NoC-MPU: a secure architecture for flexible co-hosting on shared memory MPSoCs

Joël Porquet and Alain Greiner
LIP6 Laboratory
Université Pierre et Marie Curie
joel.porquet@lip6.fr

Christian Schwarz
Nagravision
christian.schwarz@nagra.com

***Abstract*—For many embedded systems, data protection is becoming a major issue. On those systems, processors are often heterogeneous and prevent from deploying a common, trusted hypervisor on all of them. Multiple native software stacks are thus bound to share the resources without protection between them. NoC-MPU is a Memory Protection Unit allowing to support the secure and flexible co-hosting of multiple native software stacks running in multiple protection domains, on any shared memory MP-SoC using a NoC. This paper presents a complete hardware architecture of this NoC-MPU mechanism, along with a software trusted model organization.**

ACKNOWLEDGMENTS

This work has been carried out in the framework of a cooperation between UPMC/LIP6 and STMicroelectronics.

I. INTRODUCTION

For many embedded systems, especially when handling protected content, data protection is a major issue. Only a few years ago, *bi-partitioning* techniques were enough to comply with security requirements. ARM, for example, proposed the TrustZone feature, allowing a "secure world" to live alongside a "non-secure world" [1].

Nowadays, this binary *co-hosting* is no more sufficient to answer the new security requirements. Recent consumption models promote, for example, the convergence of traditional video and Internet-based content [2]. This leads to a significant increase of the security complexity.

Devices such as set-top boxes now provide a multitude of services: User Interface, Conditional Access, Digital Right Management, Personal Video Recorder, etc. Each service finds its physical representation in a mixture of hardware and software components, ranging from small security-critical software stacks running on basic processors or accelerators, up to commodity OSes on complex application processors. Although these highly heterogeneous software stacks each contains internal sensitive information that must remain protected, all of them are bound to collaborate.

A. Co-hosting and Virtualization

New techniques are emerging to co-host multiple native software stacks in parallel and to transparently partition available platform resources among them. First intended for mainframe servers [3] and later for desktop systems [4]–[7], *Virtualization* is now reaching embedded systems. The pure software approaches [8], [9], relying on the use of a Memory Management Unit (MMU) to perform the isolation between native software stacks, are soon expected to be superseded by dedicated virtualization hardware support within embedded processors. However, in both cases, isolation is always achieved at processor level.

This processor-centric property makes the sharing of processing and memory resources efficient on symmetric multiprocessor architectures that can be controlled by a common trusted basis (e.g. a virtualization *hypervisor*). However, this approach is not convenient for heterogeneous multiprocessor shared-memory embedded systems-on-chip (MP-SoCs).

For instance, multimedia-oriented MP-SoCs are often composed of a small number of general purpose processors, assisted by various specialized programmable processors as well as dedicated coprocessors frequently having a DMA (Direct Memory Access) capability. The latter two are unable to support virtualization since they are mostly MMU-less, and sometimes do not even offer several privilege levels. Establishing a common trusted basis on such heterogeneous platforms is thus difficult, as long as, for example, a software stack running on a specialized processor is able to access the whole address space, bypassing completely any virtualization layer.

B. Software stacks and protection domains

As described in [10], this MP-SoC heterogeneity must be addressed by a platform-global protection mechanism, covering the full communication infrastructure, instead of a processor-centric mechanism. We define a *protection domain* as a set of specific access rights to the shared address space. We assume that each software stack is running in a specific protection domain, with rights-overlaps possible between them. Figure 1 illustrates the ideal co-hosting of several protection domains where software stacks transparently share all the platform's resources: memory, processing and peripheral resources.

