Packet Classification and Termination in a Protocol Processor

Ulf Nordqvist and Dake Liu
Department of Electrical Engineering
Linköping University, SE-581 83 Linköping, Sweden
Phone: +46-13-28-{2903, 1256}
E-mail: {ulfnor, dake}@isy.liu.se

Abstract

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

1. Introduction

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

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

To meet these new requirement a new area of communication handling hardware platforms has emerged. These are commonly denoted as TCP Offload Engines (TOE). One of these TOE solutions is called programmable protocol processor (PPP) and it was introduced by this papers authors in [1] and [2] 1999. As most of the TOE it consist of programmable parts that can accelerate and offload a terminal host processor by handling the communication protocol processing. The protocol processor platform is a domain specific processor solution with superior performance over a general purpose CPU, that still provides flexibility through programmability within the limited application domain. The protocol processor hardware platform is further discussed in chapter 2. In chapter 3 a novel methodology and architecture for handling and distributing, control flow information to and from our protocol processor is introduced. The proposed architecture enables the protocol processor to be used in networks with fragmented packets. In chapter 4 a discussion on system performance based on behavioral models is included.

2. Programmable protocol processor

The main task of the protocol processor is to process the packets transferred between the application on the host processor and the network, so that a secure and reliable connection is provided between the sender and transmitting function. The protocol processing architecture (and the research project behind) discussed in this paper, only deals with the reception of packets. Since the transmitting

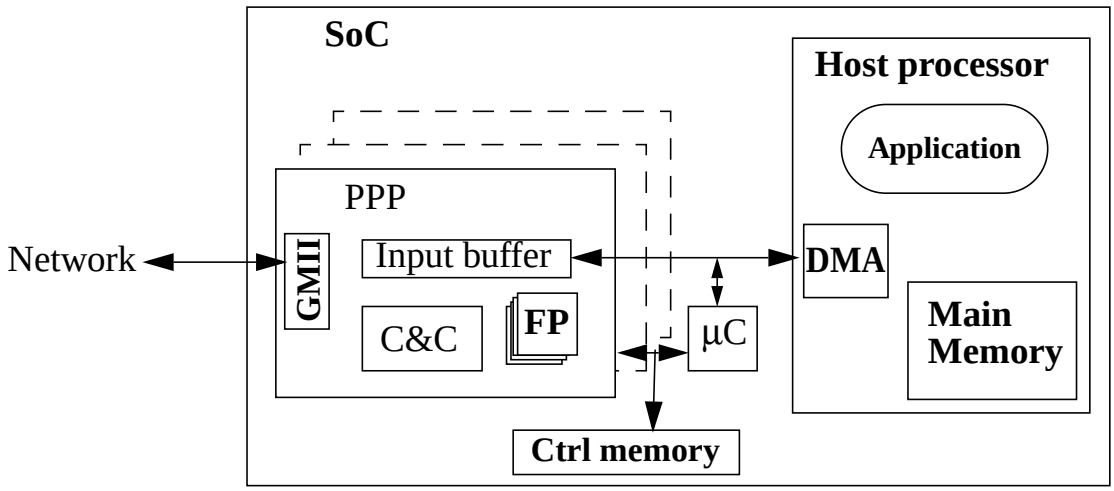

Figure 1: The PPP together with a general purpose micro controller handles the communication of one network port. In a system on chip (SoC) many PPP can be used as port-processors in order to provide high bandwidth between the application and the network. A control memory is used for storage of inter packet control variables.

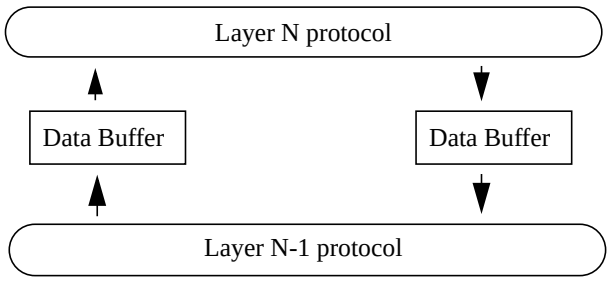

Figure 2: Using inter-layer processing the power consumption and required memory usage in the protocol processor can be reduced since all buffering of data between different protocol layers can be eliminated

of packets is limited by the applications construction of packets and have lower demands on low latency, we have chosen to concentrate our research on the packet reception problem before discussing packet creation acceleration. The goal of this research project is to process as much of the protocol stack as possible before storing the data payload to the systems main memory. By reducing the memory access and buffering, illustrated by figure 2 , both memory bottlekneck problems and power consumption can be reduced. In order to achieve this, the protocols must be processed at network speed and multiple layers are being processed simultaneously as proposed in [10].

In order to deal with the fact that the nature of the different processing task in a protocol processor is very versatile the hardware platform has been divided into two parts. This is illustrated by figure 2 . The first part is the Programmable Protocol Processor (PPP) which is dedicated for data intensive protocol processing task mainly originating from the lower level protocols in the OSI-protocol stack standard. Examples on such tasks are checksum calculations, address checks, length counters etc. Normally they are intra-packet processing tasks that have to be performed even if the protocols covered are very simple. The other part off the platform is a general purpose micro controller (µC) that deals with control intensive protocol processing tasks such as connection state handling and other inter-packet processing tasks. The micro controller is also used for the configuration of the PPP for different type of protocols as well as updates. Further the micro controller handles the control communication with the host processor and the DMA, e.g. setting up and closing sockets etc. Using DMA communication between the PPP and the host reduces the interrupts compared to bus-communication [9]. In the NP research community there is today a clear trend towards a separation of the processing in a slow and fast-path similar to our approach. In figure 3 there is an illustration showing how different layers in the protocol stack are distributed to different processing resources.

