A Data Protection Unit for NoC-based Architectures

Leandro Fiorin[†] Gianluca Palermo[‡] Slobodan Lukovic[†] Cristina Silvano[‡]

[†]ALaRI - Faculty of Informatics - University of Lugano
Via Buffi 13, 6904, Lugano, Switzerland

[‡]Politecnico di Milano - Dipartimento di Elettronica e Informazione
Via Ponzio 34/5, 20133, Milano, Italy

{fiorin,lukovics}@alari.ch {gpalermo,silvano}@elet.polimi.it

ABSTRACT

Security is gaining increasing relevance in the development of embedded devices. Towards a secure system at each level of design, this paper addresses the security aspects related to Network-on-Chip (NoC) architectures, foreseen as the communication infrastructure of next-generation embedded devices. In the context of NoC-based Multiprocessor systems, we focus on the topic, not thoroughly faced yet, of data protection.

We present the architecture of a Data Protection Unit (DPU) designed for implementation within the Network Interface (NI). The DPU supports the capability to check and limit the access rights (none, read, write or both) of processors requesting access to data locations in a shared memory - in particular distinguishing between the operating roles (supervisor or user) of processing elements. We explore different alternative implementations and demonstrate how the DPU unit does not affect the network latency if the memory request has the appropriate rights. In the experimental section we show synthesis results for different ASIC implementations of the Data Protection Unit.

Categories and Subject Descriptors

C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures (Multiprocessors)—*Interconnection architectures (e.g., common bus, multiport memory, crossbar switch)*; C.1.4 [**Processor Architectures**]: Parallel Architectures—*Distributed architectures*; D.4.6 [**Operating Systems**]: Security and Protection—*Access controls, Authetication*

General Terms

Design, Security

Keywords

Data Protection, Network-on-Chip (NoC), MultiProcessor System-on-Chip (MPSoC), Security, Embedded Systems

1. INTRODUCTION

The level of integration that silicon technology has reached in the past few years allows the use of advanced design processes for enabling applications that were to date infeasible. In fact, it is now possible to increase the number of cores per die and the complexity of the interactions among them. The complexity of new systems brings the challenge of enabling reliable communication channels between cores; a challenge that becomes more and more difficult as the number of integrated cores per design increases. The traditional solution for inter-core communication will soon become unable to guarantee sufficient levels of efficiency, both from performance and power consumption perspectives. A promising alternative, Networks-on-Chips (NoCs) [7, 4], has appeared as a new strategy to connect and manage the communication between a variety of design elements and intellectual property blocks required in complex System-on-Chips (SoCs).

However, the advantages introduced by the use of such a complex communication infrastructure may introduce new weaknesses in the system that could be potentially critical and should be subjected to careful study and evaluation. Indeed, as computing and communication increasingly pervade our lives, security and protection of sensitive data emerge as issues of extreme importance, in particular in the case of embedded systems, which, due to their intrinsic constraints, present several unique security challenges [11].

While NoC has been an emerging area of research interest, security in such systems remains mainly unexplored, even if security-aware design of communication architectures is becoming a necessity in the context of the overall embedded SoC/device security. In this scenario, this paper presents a Data Protection Unit (DPU) for NoC-based architectures. The goal of the proposed module is to contrast forms of attack such as the buffer overflow [5], aiming at obtaining unauthorized access to selected blocks of memory. Protection of memory is particularly relevant for Multiprocessor systems, where several processing elements may share the same physical memory. Access restriction is therefore necessary in order to avoid the subtraction of sensitive information or the tampering of data and instructions by some compromised core. The DPU allows performing secure accesses to memories and/or memory-mapped peripherals. In general, the proposed DPU consists of a hardware solution that enables the access to a particular memory space only if the initiator of the request is authorized to perform that particular operation. Filtering of the access is done by considering not only the memory address but also the type of the operation requested (load/store) and the status of the initiator (user or supervisor mode). The use of the Data Protection Unit gives the possibility to easily store (or load) critical data, without requiring time-consuming encryption (or decryption) and maintaining a fast memory access.

