A SCALABLE NETWORK ASIP ENABLING FLOW AWARENESS IN ETHERNET ACCESS

K. Van Renterghem, D. Verhulst, S. Verschuere,
P. Demuytere, J. Vandewege, Xing-Zhi Qiu

Department of Information Technology
Ghent University / IMEC
Sint Pietersnieuwstraat 41, Ghent, Belgium
email: Koen.VanRenterghem@intec.ugent.be

ABSTRACT

In this paper we research an FPGA based Application Specific Instruction Set Processor (ASIP) tailored to the needs of a flow aware Ethernet access node. The processor has an architecture optimized to handle flow processing tasks such as parsing, classification and packet manipulation.

The VLIW instruction set allows for high degree of parallelism among the functional units inside the ASIP and has dedicated instructions to accelerate typical packet processing tasks. This way, a single processor is capable of handling the complete throughput of a gigabit Ethernet link.

To reach the target of a 10 Gbit/s Ethernet access node several processors operate in parallel in a multicore environment. Apart from scalability, programmability is also an important feature. Therefore, the processor is developed using a retargetable tool suite, creating the hardware and an optimized C compiler out of a single processor description.

1. INTRODUCTION

Ethernet based access platforms are becoming the vehicles to offer triple play services (voice, video, data), as well as innovative and advanced services with high world-wide growth potential such as peer to peer traffic or gaming. The drivers of the success of Ethernet in enterprise networks are also driving the access technology. It is hence expected that future high bandwidth access infrastructures with fibre to the cabinet, building or home will be Ethernet based. Legacy networks for voice, data and video are converging into a single network, new services are emerging and Ethernet is becoming the transport protocol of choice [1].

A means to support existing and new services in Ethernet based access is the introduction of ATM-like features such as "flow awareness". Traffic will be treated differently depending on the subscriber to whom it belongs, and the type of service it represents [2] [3] [4].

In a flow based approach several features of packets are extracted (layer 2 up to layer 4) in an access node and then used as input for classification. The outcome of this process reveals the service tied to the inspected packet, allowing it to be further processed and routed accordingly.

The scope of this paper is the design of a processor active in the data path of such a system. The researched solution must be scalable, flexible, easy to program and maintain. These design specifications resulted in the development of a packet processing ASIP to be deployed in an FPGA based multicore processing architecture supporting line rates up to 10 Gbit/s and corresponding packet rates as high as 14.88 MPacket/s.

In order to minimize the processing time, algorithms are often described in custom languages or assembly. Here, a retargetable C compiler, capable of optimizing for instruction level parallelism allows easy ASIP code development, while maintaining efficiency. The development is targeted at a Xilinx Virtex4 LX200-11 FPGA.

2. DATA PLANE OF A FLOW AWARE ACCESS NODE

A complete flow aware Ethernet based access node consists of several building blocks each dedicated to specific tasks such as parsing, classification, filtering, queuing, statistics, traffic engineering, packet manipulation, etc... The scope of this paper is the development of an ASIP optimized for the three key building blocks: parser, classifier and packet manipulator. The specifications of the internal architecture of the ASIP are derived from a survey of typical algorithms running within the aforementioned building blocks.

Parsing is targeted at IPv4/IPv6 encapsulated in Ethernet with TCP or UDP as transport protocol. The algorithm supports a wide variety of Ethernet standards such as (stacked) VLANs and even the upcoming envelope format defined in IEEE 802.3as with frames up to 2 KB. PPPoE, often used in xDSL access networks as authentication protocol, is also supported.

The parsing algorithm decodes the protocol stack, extracts fields used to define a flow and stores them in a data structure called a 'ticket'. Control plane traffic such as ICMP, IGMP, ARP and others will be identified and flagged in the ticket to be forwarded to a dedicated control plane processor.

The verification of checksums is not part of the algorithm. At Layer 2 the CRC can easily be verified at wire rate before assigning the packet to an ASIP. The verification of Layer 3 and 4 checksums requires in depth knowledge such as the type of payload, optional headers, etc. Therefore it is performed in a dedicated hardware

649

block using the information gathered during parsing and classification.

The **classification** algorithm transforms the extracted header fields into a search key which will be applied to an external TCAM (Ternary Content Addressable Memory). The result of this process is a 'flow identifier', used throughout the system to perform further processing.

Based on the flow identifier, the **packet manipulator** can alter header or payload fields and recalculate the checksums. Typical operations are the insertion of a VLAN tag, the decrement of the IP TTL field, etc...

