International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Advanced Configuration and Power Interface (ACPI): A Cross-Layer Architecture for SoC Verification, Firmware Engineering, and Datacenter Power Management

Suri Babu Talla

Abstract: Managing power and heat across modern computing systems is not a problem any single layer of the stack can solve on its own. Silicon design, firmware, operating system software, and data center infrastructure each carry a piece of the responsibility, and without a shared contract between them, the result is fragmentation that wastes energy and limits portability. This article examines how the Advanced Configuration and Power Interface establishes that contract and maintains it across a range of platforms. The discussion covers ACPI architecture, power and performance state definitions, thermal management, pre-silicon verification, firmware implementation, and datacenter-scale orchestration, with case studies drawn from mobile SoCs, enterprise servers, and accelerator-dense racks. The article closes by examining how AI-driven optimization, chiplet-native design, and cloud-native deployment requirements will shape how ACPI develops in the years ahead.

Keywords: *ACPI, Datacenter Power Management, Firmware Engineering, SoC Verification, Thermal Management*

1. Introduction

Modern computing systems carry an integration problem that has grown harder to ignore as hardware has become more complex. The shift from single-core processors to heterogeneous multi-socket servers with integrated accelerators, memory controllers, and chiplet fabrics did not come with a corresponding shift toward unified management interfaces; instead, each hardware vendor built its own approach to power state transitions, thermal throttling, and device discovery [1]. Operating systems had no reliable way to query hardware capabilities or apply consistent power policies across platforms that were built differently. Firmware developers worked within the boundaries of their own platforms, producing power management routines that solved immediate problems but did not carry over to other system architectures. The cumulative result was a landscape of incompatible implementations, rising development costs, and energy optimization opportunities that went unrealized because no common abstraction existed to coordinate them.

At datacenter scale, the consequences of that fragmentation become more severe. A fleet of millions of servers spanning multiple processor architectures, accelerator types, and memory configurations cannot support coordinated rack-level power budgeting, cross-hardware thermal orchestration, or predictable workload scheduling unless there is a shared interface through which those activities can be directed [2]. Without one, silicon verification teams must validate power state transitions, interrupt semantics, and topology reporting against no formal specification. Firmware engineers implement power control methods with no guarantee those methods will hold across OS versions or processor generations. Datacenter operators are left enforcing power policies manually across hardware that was never designed to respond to a common set of instructions.

The Advanced Configuration and Power Interface was developed to address exactly this problem. By defining a set of standardized firmware tables and control methods, ACPI gives operating systems a device-independent way to discover hardware,

NXP, USA

manage power states, interpret thermal conditions, and apply performance policies regardless of what sits underneath [1]. The architecture is table-driven and extensible, which means it can accommodate new hardware capabilities without breaking compatibility with existing software. This article traces how ACPI resolves the fragmentation problem through that architecture, how verification methodologies confirm compliance across silicon platforms, how firmware engineering puts ACPI into practice reliably, and how datacenter operators use it to coordinate power and thermal management across heterogeneous fleets at scale [2].

2. Evolution of ACPI

Before ACPI existed, power management on personal computers was a fragmented landscape. Each manufacturer handled it differently, and the approaches rarely worked across hardware from different vendors. The Advanced Power Management specification that preceded ACPI left most decisions inside the firmware, giving the operating system little visibility into what was happening and even less ability to influence it. Plug and Play BIOS added some device discovery, but nothing tied power and thermal management together under a common framework.

ACPI 1.x and 2.x, which arrived in the late 1990s and carried through the early 2000s, replaced both of those approaches with a single table-driven architecture [1]. For the first time, firmware could express complex control logic through ACPI Machine Language and ACPI Source Language in a form that any compliant operating system could read and act on, regardless of who built the hardware underneath [2].

When multi-core processors moved into mainstream use during the 2000s, ACPI 3. x extended the specification to handle them. Individual cores within a processor, and multiple processors within a single machine, could each be managed on their own terms while the system as a whole stayed coherent [2]. ACPI 4.x followed with hot-plug support, removing the requirement to restart when adding or removing devices, and expanded thermal management to cover multiple cooling zones with responses that scaled with temperature rather than switching abruptly.

