Secure Communication with Peripherals in NoC-based Many-cores

Rafael Follmann Faccenda*, Gustavo Comarú*, Luciano Lores Caimi†, Fernando Gehm Moraes*
*School of Technology, Pontifical Catholic University of Rio Grande do Sul – PUCRS – Porto Alegre, Brazil
{rafael.faccenda, gustavo.comaru}@edu.pucrs.br, fernando.moraes@pucrs.br
†UFFS, Federal University of Fronteira Sul, Chapecó, Brazil – lcaimi@uffs.edu.br

***Abstract*—Many-core systems-on-chip (MCSoCs) contain processing elements (PEs), peripherals attached to the system, and an NoC connecting them. These systems have different flows traversing the NoC: PE-PE and PE-peripheral flows. Malicious hardware or software can hinder system security due to the resource sharing feature, such as CPU sharing for multitasking or sharing NoC links for flows belonging to different applications. Methods that isolate applications with security constraints, such as Secure Zones (SZs), protect PE-PE flows against most of the attacks reported in the literature. Proposals with methods to secure the communication with peripherals in the literature are scarce, with most of them focusing on shared memory protection. This paper presents an original approach, Secure Mapping with Access Point - SeMAP, which creates mapping policies for SZs, and communication strategies with IO devices, to protect PE-peripheral flows. Results show that the application execution time is not penalized by applying SeMAP, presenting advantages compared to a state-of-the-art approach. In terms of security, SeMAP successfully resisted an attack campaign, blocking malicious packets attempting to enter the SZ.**

***Index Terms*—Security, NoC-based Many-cores, Secure Zones, Peripherals.**

I. Introduction

Many-core systems on chip (MCSoCs) provide high computing performance due to the parallelism offered by the numerous resources inside the chip. Current applications have increasing demands on dedicated resources, such as shared memories, hardware accelerators (e.g., neural engines), and communication interfaces [1]. Thus, MCSoCs should contain, besides the set of processing elements (PEs), support for peripherals leading to the adoption of *heterogeneous architectures* instead of homogeneous ones (e.g., MPSoCs). Figure 1 presents a 4x4 MCSoC instance, with four peripherals attached to the NoC borders.

A consequence of the increasing number of features and functionalities in MCSoCs is the adoption of third-party IPs (3PIPs) to meet time-to-market constraints and reduce design costs. Such IPs come from different vendors, raising the risk of having malicious hardware and/or software inserted in the design [2]. Thus, *security* is a major design constraint.

Malicious hardware/software can hinder system security due to the resource sharing feature, such as CPU sharing for multitasking or sharing NoC links among flows belonging to different applications. Thus, methods that isolate applications with security constraints (App_{sec}) [3, 4] protect applications

978-1-6654-8128-1/22/$31.00 ©2022 IEEE

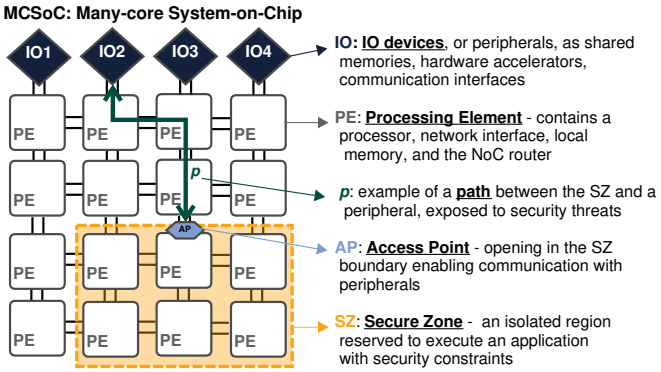


Fig. 1. MCSoC and terminology adopted in this work.

against most of the attacks reported in the literature. Secure Zones (SZ, in Figure 1) [5, 6] is an example of a defense mechanism based on spatial isolation. SZs reserve PEs and links to execute an App_{sec} without sharing the resources inside the SZ with other applications.

Literature related to many-core that present methods to secure the communication with peripheral are scarce, with most of them focusing on shared memory protection [7, 8]. On the other hand, several works present MCSoCs with peripherals but without security concerns [9, 10]. Therefore, there is a gap to fulfill: *how to protect the communication of applications with peripherals*?

