Trustworthy Silicon: An MPSoC for a Secure Operating System

Sebastian Haas Christopher Dunkel Friedrich Pauls Mattis Hasler Yogesh Verma

{name}.{surname}@barkhauseninstitut.org
Barkhausen Institut, Dresden, Germany

***Abstract*—The rapid evolution of the Internet of Things (IoT) has deeply integrated digital systems into our daily decision-making processes, making their reliability crucial. A failure in these systems not only disrupts functionality but can also pose significant risks to human safety. While software updates have become a standard measure for addressing vulnerabilities, the increasing prevalence of hardware vulnerabilities in processors is an emerging concern. These vulnerabilities can compromise security features and application isolation mechanisms provided by the operating system, undermining the overall trustworthiness of these systems.**

In this work, we introduce Masur23, the first silicon implementation of a trustworthy multi-processor chip platform that incorporates a dedicated and independent hardware component known as the Trusted Communication Unit (TCU) to enforce access control mechanisms. While communication is managed by the operating system, the TCU ensures the security of the system by enforcing the isolation of untrusted hardware and software components directly in hardware. Our measurements demonstrate that implementing a TCU-based multi-processor is not only feasible but also introduces minimal overhead in terms of latency, area, and power consumption compared to other components on the chip.

***Index Terms*—Trustworthy Chip Platform, Tiled Architecture, Microkernel-based Operating System**

I. Introduction

In recent years, the number of hardware vulnerabilities in processors has steadily increased. Vulnerabilities such as Meltdown [1] and Spectre [2] have demonstrated that affected hardware often fails to maintain the isolation guarantees it promises. This failure to uphold critical security assertions at the hardware level cascades through the entire application stack, starting with the operating system (OS), which relies on these assertions to create isolated environments for applications, abstracting them from the underlying hardware. However, the effectiveness of the isolation provided by the OS is fundamentally dependent on the integrity of the hardware assertions it relies upon. When these hardware assertions are compromised, the security guarantees provided by the OS and all subsequent layers are likewise invalidated.

As processors, even those considered "simple", grow increasingly complex, it becomes nearly impossible to verify that a processor is free of intentional or unintentional vulnerabilities that could compromise the isolation guarantees of the entire system. Intentional vulnerabilities such as hardware trojans are particularly concerning as they can be subtle and often indistinguishable from ordinary hardware bugs. This challenge is exacerbated by the fact that processors are continually becoming larger and more complex, making the detection of such vulnerabilities nearly impossible.

By physically isolating processors on a chip—either by placing them in distinct, separated areas or across multiple chips—entire classes of hardware vulnerabilities can be effectively neutralized. While complete isolation is not feasible due to the necessity of some form of communication, connecting processors via a packet-based network and securing only the communication pathways significantly reduces the complexity of ensuring system security.

This concept is further developed in [3], which introduces a HW/OS co-designed platform that supports a mikrokernel-based OS, called M^3, where each process and OS service is assigned to its own processor. Communication between processes is facilitated through capabilities that allow access to globally shared memory or enable message passing between processes. These capabilities are managed by the OS kernel and enforced by the Trusted Communication Unit (TCU), with each processor having its own TCU as the sole gateway to the Network-on-Chip (NoC). Within such a platform, the trusted computing base (TCB) is the set of hard- and software components every application must rely on to maintain a specific security objective. In particular, the TCB for an application includes only the processor it runs on as well as key components like the TCUs and the NoC. Based on the description of the TCB, an attack model can be defined, against which the platform is protected. That is, the platform prevents an attacker from compromising the whole system by exploiting vulnerabilities in untrusted hard- or software components. These untrusted components are not part of the TCB and isolated by a TCU, which strictly controls their communication capabilities.

Use cases for a security concept like the described HW/OS platform become particularly relevant when multiple applications run on the same hardware and must operate without interference. In this context, non-interference means that a compromised application—whether through a hardware vulnerability or a malicious software source—cannot threaten the execution or integrity of other applications. This is especially important in edge cloud systems and their associated hardware, such as home servers.

For instance, many commercial solutions like "Fritz!Box" or NAS servers offer an "app"-like system management interface. The risk arises when an app downloaded from a non-certified source introduces a malicious process into the system. In such scenarios, it is crucial that the attacker cannot compromise the entire system, particularly ensuring that mission-critical processes, such as basic routing functionality, remain unharmed and fully operational.

