A Trusted Communication Unit for Secure Tiled Hardware Architectures

Sebastian Haas, Nils Asmussen
Barkhausen Institut, Dresden, Germany
forename.surname@barkhauseninstitut.org

***Abstract*—Modern hardware and software in smart systems and devices need to provide high performance and energy efficiency and, at the same time, properly address security and privacy goals. M^3 proposed a system architecture that integrates cores and accelerators within a tiled hardware architecture using a security-by-design approach. Each tile includes a hardware component called trusted communication unit (TCU), which isolates all tiles from each other so that no communication is possible by default.**

In this paper, we designed, developed, and synthesized the hardware components of the M^3 system architecture comprising a network-on-chip, RISC-V cores, and TCUs. Latency measurements show that the timing overhead introduced by the data transfer and security features of the TCU is not significant compared to other latencies like cache accesses or software routines. Synthesis results reveal that the area overhead of the trusted system components is only 11 % when the system is scaled to a high number of processing tiles.

I. Introduction

Multiprocessor system-on-chips (MPSoCs) are the foundation of today's smart systems such as Internet of Things or mobile devices. Emerging technology trends such as 5G/6G communication, machine learning, and augmented reality pose big challenges to these MPSoCs in terms of data rates and power consumption [1]. While current implementations can already achieve performance and energy targets [2], properly addressing security and privacy goals is still a challenging task. Security is particularly important for these MPSoCs, because an attacker could exploit vulnerabilities in the system and cause harm, e.g., to the environment, infrastructure, or even human life. As the attacker model, we assume that the attacker gained control over a subset of the MPSoC hardware including software running on programmable processing cores (e.g., via hardware trojans or malicious software). The goal is to isolate hardware and software components from each other to increase the difficulty for attackers to spread out in the system and actually cause damage.

Naturally, some components need to work correctly to place trust in a specific system function. These hardware and software components are called *trusted computing base* (TCB). The goal is to minimize the TCB and thus reduce the probability that an attacker can compromise the entire system. Furthermore, to preserve performance and energy goals of the MPSoC, the TCB should not significantly increase latencies and should consume minimal resources such as chip area.

The M^3 system [3] has been proposed to address the aforementioned security challenges. It is a hardware/software co-design that is based on a tiled architecture. Physically separated tiles are connected by a network-on-chip (NoC) and contain computational logic (processing cores, accelerators), memory, or interfaces to external resources. Furthermore, each tile includes a hardware component called *trusted communication unit* (TCU)[1], which isolates all tiles from each other so that no communication is possible by default. A microkernel-based operating system (OS) runs on dedicated tiles and configures the TCUs. The OS manages communication channels between the tiles while the TCUs enforce them in hardware. Hence in this concept, the TCU is the main part of the hardware TCB. Besides general direct memory access (DMA) features, the TCU provides further support for OS-specific functionalities like message passing, virtual memory, and context switching.

In this paper, we took the M^3 approach and designed and implemented the hardware components of the system architecture comprising the NoC, processing cores, and TCUs. We simulated and synthesized the hardware to quantitatively evaluate the TCU and to analyze the overhead introduced by its security features.

In summary, the contributions of this paper are as follows: 1) We present a hardware component called trusted communication unit (TCU) to enable secure communication in tiled architectures. 2) We developed and synthesized a system architecture that integrates a NoC, TCUs, and multiple processing cores to run the OS. 3) We evaluate latency and area consumption of the hardware components to show the practical feasibility of the M^3 approach and to analyze the impact of the TCB on the MPSoC. Furthermore, the M^3 software as well as the complete hardware implementation used in this paper are available as open source[2].

II. Related Work

In current MPSoCs, enforcing isolation between applications and controlling access to other resources (e.g., memory, accelerators, I/O devices) is mainly achieved with security features integrated in general-purpose processors. For example, memory management units (MMUs) and IOMMUs are used to securely share memory of general-purpose processors and I/O devices, respectively. This approach assumes that the cores and their isolation features are working correctly, which is not

[1] In M^3 [3], this component was originally called data transfer unit (DTU). We changed the name to better reflect its security properties.

[2] https://github.com/Barkhausen-Institut/M3

necessarily true as recent attacks have shown [4]. In our work, we do not trust these cores, but introduce a separate hardware unit that enforces isolation between tiles.