A possible solution is to encrypt data in the communication path between the SZ and peripherals. However, cryptography is a partial solution, only ensuring confidentiality. Other security threats may compromise the communication path SZ↔IO (***p*** in Figure 1):

1) Denial of service (DOS) and side-channel attacks in the SZ↔IO path;
2) Unauthorized access to the IO. IO devices must be aware of applications with access rights to avoid attacks by malicious entities;
3) Unauthorized access to the App_{sec} running into the SZ. Communication with IO devices requires opening access points (AP, in Figure 1) at the SZ border. The AP is a vulnerability that malicious applications can explore to access the App_{sec};
4) DOS due to the lack of paths to the IO devices. A given application or SZ may isolate IO devices. Thus, reachability is a design concern, ensuring a path between SZs↔IOs.

The *goal* of this work is to define SZ mapping policies and communication strategies with IO devices, addressing issues 3 (APs) and 4 above (ensure reachability to IO devices).

The *original contribution* is the Secure Mapping with Access Point ($SeMAP$) approach, which enables the mapping of multiple SZs simultaneously, protecting App_{sec}s against unauthorized accesses, ensuring the availability of paths to the IO devices.

This paper is organized as follows. Section II presents the related work regarding peripherals in many-cores. Section III discusses the threat model assumed in this work. Section IV presents two methods to protect the communication with peripherals, DSZ and $SeMAP$, being $SeMAP$ the original contribution of this paper. Section V evaluates both methods in terms of performance and security. Section VI presents the conclusions and directions for future work.

II. Related Work

Recent works present many-cores with peripherals attached to them, addressing communication performance improvement [9, 10] or timing predictability [11, 12, 13].

Lee et al. [10] propose a message-based system calls to enhance the performance of storage IO for the MapReduce application model in many-cores. In addition, the Authors explore the intracluster locality of task allocation in the cores. As a result, the execution time of MapReduce was reduced by 29%.

Jiang et al. [11] optimize IO operations in safety-critical systems with Virtual Machines. The proposal is a hardware hypervisor to shorten the overhead of the IO communication in an application inside a VM, called *IO-GUARD*. The main objective is to enhance the IO path and resource management.

Vaas et al. [12] also focus on safety-critical systems, proposing the LOW_{IO}, an interface that reduces the interference of low-level non-deterministic IO operations on critical tasks. Hardware units control the access to peripherals, giving priority to critical transactions.

Zhao et al. [13] propose dedicated IO co-processing units and a scheduling model to provide predictability for hard real-time systems. A module named IO Processing Unit (IOPU) controls the IO tasks. The objective is to make the communications predictable.

In terms of security, Ehret et al. [14] focus on securing edge devices against attacks on their IO ports. This approach considers that the devices are installed away, making it possible for a malicious user to access the system from its ports manually. Even though this work does not consider a many-core, it is possible to apply the adopted threat model to a many-core.

Grammatikakis et al. [7] propose an NoC firewall to protect the access of a shared memory accessed through the NoC, avoiding sensitive data corruption or access of an unauthorized element. The firewall isolates the NoC, only allowing an authorized process of the MCSoC to access the memory.

Reinbrecht et. al. [8] also focus on the security of MCSoCs with shared memories. The authors propose two new attack types targeting shared memories, called Prime+Probe Arrow and Prime+Probe Firework, that can affect systems with Secure Zones when the application running inside it needs to use the shared memory. The Authors propose the *Gossip-NoC*, which includes a traffic monitor. When an anomalous behavior is detected, the monitor sends an alert message to a system manager, which changes the routing algorithm from XY to YX, avoiding malicious traffic.

In summary, proposals [9, 10, 11, 12, 13, 15] optimize the communication performance with peripherals, or systems with real-time constraints. On the other hand, the concern of [7, 8] is to protect access to shared memory. General security methods to protect access to peripherals other than shared memories are a gap in the literature. Actual many-cores have a rich set of accelerators besides shared memories, requiring the availability of security mechanisms to protect the communication.

