Data are synchronised and parallelised; thereafter one pipeline register is situated between every FP to decrease fan-out requirements*

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.

4 Control requirements

Protocol processing is a control intensive operation with different processing tasks and heavy data dependency, see Fig. 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 on the contents of that packet.

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 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.

4.2 Peripheral control

The payload has to be delivered to the application software of the host processor in a configurable way. This is taken

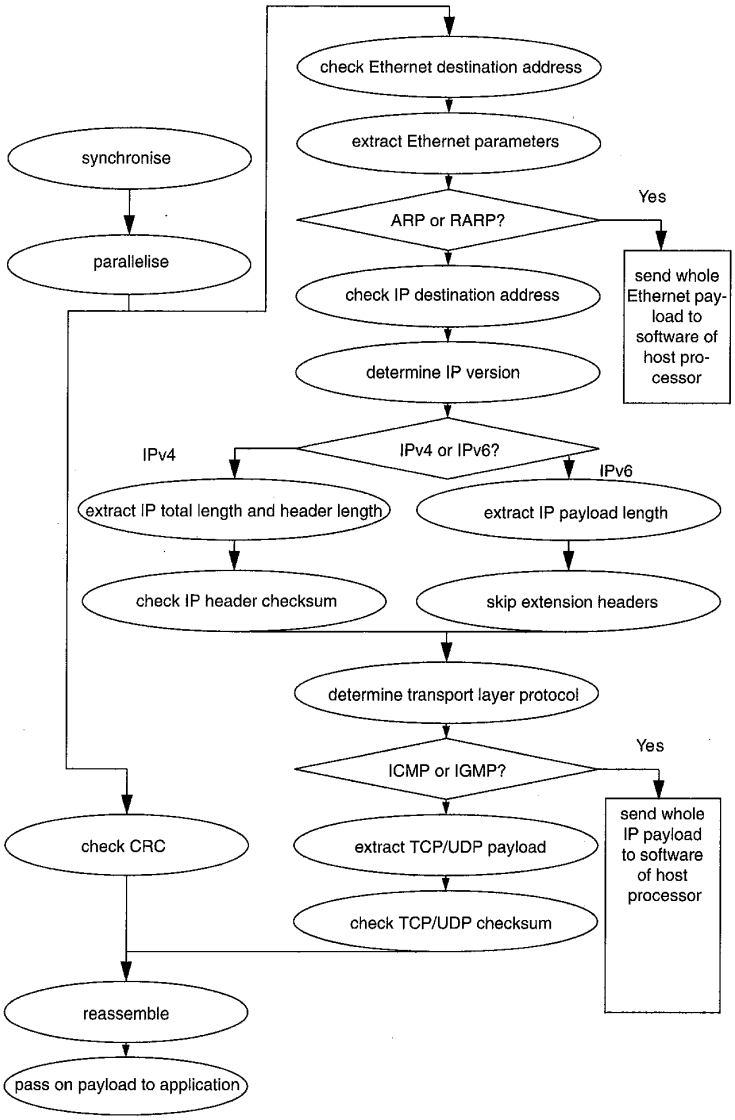

Fig. 3 *Flowchart that illustrates operation*

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.

5 Specification of functional pages

Fig. 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 are pipelined the concurrency is dependent on how the FPs are placed along the pipeline. An example of the scheduling is shown in Fig. 4. FPs will be placed in order to get the shortest possible pipeline and tightest scheduling. The interface to the FPs can be seen in Fig. 5. All signals and flags connect to the controller unit except the data and the clk. Each FP is explained in behavioural level detail below:

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

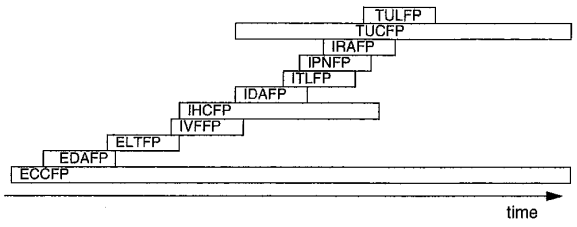

Fig. 4 *Principal scheduling of FPs for a TCP on top of IPv4 example*

the calculated value to the received frame check sequence. On nonequality a discard flag is sent to the C&C.