This paper is organized as follows: Section 2 provides an overview of the current state of the art of security solutions implemented in NoCs and of academic and industrial approaches for data protection in embedded systems. Section 3 defines the architecture of the proposed Data Protection Unit. In Section 4, several design alternatives are proposed and implementation issues are discussed. Finally, synthesis results for alternative implementations of the DPU are reported in Section 5, and conclusions and future work are discussed in Section 6.

2. RELATED WORK

In this section, we outline and discuss the related work following two complementary aspects of the object of the research presented in this paper. On the one hand, we overview academic works addressing security aspects particularly related to NoCs implementations; on the other hand, we discuss academic and industrial proposals to implement memory data protection in systems adopting conventional busses for on-chip communication.

With regard to academic work addressing security aspects in NoCs, [9] and [10] present a framework to secure the exchange of cryptographic keys within a NoC, addressing in particular the protection from power/EM attacks of a system containing not-secure cores as well as secure ones (defined as hardware IP cores which can execute one or more security applications). The framework supports authentication, encryption, key exchange, new user keys and public key storage, and similar procedures. The proposed methodology ensures that no unencrypted key leaves cores on the NoC and additionally supports IP secure cores running only trusted software. At the network level, security is based on symmetric key cryptography, where each secure core has its own security wrapper storing in a non-volatile memory a private network key. A key-keeper secure core is responsible for key distribution on the NoC and keeps encrypted keys for individual applications, encrypted user private keys or other users public keys. New keys can be downloaded and stored in the key-keeper core using encryption techniques and the functionalities offered by the secure cores in the system.

Diguet et al. [8] propose a first solution to secure a reconfigurable SoC based on NoC. The system is composed of *Secure Network Interfaces* (SNIs) and a *Secure Configuration Manager* (SCM). The SNIs act as filter for the network and handle attack symptoms that may be caused by denial of service attacks and unauthorized read/write accesses. The SCM configures system resources and network interfaces, monitoring the system for possible attacks. A routing technique based on reversed forward path calculation is proposed. The technique allows verifying that the sender of the packet arrived at a specific SNI has the right to communicate with it.

With respect to the above presented related work, the subject of our paper can be considered orthogonal and complementary, since we investigate a solution for the specific problem of data protection in NoCs.

Focusing on related work on memory data protection in embedded systems, a specific implementation of a protection unit for data stored in memory is described in [6]. The proposed module enforces access control rules that specify how a component can access a device in a particular context. AMBA bus transactions are monitored in order to discover specifics attacks, such as in the case of theuse of buffer overflow to steal the cryptographic key employed in Digital Right Management. A look-up table, indexed by the concatenation of the master identifier signals and the system address bus, is employed to store and check right accesses for the addressed memory location and to stop potential not allowed initiators.

Considering commercial implementations of on-chip memory protection units, *ARM* provides, in systems adopting the ARM TrustZone technology [3], the possibility to include a specific module - the AXI TrustZone memory adapter - to support secure-aware memory blocks. A single memory cell of up to 2MB can be shared between secure and non-secure storage area. Transactions on the bus are monitored to detect addressed memory region and security mode in order to cancel non-secure accesses to secure regions and accesses to beyond of the maximum address memory size. The module is configured through the TrustZone Protection Controller, which manages the secure mode of the various components of the TrustZone-based system and provides the software interface to set up the security status of the memory areas.

A similar solution is provided by *Sonics* [2] in its SMART Interconnect solutions. An on-chip programmable security "firewall" is employed to protect the system integrity and the media content passed between on-chip processing blocks and various I/Os and the memory subsystem. The firewall is implemented through an optional access protection mechanism to designate protection regions within the address space of specified targets. The mechanism can be dynamic, with protection region sizes and locations that can be programmed at run-time. It can also be role-dependent, with permissions defined as a function not only of which initiator is attempting to access but also which processing role the initiator is performing at that time. Protection regions subdivide a targets address space, where each target can have up to 8 protection regions. Each protection region is assigned to one of four levels of priority.