III. Threat Model

Resource sharing in PEs and NoC links introduces vulnerabilities to the applications. Considering the reference MPSoC architecture (Figure 1), the attack surface includes the access point (*AP*) and the exposed path (*p*). Malicious entities (tasks or peripherals) may explore this surface in attacks such as:

i Spoofing: falsification of identity – a malicious entity could try to pass through the AP, pretending to be a trustworthy peripheral;
ii DoS (flooding): a malicious entity could attempt to flood the SZ by injecting packets through the AP;
iii DoS (blocking): Hardware Trojans (HTs) may block, drop, or misroute flows to/from the peripherals;
iv Snooping: once the packet leaves the SZ, it is exposed to malicious entities, being vulnerable to snooping attacks.
v SCA: a malicious entity could monitor the exposed flows to execute, e.g., timing attacks [8].

Our proposal addresses threats (i) and (ii) using a key shared by the App_{sec} and the IO. This key ensures that a given packet can only enter or leave the SZ if it has the correct key. The communication protocol mitigates the threat (iii), which can detect when an expected answer to a transaction does not reach the SZ. Encrypting data in the exposed path mitigates snooping attacks. We assume in this work that packets in the exposed path are encrypted to avoid such attacks. SCA mitigation is out of the scope of the current work. Key exchange between the peripheral Network Interface (NI) and the SZ is discussed in [16].

The focus of this work is the proposal of methods to enable secure communication of applications executing in isolated resources (Secure Zone) with peripherals (outside the Secure Zone), *not* on proposing countermeasures. Examples of countermeasures include the dynamic changing of the AP location when detecting an attack or using methods to locate the attack source [17].

IV. Safe Communication with Peripherals

This work adopts the opaque SZ model [18]. Once the SZ is closed, all traffic trying to cross it is re-routed. The opaque SZ method prevents the attacks described in the threat

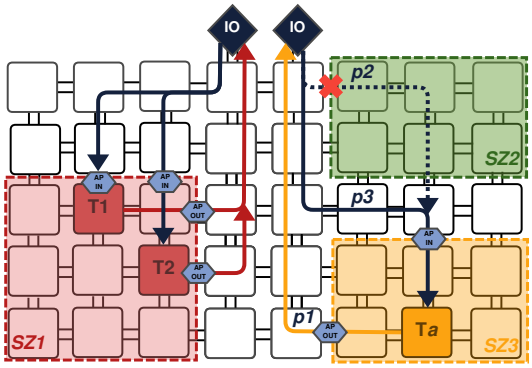


Fig. 2. Example of Dynamic SZ method, adapted from [18]

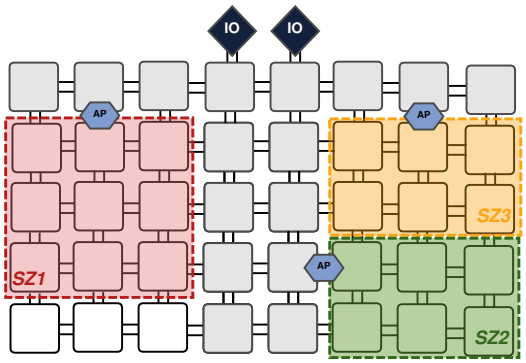


Fig. 3. Example of gray and secure areas. Three App_{sec}s mapped on the secure area, each one with an Access Point (AP).

model, including timing and logical SCA [19]. However, to allow communication between peripherals and App_{sec}, it is necessary to open the SZ boundary. The controlled opening at the boundary of the SZ is called *access point* (AP).

Opening the SZ does not violate the fundamental rule governing the SZ method: flows belonging to applications other than App_{sec} must not cross the SZ. Opening the SZ to peripherals requires a set of rules to ensure the security of App_{sec}. Work [18] presents a set of rules to meet security constraints:

1) Differentiate PE↔PE from PE↔IO communication. This differentiation prevents malicious applications from trying to inject packets into SZs.
2) Master-slave communication. PEs inside the SZ initiate all transactions with peripherals. Any unexpected packet arriving in an AP is discarded.
3) Add a key in IO packets, ensuring their authenticity and source.
4) Avoid unreachable resources, i.e., an SZ may not block the access to peripherals.