Similar to our approach, memory protection units (MPUs), also called hardware firewalls, have been proposed for tiled architectures, that only forward allowed memory accesses from the tile to external memory. For example, Fiorin et al. [5] propose a data protection unit (DPU) as a firewall together with a dedicated hardware component called network security manager which configures the access rights of the DPUs. Further, Tan et al. [6] present a per-tile hardware firewall called *isolation unit* that is not configured by a centralized authority. Instead, each application on a tile has base permissions and can transfer some of them to other tiles. In contrast to the firewalls in NoC interfaces, Sepulveda et al. [7] extend the NoC with security mechanisms. Customized routers guarantee that sensitive traffic communicates only through trusted nodes.

The aforementioned approaches are hardware-only solutions that provide the isolation features but, in contrast to our TCU, are not co-designed with the OS that inherently provides application loading and scheduling. There are other security architectures like TrustLite [8] and TyTan [9] that take the MPU hardware approach and involve the OS which configures the MPUs at runtime and allows to isolate software applications. However, these are processor-centric solutions that do not consider tiled architectures with heterogeneous hardware components. In contrast, there are NoC-centric approaches like NoC-MPU [10] and SiFive Shield [11]. They combine the extended MPU approach with the management capabilities of the OS to build a tiled security architecture.

In this paper, we present the M^3 approach based on TCUs, that further extends the NoC-centric MPU approaches by adding OS-specific functionalities like message passing, which is used to communicate between applications and OS services. Furthermore, the TCU implements MMU features like virtual memory and context switching in a lightweight manner to share tile-internal resources.

III. Hardware Design

M^3 [3] is the operating system (OS) for a new system architecture that considers heterogeneous compute units (general-purpose cores with different instruction sets, DSPs, FPGAs, fixed-function accelerators, etc.) from the beginning. The system builds upon a tiled architecture where multiple physically separated tiles are connected by a NoC. Furthermore, each tile includes a TCU, which isolates all tiles from each other so that no communication is possible by default. One tile must include a general-purpose processor that runs the OS kernel which configures the TCUs and hence manages communication between tiles. For that purpose, we implemented a tile with a RISC-V Rocket core [12] as general-purpose processor with caches, which is available as open-source and can be configured (e.g., instruction set, cache sizes, debug features). For comparison, we also implemented a tile with a BOOM core, which is the out-of-order variant of Rocket, to further evaluate the area overhead of the TCB.

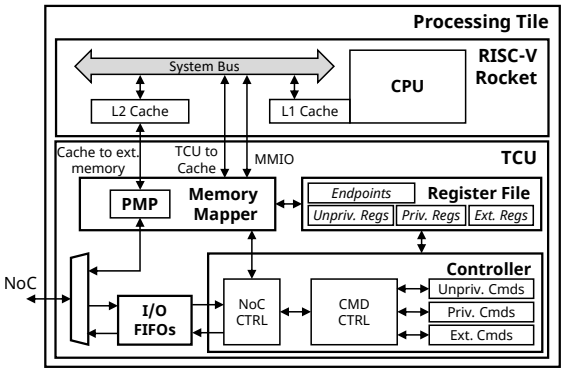

Fig. 1. Processing tile with TCU and RISC-V Rocket core

Figure 1 shows a tile that integrates the TCU and the Rocket core. The core communicates with the TCU via memory-mapped I/O (MMIO). The TCU has a tightly-coupled memory interface to access the internal caches of the core via its bus system. Other cores without caches or simple processing units with only scratch-pad memory can be connected in the same way. Interrupt signals of the core (not shown in Figure 1) are also connected to the TCU. Hence, the TCU can trigger the core at certain events such as message reception. Since the TCU interconnects the NoC and the logic within the tile, it provides a uniform interface to other tiles that simplifies the management and collaboration of heterogeneous tiles.

In our system architecture, the NoC, the processor that runs the OS kernel, and the TCUs make up the TCB. Due to their central tasks of configuring and setting up communication channels between tiles, the TCUs and the OS kernel with its underlying processor have to be trusted. The NoC must be also part of the TCB since its current implementation cannot guarantee that an eavesdropper might leak data on the network. For example, eavesdropping could be prevented via encryption of the NoC packets.