This paper faces for the first time the problem of the data protection on a NoC-based Multiprocessor System-on-Chip (MPSoC). This work presents a solution for data protection with a finest granularity (especially with respect to the ARM TrustZone), which will be more useful for the next generation of security-enhanced Multiprocessor Systems. The paper presents an architecture of the Data Protection Unit that does not affect the network latency and a set of design alternatives for the reduction of the Area/Energy overhead for different designs. Briefly, this paper represents a step over the previous three implementations of data protection.

3. SYSTEM ARCHITECTURE AND DATA PROTECTION UNIT

In this section, we describe the general characteristics of the proposed Data Protection Unit (DPU) for architectures adopting the Networks-on-Chips communication paradigm.

Before discussing the DPU module, we briefly introduce the system in which it interacts, in particular focusing on the communication infrastructure. As example, Figure 1(a) shows an overview of a simplified multiprocessor system,

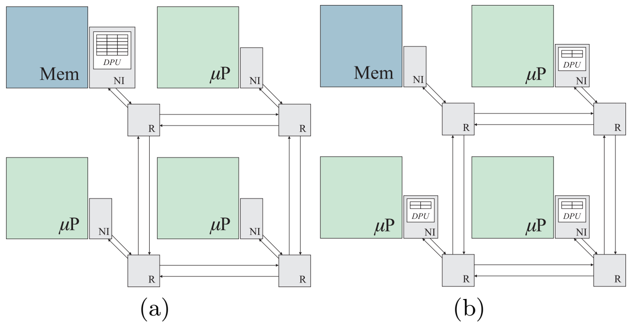


Figure 1: Simple system with three initiators and one target showing the two different network architectures using the DPU coupled (a) with the NI target and (b) with the NIs master

composed of three processors (μP) and a shared memory (Mem). A Network-on-Chip connects together the elements of the system. Without loss of generality, we assume the elements on the NoC to be memory mapped and packets created in the NI to be sent using wormhole control flow. We also assume NIs to be "secure", meaning with this term that no external attacker could modify their configuration and behavior. In particular, we assume that the NI contains a register hard-wired or not modifyable by the IPs that univocally identify the node within the network.

In order to describe the features offered by the DPU and possible architectures to implement it, we briefly present now the packet structure used within the NoC. In fact, the implementation proposed for the DPU depends on the specific structure of the packet, even if approaches similar to the one employed can be easily used for different protocols. We adopt a protocol packet such as the one shown in Figure 2. *DestID* is used to identify the target of the request (we assume a table-based routing depending on source and destination addresses); *SourceID* identifies the initiator of the transaction and depending on the granularity of the system could refer to the identification number of the node in the NoC, to the single IP in a cluster (assuming more IPs connected to the NoC through the same NI) or, in future extensions of the work, to a thread running on a specific IP core. *MemAddr* is the memory address to which the initiator is requesting access, while the *Length* field is used to communicate the length, in number of words, of the information sent or to be retrieved. *L/S* encodes which kind of operation (load, store) the initiator requests at the target memory address, while *Role* determines the particular role (status) of the processor (user, supervisor) associated with the request. In case of a *load* request, the packet is composed of just the header, while for a *store* the header is followed by the data to be written in memory.

It is important to note that in our implementation the header of the packet structure contains all the information needed by the network and by the Data Protection Unit to work correctly. As examples, the size of the source and destination fields were designed for a network with up to 255 nodes and the role field to only support the supervisor and user mode of the processor. These values could be easily extended for more nodes and more roles, increasing the size of the related field in the packet but without impacting the general behavior of the DPU architecture.