We present below two methods for communicating with peripherals: (i) *Dynamic SZ*, initially presented in [18]; (ii) Secure Mapping with Access Point ($SeMAP$), herein proposed.

A. Dynamic SZ – DSZ

The DSZ method is flexible in terms of App_{sec} mapping. SZs can be mapped at any region of the MCSoC, respecting the restriction of not blocking paths to peripherals. Figure 2 illustrates a system with 3 SZs.

APs are established for each communication transaction, using XY routing by default. The task that starts an IO communication opens two unidirectional APs (AP IN and AP OUT in Figure 2), transmitting two control packets to the SZ border. Each AP is closed when a packet traverses it. This method ensures that only one packet traverses the AP per transaction, minimizing attack attempts. On the other hand, if multiple tasks communicate with peripherals, several APs can be opened simultaneously ($SZ1$ in the Figure), increasing the attack surface. Packets to traverse the APs must meet two conditions: (i) be IO packets; (ii) match the key shared by the peripheral and the App_{sec}. The packet is discarded otherwise.

The mapping flexibility brings the *masking effect*. Consider Figure 2 having the $SZ1$ and $SZ3$ mapped and running in the system. When $SZ2$ enters in the system, it blocks path $p2$, from the IO device to $SZ3$. Thus it is necessary to compute a new path using source routing. The PE closest to the IO device computes the new path ($p3$), transmitting it to the IO device to be used in the subsequent data transmissions. This approach adds a security threat, as it involves a PE not related to App_{sec}, allowing it to know the location of the AP and use this information to initiate an attack.

Despite the DSZ application mapping flexibility, there is a restriction related to the task mapping inside the SZ, named *alignment effect*. The DSZ does not allow two or more tasks on the same X or Y coordinate to communicate with peripherals because the DSZ authorizes only one transaction per AP. If two tasks are aligned, both activate the same AP, but only one packet passes through it, thus blocking one of the tasks.

B. Secure Mapping with Access Point – SeMAP

Our proposal, named $SeMAP$, restricts the App_{sec} mapping and allows only one bidirectional AP per SZ. The goal is to have a single aperture for all the IO transactions. Our mechanism creates two logical regions, at system startup: *Gray* areas (GA) run applications without security requirements and guarantee a path between App_{sec} and peripherals, i.e., reachability. *Secure* areas only run App_{sec}s. Figure 3 illustrates an example of these two regions, with three App_{sec} mapped in the secure areas. The peripherals are attached to the North side of the system for the sake of simplicity. The approach does not restrict peripherals attached to a given system side. The mapping of App_{sec}s requires at least one side juxtaposed to a GA in such a way to have a path to the peripherals.

The process to deploy an App_{sec} into the secure area requires four steps, detailed below.

1) SZ Shape and Location: The definition of the SZ shape prioritizes shapes having the width of the secure area. This method improves system utilization, avoiding PEs without access to the gray areas. The SZ shape and coordinates selection follow a Sliding Search Window (SSW) algorithm. The starting point of the SSW is the row nearest to the peripherals. The result of this step is a set of PEs reserved to execute the App_{sec}.

2) AP Definition: Any port of any frontier router next to a GA may receive the AP. The default location is the top-left or top-right router according to the gray area position. The second case is the north port of the top-middle router if the SZ is near to the top of the GA. Figure 3 presents both cases: the AP of the $SZ1$ and $SZ3$ are in the middle-top position, while the $SZ2$ is the default case, with the AP at the top-left position. Since any port can become an AP, there is the possibility to change the AP location periodically or whenever suspicious behavior is detected.

3) Task Mapping in the Selected Shape: The system manager maps tasks that communicate with peripherals near the AP and the remaining tasks according to the hop number between communicating pairs. After the mapping execution, the SZ borders are "closed", isolating the SZ. Only packets to/from peripherals can cross the SZ through the AP.