These three algorithms were translated into C code and served as a starting point for the ASIP architecture. From then on several iterations were performed exploring architectural aspects such as instruction level parallelism (ILP) and extensions of the C language with compiler known functions (aka intrinsics) to minimize the execution time.

3. DESIGN FLOW

The ASIP was built using the Chess/Checkers retargetable tool suite of Target Compiler Technologies [5]. The environment consist of a compiler, (dis)assembler, instruction set simulator, hardware description language generator and test program generator.

The processor is described in a high level processor modeling language, which serves as input for all tools, providing consistency throughout the design flow.

After describing the ASIP, the compiler and instruction set simulator can be generated to get quantitative information about the performance of the architecture. The performance impact of design choices such as ILP and the extension of the compiler with compiler known functions can be easily evaluated. The next step is the automatic generation of VHDL or Verilog for the architecture, which can be synthesized and verified against the instruction set simulator.

4. MULTICORE SUPPORT

The requirement of handling the maximum frame rate (14.88 Mpackets/s) of a 10 Gbit/s Ethernet link leaves an ASIP only 67 ns to finish its tasks. When running at 120 MHz only 8 clock cycles are available. Clearly a multicore architecture is needed to handle the complete throughput. Several architectures are possible. Multiple cores can be put in a pipeline, each performing a part of the complete processing algorithm of the same packet. While easy to implement, this architecture requires the packet algorithm to be split in equal parts to avoid idle time on the ASIPs. An alternative is to use a pool of processors all working in parallel on different packets. ASIPs running idle are avoided, but each ASIP must have access to all external peripherals and high speed busses will need to traverse the

FPGA fabric. A mixture of both approaches, a pipeline of ASIP pools, combines the best of both worlds.

To support any of the aforementioned setups, a suitable memory architecture, taking optimal advantage of the resources scattered over the FPGA fabric, was developed.
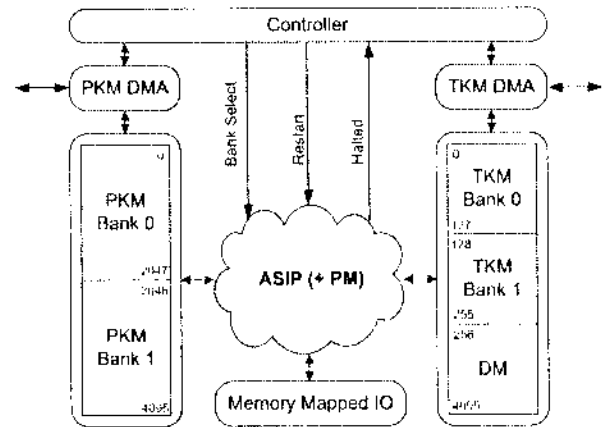

Fig. 1. External ASIP interfaces

Each ASIP has access to two 4 KiB dual ported memories. One port is exclusively reserved for the ASIP and the other one interfaces with the multicore environment. The two physical memories are divided into regions, according to their function. The Packet Memory (PKM) consists of two equal banks of 2 KiB and stores the packet to be processed. The organization of the remaining memory is somewhat different: 256 bytes Ticket Memory (TKM) and 3840 bytes Data Memory (DM). The ticket memory itself is further divided into two equal banks of 128 bytes each. This memory stores the ticket data structure containing the context linked to a packet. Each ASIP can add results to it or use it as input for its own tasks. Apart from these two physical memories, a dedicated 32 bit memory interface is available used to map external peripherals onto its memory space.

The goal of the memory architecture is to keep the processor occupied at all times. The bank select signal makes the processor operate on a specific bank. While packet processing is ongoing, the second memory port can be accessed to write new data into, or read previous results from the inactive bank, without the risk of data corruption. Buffers for rate adaptation are not needed in the system as each DMA engine interfaces with two 32 bit wide Block RAMs combined into a single 64 bit wide bus running at 200 MHz.

For the programmer, bank switches are transparent and the mechanism needs minimal support in the code. Each time the running algorithm concludes its work on a packet, a 'halt' instruction must be issued. This instruction drives the external halted signal, flushes the processor pipeline, stops the issuing of new instructions and resets the program counter to the beginning of the algorithm.

5. ASIP DATA PATH

A schematic overview of the ASIP data path is shown in Fig. 2. The data path consists of a hybrid 16/32 bit architecture.

