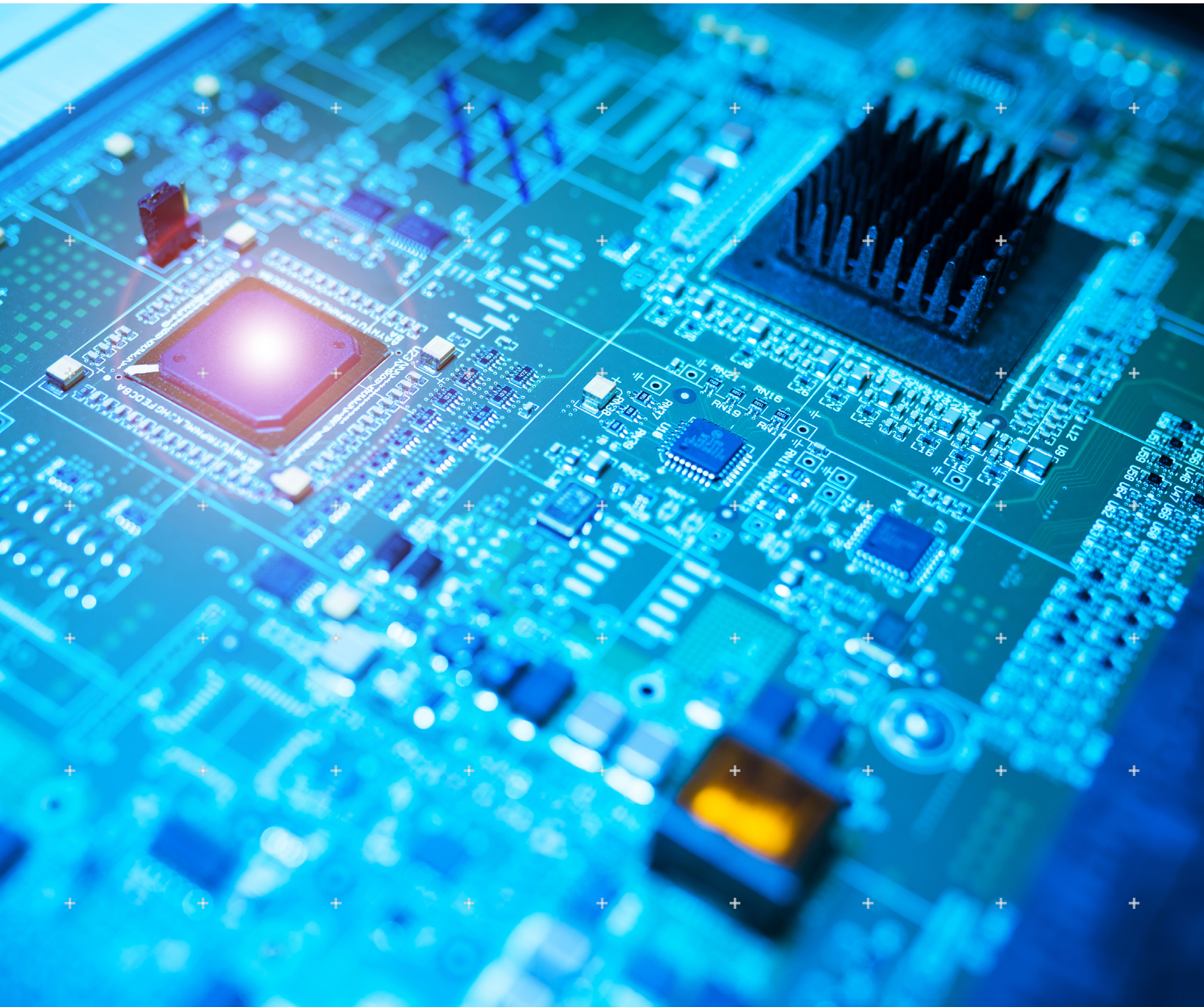
Power and Performance Management using Arm® SCMI Specification

White Paper



Abstract

This white paper provides an overview of the Arm System Control and Management Interface (SCMI) specification. It describes how an SCMI-enabled system optimizes power and performance management through a combination of abstraction and division of responsibilities between the operating system and a system controller.

