Journal of Computer Science and Technology Studies
ISSN: 2709-104X
DOI: 10.32996/jcsts
Journal Homepage: www.al-kindipublisher.com/index.php/jcsts

JCSTS
AL-KINDI CENTER FOR RESEARCH AND DEVELOPMENT

| RESEARCH ARTICLE

Secure Monitor Calls (SMCs) for AI-Centric Embedded Platforms: Bridging Software Flexibility and Hardware Control

Senthil Nathan Thangaraj
AMD, USA
Corresponding Author: Senthil Nathan Thangaraj, **E-mail**: snathan.tg@gmail.com

| ABSTRACT

Secure Monitor Calls (SMCs) are critical for AI-focused embedded platforms, acting as the interface between non-secure operating systems and secure firmware. ARM TrustZone enforces hardware isolation, allowing secure interaction between ARM cores, AI accelerators, and programmable logic. Modern embedded AI systems require more than processor-level control; they need advanced power, thermal, and resource management with deterministic behavior for safety. Two widely used interfaces rely on Secure Monitor Calls (SMCs): the ARM-standard PSCI for coordinating power states, and the Xilinx-AMD-specific EEMI for managing voltage, frequency scaling, and thermal limits. SMCs rely on register-based parameter passing to ensure secure, cross-platform communication. While platforms may add extensions for hardware differences, this model preserves portability and simplifies integration. Compared to other methods, it improves security, determinism, and system coordination. As systems evolve toward adaptive computing, SMC implementations must expand to support new AI workloads while remaining compatible with existing frameworks.

| KEYWORDS

ARM TrustZone, Embedded AI Platforms, Embedded Energy Management Interface, Power State Coordination Interface, Secure Monitor Calls, Trusted Firmware.

| ARTICLE INFORMATION

ACCEPTED: 01 November 2025 **PUBLISHED:** 21 November 2025 **DOI:** 10.32996/jcsts.2025.7.12.10

1. Introduction

ARM TrustZone is a key innovation in embedded systems, creating hardware-level isolation through secure partitions. It establishes two execution environments at different privilege levels, protecting sensitive operations from threats while maintaining performance [1]. Unlike software-only protection, TrustZone enforces secure state transitions and resource isolation at the hardware level.

Secure Monitor Calls (SMCs) act as the main communication link between the operating system and trusted firmware in AI-enabled platforms. These systems combine ARM cores with AI accelerators, which makes coordination of power, resources, and scheduling more complex [2]. Through SMCs, the non-secure world can access critical services such as power, voltage and frequency scaling, thermal control, and AI accelerator setup. These services adapt system behavior to changing workloads.

Efficient embedded AI design requires secure, deterministic communication between software layers. Traditional inter-processor methods often add latency or open security gaps, which is unacceptable in safety-critical use cases. The challenge is greater when handling diverse processors with different power, performance, and thermal needs [1].

Edge AI platforms highlight this shift toward adaptive computing. They combine general-purpose processors, AI engines, and programmable logic, demanding a careful balance of performance, power, and thermal control. Real-time AI workloads in

particular depend on deterministic responses [2]. Reliable communication protocols are essential to transfer parameters securely while preserving predictable performance and ensuring consistent system reliability overall.

This work studies SMC implementations for AI-based embedded systems. It explores strategies to optimize communication while maintaining security, analyzes latency in AI workloads, and examines payload handling for deterministic behavior. The findings provide guidelines for system architects to meet safety standards and reliability goals in AI-driven embedded platforms.

2. ARM Security Architecture and SMC Fundamentals

ARM security architecture defines privilege boundaries through exception levels. Each level provides different access rights: user applications run with the least privilege, while higher levels allow more control. The hypervisor tier enables virtualization with extended access, and at the top, the Secure Monitor manages state transitions and coordinates resources across domains [3].

Secure Monitor Calls (SMCs) handle transitions between secure and non-secure worlds using defined calling conventions. These conventions ensure secure and consistent parameter passing. When executed, SMCs raise execution privilege via synchronous exceptions while preserving context [3]. Registers identify functions and transfer parameters, avoiding memory-based handling that could create vulnerabilities. Strict adherence to these conventions prevents errors during state transitions.

The Trusted Firmware (TF-A) framework supports secure-world runtime services. It standardizes power management, system configuration, and platform-specific functions. Its modular design separates hardware-specific code from client interfaces, enabling vendor extensions while keeping compatibility [4]. The runtime also manages secure memory, interrupts, and context switching, ensuring isolation between secure services.