The general register file (REG_DATA) and the associated arithmetic and logical unit (ALU) are both 16 bit wide. As most packet header fields requiring ALU processing do not take more then 16 bit, the cost for a complete 32 bit architecture is rather high. Furthermore, eight register entries proved to be sufficient to let the envisaged algorithms execute without register spilling.

Only the Load/Store part of the processor is 32 bit. Its architecture is quite flexible as any two storages are interconnected on a common 32 bit Load/Store Bus. The source and destination addresses are calculated in two address generation units (AGU) operating on a 12 bit wide pointer register file (REG_PTR), also with eight entries.

Flexible packet processing requires the reading and writing of 8, 16 or 32 bit values from memory at byte boundaries. A memory controller selecting the right bytes from two 4 byte wide Xilinx Block RAMs (BRAMs) can provide the desired functionality with an 8 to 1 multiplexer. However, this design proved to be a critical path. Configuring the BRAMs with 2 byte wide outputs removes the possibility to handle byte aligned 32 bit accesses, but still allows them to be executed aligned at word boundaries. This configuration has slightly less flexibility, but only requires a 4 to 1 multiplexer, effectively relaxing the critical path. Memory addresses are calculated using two AGUs, one for source addresses and one for destination addresses. Three addressing mechanisms are supported: immediate addressing, postincrement addressing and indexed immediate addressing.

The processor makes use of instruction level parallelism, resulting in a VLIW ASIP with 72 bit instruction words stored in a dedicated program memory (Harvard architecture). The wide instruction word allows for a Load/Store operation to be combined with two AGUs updating the pointer registers, a checksum update and an ALU operation.

The pipeline is five stages long, although most instructions only need three stages to finish. The first stage, Instruction Fetch (IF), is followed by the instruction decode (ID) stage. Apart from generating various enable signals in this stage, both AGUs already calculate addresses and apply them to the memories, enabling the actual load or store operation in the next pipeline stage, called Execute 1 (E1). All ALU operations are read-modify-write instructions executed in the E1 stage. The next two pipeline stages E2 and E3 are exclusively used in the checksum engine.

Packet processing code is quite control intensive and therefore special care was taken optimizing the branch penalties. The compiler supports both delayed and stalling branches [6]. Furthermore, the wide instruction set allows

the encoding of multiple jump targets into a single instruction, which is quite usefull for accelerating C-style switch-case statements.
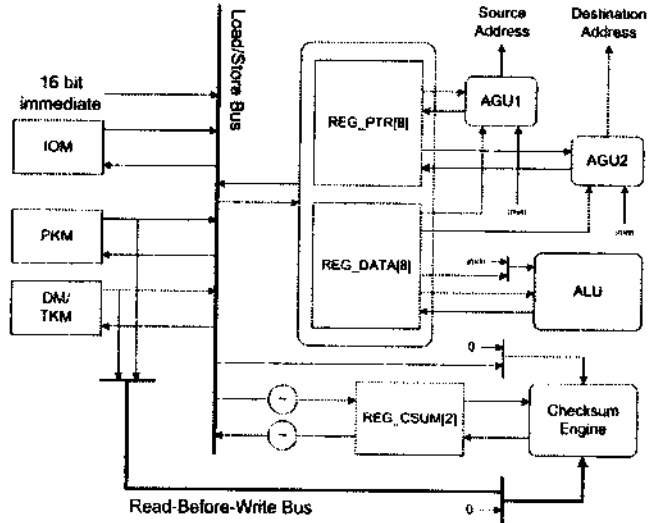

Fig. 2. ASIP Data path

Efficient algorithms have been developed to incrementally recalculate the checksums used throughout the TCP/IP stack [7]. Essentially, the checksum is updated with the ones complement subtraction of the new and the old contents, taking into account that packet headers contain the bitwise inversion of the actual checksum value.

This means that the old data has to be fetched from memory and subtracted from the current checksum before it can be overwritten with new data. The performance of such packet modifications can be greatly increased using the 'read-before-write' mode of the Xilinx BRAMs. This feature makes the old memory contents appear at the read port every time new data is written in.