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

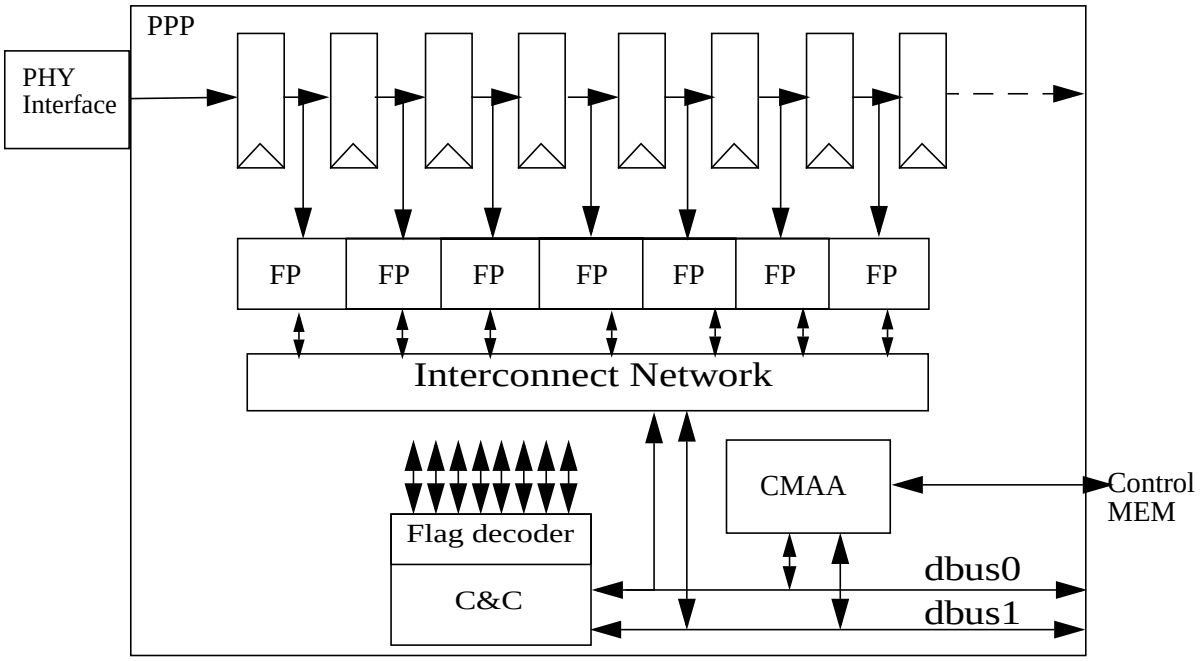

FIGURE 4. The programmable packet processor consists of 4 parts: The Counter and Controller (C&C), the input buffer chain, accelerating functional pages and a Control Memory Access Accelerator (CMAA).

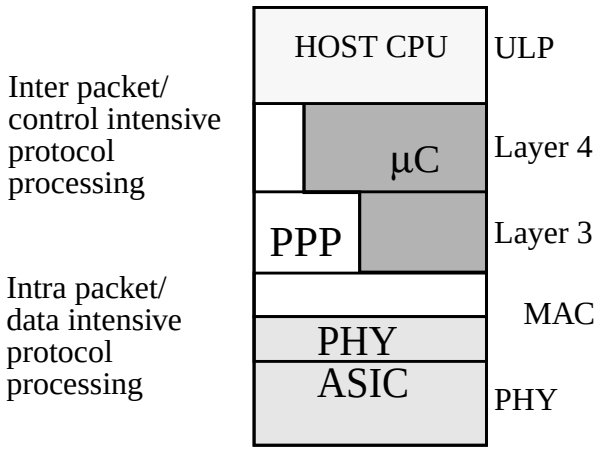

FIGURE 3. Offloading the host using various types of accelerators for different types of processing tasks and protocols. Typically higher layer protocols require more flexibility through programmability.

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

As illustrated by figure 4 the PPP hardware architecture for protocol processing consists of four main parts. One is the input buffer chain that provides the data to the accelerating functional pages (FP). By using a 32 bit wide chain of flip-flops, the fan-out from the flip-flops can be kept on a tolerable level even if the number of FP increases with new protocols and an increased protocol coverage. Using a RAM based FIFO buffer instead of flip-flops would decrease the activity but the fan-out would be a huge problem. As long as the fan-out is kept low it is still possible to replace the last flip-flops in the chain with a minimal RAM-based FIFO. The total buffer size is dependent on the decision latency of the PPP. The decision answer is normally discard or send packet to memory for further processing. Some payloads should be sent to the host memory and some to the control memory.

The control of the various accelerators (FP) in the PPP, mainly consists of start and stop flags. These flags are provided from the Counter and Controller (C&C). The flags are generated based on an internal program in the C&C, result flags from the FPs and counter values generated in the C&C. The C&C is responsible for sceduling the start of the processing in the FPs at the correct clock-cycle, as the data streams through the register chain. Based on the result from the FPs, the C&C can either discard the packet or continue the processing by configuring and starting new FPs. If a packet is discarded, e.g. because the destination address was errornous, all the FPs are immediately shut down in order to save power.