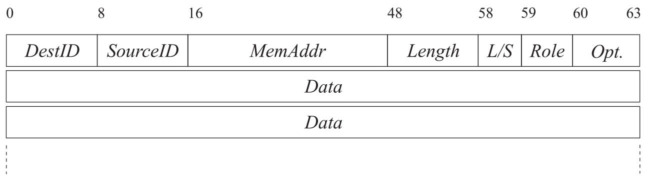


Figure 2: Structure of the packet used within the NoC

The DPU is a hardware module that enforces access control rules specifying the way in which a component connected to the NoC can access the blocks in which a memory can be divided. The division of the memory in blocks is necessary to allow a separation between sensitive and no-sensitive data of the different processors. In the basic idea, the DPU is a module embedded in the Network Interface of the target memory to protect (or the memory-mapped peripheral), supplying services similar to those offered by a classical "firewall" in data networks. The Network Interface receives packets coming from several initiators requesting access to the target memory. While processing the packet, the information contained in the header is passed to the DPU, which lookups the access rights for the requesting packet and checks if the requested operation is allowed, granting or denying the access of the data to the memory block. The most relevant part of the DPU is represented by the lookup table. In hardware this element is commonly implemented by combining a typical Content Addressable Memory (CAM) [12], used in fully associative memory and data networks routers, and a RAM storing the access rights (load, store, both or none). It is important to note that coupling the DPU to the NI guarantees that no additional latency is associated with the access right check since the protocol conversion and the DPU access are performed in parallel.

4. IMPLEMENTATION OF THE DPU

In this section, we present several alternative implementations of the DPU architecture, describing their advantages and disadvantages. We show two implementations, as described before, realized at the NI of the target memory (see Figure 1(a)) and we compare them with two implementations of the DPU distributed at the NIs master, i.e. the NI of the initiators of the network transaction (see Figure 1(b)). To distinguish them in the following discussions, we named *T0* and *T1* the DPU architectures embedded in the targets' NI and *I0* and *I1* those in the initiators' NI.

4.1 Basic architecture at the target (T0)

The first architecture proposed for the DPU is shown in detail in Figure 3. Each entry in the lookup table is indexed by the concatenation of the *SourceID* with the *Role* and the requested memory address *MemAddr*. In our implementation we assume 4KB as the size of the smallest memory block to be managed for the access rights. This means that all the data within the same block of 4KB have the same right. We use Ternary Content Addressable Memory (TCAM) [12] to compact the table, grouping ranges of keys in one entry. TCAM, in addition to logic 1 or logic 0, allows to store a *dont care* (X) value in those positions in which either a 1 or a 0 matches the entry key. A ternary symbol is encoded

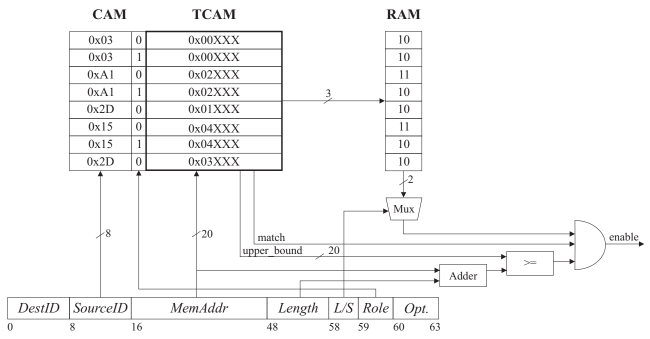


Figure 3: DPU basic architecture at the Network Interface target (T0)