We take advantage of this feature and update a dedicated bus (Read-Before-Write Bus) each time data is written to a memory. A single store instruction to memory is sufficient to produce all necessary data to update the checksum: The new data appears on the Load/Store bus in the E1 stage and the old data is made available on the Read-Before-Write Bus one cycle later in the E2 stage (Fig. 3.).
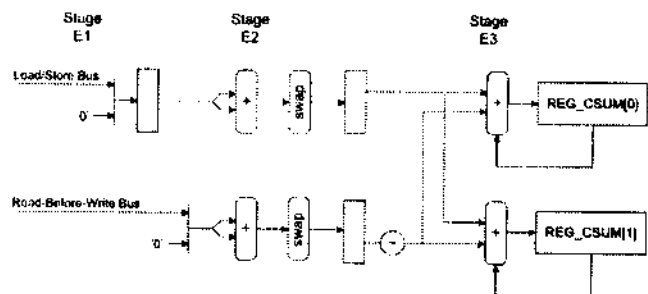

Fig. 3. Checksum Engine

651

6. PERFORMANCE ANALYSIS

6.1. Area & Speed

The design was implemented in a Xilinx Virtex 4 LX200-11 FPGA and takes about 1600 slices, of which 84% is used for the ASIP and the remainder contains the controllers supporting the interaction with the top level. This allows for 48 ASIPs to be placed inside the FPGA, if extra logic for external interfaces is not taken into account. The biggest entities in the ASIP are the register files (32%), the checksum engine (10%), the ALU (9%), the decoder (8%) and the global load/store bus (4%).

A post place and route frequency of 120 MHz can be reached. This corresponds to a clock cycle budget of 80 to complete a parsing, classification or packet manipulation algorithm, taking the maximum throughput of 1.488 Mpackets/s for a gigabit Ethernet link into account. The next paragraph shows that a single ASIP can handle throughput of a gigabit Ethernet link for one of the envisaged algorithms. Multiple cores running in parallel can easily support higher line rates.

6.2. Cycle Consumption

6.2.1. Parser
Various types of packets have been processed by the parsing algorithm, up to layer 4 if applicable. It is worth noting that the IPv6 performance is on par with IPv4 thanks to the high degree of parallelism in the processor.

Table 1. Parser performance in cycles

Packet Type	Cycle Consumption
IPv4 + UDP/TCP	57
IPv6 + UDP/TCP	57
VLAN + IPv4 + TCP	59
ICMP echo request	56
ARP	28

6.2.2. Classifier
The classification algorithm transforms the information stored in the ticket into a search key, applies it to the TCAM and reads back the result. The total operation, including TCAM latency, takes 40 clock cycles.

6.2.3. Packet Manipulator
Table 2 shows the performance figures for three typical packet manipulation scenarios, modifying the packets up to layer 4. Please note that these figures should not be added to get results for combined scenarios as each scenario shares initialisation code, executed once for each packet.

Table 2. Packet Manipulator performance in cycles

Scenario	Tasks	Cycle Consumption
1	Insert VLAN tag	32
2	Replace MAC Addresses Decrement IP TTL Update IP Checksum	29
3	Replace MAC Addresses Decrement IP TTL Change IP Source Address Change TCP Source Port Update IP & TCP Checksum	41

7. CONCLUSION

A processor architecture for a packet processing ASIP, tailored to the needs of a flow aware access node, was presented. A single ASIP is capable of handling parsing, classification or packet manipulation functionality for 1 Gbit/s Ethernet links. Furthermore, the ASIP can work seamlessly in a multi core environment, allowing excellent scalability to build a 10 Gbit/s flow aware access node.

8. ACKNOWLEDGEMENTS

9. REFERENCES

[1] K. Moerman, J. Fishburn, M. Lasserre, D. Ginsburg, "Utah's UTOPIA: An Ethernet-Based MPLS/VPLS Triple Play Deployment," *IEEE communications Magazine*, pp. 142 150, Nov 2005.

[2] Dr. Lawrence G. Roberts, "The next generation of IP - Flow Routing," in *Proc. SSGRR 2003 International Conference*, L'aquila, Italy, July 2003.

[3] S. Oueslati, J. Roberts, "A new direction for quality of service : Flow Aware Networking," in *Proc. Next Generation Internet Networks 2005*, April 2005, pp. 226 232.

[4] Y. Jiang, P. J. Emstad, A. Nevin, V. Nicola, M. Fidler, "Measurement-Based Admission Control for a flow aware network," in *Proc. Next Generation Internet Networks 2005*, April 2005, pp. 226 - 232.

[5] Target Compiler Technologies, htttp://www.retarget.com

[6] W. Stallings, "Computer Organisation and Architecture Desiging for performance," 6th Edition, pp. 484, 2003

[7] T. Mallory, A. Kullberg, "Incremental Updating of the Internet Checksum," RFC1141, January 1990.

2006 International Conference on Field Programmable Logic and Applications

Andreas Koch
Philip Leong
Eduardo Boemo

Editors