Since the processor operates on streaming data, instead

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

2.1. Functional pages

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

- 1 CRC FP described in [7]
- 2 eXtract And Compare (XAC) FP responsible for checking address numbers and port numbers against the actual host address. Further they are used to extract and compare checksums.
- 2 length counting adders.
- 2 checksum calculation adders
- 1 generic adder

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

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

3. Control Memory Access Accelerator

As mentioned earlier the micro controller is responsible for the communication control (signaling) handling. Using a general micro controller is a straightforward method similar to the traditional way of slow path processing in a GP CPU. The problem with this solution is that the control information must be transferred between the micro controller, the PPP and the control memory with low latency in order for the PPP to process its part at wire-speed and make the decision if the packet should be discarded. This is needed because of the use of fragmentation. Further, acceleration of slow path processing off-loads the micro processor. Hence, a platform including accelerating hardware assist and control interface dedicated for packet recognition and control memory access have been developed. The Control Memory Access Accelerator (CMAA) presented in this article uses 2 Look Up Engines (LUE) in order to recognize and classify the incoming packet. These LUE essentially consists of Content Addressable Memories (CAM) which are well known and commonly used in routing and switching applications. One of the early work in this area is [8].

3.1. Header data

The purpose of storing control information is to ensure that connection oriented protocols (e.g. TCP) can perform protocol processing on the payload which can be divided or segmented into many lower layer packets. These packets can arrive out-of-order and in case of connection oriented protocols the routing information is not included in all packets. Hence it is obvious that some information on the current status of a connection must be stored in order to be able to continue the processing when the next packet arrives. In the case of the protocol set discussed earlier in this chapter the following information is normally needed.

- **Protocol type**
- **Length (received so far)**
- **Total length (included in the last fragment)**

The length field(s) is provided to the length counter adder in the PPP which updates the number and finally sends the updated value to one of the XAC FP. There it is compared to the total length value which is stored in the control memory. If they are equal, the micro controller is notified that all packet fragments have been received and this entry will be removed from the search list. If unequal, the new length value is written back to the control memory.

- **Accumulated checksum results**

The checksum results is provided to one of the checksum calculating adders which adds it to the recent packets checksum using a 1-complement addition which produces a new checksum. If the length is equal to the total length which means that the hole payload message has arrived the updated checksum it is sent to one of the XAC FP for comparison with the received checksum.

- **IP Source and Destination Address.**

The source address is extracted from the data-stream by the PPP. The adress value is then used to construct a pseudo header. The pseudo header is used in the checksum calculation. Normally, only one destination address is used

for unicast packets in a terminal. This means that it is not needed to be stored in the control memory.

- **TCP Source and Destination Ports**

The type, ports and addresses identifies a specific connection. To see if a incoming packet should be discarded or accepted these fields must be checked. They are also used to identify which application the payload should be directed to.

- **Identification number**

The IP identification number is used to find the correct memory buffer in the control memory.

- **Pointers to the memory position of proceeding and succeeding packets/segments.**

In order to provide all of the services stipulated by the TCP standard, more connection related information than listed above needs to be stored. On the other hand the only information needed for the PPP to perform its processing is the information high-lighted in bulleted text. The information stored in the control memory can also be used to calculate the host memory adress. An algoritm for this type of memory address calculation remains to be implemented for the general case even if it is simple for special applications, e.g. VoIP. A general algoritm for in-order data-buffering in the host memory would significantly reduce the host processor interrupts. This type of algoritm would benefit from an accelerated access to the control memory. This issue will not be further discussed in this paper.

3.2. Accelerator interface

The CMAA interface to the rest of the PPP and the micro controller is illustrated by figure 5 .The input to the CMAA consists of flags and an instruction generated in the C&C. In table 1 the simple instruction set (6 instructions) is listed.

As output the CMAA generates a number of flags. The

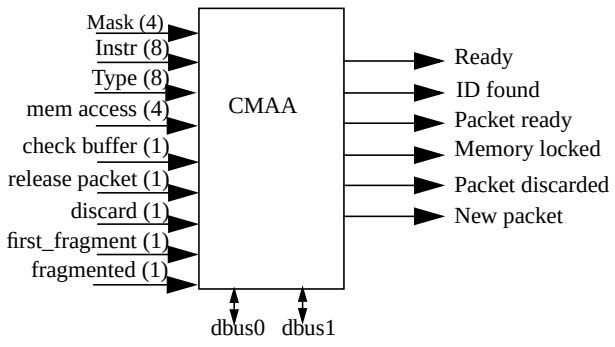

FIGURE 5. Accelerator interface.

Table 1. Lightweight instruction set

Name	Internal configuration
New packet	Packet type
Load register	Port or Address word
ID CAM operation	write, read or remove
PA CAM operation	write, read or remove
Release to micro controller	
Set memory buffer	Packet type

two data buses are being used for data transport.

3.3. Data path

An overview of the CMAA architecture is illustrated in figure 6 . The CMAA data path includes two LUE, a buffer pointer generator, and a simple memory access selector. The Primary LUE (PLUE) only includes one CAM which has 16 bit wide entries, has M entries and the result memory is W bits wide. The purpose of this unit is to check if we already have received an fragment of the incoming packet. This is checked using the IP Identification field (IP ID). If an arriving packet is fragmented, the fragmented flag will be produced in the C&C and provided to the CMAA. Then the fragment is checked in the PLUE to see if a packet buffer exist in the Control memory. If the CAM in the PLUE does not have a matching identification field entry, a new packet buffer will be created and the IP ID will be written to the PLUE CAM. In the packet buffer, inter-packet variables such as length and checksums will be stored. If the packet is non-fragmented there is no need to store its IP ID so the packet buffer is created directly on the control memory address provided from the mem buffer gen unit in figure 6 . The SLUE is a classification engine including 6 CAMs and its purpose is to check for valid connections. The two data buses is 32 bit wide. The mem buffer gen generates new buffer addresses for both packet buffers and connection buffers. The adress generation is controlled from the µC.