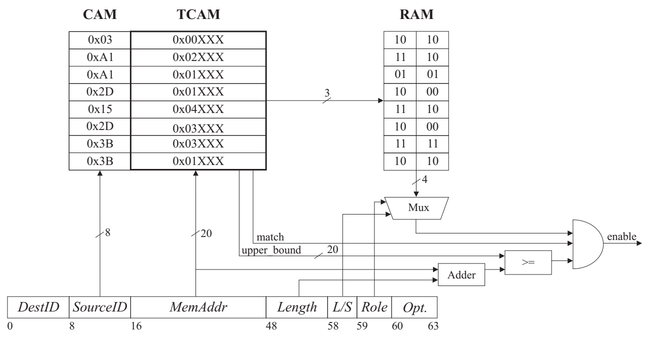


Figure 4: DPU modified architecture at the Network interface target (T1)

adding to a CAM cell the storage for a mask bit, set to logic 1 to not consider the value store in the CAM, and at the (approximated) cost of an additional memory cell.

In order to minimize the overhead, we use TCAM only for those bits looking up the requested memory address (*MemAddr*), while CAM cells for the other fields of the entry key. In fact, for efficiency, only allowed accesses are recorded in the table and we believe to be more convenient to add an entry line to specify the access right of an initiator, instead of maintaining the fixed overhead due to a global TCAM. On the other hand, the use of the TCAM for the *MemAddr* bits allows to lookup all the requests to the same memory block. Moreover, it allows us to check that the length of the data requested to be loaded or stored does not exceed the block boundaries. In fact, the address space of the memory blocks it is delimitated by the starting address stored in the CAM and by the sum of the values in the CAM and those in the mask bits.

As shown in Figure 3, we modified the common TCAM architecture in order to provide as output also the upper bound address of the memory block. Once the packet header is received, the length of the data sent or requested is added to the *MemAddr* and compared to the value provided by our modified TCAM to determine if boundaries are respected. The RAM is used to store the access rights related to the particular entry key in the CAM/TCAM and to provide them in case of successful match. Two bits are used to encode the load/store permission, with bits set to logic '1' for allowed accesses.

To summarize, a match between the packet information and the values stored in the CAM, a data dimension within the boundaries of the memory block, and a requested operation complying with those allowed, assure to the initiator the access to the desired addresses in memory. In the case of overlapping of different blocks and equal access rights of the initiator to the two (or more) memory blocks, the block with lower starting address is considered.

4.2 Modified architecture at the target (T1)

An alternative implementation to the one previously described is shown in Figure 4. In this case, the RAM of the lookup table stores the access rights for the two different possible roles of the initiator (Role 0 - load/store; Role 1 - load/store). While increasing the number of memory cells associated to the RAM, this solution, as we will show in the next subsection, can halve the number of necessary entry lines and approximately the overhead associated to the DPU implementation, in particular in the worst-case scenario (all the initiators wanting to access with both roles all of the memory blocks). For duality, if we consider an implementation of the DPU with a fixed number of entry lines for the lookup table, the second proposed architecture could double the number of allowed configurations, when compared with the previous one. However, advantages will be reduced for configurations in which a relevant number of initiators access with only one role to the memory blocks. Obviously, in configurations in which all the initiators access to the memory in the hypothesized way, the architecture T1 will be more costly then T0.

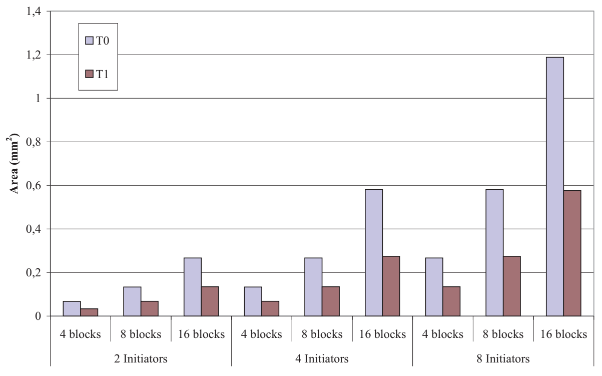
4.3 Architectures at the initiator (I0, I1)

We also explore the case in which the data protection unit is distributed and located in the Network Interfaces of the initiators. Figure 1(b) shows an overview of the distributed architecture. In this case, the number of lines in the lookup table depends on the number of targets to which the initiator can send packets and on the number of blocks in each target memory. Access control is performed at the same time than address conversion into network ID, making possible to have an overhead smaller that the one in the equivalent architectures at the target. In fact, it is not necessary to include in the lookup table *DestID* (dual of *SourceID* at the initiator), being the lookup done directly on the address coming from the processing unit.