As a given initiator device (e.g. processor or coprocessor) can be working for various software stacks, a global protection mechanism requires that all transactions released by an initiator device are tagged by an identifier defining which software

978-3-9810801-7-9/DATE11/©2011 EDAA

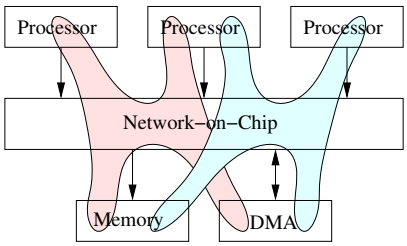

Fig. 1. Co-hosting of protection domains sharing processing and memory resources as well as a DMA device

stack is issuing the transaction. Incidentally, this information is already supported by standard communication protocols: VCI [11] and OCP [12] respectively define the `TRDID` field and `MThreadID` field. While they have been specified to support – out-of-order – simultaneous transactions issued by multi-threaded processors, they can be used to tag the transactions with a "protection domain identifier".

In this perspective, we call *multi-protection-domain* an initiator device that has the capability to tag the transactions with a protection domain identifier, and we assume then that all initiator devices in the platform have this "multi-protection-domain" capability.

[10] introduced the concept of a global architecture mechanism supporting the secure and flexible co-hosting of multiple software stacks running in multiple protection domains. This paper proposes a possible hardware/software implementation of this mechanism called NoC-MPU (Network on Chip with Memory Protection Unit) that can be used in any shared address space MP-SoC using a NoC.

This paper is organized as follows: section II is an overview of the previously proposed NoC-based protection solutions; section III presents the proposed NoC-MPU architecture; finally, we conclude and look at future work in section IV.

II. Related work

While general NoC technology has been widely studied during the last decade, security aspects of NoC-based architectures have been addressed only recently.

Diguet et al. [13] may have been the first to tackle the idea of directly integrating a security mechanism with the NoC. The proposed mechanism checks the transactions between physical devices, authorizing them only if they comply to a predefined configuration (e.g. a given processor is allowed to access a given memory bank). As a result, the granularity of the access control is the hardware device. This static assignment does not support different access rights to memory areas within the same physical memory bank. Besides, this filtering method is also incompatible with co-hosting, where several protection domains are able to share the same initiator devices.

Fiorin et al. [14]–[16] have proposed a solution for data protection in MP-SoC architectures based on NoC. Their secure NoC architecture is composed of a set of Data Protection Units (DPUs) implemented within the Network Interface Controllers. The DPU can check and limit the access rights of initiator devices accessing various regions in a shared address space (e.g. a given processor is allowed to access a certain address region), with no performance overhead. The access rights are defined for each physical device, except for processors where the DPU can actually distinguish between the two operating roles (kernel/user). However, this is not sufficient to support several protection domains sharing the same processing resources. Finally, the DPU implements a segmentation approach without any caching mechanism: the access rights tables are directly embedded within DPUs. It seems difficult for this strategy to support flexible co-hosting of multiple protection domains potentially requiring access to multiple memory regions, since the access rights tables would be too large.

III. The NoC-MPU approach

In this section, we describe the hardware/software architecture of the proposed Memory Protection Unit (MPU) for NoC-based architectures, that is the key element to support a flexible co-hosting of several protection domains on a multiprocessor platform.

For the remainder, we adopt the term *compartment*, as introduced in [10], to refer to a software stack associated to a protection domain. The *compartment identifier* (CID) allows to associate any transaction issued by a multi-protection-domain device to the corresponding compartment.

A. Principle of operation

Protecting compartments from one another means actually protecting their code, data, exclusive uses of peripheral devices, etc. against misuses. In a shared address space architecture, these assets are typically address regions accessible through read or write transactions in this shared address space. Therefore, the access control must define, for each compartment, a set of access rights on various address regions. When a compartment is executing on a processor, the CID information is attached to each transaction. Isolation is achieved in the NoC by checking all the transactions against the set of access rights defined for this CID. As a result, the granularity of the access control becomes the compartment, a flexible, logical construct, instead of a physical device.

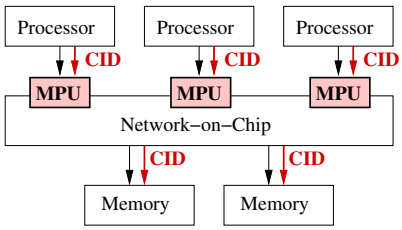

Fig. 2. Example of a NoC-based architecture equipped with MPU modules

As illustrated in Figure 2, the MPU is a hardware module embedded in the NoC, supplying services similar to those offered by a "firewall" in local area networks.