As the other accelerating devices in our protocol processor, e.g. FPs, the CMAA remains in idle mode while not in operation. Power-up will be performed when a new packet arrives. This reduces the power dissipation significantly in a network terminal due to the un-even time distribution of the packet reception.

In this paper we leave the final CAM design and implementation to be further investigated and optimized. The reason behind this is that they are extremely important for the overall performance and they require different design techniques, tools and expertise than the rest of the PPP. Final implementation of the LUE will of course have an huge impact on the performance of the CMAA. This issue is further discussed in section 3.7..

A layout of the CMAA excluding the two LUE and the buses has been produced. The number of standard cells

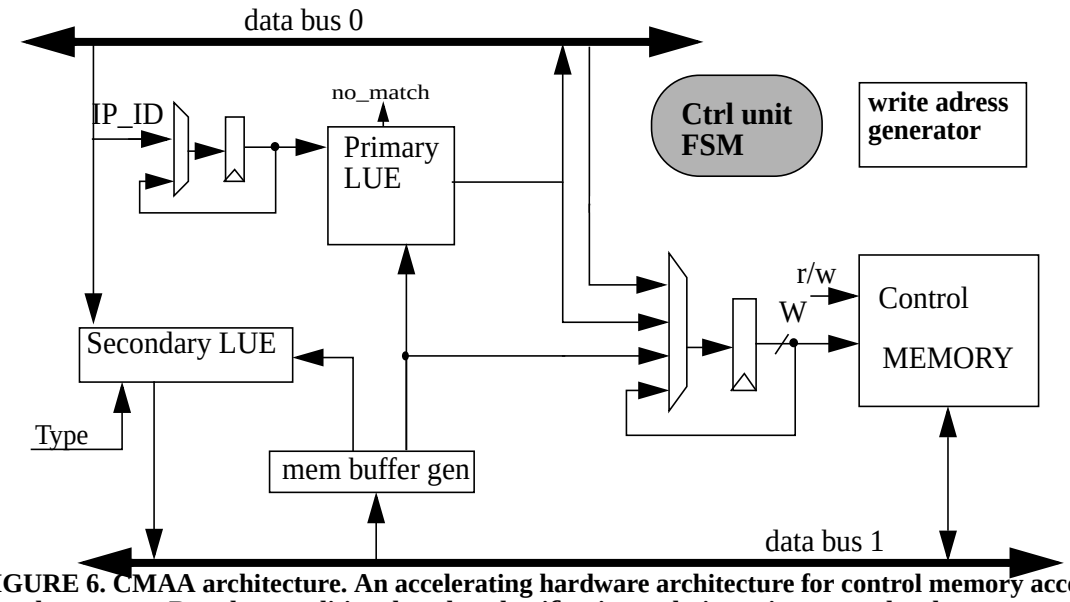

FIGURE 6. CMAA architecture. An accelerating hardware architecture for control memory access in the protocol processor. Based on traditional packet classification techniques it support low latency access to stored connection variables in the control memory

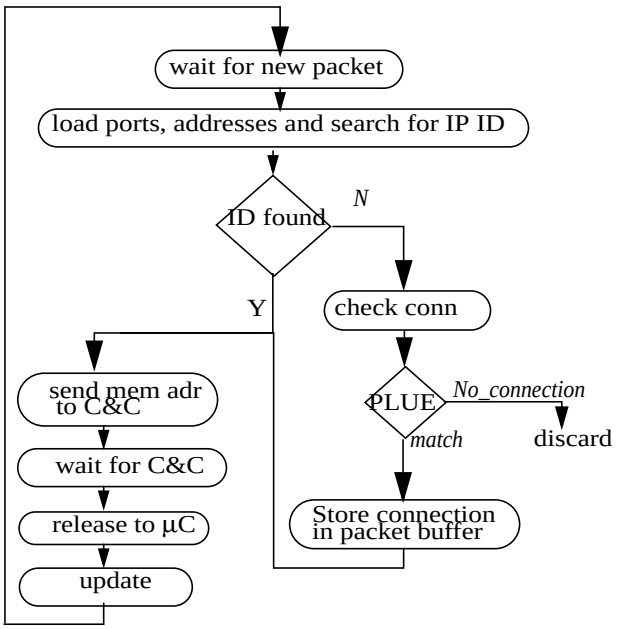

FIGURE 7. Control handling procedure within the CMAA.

and the area of the CMAA excluding the input registers, and the two LUE are 716 and 0.105 mm^2 respectively. This part of the CMAA has been simulated, using static timing analysis on the layout, to run at almost 300 MHz. This means that it is not included in the critical path of the PPP. Since we use registered inputs and outputs in the CAMs, it is the SLUE that will be the critical path of the CMAA.

3.4. Control procedure

The normal packet reception procedure of operation in the CMAA, is illustrated by figure 7 . The procedure is controlled by the control unit finite state machine (FSM) in the CMAA.

If a new packet arriving is fragmented, the PPP provides CMAA with the IP Identification number and gives a new-packet instruction to the CMAA. The IP ID is then stored in the input registers to the PLUE. Next 2 clock cycles, the CMAA continues to load ports and IP addresses while the PLUE checks if a fragment of this payload has already been received. If there is a match in the PLUE search, the corresponding address pointer to the buffer in the control memory, which is stored in the PLUE result memory, is stored in the input register to the control memory. While the PPP continues the packet processing, it can then access the control memory directly. If the new fragment contains the layer 4 header, the port, source and type fields are loaded from the PPP and then checked in the SLUE. If this loading is completed after the PLUE search, e.i. it is a IPv4 packet, the SLUE can immediately check the connection information. Otherwise the control unit remains in the check connection state while the loading continues. Based on the SLUE result, the packet is either discarded or the matching connections adress pointer is provided to the data bus 1. Next clock cycle, the data bus 1 value will be stored at the packet buffer adress which is already stored in the input register to the control memory. This means that the µC easily can access the connection information. Then the CMAA hands over to the PPP using the packet-ready flag.