Introduction

Providing efficient power and performance management through an interface that an operating system (OS) can use is a huge challenge. There are a wide variety of Arm-based system-on-chip (SoC) designs as vendors strive for differentiation. This variety creates a huge amount of complexity for the OS developer, as there are too many systems to manage in a single, generic kernel. Different SoCs use different methods to achieve similar aims in power and performance management, and it is difficult for an OS developer to judge how to deploy the different controls available. As newer, high performance processors and other compute engines such as accelerators or GPUs evolve, there is an increasing variety of these controls. There is great complexity to manage for not only OS developers, but also for silicon vendors who must provide support for multiple operating systems.

Arm recently introduced the Arm System Control and Management Interface (SCMI) [1] specification. The SCMI specification describes an OS-agnostic generic interface for power and performance management. This white paper details how an SCMI-enabled system offers improved overall system management in modern SoCs and provides a simple and effective solution for stakeholders such as OS developers and silicon vendors. This white paper is organized into five sections:

Section 1 provides a brief overview of the SCMI specification and its key aspects. Section 2 describes recent challenges in SoC designs, and provides motivations for platform abstraction. There is also a description of how platform abstraction can help achieve improved power and performance management through delegation of responsibilities to a system controller. Section 3 provides an overview of power and performance management frameworks in the Linux Operating System, and the role that SCMI plays in enabling these frameworks. Section 4 describes other standards in the industry that are commonly deployed, and SCMI's interoperability with these standards. Section 5 provides a summary.

1. Overview of SCMI

The SCMI specification provides a standardized interface for power, performance, and resource management on a SoC. The low-level management actions are performed by a system controller that directly controls the SoC hardware or *platform*. The system controller provides SCMI interfaces to its clients. A typical example of an SCMI client is an operating system kernel running on the Application Processors (AP). The SCMI specification refers to clients as *agents*. This document refers to the system controller as the *System Control Processor (SCP),* and the code it runs as the *SCP firmware* [2] or, simply, *the firmware*.

The SCMI interface comprises a set of *protocols*. Agents communicate with the SCP using these protocols. The SCMI specification defines protocols for standard platform services such as power management, performance management, and reading SoC sensors. Each protocol comprises a set of *messages*. Messages that are issued by agents to request services are called *commands*. An example of a command is a request to set the power state of a power domain. The SCP can also send messages to agents. These messages are called *notifications*. Notifications are typically used to inform the agent of a system-level event. Messages are exchanged over an underlying *transport*. **Figure 1** provides a broad overview of an SCMI-based system.