The filtering algorithm performed by the MPU is defined by a permission table, indexed by the memory address of a

transaction request and its associated CID, and containing the access rights for each combination of these two entries. To provide for a large number of compartments, sharing the same address space with different access rights, this table cannot be stored within the MPU (contrary to the DPU mechanism [14]) but in the memory. On the other hand, the permission table must be stored in on-chip memory, for both performance and security reasons. Similarly to a standard MMU, the MPU module contains a small cache (called *permission lookaside buffer*, or PLB) and a dedicated Finite State Machine (FSM) to fetch the required entry from the permission table in case of a PLB miss. As shown in Figure 3, each new transaction triggers a PLB look-up. If the PLB contains the permission information for the tuple (address,CID) the transaction is granted or denied accordingly. Otherwise, the hardware FSM accesses the permission table in memory, located at the address hold by the *permission table base*, and refills the PLB with the missing information.

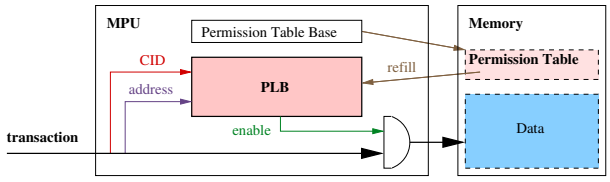

Fig. 3. Abstract architecture of the MPU module

[17] discussed alternative layouts for the permission tables. The two basic approaches are segmentation and pagination. In the segmented approach, the permission table is a linear array of segments ordered by segment base address. Although this method supports variable segment sizes, the lookup time can turn to be tremendous, as well as the update, whenever the number of segments is too high. We thus preferred the paged approach, where the address space is decomposed in fixed size pages, and the access rights are defined for each page. The worst-case lookup is deterministic and the management is quite easy.

The location of the MPU modules in the platform is relevant. The major advantage of being located at the initiators side (as in Figure 2) is that these input modules are able to prevent the Denial-of-Service attack where an initiator device tries to saturate the NoC by performing massive unauthorized accesses.

B. Hardware architecture

The proposed MPU module has been implemented in the DSPIN NoC [18], a packet-switching micro-network dedicated to shared memory multiprocessor architectures. Nevertheless, it could be embedded in any existing NoC supporting read/write transactions in a shared address space.

As pictured in Figure 4, the architecture has a flat mesh topology: one single device (initiator or target) is connected to each NoC router, via a *Network Interface Controller* (NIC). The NIC is in charge of adapting the packet-based DSPIN protocol to the device protocol (VCI/OCP in our implementation). Two types of NICs exist, depending on whether a NIC connects an initiator device (NIC/I) or a target device (NIC/T).

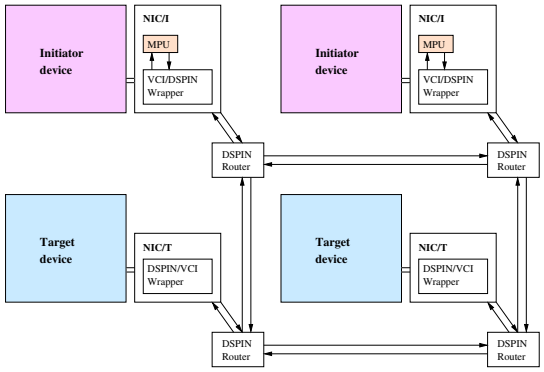

Fig. 4. DSPIN-based architecture with MPUs embedded in NICs

The MPUs are embedded inside NICs/I, and works in parallel with the protocol conversion. The permission table is implemented as a hierarchical, two levels, page table. There is one table per compartment. The page size is 4 KiB (12 bits), and the 20 MSB address bits are used to index the page table (10 bits for each level).