In this work, we present Masur23, the first silicon implementation of a multi-processor system-on-chip (MPSoC) that supports the described HW/OS security concept from [3] and utilizes the TCU from [4]. The TCU acts as the core security component for enforcing data access control and inter-processor communication. Building on this foundation, an operating system like M^3 can achieve robust application isolation. Masur23 demonstrates that a TCU-based multi-processor implementation is not only feasible but also introduces manageable overhead. Specifically, the TCU adds approximately $14\,\%$ overhead in terms of area to a RISC-V based processing unit. The TCU achieves data transfer rates of around $400\,\mathrm{MB/s}$ with a clock frequency of $100\,\mathrm{MHz}$ and a 200 cycle round-trip time for message passing between processors.

This paper is organized as follows. Section II gives an overview of related work. Next, Section III introduces our trustworthy tiled architecture concept and explains the *isolation-by-default* approach. Section IV describes the implemented chip architecture and gives details on the physical design. Section V provides an evaluation of the chip, including experimental results, followed by a conclusion in Section VI.

II. Related Work

Our proposed and implemented chip platform facilitates communication and isolation of untrusted hard- and software components within an MPSoC that contains mutliple tiles includes processing elements connected by a NoC. Several approaches in the literature also address isolation and access control mechanisms in SoC architectures. In this section, we review state-of-the-art architectures that incorporate: A) memory protection units (MPUs) as hardware-only solutions, B) MPUs configured by software or an OS, and C) NoC-centric isolation, where data transfers are routed to bypass untrusted tiles.

A. MPUs as hardware-only solutions

The concept of a dedicated hardware component, which checks all requests initiated by a tile, commonly referred to as MPU, has been extensively explored in the literature. For example, the work in [5] introduces MPUs, termed hardware firewalls, in each NoC interface of a tiled MPSoC. These firewalls check tile access requests to memory against a lookup table that contains predefined access rights.

Similarly, the work in [6] proposes a hardware firewall, called isolation unit, integrated into each NoC interface of a tile. Unlike the approach in [5], this isolation unit is not configured by a centralized authority. Instead, each application running on a tile has base permissions, with the ability to transfer some of these permissions to other tiles.

Other research enhances these MPUs by implementing a framework that enforces access control policies not only for memory but also for various shared peripherals [7]. Further extensions include runtime monitoring of security issues arising from untrusted computing elements [8] or hardware trojan-like attacks [9]. Additionally, Fiorin et al. [10] propose an MPU, called data protection unit (DPU), along with a dedicated hardware component known as the network security manager, which configures the DPUs' access rights. Incorporating a DPU results in a $17\,\%$ area and $7.5\,\%$ energy overhead, without negatively impacting system performance.

B. MPUs configured from Software

In contrast to our proposed chip platform system, the MPUs in the aforementioned approaches are not managed by privileged software or an operating system like M^3, limiting their ability to dynamically reconfigure access rights during runtime and to handle application loading and scheduling. However, some solutions in the literature introduce a software architecture to support these hardware components. For example, NoC-MPU [11] is a memory protection unit configured by trust agents—privileged software that establishes and maintains communication between tiles. Similarly, the SecBus architecture [12] secures access to external memory by attaching an MPU to the external memory interface, rather than next to each computing unit. SecBus also incorporates hardware cryptography engines to enable rapid creation of digital signatures, which are used during system boot to verify that the OS kernel has not been tampered with by an adversary.

These approaches represent a hardware/software co-design that integrates hardware-enforced access permissions with the management capabilities of the underlying operating system. However, M^3 extends these solutions by, e. g., adding support for message passing, ensuring that also communication between heterogeneous computing tiles is protected, not just access to memory. Additionally, the NoC architecture in M^3 facilitates the design of a tiled and scalable system. Once communication channels and access rights are configured by the M^3 OS, applications do not necessarily need to interact with the OS kernel, enabling scalability from the software side.

C. NoC-centric isolation

Another approach to isolating tiles within a NoC involves adapting the routing of data traffic. Specifically, customized routers ensure that sensitive data is routed exclusively through trusted nodes, thereby preventing potentially untrusted tiles from accessing confidential information [13], [14]. Wehbe et al. [15] build on this idea by utilizing the reconfigurability of FPGAs to enhance adaptive routing. During the MPSoC's boot process, computing elements on all tiles are authenticated using digital signatures. If a tile's signature is invalid, the NoC region corresponding to that tile is partially reconfigured on the FPGA to disconnect its links.