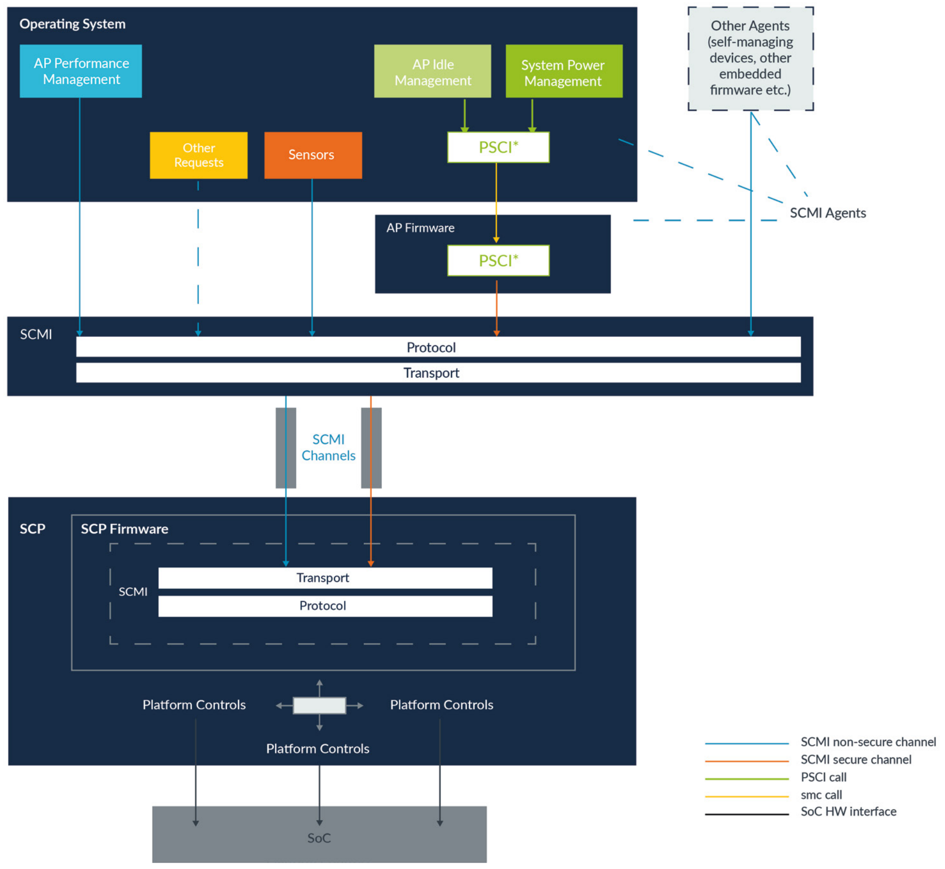


Figure 1: High-level overview of an SCMI-enabled System.
*NOTE: PSCI is described in detail in [10]

2. Motivation

Arm systems have an embedded legacy which has led to designs where the OS kernel is the only entity that manages the platform. The rationale for this is that the OS was the only entity in the system that could manage the use of resources based on the behavior and needs of workloads that it runs. Consequently, the OS directly controlled the platform's power and performance hardware to match the demands of the workload.

However, it is hard to create and maintain a single, generic OS kernel that can support a wide variety of systems from multiple vendors, as each system has its own implementation-specific power, performance and system resource control mechanisms.

In addition, modern SoCs can involve complex constraints arising from hardware and environmental conditions. For example, the platform can be subjected to thermal throttling if the temperature exceeds a threshold. These constraints impose additional safety and security requirements which might be too complex for the OS to manage while simultaneously attempting to optimize workload performance.

This gives rise to the motivation in the industry to break away from the traditional OS-based, centralized control paradigms, in favor of a delegation-based model. In the new model, the SCP firmware has sole access to the platform power and performance hardware. As before, the OS continues to be responsible for assessing the performance needs of workloads and translates them to desired performance levels. The OS sends requests to the SCP, which in turn sets the selected performance levels in the hardware. Independently from the OS, the SCP can autonomously monitor the platform for conditions that warrant enforcement of constraints. The SCP configures a net performance level that best satisfies the requests from the OS whilst meeting any imposed constraints.

Abstraction

Workload performance is typically equated to CPU clock frequency, and a corresponding voltage to drive the clock at the selected frequency. A CPU core or cluster of CPU cores can operate at one or more {frequency, voltage} pairs. Each such pair is called an Operating Performance Point (OPP), and the related performance management scheme is referred to as Dynamic Voltage and Frequency Scaling (DVFS) or Dynamic Clock and Voltage Scaling (DCVS). This framework allows workload performance to be set and measured in terms of distinct and discrete OPPs. This requires the OS kernel to possess full knowledge of the OPP values that the platform supports, and recognize how they are read and set. Each SoC offers different methods for reading and setting OPPs. Furthermore, some SoCs offer additional means of controlling performance, which have similar power and performance tradeoffs, such as CPU throttling.