Trusted Communication Unit

The TCU builds an important part of the hardware TCB in our tiled architecture, i.e. it has to enable the necessary isolation features while still allowing communication between the tiles via the NoC. The TCU requires an interface to the NoC as well as an interface to local resources of the tile. As depicted in Figure 1, our hardware implementation of the TCU contains three main blocks which cover the main functionalities: controller, register file, and memory mapper. The TCU controller implements finite state machines to executes commands as requested by the core or from the NoC. The register file includes registers such as *endpoint* registers (briefly called endpoints) to store access permissions. The memory mapper can multiplex accesses from the core to the local memory and to the register file (MMIO). Additionally, the physical memory protection (PMP) block forwards memory requests of the core to the NoC. These requests are validated according to access permissions stored in dedicated endpoints.

To enable usage and configuration of the endpoints, the TCU commands and registers can be accessed by three interfaces:

unprivileged, privileged, and external interface. The *unprivileged* interface enables commands for DMA transfers (read, write) and message passing (send, receive, reply). During command execution, the TCU checks the related endpoints and, if allowed, performs the data transfer to the target tile. A command is finished when a read response or a message acknowledgment was received. The *privileged* interface is only accessible by privileged software running on the core and enables support for virtual memory and context-switching in general-purpose cores. These features were introduced in the M^3 extension called M^3v [13]. The privileged software uses the MMU of the core to ensure that unprivileged software can only access MMIO addresses of the unprivileged interface. To support virtual addressing, the TCU holds a software-loaded translation lookaside buffer (TLB) to store recent address translations. The *external* interface is only used by external tiles (e.g., the tile running the OS kernel) to configure endpoints and to enable features of the privileged interface.

In a typical scenario, the OS kernel configures the endpoints of all tiles in the system via the external interface of the TCU. The core in the tile uses the unprivileged interface to access these endpoints and to establish communication channels to other tiles. If virtual addressing is used, the privileged software of the core can load address translations to the TLB via the privileged interface of the TCU. General-purpose cores with cache require access to shared external memory. The PMP of the TCU is used to connect the last-level cache to the external memory via NoC.

IV. Evaluation Results

In this section, we show and evaluate the experimental results of our implemented system architecture, which consists of processing tiles, each including a TCU and a RISC-V core, as well as the NoC routers. By default, the TCU includes all interfaces and features as presented in Section II. We integrated the generated Verilog code of the RISC-V cores into our design, which was generated by using the open-source Chisel generator [12]. The design is simulated and synthesized with Cadence tools by using a 22 nm FDSOI process from Globalfoundries under typical conditions (25 °C, 0.5 V). We observed the critical path of the processing tile between the RISC-V core and the SRAM of the caches. To meet all timing requirements under the selected conditions, we used 100 MHz as the maximum clock frequency in our design.

A. Latency

In the first set of experiments, we measure the latency of the TCU to evaluate its timing overhead when executing commands initiated by the core. Throughput is limited by the bandwidth of the NoC (16 bytes/cycle) since the current TCU implementation supports the full NoC bandwidth. For these experiments, we use two identical processing tiles each including a TCU and a RISC-V core. The tile which initiates commands contains the so-called *local* TCU, while the receiving tile contains the *remote* TCU. The tiles are connected to a single NoC router to minimize the delay induced by the NoC.

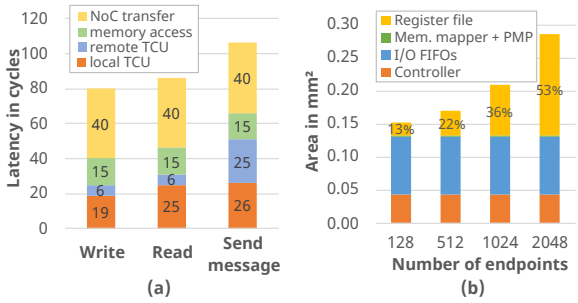

Fig. 2. (a) Latency of TCU commands, (b) Area consumption for different number of endpoint. The percentages denote the portion of the register file to the total TCU area.