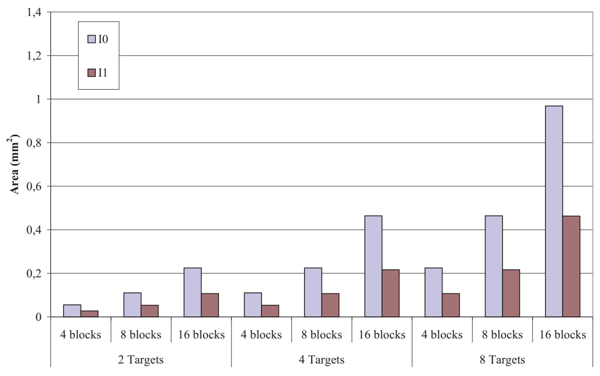
As already done for the architecture at the target, we explore the two implementations including in one case the *Role* bit in the lookup table and in the other in the RAM. As shown in particular in the next section, adopting a distributed architecture reduces the overhead associated to the protection unit. In fact, if compared to the architectures at the target, the lookup table is of smaller dimensions. Moreover, checking the access rights at the initiator's NI avoids having in the NoC the waste of bandwidth caused by the packets that would be rejected at destination, as it is in the case of the architecture at the target. However, locating the DPU at the initiator has practical drawbacks due to the increased difficulty of programming a distributed protection unit, in particular in the case of frequently changing scenarios and applications.

4.4 Interaction with the OS

We included in the Data Protection Unit the possibility to distinguish the role of the initiator during the memory request. This feature is useful for a more detailed identifica-

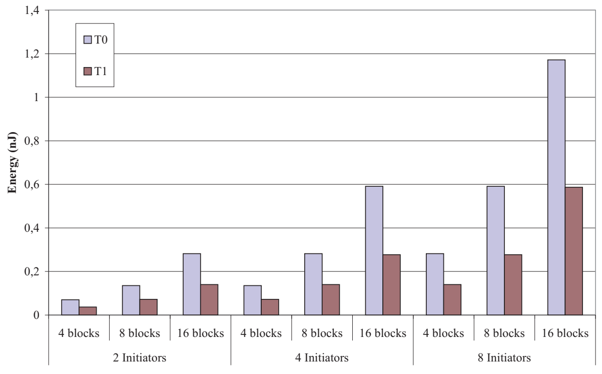


(a) DPU on the NI Target

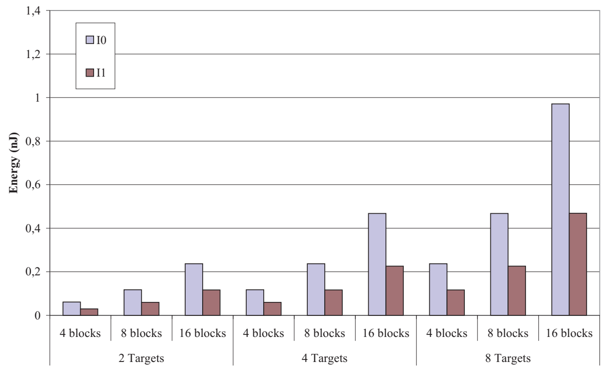


(b) DPU on the NI Initiator

Figure 5: DPU area overhead for several configurations in terms of number of initiators, targets and memory blocks, type of architectures and network position.