An alternative to the OPP-based DVFS approach is to view performance as a continuum of performance levels in an abstract, linear scale. The SCP can map individual OPPs to equivalent performance levels in this scale, and then exposes the performance scale and levels to the OS. The DVFS framework in the OS can then be built on top of this abstract scale making the OS code platform-agnostic. This facilitates kernel code development, generalization, distribution, maintenance, and porting. Specifically:

- A generic, platform-agnostic OS kernel can be maintained, which works seamlessly on different platform implementations.

- Complex platform-specific constraints that are not visible to the OS kernel can be enforced and handled by the SCP. One example is limiting the availability of higher performance levels under thermal or electrical constraints.
- Platform-specific tuning and differentiation can be implemented in firmware instead of the generic OS kernel, allowing for overall improvement in performance without modifying the OS kernel.

Safe Hardware Operation

A challenge in SoC design is providing cost-effective power supplies that can meet peak power demands of all the compute engines within the SoC. To achieve this, the power supplies are not always provisioned for maximum possible current draw. Therefore, hardware and firmware solutions are required to manage current consumption. Another common contingency that requires mitigation in real-time is thermal excursion.

Table 1 shows the typical reaction times of hardware, firmware and kernel solutions in reacting to platform contingencies.

Table 1: Typical reaction times of various components in the system

<100uS	<1mS	<1ms (often 10s of ms)
Hardware	Firmware	OS Kernel

Some power delivery issues require sub 100us reaction times and can only be solved in hardware. There are also other power delivery issues that require less stringent reaction times but are still in the sub 1 ms range. For such issues, the kernel is too slow. Moreover, the kernel might not even have visibility of all the factors that affect the safe operation of the system when these problems do arise. As a result, these issues can be mitigated by the SCP firmware. The SCP has a better reaction time than the kernel and can thus provide platform safety guarantees. The SCP can also have full visibility of the current state of the SoC hardware, which allows it to make optimized decisions for resolving specific issues. This approach maintains the simplicity of the kernel implementation and keeps it platform-agnostic.

Security

The *CLKsCREW* attack [3] exposed security vulnerabilities in energy management implementations where untrusted software had direct access to clock and voltage hardware controls. In this attack, the malicious software was able to place the platform into unsafe overclocked or undervolted configurations. Such configurations then enabled the injection of predictable faults to reveal secrets.

Many Arm-based systems today implement voltage regulator and clock frameworks in the kernel. These frameworks allow callers to independently manipulate frequency and voltage settings. Such implementations can render systems susceptible to this form of attack.

Attacks such as CLKsCREW can be mitigated if the following requirements are met:

- The kernel must not have direct and independent control of clock and voltage.
- A trusted entity, such as the SCP firmware, must be able to perform sanity checking on the requested performance levels, thereby preventing any attempted malicious programming.

These requirements can be met with a kernel implementation that delegates the management of these controls to the platform through an abstracted interface. SCMI enables such an implementation.

3. OS Power management using SCMI

Introduction

Using the Linux OS as an example, this section describes how SCMI supports frameworks such as *Energy Aware Scheduling (EAS)* [4], and *Intelligent Power Allocation (IPA)* [5]. These frameworks enable OS task scheduling and thermal management based on power consumption feedback. SCMI enables these frameworks by supplying:

- Information that aids the discovery of power and performance domains in the platform.
- Commands for configuring performance levels and power states at runtime.
- An interface that the OS can use to obtain sensor information from the SCP.

Figure 2 illustrates how an SCMI-aware, IPA governor implementation can use the SCMI specification. Firstly, the governor utilizes the SCMI sensor protocol interfaces to obtain