SMC service identifiers use a hierarchical namespace. This prevents conflicts across implementations and enables efficient routing. Encoding defines both service category and function [4]. Fast calls run atomically without interruption, while normal calls can yield to higher-priority tasks, balancing performance with flexibility.

Security of SMCs depends on strict input validation and state management. Malicious inputs must be filtered, and secure execution must resist exploitation [3]. Both the SMC interface and its services act as potential attack surfaces. Strong access control and secure coding practices are critical to preserve the integrity of the architecture.

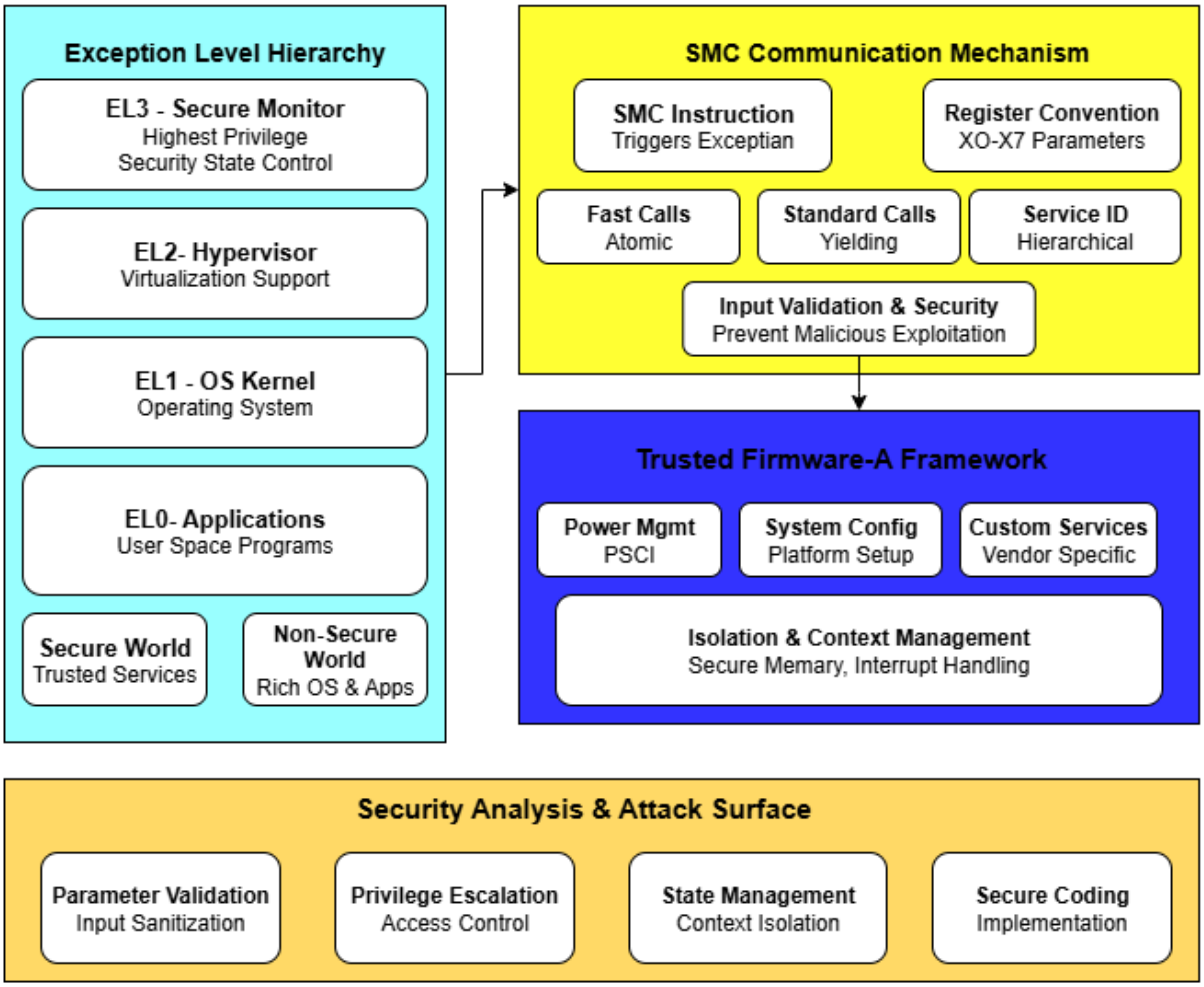

Fig 1: ARM Security Architecture & SMC Fundamentals [3, 4]

3. Power State Coordination Interface (PSCI) and Platform Extensions

The Power State Coordination Interface (PSCI) standardizes power control across ARM-based platforms. It defines common interfaces for managing power state changes between firmware and operating systems using Secure Monitor Calls (SMCs) [5]. This consistency enables portable power management across diverse embedded architectures and supports both immediate transitions and deferred requests for optimized power use.

