An Adaptive System-on-Chip for Network Applications

Roman Koch, Thilo Pionteck, Carsten Albrecht, and Erik Maehle

University of Lübeck
Institute of Computer Engineering
23538 Lübeck, Germany
{koch, pionteck, albrecht, maehle}@iti.uni-luebeck.de

Abstract

This paper presents the hardware architecture of DynaCORE, a dynamically reconfigurable system-on-chip for network applications. DynaCORE is an application specific coprocessor for offloading computationally intensive tasks from a network processor. The system-on-chip architecture is based on an adaptable network-on-chip which allows the dynamic replacement of hardware modules as well as the adaptation of the on-chip communication structure. The coprocessor leverages the active partial reconfiguration feature of modern FPGAs in order to adapt to shifting demand patterns. An embedded general-purpose processor core within the coprocessor runs software which manages the configurations of the device. With reference to a prototypical implementation targeting a Xilinx Virtex-II Pro FPGA, this paper focuses on on-chip communication issues. Topics include the integration of PowerPC processor cores into the configurable logic as well as the mode of operation of the network-on-chip.

1. Introduction

In an environment of continuously changing network protocols and ever increasing data rates, networking applications require processing devices that provide high degrees of both flexibility and computational power. Therefore in recent years, specialised network processors (NPs) were introduced. High performance NPs exploit chip-level parallel architectures comprising multiple instruction set processors as well as application specific hardware accelerators. While providing limited support for deep packet processing, that is, processing of entire packets, NPs are generally best suited for packet header processing as required by classification and forwarding applications. There is, however, an increasing demand for deep packet processing as several applications from the network security and integrity domain fall into this field. In many cases, frequently changing standards and protocols prevent these applications from taking advantage of hardware accelerators found in NPs. In contrast, software-only solutions hardly meet the performance requirements, in particular if huge amounts of payload require processing. Hence, other architectural solutions have to be found in order to cope with the increasing data rates in network applications.

FPGA based hardware accelerators are promising candidates for applications where high computational power is required in combination with flexibility. By replacing fixed hardware accelerators of NPs with dynamically reconfigurable modules, adaptation to changing standards and protocols is made possible. Futhermore, the usage of such modules also allows changing traffic profiles to be accounted for. Based on these ideas, DynaCORE [1] has been designed. DynaCORE is a run-time reconfigurable coprocessor for NPs and allows computationally intensive tasks to be offloaded from the NP. The NP still performs header processing tasks such as bridging, IP forwarding, and quality of service provisioning. Payload processing tasks like en-/decryption, compression or network intrusion detection are done within DynaCORE. NP and DynaCORE are loosely coupled. Thus, no changes to the NP architecture are necessary. In addition, the DynaCORE architecture is independent from any specific NP. A detailed description of the hardware architecture of DynaCORE is given in this paper.

This paper is organised as follows: Section 2 describes the requirements of the application area. The system architecture is presented in Section 3 while Section 4 explains how dynamic reconfiguration is exploited in DynaCORE. As DynaCORE comprises several communicating components, a network-on-

1-4244-0054-6/06/$20.00 ©2006 IEEE

chip (NoC) was chosen as the basis for the on-chip interconnection scheme. The NoC and its interfaces to the structural entities of DynaCORE, including CPU cores, are described in Section 5. Section 6 presents application scenarios before a short summary is given in Section 7.

2. Requirement Analysis

Network applications place high demands on computational power and real-time operation. Only a few ten nanoseconds per packet are available to evaluate incoming packet headers. Since capacities of optical links grow very rapidly about a factor of two per year, today's link speed already reaches 10 Gb/s (OC-192) and 40 Gb/s (OC-768). Assuming a worst case traffic scenario with TCP acknowledge packets with a packet size of 40 bytes only, this would lead to processing rates of about 25 or 100 million packets per second. Backbone traffic, however, consists in general of packets with an average size of 507 bytes [8], as traffic traces indicate.