4) Path configuration: PEs inside the SZ does not use the XY routing algorithm to reach the peripheral. The first communication of a given task with a peripheral fires a path configuration heuristic. First, the OS computes the path PE→AP, then the path AP→peripheral, according to the gray area shape. The path from the peripheral to the PE is generated using the opposite ports in the reverse order. Figure 4 illustrates an example of a path computation between "Task" and IO. There are two paths: path A, the orange arrow from Task→IO, and B, the blue arrow from IO→Task. The circles show each of the ports taken for each of the paths. After computing the path to the peripheral, the next step is to send the reverse path to the peripheral. The OS sends the path to the peripheral, which stores the path and uses it for every communication with that specific task.

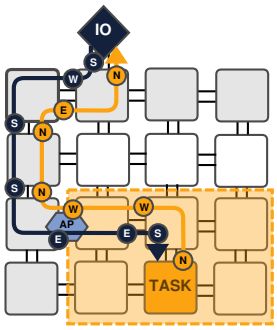


Fig. 4. Example of source routing paths from/to task to/from peripheral through the AP.

$SeMAP$ contains *security mechanisms* implemented in the APs. Packets to/from a peripheral only traverse the AP if the key embedded in the packet header matches the shared key. In addition to key verification, the AP monitors the frequency of incoming and outgoing packets, generating a suspicious alert if a threshold rate is reached. Such behavior may signalize an external peripheral (incoming packets) or App_{sec} is attempting to execute an attack.

V. Results

This section presents the performance of applications considering the two methods for communicating with peripherals. The second part of the results discusses the security aspects of the communication with peripherals. Section V-C compares both approaches.

Experiments use as baseline system the Memphis MCSoC [20], modeled at the RTL level (SystemC and VHDL). The MCSoC uses two NoCs: one for data and one for control. The data-NoC is a packet switching network with two physical channels, supporting XY (default) and source routing (when rerouting is necessary). The control-NoC is a broadcast NoC with single-flit packets and a search path mechanism, which allows path discovery for source routing [21].

A. Performance of the Communication with Peripherals

Figure 5 presents the applications mapping to evaluate the methods of communication with peripherals. The system receives the DTW (Dynamic Time Warping) application at startup. At *5 ms*, a new application enters the system (MPEG decoder). Note that the DTW DSZ (Figure 5(a)) has two paths broken by the MPEG, firing two path search computations (due to the "masking effect"). At *9 ms*, a PC (Producer-Consumer) is mapped, also blocking two DTW paths. A second scenario is evaluated, swapping the DTW with MPEG (the reference application is always mapped in the bottom-left corner of the system). In the first scenario (DTW in the left corner), the IO communication volume (number of messages exchanged with the peripherals) is higher.

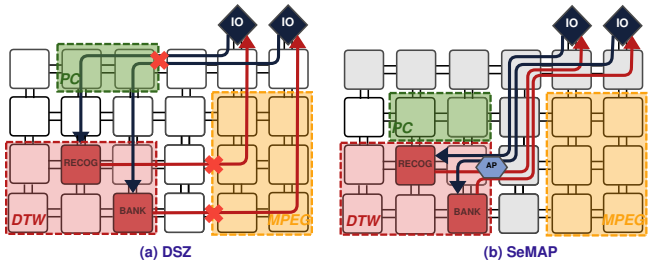
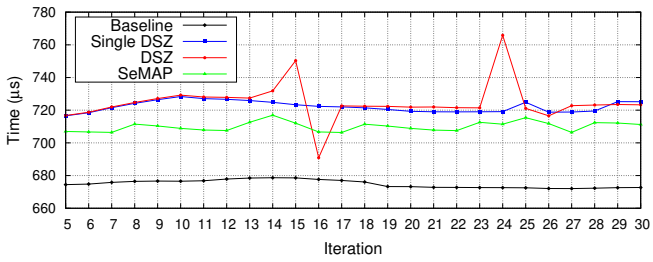


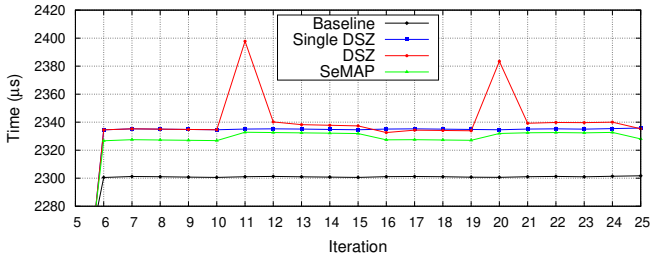
Fig. 5. Application mapping to evaluate the methods to communicate with peripherals.