PSCI provides domain-specific SMC functions to manage processor states such as deep sleep, wake-up, and dynamic core control. For example, CPU_SUSPEND, CPU_OFF, and CPU_ON use register-based parameters: X0 holds the function ID, while X1–X7 pass the power state, target CPU, and entry point. Suspend and resume operations preserve context, memory, and cache coherency during transitions. Hotplug APIs allow runtime scaling by powering cores on or off based on load.

In multi-core systems, PSCI uses SMC queries to track dependencies and coordinate state changes without race conditions. AFFINITY_INFO calls provide visibility into core status, while synchronization protocols manage shared caches, memory coherency, and interrupts, maintaining data consistency with minimal performance loss [6].

Vendors can add platform-specific extensions with custom SMC identifiers. These are used for advanced thermal control, heterogeneous domain management, or fine-grained wake-up policies. Extensions remain compatible with standard PSCI, preserving portability while supporting specialized hardware [5].

Integrating PSCI into legacy systems is challenging, as older kernels and bootloaders use different frameworks. Developers must map PSCI states to legacy equivalents, ensure drivers adopt PSCI calls, and handle mismatches across components. The goal is seamless operation that combines modern PSCI with legacy methods.

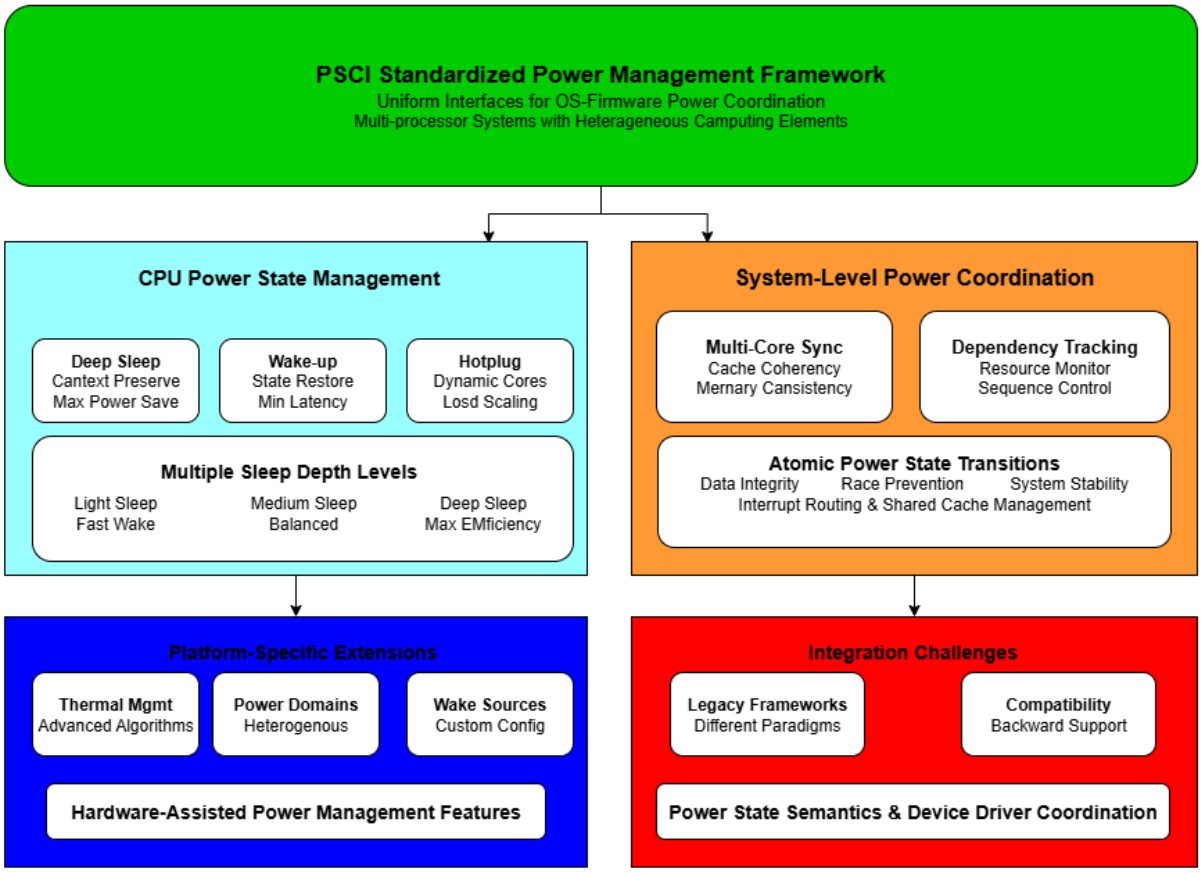

Fig 2: Power State Coordination Interface (PSCI) [5, 6]

4. Embedded Energy Management Interface (EEMI) Implementation

The Embedded Energy Management Interface (EEMI), developed by Xilinx, manages power in embedded SoCs by linking software on different processor cores with the power management controller. Communication happens through Secure Monitor Calls (SMCs), which securely request actions such as powering components on or off, adjusting performance, or saving energy.

Each EEMI API call maps to an SMC that uses ARM register conventions [11]. Requests first pass through Trusted Firmware-A (TF-A) in the secure world, which validates them before forwarding to the Platform Management Controller (PMC) via Platform Loader and Manager firmware. This ensures authenticated access while keeping processing clusters isolated.

EEMI provides detailed clock management. APIs like clock_enable, clock_disable, clock_setrate, and clock_getrate control clock operation and frequency. Advanced calls such as clock_setdivider and clock_setparent allow fine-tuned frequency scaling and reconfiguration of clock trees. All changes are authenticated before reaching the PMC, preventing unauthorized clock modifications [7].