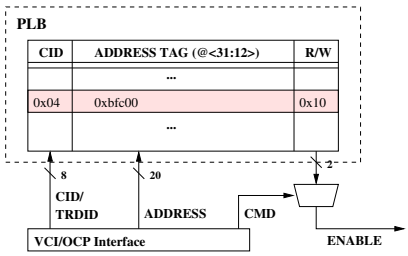

Fig. 5. Internal architecture of the PLB

As shown in Figure 5, we use three VCI/OCP signals to access the PLB and check the access rights: the transaction is defined by both the compartment identifier (*CID*, contained in the *VCI/TRDID* field), and the 20 most significant bits of the *VCI/ADDRESS* field. Both these information are looked up and access rights for read (*R*) and write (*W*) operations are supplied as output of the PLB. The *VCI/CMD* finally allows to check the transaction and to let it pass through the network or not.

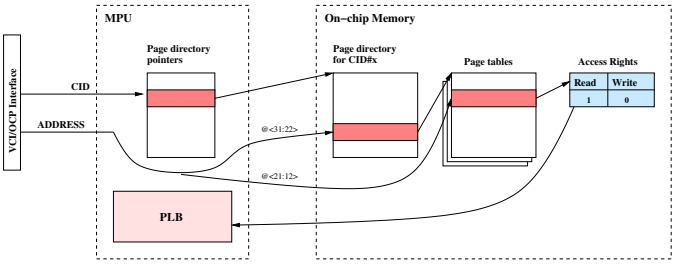

Fig. 6. Hardware page walk mechanism

Whenever the PLB does not contain the access rights

corresponding to the current transaction, the hardware page walk FSM supersedes the protocol conversion to retrieve the information in memory (see Figure 6). A buffer of page directory pointers is indexed by the *CID* field to get the base address of the page directory associated to the compartment which issued the request. The 10 MSB of the 32 bits address are used as an offset to find, in the first level page directory, the base address of the second level page table. The second level page table is then accessed, using the next 10 address bits as an offset to find the access rights information (2 bits). This information is retrieved within the MPU, and added to the PLB thus able to be consulted again for checking the access rights of the current transaction.

The MPU performs these two memory accesses by reusing the same physical identity as the initiator device which issued the transaction (i.e. the device attached to the given NIC/I). The MPU then intercepts both responses when they come back via the NIC.

Since protocol conversion and MPU access are performed in parallel, embedding the MPU within the NIC ensures that no additional latency is provoked for PLB hits. A miss in the PLB, however, causes the transaction to be temporarily frozen while the PLB is refilled, which can increase the average latency.

C. Software architecture

The NoC-MPU protection mechanism nicely fits in the layered trust model proposed in the multi-compartment approach [10]. Multi-protection-domain programmable processors are each locally managed by a software *Local Trusted Agent* (LTA). A LTA is in charge of updating a range of CID values for both, itself and the compartments (e.g. applications) running above it. A LTA is considered a compartment with additional privileges. This local management is complemented by a *Global Trusted Agent* (GTA), acting platform-wide. The GTA handles the creation/destruction of compartments and supervises the shared address space partitioning among them, particularly through "on-the-fly" definition of page tables. It also configures the different MPUs, that is the initialization of directory page pointers, by the memory-mapped interface each MPU presents. The GTA typically receives and handles violation events (in case of unauthorized accesses), which can be signaled by software interrupts.

This protection scheme allows to support complex software architectures. For example, a standard hypervisor controlling several virtual machines on a general purpose processor and a real-time OS running user applications on a specialized processor can both act as LTAs. Both can schedule their user software stacks according to their policy and update respective CID values; a software stack running on a dedicated processor can act as the GTA to administrate the whole platform.

IV. Conclusion

In this paper we presented a novel, flexible and adaptive solution for secure co-hosting of several protection domains (compartments) running concurrently on a shared memory multiprocessor System-on-Chip using a NoC. We described the hardware module embedded in the Network Interface Controllers that allows a flexible partitioning of the shared memory address space among multiple compartments. We also described the trusted software model which fits nicely with this hardware protection scheme.