These NoC-centric isolation strategies could serve as an extension to our proposed chip platform. In M^3, TCUs already enforce isolation between tiles. Implementing adaptive routing

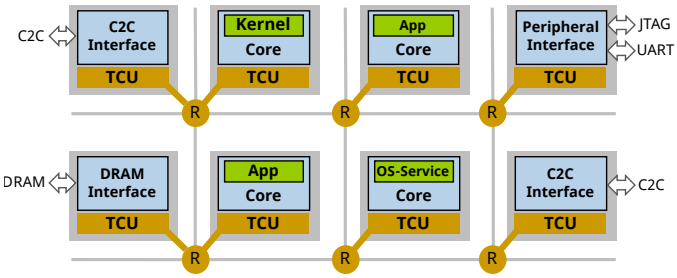

Fig. 1: Block diagram of a generic platform architecture. Processing cores and interfaces are connected with a network-on-chip through trusted communication units (TCUs). Clustering is possible using chip-to-chip (C2C) links.

could further enhance security by protecting the system against a different type of attack model: man-in-the-middle attacks. For instance, if an attacker is located on a specific tile, data leakage by monitoring traffic through the attacker's adjacent router would be prevented, as sensitive data transfers would be rerouted to avoid passing through the compromised router.

III. Trustworthy Chip Platform

In this paper, we propose a chip platform with a tiled hardware architecture designed to securely integrate untrusted heterogeneous compute units. This architecture allows individual applications and software components of a microkernel-based operating system to be mapped onto separate tiles, thereby establishing a trustworthy chip platform. In the following sections, we detail the general chip architecture, the M^3 operating system, and the specific chip implementation. Both the M^3 software and the primary components of the hardware implementation are available as open source[1].

A. General Architecture

Our proposed chip platform is based on a tiled hardware architecture, where multiple logically separated tiles are interconnected by a network-on-chip (NoC), as illustrated in Fig. 1. These tiles encapsulate various components, which can include general-purpose cores, hardware accelerators, or other compute units. Additionally, chip-to-chip (C2C) links, DRAM controllers, and other peripheral interfaces can be integrated to facilitate access to external memory or enable communication with external devices. A key element of this design is the TCU, which connects each component to the NoC, thereby enabling secure communication between all components.

By default, communication between tiles is disabled to ensure an inherently isolated system. Only the OS kernel, running on a dedicated tile, has the authority to configure TCUs by setting communication capabilities. These capabilities are stored in endpoint registers, meaning that they are managed by software but enforced in hardware by the TCUs. The TCUs also abstract the underlying communication fabric (e.g., NoC) by implementing various communication protocols for the functional units they host (e.g., processing cores). Supported protocols are: 1) RDMA-like read and write operations for accessing memory blocks of a remote or potentially shared memory (e.g., external DRAM); 2) message-passing channels for high-level communication between applications on different tiles, as well as between applications and the kernel (e.g., system calls); and 3) a caching interface that allows cores with caches to directly access external DRAM with physical memory protection.

Before any communication channel is established, the TCU verifies whether the relevant capabilities are configured; if not, access is denied. OS features such as virtual memory and context-switching enhance the performance of the software but also require support from the TCU. For example, software running on the core initiates an RDMA request and provides a virtual address to the TCU. The TCU must translate this virtual address to a physical address to access the right location in the DRAM. For that purpose, the TCU implements a small translation lookaside buffer (TLB) to speed up the address translation process [16].

In general, the performance of MPSoCs can be enhanced by increasing the parallelism on system-level. In our architecture, this can be achieved by scaling the system to include more tiles and expanding the NoC with additional routers. Another approach we adopt in our Masur23 chip implementation involves scaling the system by connecting multiple chips via C2C links. Although C2C links introduce additional communication delays, this scaling strategy allows for the creation of flexible chip clusters.

From the perspective of the M^3 OS, tiles on different chips are managed similarly. However, when scheduling applications across tiles, the OS may take the increased latencies into account. For instance, applications with higher communication demands should be placed on tiles within the same chip, while those with lower communication demands can be distributed across tiles on different chips.