ACPI 5.x tackled a different problem in the early 2010s, namely how to describe power management on architectures beyond x86. Adding ARM support brought the specification into mobile and embedded systems, where low-power idle states had previously required entirely separate management approaches [6]. The same interface that governed servers and desktops could now describe the power behavior of a mobile device.

ACPI 6.x, arriving in the mid-2010s and continuing forward, addressed the growing diversity of compute hardware [2]. As GPUs, AI accelerators, and purpose-built processors became standard components, the specification grew to describe their power domains and thermal characteristics alongside conventional CPUs. Virtualization support added another layer, giving hypervisors a way to present virtual hardware to guest operating systems while retaining control over the physical resources beneath [6].

What the version history shows is that each revision solved a coordination problem the previous one could not reach and did so by extending the existing standard rather than replacing it, keeping older systems compatible while opening the door to new capabilities [2].

ACPI Version	Key Features	Impact on Platforms	Notes
Pre-ACPI (APM, PnP BIOS)	Fragmented power management, limited OS visibility	Vendor-specific routines, poor portability	OS lacked control over thermal/power states
ACPI 1.x/2.x	Table-driven architecture, AML/ASL methods	Unified discovery and control across hardware	First major standardization effort
ACPI 3.x	Multi-core processor support	Per-core management, coherent system control	Enabled scalable CPU power states

ACPI 4.x	Hot-plug support, scalable thermal zones	Dynamic device addition/removal, improved cooling	Reduced downtime, flexible thermal responses
ACPI 5.x	ARM architecture support	Mobile/embedded integration	Unified servers, desktops, and mobile devices
ACPI 6.x	GPUs, accelerators, virtualization	Heterogeneous compute domains	Extended to modern datacenter workloads

Table 1. Evolution of ACPI Versions [1, 2, 6]

3. ACPI Architectural Overview

3.1 OS-Directed Power Management (OSPM)

Approaches such as APM concentrated decision-making authority in firmware, which had no visibility into what the system was actually doing or what the user needed. ACPI inverts that relationship through the OS-Directed Power Management model, shifting authority to the operating system while giving firmware the mechanisms it needs to describe what the hardware supports and how transitions must be executed [5]. The contract between the two sides is precise. Firmware documents the available power states, the registers that govern transitions between them, and the exact sequence of operations each transition requires. The OS reads those descriptions from ACPI tables, monitors workload and thermal conditions, and calls the appropriate AML methods when a transition is warranted. Firmware is no longer in the business of deciding when the system should sleep; it is responsible for executing the sleep entry sequence correctly when asked to do so [5]. That division of responsibility removes a coordination bottleneck that had made platform power management difficult to scale. Hardware teams can tune for low-power operation without tracking application behavior. OS teams can build and refine power policies without reverse-engineering hardware implementation details. Hardware designers can document their constraints once and trust that any compliant firmware and OS stack will honor them [5].

3.2 ACPI Tables

Embedding hardware capability information in OS driver source code creates a maintenance problem: every new platform requires OS changes, and the OS must be rebuilt to support hardware its developers have not yet seen. ACPI solves this by placing that information in firmware tables that the OS reads at runtime, making capability discovery a dynamic process that works across any compliant platform without modification [2]. The Root System Description Pointer gives the OS its starting point, directing it to the root or extended system description table, which holds pointers to every other ACPI table the firmware provides. The Fixed ACPI Description Table records the locations of power management registers and the availability of specific hardware mechanisms. The Differentiated System Description Table carries the platform-specific AML control methods, and Secondary System Description Tables let firmware extend the description with platform-specific additions without touching the core DSDT. Processor topology reaches the OS through the Multiple APIC Description Table, which lists every processor alongside its local APIC address so the OS can determine how many processors exist and how they are arranged. Memory affinity and access latency in NUMA systems are communicated through the System Resource Affinity Table and System Locality Information Table, giving the scheduler the information it needs to place threads close to the memory they use [2, 4]. IO remapping, debug interfaces, and security module descriptions extend the same framework to additional platform elements, and because every structure follows a standardized format, any compliant OS can interpret any compliant platform without custom adaptation [2, 4].