Figure 6 presents the iteration latency for DTW and MPEG applications. The y-axis is the time required to execute each application iteration (in μs), and the x-axis is the iteration number. Graphs omit the first five iterations, considering these as the warm-up period. Each graph has 4 curves:

- **Baseline** (black line): execution of the applications without communication with peripherals. Input data is assumed to be stored in the local memories, and results are also stored in the local memories. Simulating the baseline MCSoC aims to evaluate the overhead due to the communication with peripherals.
- **Single DSZ** (blue line): only the reference application (DTW or MPEG) executes in the system. The goal of simulating the DSZ approach without other applications is to evaluate the DSZ method in the absence of the masking effect.
- **DSZ** approach (red line) using evaluation scenario presented on Figure 5(a).
- **SeMAP** approach (green line) using evaluation scenario presented on Figure 5(b).



(a) DTW iteration latency. Execution time (ms): 18.26 (baseline), 18.44 (single DSZ), 18.51 (DSZ), 18.37 ($SeMAP$).



(b) MPEG iteration latency. Execution time (ms): 13.82 (baseline), 13.91 (single DSZ), 13.94 (DSZ), 13.98 ($SeMAP$).

Fig. 6. Iteration latency using the baseline MCSoC, DSZ and $SeMAP$.

Figure 6 shows that:

- Comparing **SeMAP** and **Single DSZ** versus **Baseline**, the latency per iteration increases 1.2% (MPEG) to 7.3% (DTW) when there is IO communication (average values). The latency increases due to the: (i) non-minimum paths; (ii) master-slave communication protocol, i. e., all transactions started by the App_{sec}; (iii) management of APs in the DSZ method (opening and closing of APs at each transaction).
- Total execution time has a minimal overhead - 1.3% for DTW (**Baseline** versus **DSZ**) and 0.5% for MPEG (**Baseline** versus **SeMAP**).
- **SeMAP** reduces the latency per iteration (0.33% for MPEG and 2.7% for DTW, best cases) compared to **Single DSZ** because it does not need to manage APs. On the other hand, there is a latency per AP to start the application execution, as it is necessary to compute the paths for each AP (25μs@100MHz - average value per path).
- The masking effect, observed in **DSZ** (red curves), increases the iteration latency when a new application enters the system, blocking the PE→Peripheral communication, requiring to reroute the broken paths. In both scenarios, the masking effect only affects few iterations, since this mechanism is activated once, being the alternative path taken for the subsequent communications. The valley observed in the first scenario is due to the application pipeline behavior, i.e., while a task is blocked waiting for a new path, the other tasks continue to run. The performance degradation increases in scenarios with a larger number of broken paths,
- **SeMAP** is immune to the masking effect, presenting a small latency increase (0.2% to 1.8%) when a new application enters the system. The network traffic increases when a new application is admitted due to the transmission of the object code of the tasks. This increase in network traffic explains the slight increase observed in latency.

B. AP Security Evaluation

We executed an attack campaign with three different packet types arriving on APs: (i) application packets (PE-PE); (ii) IO packets with incorrect key; (iii) IO packets with a forged key. In scenarios i and ii, the APs correctly dropped the packets and notified the arrival of a suspicious packet to the system manager.

The master-slave protocol adds a random sequence number in the request packet (for read or write operations). The IO must answer with this number. Consider scenario iii, where the malicious peripheral forges the key and the AP address. In both methods, the packet reaches the PE, which notifies the system manager to isolate the malicious peripheral upon the reception of an unexpected packet or a packet with a wrong sequence number (it would be costly in terms of silicon area to have registers in the AP to store a list of malicious peripherals). Thus, it is necessary to meet four conditions to execute a successful attack (i) correctly forge the key; (ii) send the packet to the AP address; (iii) insert the packet into the SZ when a task is waiting for a peripheral answer; (iv) generate the correct sequence number. We consider that the fulfillment of these four conditions has a minimal probability of occurring, being sufficient to guarantee a secure communication between App_{secs} and peripherals.