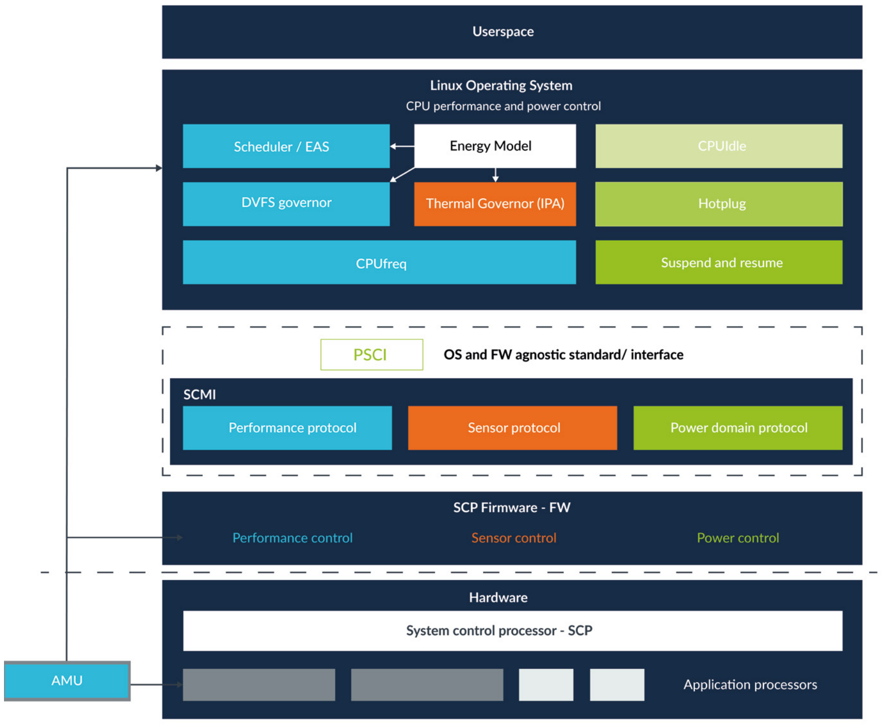


Figure 2: Linux power and performance management architecture and SCMI

instantaneous temperature data of the SoC to compute the current thermal headroom. It then converts the thermal headroom into a power budget, in accordance with the core IPA control algorithm. Next, by combining utilization and priority information, it determines how available power budget must be divided between the domains that allow DVFS control. The domain power budgets are then converted to available performance levels using power costs provided by the SCMI Performance Management protocol. If the current performance level of a domain exceeds the budget, it is restricted to a calculated maximum performance level. This level is then requested through the SCMI Performance Management protocol.

Energy model

The EAS implementation in Linux kernel [6] gives the scheduler the ability to predict the impact of its decisions on the energy consumed by the CPUs. Like IPA, EAS relies on CPU power and performance information to select an energy-efficient CPU for each task, with a minimal impact on throughput. Since there are multiple consumers of this information in the kernel, Linux has an Energy Model (EM) framework [7] for holding the information.

The EM information can be delivered through description data such as a device tree. As an alternative, an SCMI-enabled SCP can provide direct power information though a standard SCMI protocol interface. The interface enables discovery of the available performance levels for each performance domain in the platform, and their associated power costs. The power cost can be expressed in milliwatts or in an abstract linear scale. SCMI thus enables vendors and kernel developers to work towards maintaining generic, portable, and platform-agnostic power-management frameworks in the OS kernel.

Performance levels

In Linux, the `cpufreq` governor decides the performance level needed for each CPU performance domain. Similarly, `devfreq` governors decide the performance levels needed for device performance domains. These decisions are based on characteristics of the system and the governors' own internal policies.

For example, in the case of `cpufreq`, the governor executes its internal policy to select an appropriate performance level for a given performance domain. When implemented with SCMI support, the `cpufreq` driver invokes the SCMI Performance Management protocol, and in turn, the SCP sets the selected performance level.

As already noted in the preceding sections, the SCP can only make a best-case effort to fulfil the request. In most cases the SCP delivers the requested performance. However, there might be prevailing conditions that cause the SCP to enforce constraints and modify the OS request. With high-performance CPUs, the hardware and firmware are better equipped for managing platform constraints because they can provide mitigation in real-time to ensure that the platform always operates within safe limits.