For the last ten years, the composition of internet backbone traffic has been rather stable in respect of the protocol layers 1 to 4 of the OSI layer model. In contrast to lower layers, higher layer protocols vary more frequently. IP backbone traffic comprises of about 85.1% of TCP packets or 95.7% of TCP-based data volume. TCP is a stream-based, connection-oriented, and lossless transport protocol that provides congestion control. The residual traffic is mainly dominated by UDP. Only about 2% of packets and less than 1% of the data volume make up the non-TCP/UDP part [8]. At first view, IPsec and other secure traffic seem to be marginal, but secure connections are hardly visible at lower layers. Most secure connections are established on top of TCP so that they are invisible from the point of IP. Just a quarter of IP traffic is generated by Telnet, SSH, FTP, and SCP. Web applications using the http and https protocols generate the lion's share [4]. State-of-the-art network processors such as the Intel IXP 2400 provide bandwidths of 4 Gb/s and more. So, the IXP 2400 for example has to deal with 3.8 Gb/s TCP traffic including at least 0.125 Gb/s traffic for secure socket layer connections and about 0.125 Gb/s of IPsec traffic.

In the area of edge routers the traffic composition deviates. Virtual private networks (VPNs) often combine encryption with compression so that the computational effort is high. It has to be assumed that most routers transport similar packet flows without any need for encryption or compression. Conversely, a small number of routers generate a high amount of encrypted traffic. In those routers the ratio of encrypted to unencrypted traffic is several times higher than in a backbone. Reliable numbers, however, can hardly be given because of the lack of freely available statistics on edge-router traffic.

Encrypted and/or compressed data rarely uses the complete maximum transfer unit, but minimal-sized packets can be assumed to be rare as well. Design parameters such as buffer sizes and time-out values are to be adjusted to best ratio of performance and resource utilisation. Therefore, DynaCORE has to be designed to deal with average-sized packets. Worst-case assumptions base on 1 Gb/s incoming traffic stream and a packet size of 507 bytes. DynaCORE has about 4 ms time to classify and dispatch an incoming packet as well as for a dynamic exchange of functional units, if applicable.

3. System Architecture

This section covers the system level architecture of DynaCORE from a structural point of view. The dynamic behaviour of the system is considered in detail in subsequent sections.

The DynaCORE architecture is an adaptive system-on-chip (SoC) architecture specifically designed for high-speed networking applications. It relies on an underlying technology which allows blocks of logic circuitry within the chip to be replaced while the system is running. In addition, a general-purpose processor is supposed to be available for software based management and processing tasks. Appropriate technology is available in the form of recent FPGA families which are capable of *dynamic* or *active partial reconfiguration* [3] and, furthermore, include CPU cores in the FPGA. In the following, the term *reconfiguration* is used for the process of replacing blocks of logic circuitry. Reconfiguration enables DynaCORE to adapt to changing requirements by applying, as needed, different algorithms to data being transferred over a network.

As depicted in Figure 1, the DynaCORE architecture resolves into static and reconfigurable parts. The static part of DynaCORE is made up of global control and system management logic. In particular, it contains a dispatcher, a reconfiguration manager (RM) and interfaces for communication with the outside world. Utilising the I/O interfaces the receive unit receives data from the network. Subsequently, data is directed to the appropriate hardware assist (HA) by the dispatcher. Processed data is then sent back to the external network by the transmit unit making use of the I/O interfaces again. The RM takes care that the appropriate processing circuitry is available for the required type of processing. If necessary, it trig-

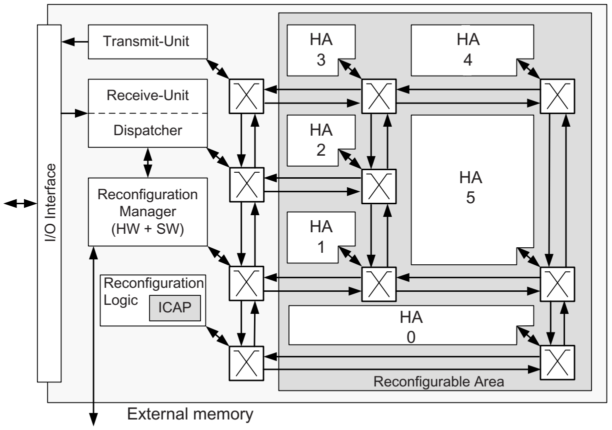

Figure 1. DynaCORE Block Diagram

gers and controls reconfiguration of DynaCORE. For that purpose, the RM communicates with reconfiguration logic encapsulating a particular hardware instance which is referred to as *internal configuration access port* (*ICAP* [9]). The ICAP interfaces DynaCORE logic with the underlying reconfigurable hardware. Besides the static part of the architecture there is a reconfigurable part which contains the actual processing logic in the form of HAs. Finally, the component interconnect is provided by a NoC which comprises both static and reconfigurable elements. The reconfigurability of the NoC structure allows for HAs of different sizes.