C. Discussion

According to the threat model (Section III), both DZS and $SeMAP$ methods avoid spoofing and DoS (flooding) by detecting malicious packets arriving at an AP, blocking them, and notifying the address of the malicious entity (PE ou peripheral) to the system manager. The communication API avoids DoS (blocking) attacks when the communication started in the SZ does not receive an answer after a given period (watch-dog timer).

Table I compares the DSZ and $SeMAP$ methods qualitatively. The DSZ is recommended for scenarios with few SZs coexisting simultaneously (due to the masking effect) and a few tasks communicating with IO (due to the alignment effect). *The proposed SeMAP is a generic approach to map* $App_{sec}s$, without the restrictions observed in the DSZ method. Despite the advantages, $SeMAP$ has limitations related to the use of resources and possible congestion in the AP. It is possible to mitigate the first limitation with defragmentation techniques, and the second limitation would only occur in cases of very intense communication with peripherals.

VI. Conclusion

The Introduction raised the following question: *how to protect the communication of applications with peripherals*? The answer is to use the DSZ or $SeMAP$ methods, which secure the SZ↔IO communication against spoofing, DoS,

TABLE I
DSZ AND SEMAP QUALITATIVE COMPARISON.

	DSZ	SeMAP
PROS	Better resource utilization due to the mapping flexibility	Single bidirectional AP per App_{sec}, reducing the attack surface
	Communication with peripherals is not concentrated in a single AP (better traffic distribution)	The AP stays opened during the App_{sec} execution, avoiding configuration messages per IO transaction, reducing its management cost
		No alignment effect
		No masking effect
		The nearest PE to the peripheral does not need to be interrupted to compute a path to the SZ and is unaware of the AP position
CONS	Masking effect (Section IV-A)	Smaller resource utilization than DSZ due to the partition of the system into *gray* and *secure* areas
	Alignment effect (Section IV-A)	Fragmentation of the secure areas at runtime. It is possible to defragment the system using task migration
	Several APs opened in SZ simultaneously, increasing the attack surface	Peripheral traffic concentrated in a single AP may lead to NoC congestion

and snooping attacks. DSZ is flexible in mapping the SZ at any place of the MCSoC but presents limitations related to the SZ↔IO paths and APs (larger attack surface). $SeMAP$ adopts a restrictive mapping (secure and gray areas) and one bidirectional AP per SZ. Results show that the iteration latency increases up to 7.3% when communicating with peripherals, but the overhead is minimal considering the total execution time (worst-case 1.3%).

$SeMAP$ is the architecture to adopt due to the absence of the limitations compromising the DSZ method.

Future work includes: (i) define a Secure NI to be inserted between the MCSoC and an IO device; (ii) add a reservation protocol in the IO communication API to avoid a peripheral answering to a malicious request; (iii) study defragmentation techniques to be deployed at runtime.

ACKNOWLEDGMENT

This work was financed in part by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – CAPES (Finance Code 001), and CNPq (grant 309605/2020-2).