After the PPP has received the packet-ready flag, it continues to process the packet and updates the control memory.

After successful packet processing, the PPP releases

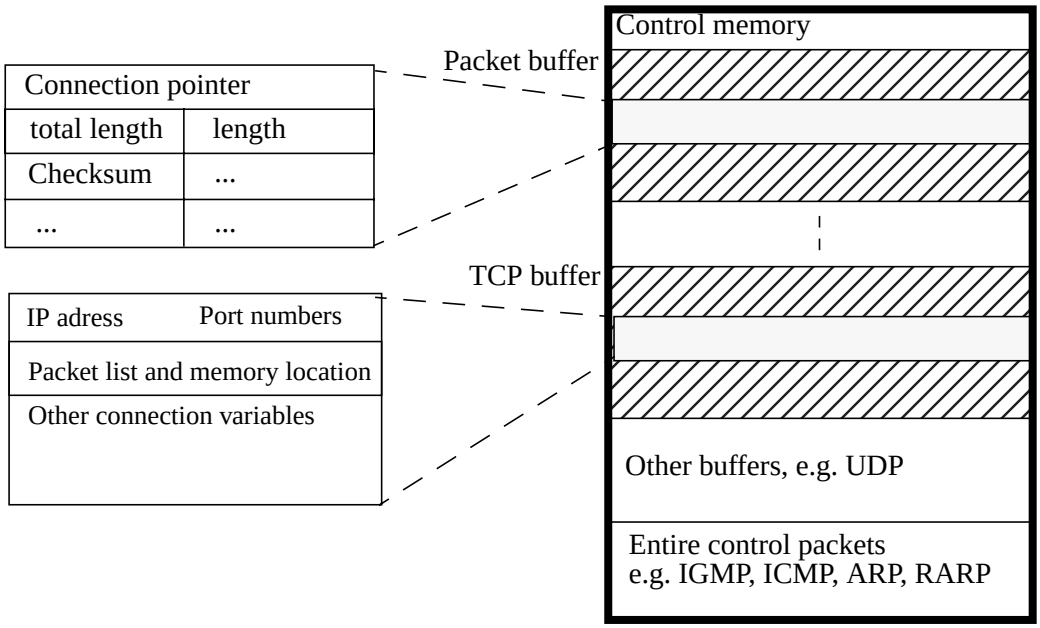

FIGURE 8. Memory organization in the control memory.

the packet to the CMAA. Next clock cycle, the CMAA releases the lock of the control memory, starts buffer pointer updating and sends the new-packet flag to the µC. During the update state, the CMAA also updates the write adress for new entries to the two LUE. This is only done if a write operation has been performed. During the write adress search, the CMAA uses one of the generic adders in the PPP to search for empty entries. When the pointer updating and the CAM write search is finished the CMAA returns to the wait-for-new-packet state.

3.5. Control memory organization

The control memory is organized according to figure 8 . As illustrated the control memory consists of a number of different buffers storing inter-packet information. Further the memory include all the control oriented packets that is going to be processed in the micro controller software. Since these protocols is completely processed by the micro controller, also the payload of these packets is stored in the control memory. For TCP and UDP type of packets only preprocessed header information is stored. In the packet buffers, layer 3 information needed for reassembly is stored. Each packet buffer is deleted when all fragments have arrived.

3.6. Look-up Engine architectures

The SLUE consists of 6 CAMs as illustrated by figure 9 . The outputs generated by the CAMs are vectors containing zeros or ones describing table matches. These are used to select the address pointer in the result memory, e.i. the control memory address for the received packet.

The 7 different CAMs we propose to be used in the CMAA architecture will have an huge impact on the performance, size and power figures of the entire design. Therefor they require a thorough investigation and optimization procedure, in order to obtain the optimal system performance. Even if the optimization of these CAMs not is in the scope of this paper, some characteristics and requirements on the CAMs can be noted. First of all we propose that CAMs should be instead of TCAMs ([11] and [12]). This reduces the cell size and power dissipation. The primary LUE is a standard CAM memory 16 bit content and M entries. The result memory is M times the length of the control memory adress W.

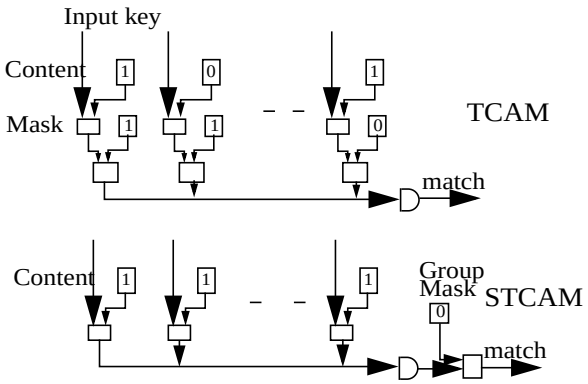

FIGURE 10. Simplified TCAM principle.