A standard HA is composed of an HA core, a monitor and logic interfacing the NoC. HA cores may be based on off-the-shelf intellectual property cores (IP cores). The monitor of an HA processes core state information and supplies the RM with data necessary for reconfiguration decisions. In addition to these standard HAs, there is also an HA that is based on an embedded processor core instead of an IP core. That is, it performs data processing tasks in software rather than in hardware. The software based HA serves as a universal low-performance backup for situations where the external demand pattern cannot be matched by a configuration of solely standard HAs. Futhermore, the software instance attenuates the effect of standard HAs being temporarily non-operational in the course of reconfiguration.

4. Reconfiguration

Adaptivity is the main feature of DynaCORE. In the following, the basis for reconfiguration decisions is explained and implementation specific issues are covered.

4.1. Reconfiguration Control

A conceptual entity, the reconfiguration control, is set on top of the system to guard and observe its complete functionality. The reconfiguration control is responsible for allocating reconfigurable logic and configuring functional cores. Its goals are to provide all functions required by the traffic profile and to achieve a balanced load distribution. It is comprised of three components to observe and optimise the coprocessor's performance: the *reconfiguration manager* (RM), the *dispatcher*, and the *performance monitors*.

Each HA hosts a performance monitor to collect information about its internal state. The collected data is forwarded to the RM. It may consist of, e.g., number of processed words and function-specific values. For example, a DES encryption core could be observed by the number of words processed, the number of keys used and the frequency of key changes in respect to a certain time period.

Decisions of the RM including creation and removal of new HAs solely base on this current state information. The set of potential configurations is precomputed and constitutes the state space of a finite state machine. Transitions are defined by rules that are basically composed of comparisons of thresholds and measurands [1]. Furthermore, the RM manages the flow mapping. The short-term assignment of incoming packets is performed by the dispatcher. Strictly speaking, the dispatcher executes the order of the RM for the crucial parts of protocol wrapping between the network processor, the Internet-protocol world, and DynaCORE with its proprietary internal protocol [2].

4.2 Implementation Issues

During the initial phase of the DynaCORE project Xilinx Virtex-II Pro FPGAs [9] were the only platform which provided for dynamic partial reconfiguration and which also met the hardware and software performance requirements. Therefore, the DynaCORE architecture has been designed with a prototypical implementation on a Xilinx Virtex-II Pro FPGA in mind. As the newer Virtex-4 FX family features a similar chip-level architecture, that implementation can be easily migrated to this platform just with marginal modifications. Following the decision for a specific implementation platform, the DynaCORE architecture was, to a certain degree, tailored according to the specific properties of that platform. Therefore, DynaCORE does not only rely on active partial reconfigurability, it also features components which directly map to the embedded PowerPC 405 processor cores, the ICAP and blocks of dual ported SRAM (BlockRAM), all of which can be found in Xilinx Virtex-II Pro and Virtex-4 FX FPGAs. External memory is used to store descriptions of all types of HAs and reconfigurable NoC elements. These descriptions are stored in the form of configuration bitstreams which are sequences of bytes that, when written to the ICAP, cause the respective HA or NoC element to be configured into the FPGA.

5. Network-on-Chip

This section covers the NoC which serves as the on-chip interconnection structure of DynaCORE as well as communication aspects of the components attached to the NoC.

5.1 Concept

DynaCORE aims at network applications or, more specifically, at packet payload processing tasks. The principle way of operation is therefore easily described as data packets being taken from an external network, fed into DynaCORE, directed through on-chip processing facilities and, finally, handed over to the external network. As a consequence, on-chip traffic is dominated by data bursts along certain paths rather than by an arbitrary exchange of messages between components. The actual data path, however, may vary depending on the particular task. In addition, the dynamically replaceable HAs may occupy areas of different sizes and thus require changes to the interconnection structure. Hence, a static interconnection scheme would not be suitable for DynaCORE. Therefore, DynaCORE builds up upon a reconfigurable NoC.