3.3 AML and ASL

Hardcoding platform-specific hardware procedures into an OS kernel creates exactly the portability problem that ACPI is designed to avoid. The solution is to let firmware express complex control logic in a standardized bytecode that any compliant OS can execute without understanding the underlying hardware [2]. Firmware engineers write control

methods in ACPI Source Language, a high-level syntax that resembles conventional imperative code. Those methods are compiled into ACPI Machine Language bytecode and stored in firmware, where the OS ACPI subsystem interprets them at runtime.

A power-down sequence that requires GPIO pins to toggle in a defined order, voltage rails to stabilize, and interrupts to be masked before the sequence completes can be written once as an AML method. The OS invokes it by name without needing any knowledge of the hardware it controls [2]. Thermal zone logic follows the same pattern. Within AML, the firmware engineer defines which temperature sensors map to which cooling devices, sets the trip points at which fan speed adjustments or frequency reductions are triggered, and ties those responses directly to the sensor readings that cross the defined threshold. This description encodes the full feedback loop between the thermal behavior of the hardware and the power management layer of the operating system in one place, so the OS requires no platform-specific logic of its own to participate in thermal control [4].

Component	Description	Role in System	Example Use
OS-Directed Power Management (OSPM)	OS controls transitions; firmware documents states	Separation of responsibilities	The OS decides the sleep entry, and the firmware executes the sequence.
ACPI Tables	Structured firmware data for OS discovery	Dynamic capability reporting	MADT for processor topology, SRAT for memory affinity
AML/ASL	High-level source compiled into bytecode	Encodes hardware control logic	Thermal zone trip points, power-down sequences
Root System Description Pointer (RSDP)	Entry point for ACPI tables	OS locates ACPI structures	Directs OS to RSDT/XSDT
Differentiated System Description Table (DSDT)	Platform-specific AML methods	Custom hardware control	GPIO sequencing, device enumeration

Table 2. ACPI Architectural Components [2, 4, 5]

4. Power and Performance States

4.1 Global System States

ACPI defines a set of system-level power states that together cover the full range of operating conditions a platform must handle [2, 5]. S0 is the normal running state where all components are powered and the system is fully functional. Sleeping states S1 through S4 step progressively deeper into low-power territory: S1 halts the processor while keeping memory and IO powered; S3 suspends to RAM, powering off most devices while keeping DRAM alive through self-refresh so the system can return to operation quickly; S4 writes memory contents to disk and powers off nearly everything, allowing recovery even after a complete loss of power; and S5 presents as fully off to the user while retaining just enough power to detect wake events such as a button press or network signal [5].

Mobile and embedded platforms extend S0 with the S0ix substates, which provide intermediate idle modes that hold some system responsiveness while consuming very little power. A system with only S0 and S3 has no way to express the range of power-performance tradeoffs that modern mobile hardware makes available, and S0ix fills that gap without requiring a new top-level state.

4.2 CPU Idle States

C-states define how deeply the processor idles between instruction dispatches [2, 5]. C0 is the active execution state. C1 halts the pipeline while preserving

the ability to resume with minimal latency. Deeper states from C2 onward apply increasingly aggressive techniques, including clock gating, cache flushing, and voltage reduction, each saving more power at the cost of a longer return to C0.

The practical value of this hierarchy is that idle periods vary enormously in duration, and the right depth depends on how long the processor will actually be idle. A ten-microsecond gap should stay in C1 because the energy cost of entering and exiting a deep state would exceed any savings from doing so. A multi-second idle can safely reach C6 or C7, where voltage drops near zero and savings are substantial. The OS, by tracking idle duration patterns and knowing the exit latency of each state, selects the deepest level the application's response time requirements will accommodate [5].

4.3 Performance States

P-states address the problem that running a lightly loaded workload at full processor frequency wastes energy without delivering any benefit the workload can use [5]. P0 represents the highest frequency and voltage the processor will run at, reserved for workloads that need full throughput or cannot tolerate added latency. Moving to P1 and beyond steps both frequency and voltage down in increments, with each level giving back some performance in exchange for reduced power draw.