B. Operating System

In contrast to monolithic OS'es, where the kernel has full access to the entire system, microkernel-based systems like M^3 [17] or L4 [18] decompose the kernel into multiple isolated components (e. g., drivers, file systems, protocols). These OS services run as unprivileged software within their own address spaces. Consequently, compromising the entire system typically requires an attacker to exploit a series of vulnerabilities across multiple software components, rather than just one, thereby inherently enhancing system security.

The security concept is further reinforced by combining the already isolated software components of the microkernel-based system with the tiled hardware architecture. In this setup, isolation between OS services and applications is not solely dependent on software mechanisms, such as processor privilege modes; instead, it is enforced at the hardware level by the TCUs. Our proposed chip platform leverages this tiled hardware architecture, running the microkernel-based M^3 OS, to provide a robust security framework.

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

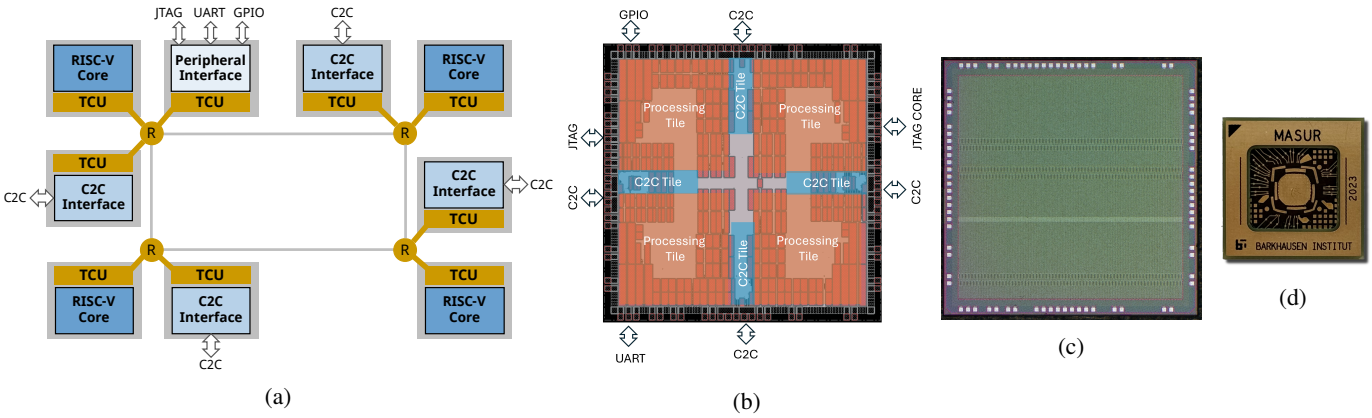

Fig. 2: Masur23 design. (a) Top-level architecture. (b) Post-layout design. (c) Fabricated die. (d) Custom-made wirebond BGA-package on FR-4 basis.

IV. Masur23 Chip Implementation

In this section, we introduce Masur23, our first silicon implementation of the M^3 platform. The subsequent subsections provide a detailed overview of both the architectural and physical implementation of the chip.

A. Architecture

For the Masur23 chip, we implemented a subset of the platform depicted in Fig. 1 to prototype and test the core functionalities of the M^3 system. This implementation allows us to perform communication using the aforementioned protocols. Specifically, we can access memory locations via RDMA requests, enable communication between software components through message passing, and utilize physical memory protection to forward cache accesses to external memory.

Our design includes four routers connected in an orthogonal mesh configuration, as shown in Fig. 2a. Each router connects to a processing tile that contains a 64-bit RISC-V Rocket core [19] and a fully featured TCU. The Rocket core is equipped with 32 kB of L1 and 256 kB of L2 cache. In this implementation, since no DRAM is directly attached to the chip, the cores access external memory via C2C links.

Additionally, each router connects to a C2C tile, enabling the system to scale outward in all directions. The C2C links utilize a full-duplex single-lane Xilinx Aurora 8b10b serial link over two pairs of Low-Voltage Differential Signaling (LVDS) lines, with one pair dedicated to sending and the other to receiving data. Each C2C tile includes a TCU to provide a uniform interface with the NoC router, although fully featured TCUs are not necessary in this context. Instead, these TCUs are configured to forward all packets directly to the C2C link and the corresponding peering chip. A third type of tile, the periphery tile, is also included in the design. This tile houses a collection of control and debug interfaces, such as JTAG and UART, to facilitate system management and troubleshooting.