In order to provide flexibility for different protocols we use a concept we call Simplified TCAM (STCAM) illustrated by figure 10 in the secondary LUE. Instead of using ternary bit comparisons as in TCAMs we only provides an wildcard function to the entire CAM. In figure 11 there is an illustration showing how the secondary LUE uses the

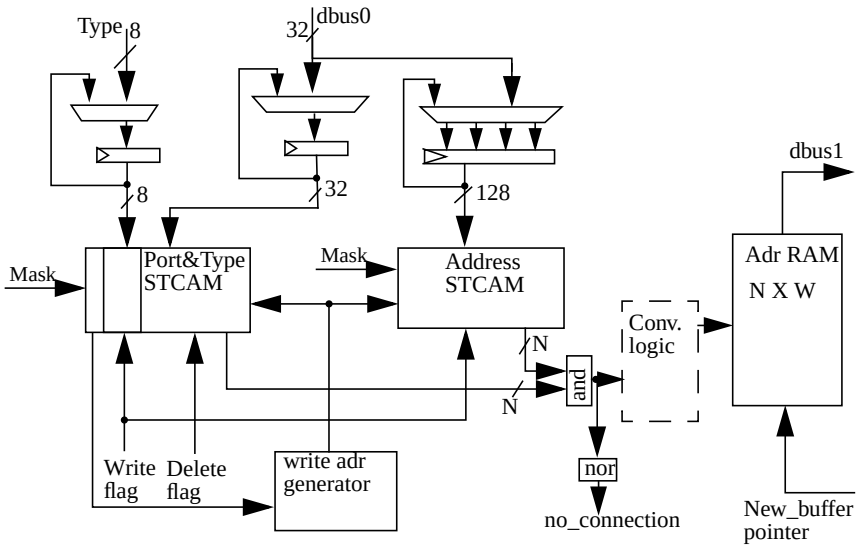

FIGURE 9. Secondary Look-Up Engine (SLUE) architecture. Note that the conversion logic that converts the matching vector to a result memory adress can be eliminated if the matching vector is used directly as word lines in the memory. This however require that the RAM must be implemented in the same manufacturing process.

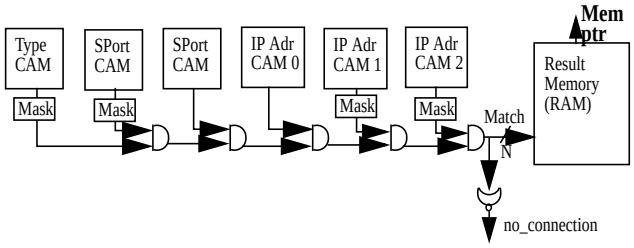

FIGURE 11. The two different STCAM in the SLUE each consists of three ordinary CAMs and masking functions. Each of the CAMs uses N entries.

STCAM principle. The mask input enables a wildcard functionality for different fields when recognizing an incoming packet according to table 2. The table shows that the proposed SLUE architecture can be used for various types of protocols. A careful use of these wild cards is needed in order to avoid multiple matches. By using the type field, which is an internal type, it is possible to avoid multiple matches which means that the priority logic in the SLUE can be eliminated. Further it enables the connections to be written in the CAM in an arbitrary order.

Table 2. Configurations using masking for different packet types and applications.

Protocol examples	Type	Source Port	Destin. Port	Source Address	Destin. Adr
IPv6 Unicasting	Optional	Optional	16	128	*

Table 2. Configurations using masking for different packet types and applications.

Protocol examples	Type	Source Port	Destin. Port	Source Address	Destin. Adr
IPv6 Broad or multi cast-ing	Optional	Optional	16	*	128
IPv6 alt	Optional	Optional	16	64	64
IPv4	Optional	Optional	16	32	32
IPv4	Optional	Optional	16	32	*
UDP	Optional	16	16		32

It can always be discussed how much the IP version 6 (IP v6) protocol will be used in the future. We have chosen to include it since the penalty is not as severe in network terminals as it is in routers. The reason for this is that in network terminals we only have one destination address to check for unicasting. This can be done in other parts of the PPP. Hence 128 bits can be excluded from the CAMs entries. For broad and multicasting packets a different type field is generated and only the destination is checked (instead of the source address). This reduces the penalty we have to pay in forms of larger CAMs when including IP v6. There exist however routers where only 64 bits out of the 128 in the IP v6 address is used for packet classification. The reason is that in such networks the other 64 bits is just a copy of the MAC address. If such method would be applied the CAMs can reduce the wordlength of the content with additionally 64 bits by eliminating the IP Adr 2 in figure 11 Since this architecture will be used in a network terminal the activity will not be as high in the CAMs as it

would be in a router. The reason is that we only do a load and search operation when a new packet arrives, not every clock cycle. The low activity significantly reduces the power consumption in the CAMs.

3.7. CAM implementation issues

The total size of the 7 CAMs and there result memories will be a major part of the system chip area. It is very hard to make predictions on the sizes of these CAMs since that is a matter of optimization effort and implementation strategy. Further the complex placement and routing requires a full custom approach even for standard cell based designs. Even without a final layout, a lower bound on the chip-area can be estimated. Using standard cells from our design process (AMS 0.35 µm 3.3 V 3-M) an optimized bit-slice cell in a CAM is approximately 350 μm^2 which results in a lower bound on the combined CAM area according to EQ 1. The result memories must store M + N times W bits using approximately 180 mm^2 each.

$$A_{CAM} = (16 \times M + (128 + 40) \times N) \times 350 + (M + N) \times W \times 180 \quad (1)$$

As an example M=16, N=64 and W=20 can be considered. The chip-area for the two LUE would then be at least 4 mm^2. This figure is acceptable but if more entries are to be considered a process migration to smaller geometries is natural. The number of entries to implement is a matter of optimization. This optimization procedure requires a careful analyze of application requirement and network traffic. Never the less it is clear that in NT, the required number of network links is not as high as in routers. Hence M and N does not need to be very large for most applications and networks.