When the operating system observes that a workload is not pressing the processor hard enough to justify its current state, it moves down to whichever P-state matches the actual demand, recovering energy without any effect the user would notice. Turbo and boost modes sit above P0 and work in the opposite direction, permitting the processor to run briefly beyond its nominal maximum frequency when there is enough thermal and power headroom available to do so safely, an arrangement that benefits workloads whose demand arrives in short bursts rather than as a continuous sustained load [5].

State Type	Levels	Purpose	Example Actions
Global System States	S0–S5, S0ix	System-wide power modes	S3 suspend to RAM, S4 hibernate
CPU Idle States (C-states)	C0–C7	Processor idle depth	C1 halt the pipeline, and C6 voltage near zero
Performance States (P-states)	P0–Pn, Turbo	Frequency/voltage scaling	P0 max throughput, P2 reduced frequency
Thermal Zones	Passive, Active, Critical	Temperature-based responses	Passive throttling, active fan, critical shutdown
Trip Points	Defined thresholds	Trigger cooling or throttling	Passive trip at 85°C, critical trip at 100°C

Table 3. ACPI Power and Thermal States [5, 8, 11]

5. ACPI Thermal Management

Processors, memory controllers, accelerators, and voltage regulators all generate heat within a shared thermal envelope, and a localized hotspot in any one of them can cause damage or trigger throttling even when the system-wide average temperature appears acceptable. A thermal management approach that monitors one point in the system cannot catch these localized conditions before they become problems.

ACPI addresses this through thermal zones, each of which groups a related set of sensors with the cooling devices that respond to them [8]. A processor thermal zone might combine sensors spread across the die with controls for fan speed and CPU frequency. Trip points within each zone define the temperature thresholds at which specific actions are taken: the critical trip point triggers an immediate shutdown to prevent hardware damage, the passive trip point engages power reduction techniques such as frequency throttling that lower heat generation at some performance cost, and the active trip point activates fans or other powered cooling mechanisms [8, 11].

Because firmware defines the zones and their relationship to cooling devices, the same ACPI framework accommodates very different platform designs without requiring OS changes. A passively cooled embedded device can apply aggressive throttling thresholds to avoid needing a fan at all. An

actively cooled server can tolerate higher temperatures while running fans faster. Both platforms use the same thermal zone model, and the OS policy that reads from it does not need to know which kind of platform it is running on [8, 11].

6. ACPI for SoC Verification

As SoC complexity has grown, the gap between validating individual components and validating the integrated system has widened. A design incorporating tens of thousands of devices across multiple power domains, clock domains, and interconnect protocols cannot be verified by checking each block against its own specification in isolation; the interactions between blocks are where many of the most consequential bugs live.

ACPI provides a system-level specification precise enough to serve as the acceptance contract for that integrated validation. Verification teams define the expected behavior of every power state transition, the register changes each transition must produce, the sequencing of internal signals, and the timing relationships between domains [14]. Those definitions become the criteria a design must satisfy before tape-out.

6.1 ACPI as a Verification Contract

The specification creates obligations that bind all three parties involved in bringing a platform to silicon [14]. Hardware must support every state transition the contract defines. Firmware must implement every control method ACPI requires. The OS must correctly interpret the power states the hardware reports. Verification work tests the hardware side of that contract directly. Register access sequences must produce the effects the specification describes. Every state transition must finish within the timeout the specification sets. When a temperature threshold is crossed, the corresponding thermal zone must activate the correct cooling response. Wake events generated by interrupts must reach the operating system through exactly the delivery path the specification prescribes.

6.2 UVM-Based ACPI Verification

UVM environments built around ACPI verification typically include monitors observing register accesses and state transitions, scoreboards comparing actual hardware behavior against the behavior the specification mandates, functional coverage collectors tracking which combinations of power states and thermal conditions have been exercised, and assertion checkers enforcing protocol rules throughout the run [6, 12].

Hybrid memory subsystems that combine DRAM, low-power DRAM, and persistent memory add a further dimension: state transitions must correctly manage the distinct refresh and retention requirements of each technology. CPU hotplug sequences must be validated to confirm that newly added processors are discovered and brought online without disrupting the rest of the system. Embedded controller communication during power transitions must be checked for correct sequencing at every step [6, 12].