5.2 Ethernet destination address extraction and comparison FP (EDAFP)

The EDAFP is configured with the address of the terminal where 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

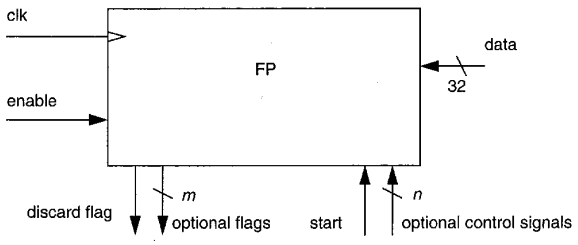

Fig. 5 *General interface of an FP*

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

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.

5.4 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 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.

5.6 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 unrecognised address a discard flag is sent to the C&C.

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.

5.8 IP total.length extraction FP (ITLFP)

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

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.

5.10 IP reassembly FP (IRAFP)

The IRAFP extracts the fragment fields from the 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.

5.11 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 nonzero 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 nested.

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.

6 Ethernet checksum calculation functional page

This section provides a more detailed description of the ECCFP and describes the issues with a 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 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], 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 the 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 [8]. 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. As in [9] 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 minimise 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 μm process technology implementation, the static timing analysis (STA) provides the result of 10.53 Gbit/s throughput operation for the ECCFP. The STA was done with extracted parasitics from the layout.

7 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 straightforward 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 by two bit positions.

The second observation, that complicates the operation much 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 are 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 being 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 μm process technology can support transmission speeds of up to 11.55 Gbit/s [10]. The STA was done in the same manner as for the ECCFP.

8 Controller and counter unit

Fig. 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, 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.

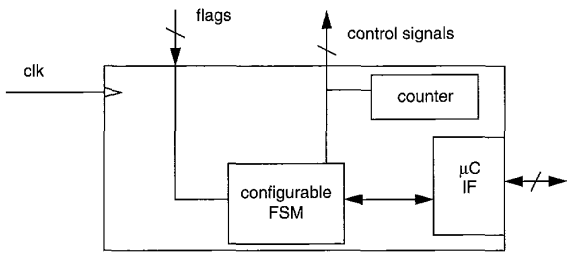

Fig. 6 *Controller and counter unit overview*

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 storage of payload with help from the μC.

9 Discussion

The unique architecture introduced by us 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 Gbit 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 dissipation than the GPPP. Also other dedicated protocol processing solutions normally make use of a layer-based pipelining technique [11], which introduces latency. This latency is eliminated in the GPPP since all layers are processed concurrently.

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 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.

11 Acknowledgments

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

12 References

1 LIU, D., NORDQVIST, U., and SVENSSON, C.: 'Configuration-based architecture for high speed and general-purpose protocol processing'. Proceedings of SIPS'99, Taiwan, pp. 540–547
2 GERORGIOU, C. J., and LI, C.-S.: 'Scalable protocol engine for high-bandwidth communications'. Proceedings of IEEE international Conference on Communications, Towards the knowledge millennium, Montreal, 1997, Vol. 2, pp. 1121–1126
3 YANG, M., and TANTAWY, A.: 'A design methodology for protocol processors'. Proceedings of Fifth IEEE Computer Society workshop on Future trends of distributed computing systems, 1995, pp. 376–381
4 HENRIKSSON, T., NORDQVIST, U., and LIU, D.: 'Specification of a configurable general-purpose protocol processor'. Proceedings of CSNDSP 2000, Bournemouth, UK, pp. 284–289
5 TANENBAUM, A. S.: 'Computer networks' (Prentice Hall PTR, 1996, 3rd edn.)
6 KADAMBI, J., CRAYFORD, I., and KALKUNTE, M.: 'Gigabit Ethernet' (Prentice Hall, 1998)
7 NORDQVIST, U., HENRIKSSON, T., and LIU, D.: 'CRC generation for protocol processing'. Proceedings of Norchip 2000, Turku, Finland, pp. 288–293
8 PEI, T.-B., and ZUKOWSKI, C.: 'High-speed parallel CRC circuits in VLSI', *IEEE Trans. Commun.*, 1992, **40**, (4), pp. 653–657
9 HENRIKSSON, T., ERIKSSON, H., NORDQVIST, U., LARSSON-EDEFORS, P., and LIU, D.: 'VLSI implementation of CRC-32 for 10 Gigabit Ethernet'. Proceedings of ICECS 2001, Malta, pp. 1215–1218
10 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
11 KAISERWERTH, M.: 'The parallel protocol engine', *IEEE/ACM Trans. Netw.*, 1993, **1**, (6), pp. 650–663

13 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.