REFERENCES

[1] A. Kamaleldin and D. Göhringer, "AGILER: An Adaptive Heterogeneous Tile-Based Many-Core Architecture for RISC-V Processors," *IEEE Access*, vol. 10, pp. 43 895–43 913, 2022.
[2] H. Li, Q. Liu, and J. Zhang, "A survey of hardware Trojan threat and defense," *Integration, the VLSI Journal*, vol. 55, pp. 426–437, 2016.
[3] L. Caimi, R. Faccenda, and F. G. Moraes, "A Survey on Security Mechanisms for NoC-based Many-Core SoCs," *Journal of Integrated Circuits and Systems*, vol. 16, no. 2, pp. 1–15, 2021.
[4] S. P. Azad, G. Jervan, M. Tempelmeier, and J. Sepúlveda, "CAESAR-MPSoC: Dynamic and Efficient MPSoC Security Zones," in *ISVLSI*, 2019, pp. 477–482.
[5] S. Pinto, P. Machado, D. Oliveira, D. Cerdeira, and T. Gomes, "Self-secured devices: High Performance and Secure I/O Access in TrustZone-based Systems," *J. Syst. Archit*, vol. 119, p. 102238, 2021.
[6] E. M. Benhani, C. M. López, and L. Bossuet, "Secure Internal Communication of a Trustzone-Enabled Heterogeneous SoC Lightweight Encryption," in *FPT*, 2019, pp. 239–242.
[7] M. D. Grammatikakis, P. Petrakis, A. Papagrigoriou, G. Kornaros, and M. Coppola, "High-level Security Services based on a Hardware NoC Firewall Module," in *WISES*, 2015, pp. 73–78.
[8] C. Reinbrecht, A. A. Susin, L. Bossuet, G. Sigl, and J. Sepúlveda, "Timing attack on NoC-based systems: Prime+Probe attack and NoC-based protection," *Microprocess. Microsystems*, vol. 52, pp. 556–565, 2017.
[9] C. Lee, J. Lee, D. Koo, C. Kim, J. Bang, E.-K. Byun, and H. Eom, "Towards enhanced I/O performance of a highly integrated many-core processor by empirical analysis," *Cluster Computing*, pp. 1–13, 2021.
[10] C. Lee, J. Cho, J. Kim, and H. Jin, "Transparent many-core partitioning for high-performance big data I/O," *Concurr. Comput. Pract. Exp.*, vol. 33, no. 18, 2021.
[11] Z. Jiang, K. Yang, Y. Ma, N. Fisher, N. C. Audsley, and Z. Dong, "I/O-GUARD: Hardware/Software Co-Design for I/O Virtualization with Guaranteed Real-time Performance," in *DAC*, 2021, pp. 1159–1164.
[12] S. Vaas, P. Ulbrich, C. Eichler, P. Wägemann, M. Reichenbach, and D. Fey, "Taming Non-Deterministic Low-Level I/O: Predictable Multi-Core Real-Time Systems by SoC Co-Design," in *ISORC*, 2021, pp. 43–52.
[13] S. Zhao, Z. Jiang, X. Dai, I. Bate, I. Habli, and W. Chang, "Timing-Accurate General-Purpose I/O for Multi- and Many-Core Systems: Scheduling and Hardware Support," in *DAC*. IEEE, 2020, pp. 1–6.
[14] A. Ehret, E. D. Rosario, C. Schwicking, K. Gettings, and M. A. Kinsy, "Reconfigurable Hardware Root-of-Trust for Secure Edge Processing," in *HPEC*, 2021, pp. 1–7.
[15] A. Suyyagh and Z. Zilic, "Energy and Task-Aware Partitioning on Single-ISA Clustered Heterogeneous Processors," *IEEE Trans. Parallel Distributed Syst.*, vol. 31, no. 2, pp. 306–317, 2020.
[16] L. L. Caimi, V. Fochi, E. Wächter, D. Munhoz, and F. G. Moraes, "Secure Admission and Execution of Applications in Many-core Systems," in *SBCCI*, 2017, pp. 65–71.
[17] S. Charles, Y. Lyu, and P. Mishra, "Real-Time Detection and Localization of Distributed DoS Attacks in NoC-Based SoCs," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 39, no. 12, pp. 4510–4523, 2020.
[18] L. L. Caimi and F. G. Moraes, "Security in Many-Core SoCs Leveraged by Opaque Secure Zones," in *ISVLSI*, 2019, pp. 471–476.
[19] C. Reinbrecht, A. Aljuffri, S. Hamdioui, M. Taouil, B. Forlin, and J. Sepúlveda, "Guard-NoC: A Protection Against Side-Channel Attacks for MPSoCs," in *ISVLSI*, 2020, pp. 536–541.
[20] M. Ruaro, L. L. Caimi, V. Fochi, and F. G. Moraes, "Memphis: a framework for heterogeneous many-core socs generation and validation," *Design Automation for Embedded Systems*, vol. 23, no. 3, pp. 103–122, 2019.
[21] E. Wachter, L. L. Caimi, V. Fochi, D. Munhoz, and F. G. Moraes, "BrNoC: A broadcast NoC for control messages in many-core systems," *Elsevier Microelectronics Journal*, vol. 68, pp. 69–77, 2017.