5.1.1 NoC Topology

The NoC consists of full-duplex links and switches with a maximum of three links to adjacent switches and one link to a local HA. Depending on the number of non-local links a linear network or a two-dimensional grid can be instantiated. The routing of packets inside the NoC is indirectly controlled by the RM which supplies the individual switches with routing tables.

In order to support HAs of varying sizes, switches can be dynamically inserted into and removed from the NoC. This provides the basis for replacing a large HA with two or more smaller HAs or vice versa. The mechanism for dynamically changing the NoC is essentially the same as the one applied for dynamically replacing HAs. It is based on dynamic reconfiguration controlled by the reconfiguration control unit in conjunction with the dispatcher. The RM maintains a global view of the NoC structure and ensures that connectivity constraints are always met.

5.1.2 Protocol Stack

While other NoCs like [6, 7, 10] make use of relatively lean protocols, the NoC used in DynaCORE utilises a three-layer protocol stack. The decision for a more complex protocol was made, because processing instances should be relocatable, there should be support for prioritised traffic, and a future expansion of the DynaCORE implementation to multiple FPGAs should be taken into account. The header structures of the individual layers are depicted in Figure 2.

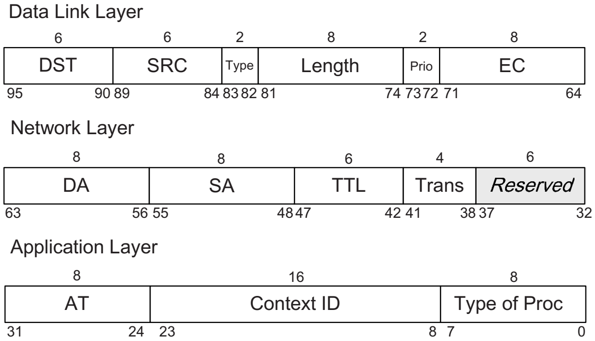

Figure 2. Protocol headers

Each header consists of 32 bits of data. At the lowest level, there is the data link header. It contains physical destination (DST) and source addresses (SRC), gives information on the number of subsequent 32 bit words that belong to the current packet (Length) as well as it differentiates several message types (Type) and priorities (Prio). The network-layer header provides logical destination (DA) and source addresses (SA) and a time-to-live (TTL) field that serves for breaking loops. For data packets, another logical destination address may be supplied in a compressed form (Trans = transit address). This address specifies to which component a packet should be sent after processing. At the highest level, the application-layer header contains a sub-address (AT = application type) and a parameter (Type of Proc = type of processing) for evaluation by an HA. In addition, this header carries a context identifier (Context ID) that allows different packets to be associated with each other.

While physical addresses refer to specific switches at specific locations within the NoC topology, the logical addresses refer to processing entities. Packets are routed from one switch to another solely based on the physical address DST. The addressed switch eventually passes the packet on to the locally connected HA. The HA, in turn, determines by the logical address DA whether any and which of its processing entities is addressed. A single HA may listen to multiple logical addresses, and it may also give its switch information on where to send packets which are misdirected, that is, which contain a DA the HA is not responsible for. This mechanism is primarily used in combination with dynamic reconfiguration as described in Section 4. The differentiation between physical and logical addresses allows HAs and sub-ordinate processing entities to be moved or combined within the NoC. Moreover, the protocol is prepared for an envisioned multi-chip system which comprises one NoC per chip. Here, physical addresses would be local to each chip and refer to switches as well as to inter-chip gateways while logical addresses remain valid in the system level context.

5.1.3 Reconfiguration

The NoC is prepared for two reconfiguration scenarios: First, it actively supports the replacement of individual HAs. Second, the NoC topology can be adapted by dynamically inserting or deleting switches.