This complete hardware/software secure architecture is currently being implemented. First results concerning the performance overhead (i.e. due to PBL misses of the hardware protection modules) are encouraging, as well as the silicon area overhead of the hardware modules over a NoC infrastructure. The next logical step is to thoroughly characterize this performance overhead, evaluating typical execution scenarios. This involves benchmarking applications on multiple compartments and measuring the silicon overhead after synthesis of the hardware modules.

References

[1] T. Alves and D. Felton, "ARM TrustZone: Integrated Hardware and Software Security," July 2004. [Online]. Available: http://www.arm.com

[2] Cloakware Inc., "Security Impacts of Next-Generation Set-Top Boxes," White paper, 2009. [Online]. Available: www.csimagazine.com/pdf/Cloakware-STB.pdf

[3] R. J. Creasy, "The Origin of the VM/370 Time-Sharing System," *IBM Journal of Research and Development*, vol. 25, no. 5, pp. 483–490, 1981.

[4] I. Pratt, K. Fraser, S. Hand, C. Limpach, A. Warfield, D. Magenheimer, J. Nakajima, and A. Mallick, "Xen 3.0 and the Art of Virtualization," in *Proceedings of Linux Symposium 2005*, July 2005.

[5] VMWare. [Online]. Available: http://www.vmware.com

[6] R. Uhlig, G. Neiger, D. Rodgers, A. L. Santoni, F. C. M. Martins, A. V. Anderson, S. M. Bennett, A. Kagi, F. H. Leung, and L. Smith, "Intel Virtualization Technology," *Computer*, vol. 38, no. 5, pp. 48–56, 2005.

[7] Advanced Micro Devices, Inc., *AMD64 Virtualization Codenamed "Pacifica" Technology: Secure Virtual Machine Architecture Reference Manual*, May 2005.

[8] O. K. Labs. [Online]. Available: http://www.ok-labs.com

[9] VirtualLogix. [Online]. Available: http://www.virtuallogix.com

[10] J. Porquet, C. Schwarz, and A. Greiner, "Multi-compartment: a new architecture for secure co-hosting on SoC," in *SOC'09: Proceedings of the 11th international conference on System-on-chip*. Piscataway, NJ, USA: IEEE Press, 2009, pp. 124–127.

[11] VSI Alliance, *Virtual Component Interface Standard*, April 2001, version 2, OCB 2 2.0.

[12] OCP-IP Alliance, *Open Core Protocol Specification*, 2005, release 2.1.

[13] J.-P. Diguet, S. Evain, R. Vaslin, G. Gogniat, and E. Juin, "NOC-centric Security of Reconfigurable SoC," in *NOCS '07: Proceedings of the First International Symposium on Networks-on-Chip*, 2007, pp. 223–232.

[14] L. Fiorin, G. Palermo, S. Lukovic, and C. Silvano, "A data protection unit for NoC-based architectures," in *CODES+ISSS '07: Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*. New York, NY, USA: ACM, 2007, pp. 167–172.

[15] L. Fiorin, G. Palermo, and C. Silvano, "A security monitoring service for NoCs," in *CODES+ISSS '08: Proceedings of the 6th IEEE/ACM/IFIP international conference on Hardware/Software codesign and system synthesis*. New York, NY, USA: ACM, 2008, pp. 197–202.

[16] L. Fiorin, G. Palermo, S. Lukovic, V. Catalano, and C. Silvano, "Secure Memory Accesses on Networks-on-Chip," *IEEE Trans. Comput.*, vol. 57, no. 9, pp. 1216–1229, 2008.

[17] E. Witchel, J. Cates, and K. Asanović, "Mondrian Memory Protection," in *ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, 2002, pp. 304–316.

[18] I. M. Panades, A. Greiner, and A. Sheibanyrad, "A Low Cost Network-on-Chip with Guaranteed Service Well Suited to the GALS Approach," 2006, First International Conference on Nano-Networks (Nano-Net).