(a) DPU on the NI Target



(b) DPU on the NI Initiator

Figure 6: DPU energy for access for several configurations in terms of number of initiators, targets and memory blocks, type of architectures and network position.

tion of the initiator, especially when it operates as a heterogeneous software entity. In a complex architecture running an Operating System (OS), the role of the initiator can be associated to the software *mode* (supervisor or user) of the processor.

In our approach, this information is supposed to be propagated from the processor to a particular register in the Network Interface or, if adopting an interface to the processor similar to the one suggested by the OCP/IP specifications [1], special signals to be driven only if the processor is in a secure mode. The first hypothesis can be easily satisfied by considering the possibility to have a memory mapped register on the NI close to the processor and the possibility to execute a couple of functions executable only in supervisor mode. In this case, the memory mapped register will be used to store the status (*role*) of the software execution (user or supervisor mode) while the two functions, both implementing a simple store operation, will be used to change the status within this role-register. In fact, passing from supervisor to user mode the last function call in supervisor mode will set the role-register on *user*, while passing from user to supervisor the first call to be executed will be the function used to set the role-register to *supervisor*.

The role-register within the network interface can be also used in the same way to store information about the execution of secure services, drivers or kernels with respect to normal OS or applications, as in [3].

5. SYNTHESIS RESULTS

In this section, we present the synthesis results for the four different implementations (T0, T1, I0, I1) of the DPU architecture presented in Section 3 obtained by using the 0.13μm HCMOS9GPHS STMicroelectronics technology library. In the second part, we also show two case studies for the whole system architecture to underline the overhead associated with the data protection, comparing different implementations of the DPU (T1 and I1).

The graphs in Figures 5 and 6 show area (in mm^2) and energy (in nJ) associated with different combinations of initiators/memory blocks/architectures (T0, T1) and targets/memory blocks/architectures (I0, I1). All the synthesis were performed for a given clock frequency of 500MHz and timing constraints are met for all the configurations explored. As already introduced, the graphs show results in the worst case scenario, corresponding to implementations of the DPU allowing covering all possible combinations of

Table 1: Area and energy overhead due to the Data Protection Unit for the whole system, considering two different target architectures of the network and two different DPU implementations.

		Area $[mm^2]$	**Energy** $[pJ]$
Arch1	T1	0.268	36.5
	I1	0.221	117.4
Arch2	T1	0.540	508.6
	I1	0.443	59.9

parameters. In case of T0 (I0), this would be equivalent of having a number of entries in the lookup table equal to the product of the number of initiators (targets) with the number of memory blocks and roles. In case of T1 (I1), being the role directly encoded in the RAM, for a given combination of parameters the maximum number of lines in the lookup table is half than in the dual architecture, while the dimension of the RAM doubles.

Table 1 shows area and energy overhead due to the proposed data protection mechanism, applied to all the memory elements of the system. We consider two different target architectures:

- *Arch1* considers 2 initiators and 8 target-memories each one partitioned into 4 memory blocks;
- *Arch2* considers 8 initiators and 1 target-memory partitioned into 16 memory blocks.

In the column *Area* of Table 1, we show the sum of the area of all the DPUs distributed on the different NIs, while column *Energy* represents the cost for access to the DPU for each memory request. As the table shows, we obtain always a smaller area in the I1 architecture, due to the relevant influence in the dimension of T1 given by the look up of also the *SourceID* bits. In particular, T1 results to be in both *Arch1* and *Arch2* about 21% bigger than I1. However, if we consider the energy aspects of the architectures, we can observe that adopting the T1 architecture in *Arch1* allows saving about 70% of the energy consumed in the case of considering the I1 architecture. This result is obtained because the DPUs implemented at the targets NI in T1 are smaller than those implemented at the initiators NI in I1 and consequently the energy to perform the look up is smaller. When considering *Arch2*, the opposite holds, being the DPU implemented at the target in T1 sensibly bigger than those at the initiators in I1. Compared to common NI implementations [13], the overhead associated with the DPU is quite significant and represents one of the main drawback of this implementation. In fact, considering for instance a T1 configuration with 16 entry lines for the lookup table, the area occupied by the DPU is about the 21.5% of the total area.