Dynamic replacement of an HA. An HA *A* may be scheduled to be replaced by an HA *B* when the RM determines that the functionality provided by HA *B* is more urgently needed than that of HA *A*. At the time the replacement process starts, however, there may still be packets in transit within the NoC that are destined for HA *A*. Hence, precautions are necessary to ensure that these packets are properly processed even when HA *A* has already become absent. Therefore, the RM determines a backup HA *C* which provides the same functionality as HA *A* and assigns the logical addresses previously used by HA *A* to HA *C*. Generally, a software based HA as described in Section 3 serves as a universal backup HA. In case of the Virtex-II Pro implementation one of the two PowerPC cores is reserved for this purpose. Subsequent to actually replacing HA *A* by HA *B*, HA *B* is supplied with a mapping of the logical addresses of HA *A* to the physical address of backup HA *C*. Packets destined for HA *A* will now be routed to HA *B*, because HA *B* is located at the physical address of former HA *A*. HA *B* can now update the physical address DST of such packets according to the aforementioned mapping so that these packets are eventually forwarded to the backup HA *C*. This forwarding mechanism is supported by the generic HA interface which is a subcomponent of any HA and described in Section 5.2.

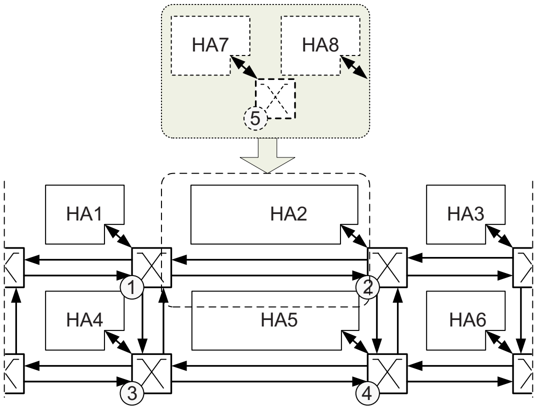

Figure 3. Adaptation of NoC structure

Dynamic insertion and removal of switches. When the structure of the NoC is changed, special care has to be taken not to isolate parts of the NoC. It has to be made sure that a new switch can be accessed by the neighbouring switches. A typical scenario is depicted in figure 3. Here, HA2 is replaced by two smaller HAs (HA7 and HA8). In order to connect HA7 to the NoC a new switch ⑤ has to be inserted in between of switches ① and ②. Since the dynamic reconfiguration should not affect other hardware modules and their communications, it must be guaranteed that packets are not sent directly from switch ① to switch ② or vice versa. Therefore, the routing tables of these switches have to be adapted. In addition, the routing table of switch ① should hold information to access the new switch ⑤. These updates are done under the control of the reconfiguration control unit which sends NoC internal messages containing the routing tables to the affected switches. The messages are addressed to the physical addresses of the switches and tagged highest priority, ensuring that they are processed with precedence within the packet switched NoC.

As soon as the diversion for packet traffic has been set up, switch ⑤ is added to the system together with the new HAs. Now the first step is to initialise the routing table of the new switch. Therefore, a corresponding message is generated by the reconfiguration control unit. The message can successfully be forwarded by switch ① as it obtained knowledge of switch ⑤ and its physical address with the recent update of its routing table. The final step consists of updating the routing tables of all surrounding switches, thereby cancelling the diversion and activating the connections between switches ①, ⑤, and ②. In the case of removing a switch from the NoC essentially the same procedure is applied.