B. Physical Implementation

The Masur23 chip was engineered using a comprehensive suite of EDA tools: Cadence Genus for synthesis, Innovus for place-and-route, Tempus for static timing analysis, and Calibre for physical verification. The chip fabrication was carried out at GlobalFoundries' Dresden facility, utilizing the 22 nm FD-SOI technology.

Module	Count	Area per Instance [mm^2]			Total area
		Total area	SRAM area	SRAM %	[mm^2]
Processing tile	4	1.149	0.850	73.97	4.596
Rocket	4	0.874	0.681	77.91	3.496
TCU	4	0.141	0.11	78.01	0.564
NOC-IF	4	0.133	0.059	44.36	0.532
C2C tile	4	0.176	0.077	43.75	0.704
C2C Aurora	4	0.036	0.018	50.0	0.144
TCU	4	0.005	0	0	0.02
NOC-IF	4	0.134	0.059	44.02	0.536
Periphery tile	1	0.135	0.059	43.70	0.135
TCU	1	0.005	0	0	0.005
NOC-IF	1	0.130	0.059	45.38	0.130
NoC Router	4	0.014	0	0	0.056

TABLE I: Masur23 area utilization summary.

The chip has a total die area of 9.9 mm^2, with dimensions of 3.15×3.15 mm^2. Fig. 2b illustrates the post-route design, highlighting the four tiles, each surrounded by their respective SRAM cells. NoC and C2C links are positioned between the RISC-V tiles, ensuring efficient communication across the chip.

The area utilization breakdown of the Masur23 chip is detailed in Tab. I. Despite their compact design, the RISC-V Rocket cores occupy 63.7% of the total area, while SRAM cells account for 68.6%, with the majority allocated to the Rocket core's cache—specifically, 32 kB + 256 kB per instance. Notably, the TCUs, critical for the system's security features, contribute only 10.7% to the total area. Two variants of tiles are employed. The first variant are processing tiles featuring a full-featured TCU connected with a Rocket core. These TCUs require SRAM resources for I/O FIFOs, the TLB, and endpoint registers. The second variant includes interface tiles with a smaller TCU. These tiles connect to an external interface like C2C links and peripherals. I/O FIFOs, TLB, and endpoint registers can be omitted, leaving only the logic to connect the NoC interface. In this configuration, the total TCU area is reduced by 96.1% to 0.005 mm^2.

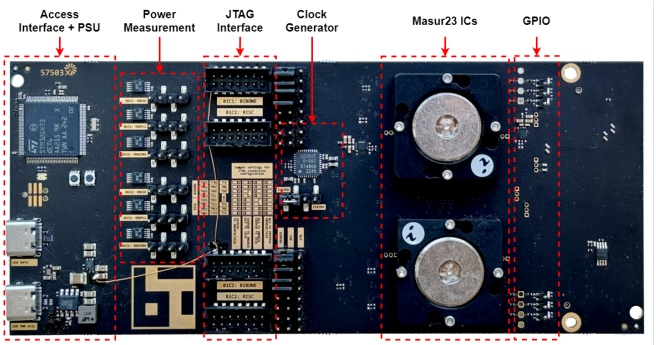

Fig. 3: PCB hosting two chips in BGA sockets. Chips are connected by C2C links. PCB provides voltage control and measurement, test points, USB-C connector and JTAG pin headers.

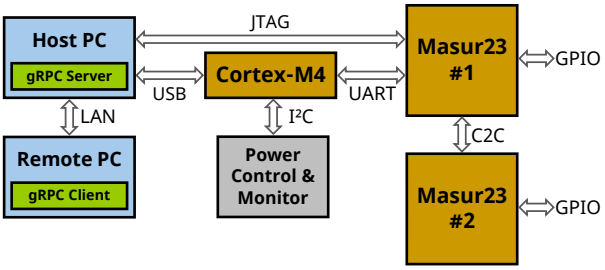

Fig. 4: Evaluation setup overview.

V. Evaluation

After the physical implementation, the Masur23 chips were packaged and mounted on our printed circuit board (PCB). This section provides an overview of our setup and presents the measurement results obtained from the implemented system.