Power and reset controls use dedicated SMCs. Commands like reset_assert and reset_get_status manage resets, while node control APIs handle access to PM slave devices with capability and QoS requirements. Functions such as self_suspend and request_suspend allow clusters to suspend themselves or request other clusters to power down securely [11].

Compared to memory-mapped methods, EEMI's SMC approach improves security by routing all platform control through trusted monitors. Standardized identifiers and parameter passing make it easier to integrate across platforms. Both blocking (REQUEST_ACK_BLOCKING) and non-blocking (REQUEST_ACK_NONBLOCKING) modes are supported, giving flexibility for real-time systems [8].

EEMI APIs are grouped into classes for suspend/wake operations, Power & reset management, clock management and pin control.. Suspend and wake calls coordinate timing across clusters, while Pin Control APIs (pinctrl_request, pinctrl_set_function, etc.) manage pin configuration securely with centralized validation.

Functional Area	Key APIs	Purpose
Architecture & Communication	Firmware APIs, OS APIs, SMC calls	Secure, layered control between clusters and PMC
Clock Control	clock_enable, clock_disable, clock_setrate, clock_getrate	Fine-grained clock and frequency management
Power & Reset Management	reset_assert, request_node, self_suspend	Secure power transitions and reset operations
Suspend & Wake	request_suspend, request_wakeup	Coordinated cluster suspend and wake control
Pin Control	pinctrl_request, pinctrl_set_function	Secure pin configuration with access validation

Table 1: Summary of Embedded Energy Management Interface (EEMI) Functions [7, 8, 11]

5. Standard SMC Implementation Considerations and Best Practices

Secure Monitor Calls (SMCs) use ARM's register-based parameter passing, which limits data exchange to registers X0–X7 defined by the ARM calling convention [3]. X0 holds the function ID, while X1–X7 carry parameters, with return values using the same scheme [4]. This uniform model simplifies integration across ARM platforms, ensures portability, and maintains TrustZone security guarantees. It also provides deterministic timing, which is critical for real-time systems.

By avoiding memory-based transfers, this approach reduces complexity and minimizes security risks. However, strict input validation is essential. Services must perform bounds checking and type validation to block malicious requests while keeping performance efficient [3]. The limited parameter space also narrows the attack surface compared to more complex communication methods.