Figure 2a shows the measurement results of a DMA-write and read as well as for sending and receiving a message (with 8-byte payload data each). For each transfer, the local TCU requires 11 cycles to initiate the command by reading the corresponding endpoint and validating the access permissions. The TCU also performs a TLB lookup which requires at least 4 cycles if the associated TLB entry is found immediately. If more TLB entries have to be scanned, a higher number of cycles is needed. If the privileged interface (virtual memory support) of the TCU is disabled, the TLB access is skipped and the latency of the command is reduced accordingly. The remaining cycles of the local TCU are spent to prepare the access to the RISC-V cache and to finish the command as soon the response/acknowledgment packet has been received. The time of a memory access of the RISC-V is specific to this core and depends on its internal cache access times. For the write and read commands, the remote TCU requires 6 cycles to forward the requests to the memory. For message passing, the remote TCU takes about 25 cycles on average. This includes finding a free slot in the memory-mapped receive buffer, setting up an endpoint for a reply, and informing the software in case the received message belongs to a currently paused application on the core (only if the privileged interface is enabled). The NoC transfer delay is about 20 cycles per packet via the single router, which is mainly caused by the synchronization registers in the NoC interface. Asynchronous transitions enable to set different clock frequencies of the NoC and the tile. Since each command consists of a data and a response/acknowledgment packet, we measured 40 cycles for the total transfer.

In summary, the latency introduced by the TCU functionalities and security features is not significant. Memory read delays to the RISC-V caches and packet transmission times via NoC are in the same range or higher. Furthermore, processor-intern routines of the RISC-V (e.g., interrupt handling) typically show up to two magnitudes higher latencies [14].

B. Area consumption

In the next set of experiments, we evaluate the area consumption of the synthesized hardware components. Table I lists the area of the two RISC-V variants, a single NoC router, and the full-featured TCU. Compared to the in-order Rocket core, the higher complexity of the out-of-order BOOM core leads to the 1.7× area increase. The TCU takes 11 % and

TABLE I
AREA CONSUMPTION OF HARDWARE COMPONENTS

	Total (mm^2)	SRAM (mm^2)
BOOM	2.499	1.716
Rocket	1.454	1.280
NoC router	0.022	0
TCU	0.154	0.120
Controller	0.044	0.017
NoC CTRL	0.012	0
CMD CTRL	0.032	0.017
Unpriv. cmds.	0.012	0
Priv. cmds.	0.019	0.017
Ext. cmds.	0.001	0
Register file	0.021	0.017
Memory mapper + PMP	0.003	0
I/O FIFOs	0.086	0.086

$6\,\%$ of the area of the Rocket and BOOM core, respectively, and thus only adds a small area overhead to the processing tile. In the TCU, the register file, the TLB, and the I/O FIFOs are implemented in SRAM since we obtained a slightly higher area consumption for an implementation in hardware registers. For example, the register file holds 128 24-byte endpoints and ten 8-byte registers for all commands, which fit into one $4\,\mathrm{kB}$ SRAM block. I/O FIFOs take a large portion of the TCU area because they implement in total five SRAM blocks for send and receive direction each to hold at least one complete NoC packet with up to $2\,\mathrm{kB}$ payload data. This is important to allow a deadlock-free data transfer. The TCU can be configured individually, e.g., if a tile only contains the interface to external memory, there are no I/O FIFOs, no register file, but only the NoC controller required. In this case, the total TCU area is reduced by $92\,\%$ to $0.012\,\mathrm{mm}^2$.

Endpoints are used for exchanging data between applications, accessing files, or for socket-based communications over the network. Therefore, applications that work with many files or sockets simultaneously require many endpoints and in particular, sharing a tile among multiple applications may increase the number of required endpoints significantly. In Figure 2b, we evaluate the area consumption of the TCU with an increasing number of endpoints. Compared to the total TCU area, the percentage of the register file increases from $13\,\%$ at 128 endpoints to $53\,\%$ at 2048 endpoints. This shows that scaling the number of endpoints by increasing the number of SRAM blocks is unacceptable with the current design. In future work, we plan to outsource endpoints to external memory and prevent unauthorized access by applying encryption mechanisms.