A. Evaluation Setup

Fig. 2d illustrates the custom-made package designed using the Allegro Package Designer Plus tool. The package features an eight-layer FR-4 substrate with a glued-on frame, specifically designed to accommodate a polymer fill. This polymer fill enhances thermal conductivity, electrical insulation, mechanical stability, and environmental resistance. The chip is wire-bonded to the package.

The chips are mounted in ball grid array (BGA) sockets on a PCB, as shown in Fig. 3. This PCB is designed to accommodate two Masur23 chips and to interconnect them via their C2C links. The PCB also provides USB, JTAG, and UART connectivity, facilitating communication between the chips and a host PC. As depicted in Fig. 4, a host PC is required for the initial bring-up process of the chips, while a remote PC can optionally be used for remote access via gRPC. Additionally, the PCB has adjustable power sources for the chip's various power rails, allowing independent tuning of the chip's I/O ring, the on-chip all-digital phase-locked loops (ADPLL), and the main processing core. The board also provides a $100\,\mathrm{MHz}$ reference clock for the chips.

B. Experimental Results

In the lab, we conducted transfer workload tests to validate the Masur23 chip's suitability for running a distributed, microkernel-based operating system like M^3. These tests inherently included the isolation mechanisms provided by the TCU, as these cannot be disabled. The key metrics for evaluating the chip's performance in running a distributed operating system, or distributed applications in general, are communication bandwidth and latency. Communication bandwidth refers to the amount of data that can be transmitted between processors per unit of time, which may be expressed in natural time units such as seconds or normalized to the base clock frequency in cycles. Latency is measured as the time required to send a message and receive an acknowledgment that the message has successfully reached its destination, a critical factor in operating systems. For scalability testing, communication was also evaluated across chip boundaries to ensure that the chip can operate as part of a seamless cluster setup, running a single operating system across multiple chips linked by C2C connections. The measured communication speeds are summarized in Tab. II. These measurements were performed end-to-end, with timestamps recorded before and after invoking a TCU driver function to send a message. The driver function returns only when the acknowledgment is received. To contextualize the measured values, we calculated the theoretical transfer speed by considering the protocol overheads. The NoC transfers only the header in the first cycle, followed by 128 bits per cycle. At a clock frequency of$100\,\mathrm{MHz}$, this results in a bandwidth range of $800\,\mathrm{MB/s}$ to $1592\,\mathrm{MB/s}$, depending on the message length. Long messages were used for bandwidth measurements, while short messages were used to measure latency. The C2C links utilize single-line serial communication based on the Aurora protocol, which employs 8b10b encoding at the lowest layer. The protocols introduce overhead factors of 0.93 and 0.8, resulting in a combined total overhead factor of 0.47, including the NoC overhead. With a baseline frequency of $800\,\mathrm{MHz}$, the theoretical bandwidth of a C2C link is $47\,\mathrm{MB/s}$. Considering these theoretical communication speeds, a single processor with a TCU and the M^3 software stack achieves approximately $26\,\%$ utilization of the NoC bandwidth and over $95\,\%$ utilization of the C2C links (cf. Tab. II). The relatively low NoC utilization can be attributed to the M^3 software stack, which serializes transfers, and the requirement for each message to be acknowledged, causing the sender to wait for a response from the receiver before sending the next message.

As shown in Tab. II, the round-trip time for transporting a message from one processor to another is 193 cycles for on-chip transfers and 306 cycles for cross-chip transfers. This transfer process includes the time for the software stack, TCU command handling with rights management, the actual transfer over the NoC, and the acknowledgment sent back to the original sender. This entire pipeline is sufficient to implement safe and secure remote system calls in a distributed operating system.

In this context, security refers to the OS'es requirement to control all communication channels between processors, while safety ensures that sent messages arrive at their intended destination and that the order of requests and replies is

	Bandwidth [MB/s]		Latency [cycles]
Test	measured	theoretical	
on-chip	395	1592	193
chip-to-chip	45	47	306

TABLE II: Communication speed @100 MHz

Processing tiles	1	2	3	4	isolated
Scenario	Power [mW]				
Chip idle	39.0				
Tile enabled	41.6	44.8	48.0	52.8	3.4 / tile
Tile working	43.0	49.5	55.1	59.6	2.1 / tile
Chip-to-Chip enabled	43.2				4.2 / link
Chip-to-Chip transfer	46.9				3.7 / link

TABLE III: Power Consumption

preserved. Simulations conducted in [4] indicated that the round-trip time for a transfer without involving the software stack is roughly 100 cycles. This implies that each of the three components—the software stack, the TCU/NoC, and the C2C links—add about 100 cycles of latency.