SMC services benefit from modular firmware design, keeping service logic separate from communication mechanisms. This improves maintainability, testing, and compliance with ARM specifications. The architecture supports both fast calls (atomic execution) and normal calls (yielding to higher-priority tasks), allowing flexibility for services with different timing needs and adapting seamlessly to diverse embedded system requirements.

Portability depends on strict adherence to ARM conventions and service identifier rules. Following these standards ensures compatibility with existing tools, software reuse across platforms, and reliable integration in AI-focused embedded systems [3].

6. Comparison with Alternative Communication Methods

While Secure Monitor Calls (SMCs) are widely adopted, embedded platforms have long used other methods for communication between software and hardware controllers. The most common are memory-mapped I/O, mailbox protocols, and shared-memory schemes. Each offers certain benefits but falls short when compared with SMC-based approaches.

- **Memory-Mapped I/O (MMIO):**
  MMIO gives software direct access to hardware registers through fixed memory addresses. It is simple and works well for straightforward control operations such as toggling GPIO pins or accessing timers. However, it bypasses the secure world entirely, leaving hardware exposed to untrusted software. In safety-critical domains like automotive, this lack of isolation increases the risk of faults or malicious attacks.

- **Mailbox Protocols:**
  Mailboxes use message-passing between processors or between secure and non-secure domains. They allow larger payloads and asynchronous communication, making them popular in multi-core DSP systems. The drawback is added latency and synchronization complexity. For instance, a thermal event notification sent via mailbox may experience variable delays, making it unsuitable for real-time AI workloads where predictable responses are required.

- **Shared-Memory Communication:**
  Shared memory supports high-throughput data exchange, ideal for tasks such as video frame transfers. However, it demands careful locking to avoid race conditions and buffer corruption. If memory is not tightly isolated, attackers could tamper with shared regions, leading to unpredictable system behavior. This trade-off makes shared memory risky for security-sensitive control operations.

Compared to these methods, **SMCs deliver three clear advantages**:

1. **Security** – All requests are mediated by trusted firmware, protecting critical hardware resources.
2. **Determinism** – Register-based calls guarantee predictable timing, vital for real-time applications like ADAS.
3. **Portability** – Standardized conventions work across ARM platforms, lowering development and integration costs.

Although MMIO, mailboxes, and shared memory remain useful in certain high-bandwidth or non-critical contexts, SMCs are the preferred option when systems must balance strong security with real-time performance. This explains why modern AI-centric embedded platforms increasingly rely on SMCs as their primary communication mechanism.

7. Performance Analysis of SMC Implementations

The efficiency of Secure Monitor Calls directly affects system responsiveness, especially in AI workloads that demand real-time performance. Unlike memory-mapped I/O or shared-memory communication, SMCs provide lower and more predictable latency because all parameters are exchanged through processor registers. This reduces overhead and simplifies synchronization across secure and non-secure domains.

Latency measurements show that SMC calls typically add only a few hundred nanoseconds to a few microseconds of overhead, depending on the platform and complexity of the service. For AI inference tasks, where decisions must be made in milliseconds, this overhead is negligible compared to the benefits of secure, deterministic communication.

Throughput is another important factor. In multi-core and heterogeneous systems, the ability to handle many concurrent requests without contention is critical. Standardized SMC implementations support fast calls for atomic operations and normal calls for longer tasks. This design ensures that lightweight services do not block more complex operations, thereby improving both efficiency and overall system scalability.

Performance is also tied to workload type. In automotive and robotics platforms, deterministic timing ensures compliance with safety standards. In telecom and 5G systems, higher throughput is more important, and SMCs provide predictable coordination of power and thermal resources across diverse accelerators.

Finally, SMC performance depends on proper validation and minimal context-switch overhead. Optimized firmware implementations, such as Trusted Firmware-A (TF-A), reduce latency by streamlining secure-world service handling. Benchmark studies consistently show that well-designed SMC frameworks achieve a balance of low overhead, strong security, and portability across ARM-based platforms.

8. Case Studies and Practical Implementations

Automotive Systems (ADAS Use Case): In Advanced Driver Assistance Systems (ADAS), timing and safety are critical. Features such as automatic emergency braking, lane detection, and collision avoidance all rely on AI accelerators working alongside ARM cores. Any delay in communication between operating systems and secure firmware could cause unpredictable behavior.