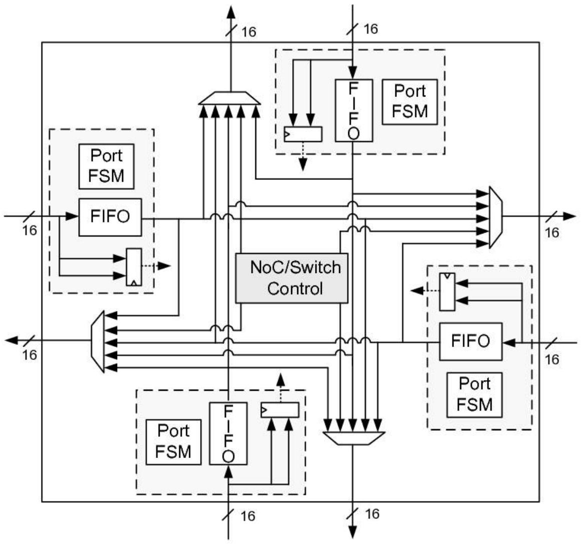

Figure 4. Internal switch architecture

5.2 Implementation

The main component of the NoC is the switch. Its block structure is depicted in Figure 4. For each of the 16 bit wide input ports of the switch, a Virtex-II Pro BlockRAM is used as a FIFO allowing storage of a complete packet. The first 32 bit of the packet header are bypassed to the switch controller so that header analysation can start immediately. Beside the TTL field, only the data-link layer is processed. At first, the priority of the incoming packet is determined in order to regulate the access to the routing table. Arbitration is done by using the Prio and the Type field of the header. In the case of equal priorities, access to the routing table is granted round robin. The routing table consists of 64 entries of 2 bit and is mapped onto DistributedRAM. Depending on the destination address in the packet header a port is assigned and the packet is immediately forwarded to the next switch or to the local HA. In case of a free destination port the header processing inside a switch is done in five clock cycles. If the destination port is not available, the complete packet is buffered in the FIFO of the corresponding input port. Meanwhile, the switch sends a NoC internal message to its predecessor in order to prevent the arrival of further packets.

The routing tables for the switches are provided by the RM. They are generated after initialisation, a change in the structure of the NoC and when HAs are exchanged. For their immediate distribution over the NoC, a special NoC internal message class is used. Table 5.2 provides an overview over the different message classes of the NoC. In total there exists six message classes. Differentiation is done using the Type (T) and Prio (P) field of the data link layer.

Table 1. NoC message classes

T P	Class	Description
00 00	Payload	Payload data for HAs
00 01	Monitor	Data to/from HA monitor
01 10	Log_Addr Forw_Addr	Logical addr. for HA interface Addresses for packet forwarding *AT field is used for distinction*
10 10	Route_Tbl	Routing tables for the switches
10 11	NoC_Msg	Internal message for flow control
11 11	Rec_Data	Data for dynamic reconf.

When the physical address of a packet matches the address of a switch, the packet is forwarded to the HA. The structure of the HA is shown in Figure 5. A special interface inside the HA separates the local port of the switch from the actual IP core. This avoids any adaptation of the switch to the bitwidth or communication protocol of an IP core. The HA interface consists of a receive and transmit unit, an HA identifier block and the HA monitor. According to the logical address and the AT field, the receive units transfers the incoming packet to one of the components inside an HA. In addition, it converts the bitwidth of the different components and provides an additional buffer. The actual determination of the destination of a packet is done inside the HA identifier block. Therefore, local addresses of the HA and potential forwarding addresses are stored inside this block. If the packet matches a logical address of the HA, the payload is transferred to the IP core and header information is bypassed to the transmit unit and the HA identifier. The HA identifier block determines the new destination address for the processed payload data. This information is passed to the transmit unit which creates a new header.

If a packet matches the physical address of the switch but not any of the logical addresses of the HA, it is either targeted to the HA monitor or to an HA which was formely attached to the switch. In the second case it is directly transferred from the receive unit to the transmit unit of the HA. Inside the transmit unit, the physical and logical destination addresses of the original header are overwritten by the forwarding addresses stored in the HA identifier block. Afterwards, the packet is transferred to the switch. If a packet is assigned to the HA monitor, the application layer and optional payload are forwarded to the HA monitor. The HA monitor is capable of sending packets to the reconfiguration control unit without being trig-

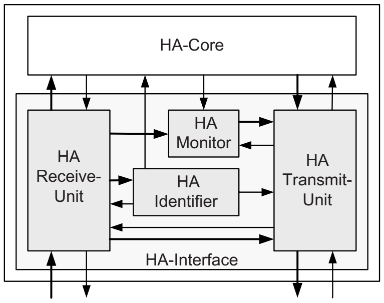

Figure 5. Generic structure of an HA

gered. Therefore, the HA monitor can create packet headers independently from formerly received packet headers.

5.3 Integration of PowerPC Cores

The prototypical implementation of DynaCORE is based on a Virtex-II Pro XC2VP30 FPGA that contains two PowerPC 405 processor cores, one of which is dedicated to the RM software while the other is used for software based HAs. Both cores are connected to external memory which serves as the storage for both program code and data. As described in Section 3, the external memory also stores configuration bitstreams.