When comparing these experimental results with the requirements outlined for the M^3 operating system in [16], which states that system calls with a duration of around 500 cycles are sufficient, it is evident that the presented hardware is more than capable of supporting the smooth operation of M^3.

Power consumption of the chip was measured across various use cases to isolate the power consumed by a processor, on-chip communication, and off-chip communication. As shown in Tab. III, simply powering on the chip without starting any processors consumes 39 mW. Enabling a processing tile (by opening the clock gate) increases the power consumption by approximately 3.4 mW per tile. Data transfer between tiles consumes an additional 2.1 mW per processor. Using the C2C communication links draws more power than operating a processing tile, primarily due to the LVDS drivers, which operate at significantly higher voltage and frequency levels. A single C2C link consumes 4.2 mW when idle, with an additional 0.3 mW consumed during data transfer. The minimal additional power consumption during data transfer can be attributed to the fact that, once activated, the Aurora protocol begins constant communication, i. e. it is sending messages back and forth regardless of whether actual data must be transferred.

C. Usability for Real-World Workloads

The aforementioned use case of a home router or NAS system might involve a chipset similar to the one used in a Synology Beestation. This NAS system is equipped with an RTD1619B chipset [20] that features four Arm Cortex-A55 cores clocked at 1.7 GHz, along with an additional GPU, 1 MB of cache, and components such as video accelerators and on-chip Ethernet controllers. The RTD1619B is designed using a 12 nm process and has a thermal design power (TDP) of 5 W. Although the performance of our presented chip, being the first iteration of this platform to be manufactured in silicon, does not yet match the performance of RTD1619B, it is possible to extrapolate the potential performance based on results from other silicon-proven designs. A previous chip implementation described in [21], which used the same technology, the exact same NoC, and a similar network interface unit (though without security features), demonstrated similarly low power consumption at a base frequency of 500 MHz. Assuming that the base frequency of our chip is scaled to 500 MHz, a cluster of four chips, each containing four RISC-V cores capable of executing 500M instructions per second, could achieve a combined total of approximately 8000M instructions per second. In comparison, the RTD1619B, with its four cores running at 1700M instructions per second each, achieves 6800M instructions per second in total. This comparison indicates that our platform would operate within the same order of magnitude in terms of computational performance. Based on the power consumption data from [21], it is reasonable to apply a power scaling factor of 10 to extrapolate from 100 MHz to 500 MHz. Therefore, the total power consumption of the envisioned chip cluster in milliwatt is estimated to be:

	idle		4x tile		2x C2C	
(	39	+	4(3.4 + 2.1)	+	2(4.2)	)
	scaling		# of chips		total	
*	10	*	4	=	2776	

Compared to the RTD1619B's TDP of 5 W, our envisioned chip cluster would operate within a similar power consumption range, indicating that our platform is competitive in terms of both performance and energy efficiency.

VI. Conclusion and Outlook

We presented Masur23 as the first silicon implementation of an MPSoC architecture that utilizes TCUs as core security components to enforce OS-provided policies and isolate untrusted software and hardware components. Notably, the TCU, which provides security for the entire chip, occupies only 10.7 % of the total area. Transfer workload tests have demonstrated that the platform is capable of running a distributed operating system like M^3. Furthermore, compared to a chip from a real-world application—lacking the security features introduced by our TCU—we have shown that Masur23 operates within a similar range of performance and power consumption, highlighting its competitiveness and efficiency.

As part of our future work, we plan to showcase real-world attack scenarios such as man-in-the-middle and side-channel attacks as well as the impact of malicious components like hardware trojans. Furthermore, to validate the effectiveness of the HW/OS isolation approach, we plan to extend the chip platform to support security features such as trusted execution environments and remote attestation.

VII. Acknowledgment

This research was partly funded by the German Federal Ministry of Education and Research under grant number 16ME0527. This research is also financed on the basis of the budget passed by the Saxon State Parliament in Germany. Furthermore, we thank Racyics GmbH for providing EDA tools and the flip-chip packaging solution as well as Contronix GmbH for PCB design and fabrication.