It is important to note that exists the possibility to have systems where not all the targets need data protection. In these architectures, if we adopt a T1 configuration for the target to protect we do not need to pay any energy overhead for the accesses to non protected memories, since we do not have any access to the DPU. On the other hand, if we adopt the I1 configuration, we have to pay the access to the DPU for each network transaction. This is due to the fact that the protocol translation in the NI initiator and the DPU access is done in parallel to hide the latency overhead.

6. CONCLUSIONS AND FUTURE WORKS

In this paper we presented a novel solution for data protection in Multiprocessor System-on-Chip architectures based on NoC. We proposed a hardware module coupled with the network interface that gives the possibility to perform secure accesses to memories and memory-mapped peripherals. The paper also presented four different implementations of the Data Protection Unit, showing advantages and disadvantages of each solution with respect to the system architecture.

A deeper interaction with the Operating System is the main future work of the paper. In fact, our objective is the possibility to secure the memory access not only referring to the initiator but, with finest granularity, to the threads running on the processor. This could enable the possibility to perform the migration of the threads across the network together with their associated access rights.

7. ACKNOWLEDGMENTS

This work has been carried out under the MEDEA+ LoMoSA+ Project and is partially funded by KTI - The Swiss Innovation Promotion Agency - Project Nr. 7945.1 NMPP-NM.

8. REFERENCES

[1] *Open Core Protocol Specification 2.2.*

[2] *SonicsMX SMART Interconnect Datasheet.* http://www.sonicsinc.com.

[3] T. Alves and D. Felton. *TrustZone: Integrated Hardware and Software Security, White Paper.* ARM, 2004.

[4] L. Benini and G. De Micheli. Networks on Chips: A New SoC Paradigm. *IEEE Computer*, 2002.

[5] E. Chien and P. Szoe. *Blended Attacks Exploits, Vulnerabilities and Buffer Overflow Techniques in Computer Viruses.* Symantec White Paper, Sept. 2002.

[6] J. Coburn, S. Ravi, A. Raghunathan, and S. Chakradhar. SECA: Security-Enhanced Communication Architecture. In *Proceedings of the 2005 International Conference on CASES*, 2005.

[7] W. J. Dally and B. Towles. Route packets, not wires: on-chip inteconnection networks. In *Proceedings of DAC '01*, pages 684–689, New York, NY, USA, 2001. ACM Press.

[8] J. P. Diguet, S. Evain, R. Vaslin, G. Gogniat, and E. Juin. NoC-centric security of reconfigurable soc. In *Proceedings of the First International Symposium on Networks-on-Chip (NOCS'07)*, May 7-9 2007.

[9] C. H. Gebotys and R. J. Gebotys. A framework for security on NoC technologies. In *Proceedings of the Annual Symposium on VLSI*, pages 113–117. IEEE Computer Society, Feb. 20-21 2003.

[10] C. H. Gebotys and Y. Zhang. Security wrappers and power analysis for SoC technology. In *First IEEE/ACM/IFIP International Conference on CODES+ISSS*, pages 162–167, Oct. 1-3 2003.

[11] P. Kocher, R. Lee, G. McGraw, A. Raghunathan, and S. Ravi. Security as a New Dimension in Embedded System Design. In *Proceedings of DAC 2004*, Jun. 7-11 2004.

[12] K. Pagiantzis and A. Sheikholeslami. Content-Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey. *IEEE Journal of Solid-State Circuits*, 41(3), March 2006.

[13] A. Radulescu, J. Dielissen, S. G. Pestana, O. Gangwal, E. Rijpkema, P. Wielage, and K. Goossens. An Efficient On-Chip NI Offering Guaranteed Services, Shared-Memory Abstraction, and Flexible Network Configuration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(1), January 2005.