The PowerPC 405 cores provide two interfaces which are suitable for communication with the surrounding logic: The processor local bus (PLB) and the on-chip memory bus (OCM bus). Unlike a microcontroller, the PowerPC does not feature any programmable I/Os. However, although the OCM bus is exclusively specified for use with BlockRAM blocks on the FPGA, it has been successfully used for also interfacing the PowerPC core with arbitrary logic. That way, general purpose I/Os have been implemented very efficiently with regard to resource usage. In view of the complexity inherent to logic connecting to the PLB, the OCM bus was thus chosen for connecting the PowerPC cores to the NoC. The PLB is solely used to connect the PowerPC cores to external memory.

As a consequence, the RM software must actively transfer configuration data from the external memory to the reconfiguration logic word by word. At a first glance, this approach may appear to be sub-optimal compared to a solution based on direct transfers from memory to reconfiguration logic. A closer view, however, reveals that there is not any negative impact on the performance of the system. The processor core may be clocked at a frequency several times as high as the surrounding logic. Thus, the need for multiple machine instructions per data word to be copied does not necessarily slow down the process. Moreover, the copy routine can safely be assumed to reside either in the L1 instruction cache or to be stored in on-chip memory so that instruction fetches do not require access to external memory. Accordingly, the external memory bandwidth is exclusively available to the bitstream copying process. More specifically, the processor cores of an XC2VP30FF896-6 can be clocked at up to 350 MHz. In the experimental system, the external DDR-SDRAM is clocked at 100 MHz and connected by a 32 bit data bus. These figures indicate that is quite feasible to transfer 8 bit words to the ICAP at a rate of 50 MHz which is the maximum frequency allowed for configuration. Compared to a solution involving a second PLB master besides the PowerPCs, the overall complexity of the system is significantly reduced.

6. Application Scenario

The current focus of applying DynaCORE is set on stream-based networking applications. These applications demand a huge variety of different algorithms which can hardly be supported by a single static device. In order to support networking applications such as edge router functionalities, e.g., packet filtering for firewalls or network intrusion detection systems, establishing VPNs, content-based switching, etc., DynaCORE is attached to an NP. The NP allows an appropriate integration of DynaCORE because of its flexible software programmability.

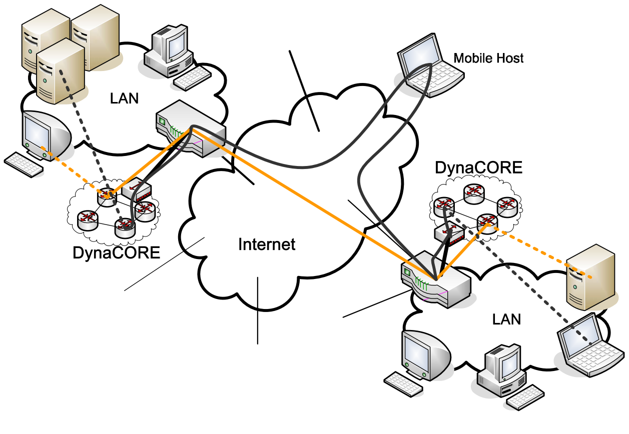

Figure 6. VPN configuration example

A suitable scenario for DynaCORE is a VPN to connect at least two separated local area networks (LANs) and/or other spread VPN participants. VPNs usually encrypt their packet payload to ensure security and frequently compress it to reduce the amount of data. The need for compression is generally motivated using strongly limited carriers such as wireless technologies. Figure 6 shows a feasible configuration. Two LANs are combined to a VPN using an edge-to-edge tunnel with encryption so that any internal VPN connection is passed through this tunnel. Additionally, a mobile host with a probably low-bandwidth connection is incorporated into the VPN by an end-to-edge tunnel with encryption and compression. DynaCORE is applied at the edge routers to perform the deep-packet processing of these applications. The solid line shows the connection of the VPN terminals, whereas the dashed line indicates the internal LAN link to the destination. In detail, the en-/decryption for both types of connection is performed as well as the compression for the mobile-host connection. The internal set-up of DynaCORE highly depends on the traffic volume. It is reasonable to assume that the edge-to-edge tunnel is more frequently used than the end-to-edge connection. Moreover, it is assumed that both connections use the same cryptography algorithm, e.g. 3DES, with different keys. So, including the figures of Section 2 three components may be installed. Two 3DES cores and a compression core would fit this scenario. Note that the load distribution between these two 3DES cores cannot be uniform. Because of the different keys, one core might deal with two keys. Each key exchange costs a couple of cycles so that its performance falls off in comparison to the other. Thus, less packets will be forwarded to key-exchanging core.