In order to examine our architectural performance, it is crucial to know how many clock cycles each search operation in the two LUE requires. We expect the system clock to have a period of maximally 7.5 ns in a 0.35 micron process, based on timing analysis on other parts of the PPP. Hence the maximum network speed is 4.3 Gbit/s using the specified 32 bit wide input buffers. Since we are sure that there is only one packet being processed at any given time, we do not necessarily need the LUE:s to be pipelined, e.i. we do not need any internal intermediate results to be stored. Instead a multi-cycle-path design technique can be used. To use pipeline stages or not is an implementation issue for the CAM designers. Simulations shows that the small PLUE will not require more than 2 clock cycles to complete one search, e.i. it has a critical path shorter than 15 ns. Then we assume M is maximally 64.

The number of clock cycles required for a search operation in the SLUE is equal to the critical path divided by 7.5 ns. The critical path consists of circuit delays and wire delays. If the SLUE are being implemented using standard cells the logic delay is simple to calculate. For N=64 there will be approximately 15 logic cells in the critical path which leads us into believing that 2 clock cycles is enough. The problem is that in larger CAMs a big part of the critical path, is wire delay. In my research design (N=256) I have used synthesis and P&R tools from Cadence. The resulting implementation result is very far from optimal and does not meet my requirement 3 clock cycles. The design is simply to large and hence the P&R problem to complex. Therefor the conclusion is that the design strategy must be changed to something more custom oriented even if the CAM is rather small compared to the one used in routers. Clearly a bitslice manipulating placement strategy has to be used for efficient CAM design regardless of the size. Anyway the conclusion after studies of other comparable CAM designs and discussion with industry CAM designers is that, for N less than or equal to 256, a search operation will require maximally 4 clock cycles (or pipeline stages). For N=64, 3 clock cycles is definitely enough. These figures apply to standard cell based designs.

Even with a pessimistic feature size projection (Moores law), there is no reason to believe that scaling not can support the CMAA to run at clock periods around 3 ns using 3 clock cycles for one search operation. Hence the CMAA could be used in a 10 Gbit/s network such as 10 Gigabit Ethernet, using already available processes, e.i. 0.13 micron. The resulting latency for CMAA operations is further discussed in section 3.8..

The latency, critical path, power consumption in the LUE is of course depending on M, N and W. To optimize these variables simulation on real world network traffic is required. Until this optimization phase is completed the numbers M=16, N=64 and W=20 will be considered for further architectural development.

3.8. Latency

The proposed architecture for access of the control memory, reduce the control memory access latency to a few clock-cycles. The fast path latency determines how big the input buffer chain has to be. The latency of the CMAA must be added to the latency of the PPP in order to calculate the total fast path latency. We propose that the SLUE should use 3 clock cycles to perform a search. A 3-clock-cycle type of SLUE would give a maximum memory access latency of 11 according to table 3 when a new packet has been received. Further the table shows that a four cycle type of CAM architecture, will give a maximum memory access latency of 12 clock cycles. This of course have an impact on the pipeline register chain in the PPP and the total latency for a packet reception and delivery to the micro controller.

The PPP can start the processing of an incoming packet before the control data has been accessed from the control memory. Therefore this latency only sets a lower limit on the latency of the total packet reception. The total latency

is however mainly dependent on the processing activities, including interrupts and stalls, in the micro controller.

Table 3. Memory access latency for various packets received. (PLUE requires 2 clock cycles to perform a search)

Layer 3 protocol	# clock cycles latency for CMAA operation 3 stage SLUE	# clock cycles latency for CMAA operation 4 stage SLUE
IPv4 - new packet	9	10
IPv4 - old packet, new fragment	4	4
IPv6 - new packet	11	12
IPv6 - old packet, new fragment	4	4

3.9. Enabling flow based QoS

Using the fast control memory access, it is possible to enable quality of service (QoS) to the reception. Any kind of priority parameters or flow parameters can be stored in the different buffers in the control memory. These can then be used for multiplexing of the incoming data stream, if a flow based operation is demanded.

3.10. Shared control memory

The motivation for separating the protocol processing into one PPP-part and one µC-part is of course to use the programmability of the µC when processing control intensive tasks, and still have high-performance and low-power implementation of the data intensive processing. This distributed architecture however requires an interface, and that interface is the control memory unit together with control flags to and from the C&C. As mentioned before, the PPP only need to access the memory when a new packet is received and then only a limited part of the control information is used. Since the latency of this access directly effects the length of the input buffer chain, the PPP must have priority over the µC when it comes to memory access. In fact the µC only have access to the control memory when the CMAA resides in the update or wait-for-new-packet state according to figure 7 .

4.System performance

Each hardware acceleration block in the PP has been seperatively implemented and simulated using static timing analasys.The conclusion is that they can operate at network speeds of moore than 170 Mhz. Since all parts of the fast path operates on streaming data it means that the network can run at this clock frequency. The fast path architecture processes each packet, delivers data and control signals to the micro controller and then returns to idle mode. When the fastpath, e.i. the C&C, has returned to idle-mode it can start processing the next packets. Hence, the proposed fast path architecture can operate in high speed networks as long as the gap between the incomming packets is sufficient for the processor to return to idle mode. This must be supported by the network protocol.

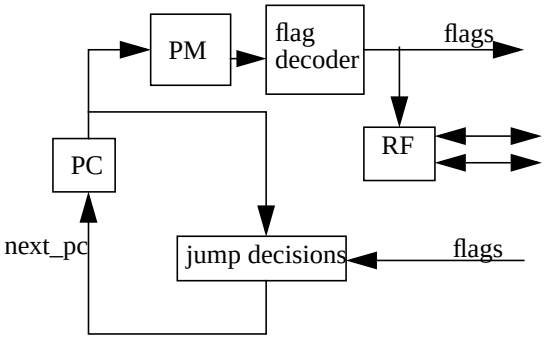

Figure 12: The C&C architectural model.