SMCs solve this problem by providing deterministic, low-latency communication. For example, when an ADAS system detects a potential collision, the AI accelerator must quickly increase processing frequency to analyze sensor data in real time. Through SMCs, the operating system securely requests a voltage and frequency scaling (DVFS) operation from trusted firmware. The request is validated and executed by the secure world with guaranteed timing, ensuring that the accelerator runs at the required performance level without delay.

At the same time, thermal safety is maintained. If the system is running hot, SMCs coordinate with the power controller to balance performance and temperature without exposing direct hardware control to non-secure software. This prevents both overheating and unsafe throttling. Because all interactions are managed through SMCs, developers can be confident the system meets ISO 26262 requirements for functional safety, including predictable timing and safe state transitions under varied and demanding real-world driving conditions.

Other Implementations: In 5G and telecom SoCs, SMCs coordinate power and thermal resources across ARM cores, DSP engines, and accelerators. Adaptive platforms like Xilinx Versal AI Edge use SMCs via EEMI for fine-grained power and clock control of programmable logic. Similarly, NVIDIA Jetson and Qualcomm Snapdragon platforms rely on SMCs to unify secure management of CPUs, GPUs, and AI accelerators, improving portability and simplifying development.

Together, these examples show how SMCs are more than a theoretical mechanism. They are essential for enabling secure, deterministic, and efficient operation in AI-driven embedded systems.

9. Conclusion

Secure Monitor Calls (SMCs) are vital in AI-focused embedded platforms. They provide secure and portable communication between software layers, with hardware-enforced isolation, deterministic timing, and better integration than memory-mapped schemes.

Modern platforms extend SMCs to handle complex parameters and larger data transfers, enabling advanced control algorithms for heterogeneous systems. Future work will focus on higher bandwidth, lower latency, and broader standardization to support AI accelerators and adaptive computing.

For embedded engineers, key considerations include strict interface validation, backward compatibility, and performance testing. These practices ensure reliable operation under varying conditions. With their predictable timing and strong security, SMCs are especially suited for safety-critical applications like ADAS.

Beyond automotive and industrial domains, SMCs are expected to play a growing role in 5G, robotics, and edge-to-cloud systems, where secure state management, memory isolation [9], and compliance with information security frameworks [10] are equally critical. By combining hardware isolation with standardized firmware, SMCs will remain a cornerstone of secure and adaptive embedded computing.

Funding: This research received no external funding.
Conflicts of Interest: The authors declare no conflict of interest.
Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors, and the reviewers

References

[1] AMD, (2021) Versal ACAP System Software Developers Guide (UG1304), 2021. Available: https://docs.amd.com/r/2021.1-English/ug1304-versal-acap-ssdg/Revision-History
[2] ARM Developer, (n.d) Arm Architecture Reference Manual for A-profile architecture. Available: https://developer.arm.com/documentation/ddi0487/latest/
[3] ARM Developer, (n.d) Arm Power State Coordination Interface Platform Design Document. Available: https://developer.arm.com/documentation/den0022/latest/
[4] ARM Developer, (n.d) ARM System Memory Management Unit Architecture Specification - 64KB Translation Granule Supplement,. Available: https://developer.arm.com/documentation/ihi0067/latest/
[5] IEEE-SA Standards Board, (1998) IEEE Standard for Software Verification and Validation, 1998. Available: https://people.eecs.ku.edu/~hossein/Teaching/Stds/1012.pdf
[6] ISO, (n.d) ISO/IEC 27001:2013(en). Available: https://www.iso.org/obp/ui/#iso:std:iso-iec:27001:ed-2:v1:en
[7] Linux Kernel Community, (n.d) Power Management. Available: https://www.infradead.org/~mchehab/kernel_docs/admin-guide/pm/index.html
[8] Robert P et al., (2017) Secure Edge Computing with ARM TrustZone, ScitePress, 2017. Available: https://www.scitepress.org/papers/2017/63086/63086.pdf
[9] Trusted Firmware, (n.d) Trusted Firmware-A Documentation. Available: https://trustedfirmware-a.readthedocs.io/en/latest/
[10] Xilinx Inc., (n.d) Versal™ AI Edge Series. Available: https://www.xilinx.com/content/dam/xilinx/publications/solution-briefs/xilinx-versal-AI-edge-product-brief.pdf
[11] XILINX, (2020) Embedded Energy Management Interface (v1.0), 2020. Available: https://docs.amd.com/v/u/en-US/ug1200-eemi-api