Current edge-router technologies such as the router extension Encryption Services PIC [5] of Juniper Networks support loads of VPN tunnels and deliver reasonable performance. Nevertheless, the system integration is limited to appropriate router systems and cryptography algorithms are fixed. Therefore, the flexibility degree is improvable and demands more flexible stream-processing architectures.

7. Summary

Network applications span a very agile field of computing tasks. Current technologies achieve their limits to fulfil all requirements. Network processors lack high-performance deep packet processing capabilities whereas pure hardware solutions cannot cope with changes in the field. Architectural features of DynaCORE presented here apply run-time reconfigurability to close this gap. The ability to compete with the bandwidth required bases on the NoC communication structure. The advantage of high bandwidth prevails over the latency that is included in switched networks. Basic NoC components are switches and HA interfaces, complete HAs are built by attaching standard IP cores to the generic HA interface. Supervised by the reconfiguration manager the reconfiguration process with a changing NoC topology is described. The protocol features, in particular the layers, are emphasised in this context. Moreover, the system's feasibility is demonstrated by describing the crucial implementation details on a partially reconfigurable FPGA with embedded processor cores, the Xilinx Virtex-II Pro.

8. Acknowledgement

This work was funded in part by the German Research Foundation (DFG) within priority programme 1148 under grant reference Ma 1412/5.

References

[1] C. Albrecht, J. Foag, R. Koch, and E. Maehle. DynaCORE – a dynamically reconfigurable coprocessor architecture for network processors. In *Proc. of the Euromicro Conference on Parallel, Distributed and Network-Centric Processing 2006*, Feb. 2006.

[2] C. Albrecht, R. Koch, and T. Pionteck. On the design of a loosely-coupled run-time reconfigurable network coprocessor. In *Proc. of the Workshop on Media and Streaming Processors 2005 (MSP-7)*, Nov. 2005.

[3] P. Butel, G. Habay, and A. Rachet. Managing Partial Dynamic Reconfiguration in Virtex-II Pro FPGAs. *Xcell Journal*, 50:32–37, Aug. 2004.

[4] Y. Fei, J. Jones, K. Lakkas, and Y. Zheng. Measurement of the Usage of Several Secure Internet Protocols from Internet Traces. Student Project, Dept. of Computer Science and Engineering, UCSD, CA, USA, 2002.

[5] Juniper Networks Inc., 1194 North Mathilda Avenue, Sunnyvale, CA 94089 USA. *Encryption Services (ES) PIC*, Nov. 2004.

[6] A. Mello, L. Moller, N. Calazans, and F. Moraes. MultiNoC: A multiprocessing system enabled by a network on chip. In *DATE '05: Proc. of the conference on Design, Automation and Test in Europe*, pages 234–239, Washington, DC, USA, 2005.

[7] R. Soares, I. S. Silva, and A. Azevedo. When reconfigurable architecture meets network-on-chip. In *SBCCI '04: Proc. of the 17th symposium on Integrated circuits and system design*, pages 216–221, New York, NY, USA, 2004. ACM Press.

[8] Sprint Corporation. IP Monitoring Project. URL: http://ipmon.sprint.com/, 2001–2004.

[9] Xilinx, Inc. *Virtex-II Pro and Virtex-II Pro X FPGA User Guide*, Mar. 2005.

[10] H. Zimmer and A. Jantsch. A fault model notation and error-control scheme for switch-to-switch buses in a network-on-chip. In *CODES+ISSS '03: Proc. of the 1st IEEE/ACM/IFIP int'l conference on Hardware/software codesign and system synthesis*, pages 188–193, New York, NY, USA, 2003. ACM Press.