C. Overhead of the TCB

With the help of the area results we can conclude the costs of the TCB compared to the whole hardware architecture. As stated in Section II, the NoC, the processor that is running the OS kernel, and the TCUs make up the TCB. Assuming a system architecture with n tiles including RISC-V Rocket cores and TCUs while each tile is connected to a single NoC router, we can estimate the ratio of the area of the TCB to the entire area. For example for a system architecture with n=8, we obtain a TCB overhead of $23\,\%$. The overhead decreases with increasing n and reaches the limit at $11\,\%$ for n>100.

V. CONCLUSION

We investigated the overhead of the TCB in the M^3 system by designing and developing the required hardware components of the underlying tiled architecture. Latency measurements have shown that the timing overhead introduced by the communication functionalities of the TCU is in the same range as typical data transfers of the NoC or memory accesses of the general-purpose RISC-V processor. Synthesis results reveal that the area overhead of the trusted system components is only $11\,\%$ when the system is scaled to a high number of processing tiles (>100). Hence, the added security properties in the M^3 architecture have only a low impact on the overall performance and area consumption of the system.

ACKNOWLEDGMENTS

This research received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No. 957216 (iNGENIOUS). It is also financed on the basis of the budget passed by the Saxon State Parliament in Germany.

REFERENCES

[1] IRDS, "International Roadmap for Devices and Systems: 2021 Update - Executive Summary," 2021. [Online]. Available: https://irds.ieee.org/images/files/pdf/2021/2021IRDS_ES.pdf

[2] A. Pullini, D. Rossi, I. Loi, G. Tagliavini, and L. Benini, "Mr.Wolf: An Energy-Precision Scalable Parallel Ultra Low Power SoC for IoT Edge Processing," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 7, 2019.

[3] N. Asmussen, M. Völp, B. Nöthen, H. Härtig, and G. Fettweis, "M^3: A Hardware/Operating-System Co-Design to Tame Heterogeneous Manycores," in *21st International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, ser. ASPLOS'16. ACM, 2016, pp. 189–203.

[4] A. Nilsson, P. N. Bideh, and J. Brorsson, "A Survey of Published Attacks on Intel SGX," *arXiv preprint arXiv:2006.13598*, 2020.

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

[6] B. Tan *et al.*, "A System-level Security Approach for Heterogeneous MPSoCs," in *Conference on Design and Architectures for Signal and Image Processing (DASIP)*, 2016, pp. 74–81.

[7] J. Sepulveda, R. Fernandes, C. Marcon, D. Florez, and G. Sigl, "A Security-Aware Routing Implementation for Dynamic Data Protection in Zone-Based MPSoC," in *30th Symposium on Integrated Circuits and Systems Design: Chip on the Sands*, ser. SBCCI '17, 2017.

[8] P. Koeberl, S. Schulz, A.-R. Sadeghi, and V. Varadharajan, "TrustLite: A Security Architecture for Tiny Embedded Devices," in *9th European Conference on Computer Systems*, ser. EuroSys '14. ACM, 2014.

[9] F. Brasser, B. El Mahjoub, A.-R. Sadeghi, C. Wachsmann, and P. Koeberl, "TyTAN: Tiny Trust Anchor for Tiny Devices," in *52nd Annual Design Automation Conference*, ser. DAC '15, 2015.

[10] J. Porquet, A. Greiner, and C. Schwarz, "NoC-MPU: A secure architecture for flexible co-hosting on shared memory MPSoCs," in *Design, Automation & Test in Europe Conference & Exhibition*, ser. DATE'11, March 2011.

[11] J. Prior, "SiFive Shield: An Open, Scalable Platform Architecture for Security," 2019. [Online]. Available: https://www.sifive.com/blog/sifive-shield-an-open-scalable-platform-architecture

[12] K. Asanović *et al.*, "The Rocket Chip Generator," *EECS, University of California at Berkeley, Tech. Rep. UCB/EECS-2016-17*, 2016.

[13] N. Asmussen, S. Haas, C. Weinhold, T. Miemietz, and M. Roitzsch, "Efficient and Scalable Core Multiplexing with M^3v," in *27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2022, p. 452–466.

[14] B. Sa, J. Martins, and S. Pinto, "A First Look at RISC-V Virtualization from an Embedded Systems Perspective," *IEEE Transactions on Computers*, no. 01, pp. 1–1, Nov. 2021.