The slow path consists of the micro controller which is enough flexible to fully offload the TCP and protocols alike. The micro controller is not capable of processing the packets at wire speed. This may limit the performance of the entire system for extreme traffic situations. The fast path does however offload some of the tasks traditionally processed on general purpose hardware. This will relaxe the slow path. The amount of off-loading depends on the traffic flow and requires further simulations.

In order to verify the functionality of the CMAA block used as proposed in a fast path, a cycle-true and bit-true behavioral model has been simulated. The simulation model covers the fast path packet reception. In the model the C&C has been modeled as a simple version of the architecture presented in [13] The principle is shortly described in figure 12 . The C&C is modelled as a program counter (PC) which selects instructions in the program memory. These instructions is then decoded to produce all the control signals in the PP. In order to make conditional jumps without loosing any clockcycles, a programmable jump decion block calculates the next pc value based on the current PC value and result flags from the rest of the PP. This architecture supports wire speed processing of non-fragmented packets.

The model includes GMII network interface (32 bit wide input). Further a behavioral model of the CMAA including 16-entry, 3-stage pipeline CAMs has been used.

So far the only protocols simulated are TCP and IPv4. Fast path tasks simulated includes CRC, IP reassembly, checksum calculation and data stream demultiplexing. The C&C is programmed to cover these protocols using a the program memory in the C&C model. The network traffic simulated is random.

The simulations verifies that the proposed CMAA architecture can be used in the protocol processor environment. Further it shows that when programmed for TCP the

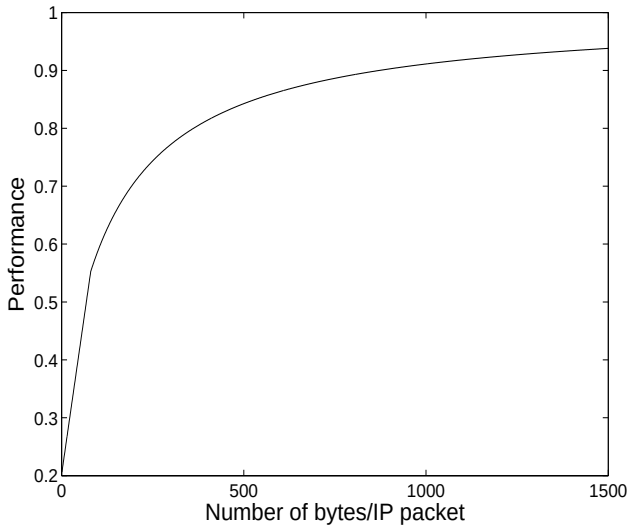

Figure 13: Performance degradation due to packet processing latency and increased inter-packet gaps. In comparison with ideal network capacity.

minimal distance allowed between 2 packets is 23 clock cycles (ideal is none). In addition to this limitation the simulations also shows that the minimal number of clock cycles required per received ethernet packet processed is 38. The decision latency in the CMAA contributes to this figure. These architectural limitations strongly effects the performance compared to an ideal solution. Especially for small packets. This is illustrated by figure 13 . The overall system performance is on the other hand much dependent on the network traffic. If the traffic does not include many fragmented packets the CMAA may not be worth using due to the performance degradation. In that case a PP without CMAA acceleration should be used.

To illustrate how much processing that is offloaded from the slow path using a CMAA, we can count microcode instructions needed for a address check using a general purpose RISC machine. Assume that there is N possible addresses that we have to check against a received address. The setup requires four instructions for loading addresses and a loop variable. Then 6N instructions are needed for a comparison loop. With N=16 entries, 100 instructions are needed in the worst case. 32 of them are conditional jumps. This processing is performed on every received packet and it is just a small part of the CMAA processing.

5. Conclusions

A novel architecture for acceleration of control memory access in a protocol processor for network terminals was presented. The architecture uses classification engines and concepts which has traditionally been used for network infrastructure components. The proposed architecture enables low latency access to connection state variables, partial checksum results and any other control information stored in the shared control memory. Hence inter-packet processing such as reassembly has been accelerated using the proposed architecture in network terminals. Further the architecture offloads the slow path which is very important in high-speed networks. The proposed architecture can process the fast path in a multi gigabit network, implemented in a mature standard cell process such as AMS 0.35 µm. Simulations does however show that the protocol processor requires increased packet gaps in order to manage fragmented packets.

6. Further work

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

7. Acknowledgments

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

References

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

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

[3] M. Yang, A. Tantawy, “A design methodology for protocol processors”, Proceedings of the Fifth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems, 1995, pp. 376 -381

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

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

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

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

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

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

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

[11] van Lunteren J., Engbersen A.P.J., “Multi-field packet classification using ternary CAM”, Electronics Letters, Volume: 38 Issue: 1, 3 Jan. 2002, pp 21 - 23

[12] Huang N-F, Chen W-E, Luo J-Y, Chen J_M, “Design of multi-field IPv6 packet classifiers using ternary CAMs”, GLOBECOM’01. IEEE, Volume: 3, 2001, pp 1877 -1881

[13] T. Henriksson, U. Nordqvist and D. Liu, “Embedded Protocol Processor for Fast and Efficient Packet Reception”, in Proceedings of International Conference on Computer Design, September 16-18, 2002, Freiburg, Germany, pp. 414-419.