Performance Feedback

Operating systems need to understand what performance is being delivered by a platform at any point in time. Firmware or hardware can provide this feedback in different forms. In the firmware-based approach, the SCP can send SCMI performance level change notifications. The OS can register for these notifications to become aware of a change in the performance level of a domain that the OS itself did not request. SCMI allows the SCP to report the identity of the agent that caused this performance change. This helps the OS become aware of performance level changes that were caused by platform constraints. However, notifications can be disruptive since they are delivered as interrupts to the OS. In addition, SCMI also supports providing current performance information through a statistics table.

In the hardware-based approach, free-running performance counters can be deployed in hardware. For instance, the Armv8.4-A architecture introduces the Arm Activity Monitors Extension (AMU) [8]. This optional extension can be used to obtain performance feedback. The AMU feature provides activity monitors that track delivered CPU clock frequency. The AMU also provides additional counters for measuring instructions retired and memory stall cycles. The OS can collect and correlate all this information to gain full insights into delivered CPU performance. The Activity Monitors Extension provides a system register interface to the activity monitors counters. It also supports an optional memory-mapped interface that allows the SCP to directly access performance information from the CPUs.

4. SCMI and ACPI

ACPI is an industry standard that provides interfaces for power and performance management [9]. In particular, ACPI provides platform abstraction that allows for generalization of ACPI-based operating systems.

Although used extensively in other markets, ACPI is not commonly deployed in mobile and embedded. For these markets, SCMI can provide platform abstraction, thus allowing for generalization of the OS kernel. Though the abstractions differ, SCMI and ACPI have a high degree of compatibility. Therefore, it is possible to provide SCMI-based SCP implementations that support ACPI based operating systems.

5. Conclusion

This white paper introduces the SCMI specification, which provides platform-agnostic interfaces for power, performance and system management in a SoC. The SCMI interfaces allow SoC vendors to provide a single firmware solution that supports multiple OS kernels, and OS vendors to develop a platform-agnostic OS kernel that runs seamlessly on multiple platforms. This contrasts with the current situation in the Arm ecosystem, where the SoC vendors must develop a custom firmware for each supported OS kernel, and where the OS kernel must be customized to support each SoC. The white paper presents key motivations and industrial trends leading to the requirement for delegation of performance and power management duties to a system controller running an SCMI-aware firmware. Finally, the white paper provides illustrative examples based on the Linux kernel power and performance management frameworks to showcase SCMI supporting them in a standardized manner.

Arm recommends that silicon designers, OS vendors, and kernel developers contributing to the Arm ecosystem adopt the SCMI-based design approach outlined in this white paper. For further information on SCMI, please see the Arm SCMI specification [1].

6. References

[1] **Arm® System Control and Management Interface.**

[2] SCP firmware.

https://github.com/ARM-software/SCP-firmware

[3] *CLKSCREW: Exposing the Perils of Security-Oblivious* Energy Management, A. Tang, S. Sethumadhavan, and S. Stolfo, Columbia University, Proceedings of the USENIX Security Conference, 2017.

https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/tang

[4] Energy Aware Scheduling (EAS).

https://developer.arm.com/tools-and-software/open-source-software/linux-kernel/energy-aware-scheduling

[5] Intelligent Power Allocator (IPA).

https://developer.arm.com/tools-and-software/open-source-software/linux-kernel/intelligent-power-allocation

[6] Energy Aware Scheduling: Linux kernel implementation.

https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/Documentation/scheduler/sched-energy.txt?h=v5.1.6

[7] Energy models of CPUs and the Energy Model Framework.

https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/Documentation/power/energy-model.txt?h=v5.1.6

[8] Arm® Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile.

https://static.docs.arm.com/ddi0487/da/DDI0487D_a_armv8_arm.pdf?_ga=2.64170153.706104037.1551267858-1587319438.1538587147

[9] Advanced Configuration and Power Interface, Version 6.3.

https://uefi.org/specifications

[10] Arm ® Power State Coordination Interface, Version D.

https://developer.arm.com/architectures/system-architectures/software-standards/psci