References

[1] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading Kernel Memory from User Space," *meltdownattack.com*, 2018. [Online]. Available: https://meltdownattack.com/meltdown.pdf.

[2] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre Attacks: Exploiting Speculative Execution," *meltdownattack.com*, 2018. [Online]. Available: https://spectreattack.com/spectre.pdf.

[3] F. Pauls, S. Haas, S. Köpsell, M. Roitzsch, N. Asmussen, and G. Fettweis, "On Trustworthy Scalable Hardware/Software Platform Design," in *Smart Systems Integration Conference and Exhibition (SSI)*, Apr. 2022.

[4] S. Haas and N. Asmussen, "A Trusted Communication Unit for Secure Tiled Hardware Architectures," in *2022 29th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2022, pp. 1–4.

[5] M. D. Grammatikakis, K. Papadimitriou, P. Petrakis, A. n. Papagrigoriou, G. Kornaros, I. Christoforakis, and M. Coppola, "Security Effectiveness and a Hardware Firewall for MPSoCs," in *2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS)*, 2014, pp. 1032–1039.

[6] B. Tan, M. Biglari-Abhari, and Z. Salcic, "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] F. Restuccia, A. Meza, and R. Kastner, "Aker: A Design and Verification Framework for Safe and Secure SoC Access Control," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–9.

[8] A. Basak, S. Bhunia, T. Tkacik, and S. Ray, "Security Assurance for System-on-Chip Designs With Untrusted IPs," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 7, pp. 1515–1528, 2017.

[9] H. Salem and N. Topham, "Trustworthy computing on untrustworthy and Trojan-infected on-chip interconnects," in *2021 IEEE European Test Symposium (ETS)*, IEEE, 2021, pp. 1–2.

[10] 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, Sep. 2008, ISSN: 0018-9340.

[11] J. Porquet, A. Greiner, and C. Schwarz, "NoC-MPU: A secure architecture for flexible co-hosting on shared memory MPSoCs," in *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition*, ser. DATE'11, Mar. 2011, pp. 1–4.

[12] J. Brunel, R. Pacalet, S. Ouaarab, and G. Duc, "SecBus, a Software/Hardware Architecture for Securing External Memories," in *2014 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, 2014, pp. 277–282.

[13] 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 *Proceedings of the 30th Symposium on Integrated Circuits and Systems Design: Chip on the Sands*, ser. SBCCI '17, Fortaleza, Ceará, Brazil: Association for Computing Machinery, 2017, pp. 59–64.

[14] S. Charles and P. Mishra, "Lightweight and Trust-Aware Routing in NoC-Based SoCs," in *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2020, pp. 160–167.

[15] T. Wehbe and X. Wang, "Secure and Dependable NoC-Connected Systems on an FPGA Chip," *IEEE Transactions on Reliability*, vol. 65, no. 4, pp. 1852–1863, 2016.

[16] N. Asmussen, S. Haas, C. Weinhold, T. Miemietz, and M. Roitzsch, "Efficient and Scalable Core Multiplexing with M^3v," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Lan uages and Operating Systems*, ser. ASPLOS'22, Lausanne, Switzerland, 2022, pp. 452–466.

[17] 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, 2016, pp. 189–203.

[18] H. Härtig, M. Hohmuth, J. Liedtke, S. Schönberg, and J. Wolter, "The performance of μ-kernel-based systems," *ACM SIGOPS Operating Systems Review*, vol. 31, no. 5, pp. 66–77, 1997.

[19] K. Asanović, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, S. Karandikar, B. Keller, D. Kim, and J. Koenig, "The Rocket Chip Generator," *EECS, University of California at Berkeley, Tech. Rep. UCB/EECS-2016-17*, 2016.

[20] GadgetVersus.com. "RealTek RTD1619B." (2024), [Online]. Available: https://gadgetversus.com/processor/realtek-rtd1619b-specs/ (visited on 08/27/2024).

[21] M. Hasler, S. Haas, R. Wittig, S. Scholze, A. Dixius, S. Höppner, G. Fettweis, and C. Mayr, "A Random Linear Network Coding Platform MPSoC Designed in 22nm FDSOI," in *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2022, pp. 217–222.