6.3 Emulation and FPGA Prototyping

RTL simulation provides cycle-accurate coverage of individual components but cannot easily sustain the extended workloads that surface system-level integration problems. Emulation and FPGA prototyping address that limitation by allowing actual OS images to boot against the design, sleep and wake sequences to be exercised from end to end, and thermal throttling behavior to be observed under conditions that represent realistic heat generation [6]. The result is a level of validation confidence that component-level simulation cannot provide. Problems that would only appear when real OS software drives the hardware through its full behavioral range are found before silicon is committed. By tape-out, the design has been exercised against realistic conditions, and the remaining uncertainty is bounded rather than open-ended [6].

7. ACPI for Firmware Engineering

Firmware teams must understand what the hardware does, express it accurately in ACPI tables and AML methods, and ensure the result gives the OS the interface it needs to control the platform correctly.

7.1 ACPI Table Development

ACPI tables must accurately represent physical hardware topology, and errors in that representation create mismatches between what firmware claims the hardware supports and what the hardware actually does. The DSDT must list every device and describe its power domain relationships. DSDT entries must correctly record processor counts, interrupt controller types, and IO device locations. A wrong entry does not fail at authoring time; it fails at runtime under conditions that may be difficult to reproduce [12, 13]. Getting tables right requires hardware teams to share physical topology, power capabilities, and thermal

characteristics with firmware teams, who translate that information into ACPI structures, while verification teams confirm consistency with the RTL and OS teams validate that the tables produce correct behavior. Automated generation tools that derive tables from hardware specifications reduce the manual error risk and keep tables synchronized when hardware changes rather than requiring a separate manual update cycle [13].

7.2 AML/ASL Development

Control methods execute at runtime and carry more risk than static tables because errors in them produce runtime failures rather than build-time errors. Power-down sequences must correctly order rail shutdowns, clock disables, and low-power mode entry. Thermal logic depends on sensor readings being translated into the correct mechanical response at precisely the thresholds the specification defines, with no margin for the wrong mechanism engaging at the wrong time. Device enumeration must handle hot-plug discovery with equal reliability, including the edge cases where a device appears or disappears while the enumeration sequence is still in progress. Embedded controller communication methods must synchronize processor and system controller activity correctly throughout every transition [12, 13].

An incorrect power-down sequence can leave a device consuming power in a state from which it cannot cleanly recover. Thermal logic that reacts too sharply to brief temperature spikes causes throttling that degrades performance without a genuine thermal justification. These problems are harder to find than functional bugs because they depend on specific combinations of state and timing that do not arise during routine testing.

7.3 Sleep/Wake and SCI/SMI Handling

Sleep and wake sequences are among the most demanding firmware responsibilities on any platform. S3 sleep entry requires saving processor state to memory, executing the platform-specific quiesce sequence, disabling clocks and power supplies that are not needed during sleep, and leaving the system in a configuration that maintains memory refresh while consuming as little power as possible. A wake event must restore hardware to a state the OS can resume from without encountering any inconsistency the sleep entry path left behind [4, 12].

System Control Interrupt routing determines how hardware events reach the OS through the ACPI event mechanism. A misconfigured SCI routing causes events to be lost or delivered incorrectly, and either outcome breaks the power management behavior that depends on those events being handled. Wake event configuration must be selective enough to preserve the power savings that sleep provides, allowing only the events that should interrupt sleep to do so rather than treating any hardware activity as a wake trigger [4, 12].

8. ACPI in Datacenter Power Management

Managing power across a large datacenter cannot be done effectively if every server makes decisions in isolation without awareness of the broader context it operates in. A fleet containing millions of processors, tens of millions of memory modules, and tens of thousands of power supplies require a coordination layer that can see and influence power and thermal behavior across the whole infrastructure, and that layer depends on every server speaking the same language about its capabilities and current state.

Feature	Description	Datacenter Benefit	Example
Datacenter-Scale Control Plane	ACPI tables unify fleet management	Consistent orchestration across heterogeneous servers	MADT for CPU layout, SRAT for memory
Rack-Level Power Budgeting	Interfaces enforce proactive caps	Prevents overload, maintains stability	RAPL limits adjusted before supply stress
Power Capping	OS throttles CPU/memory to meet caps	Predictable aggregate consumption	Cloud workloads constrained to budget

Distributed Thermal Zones	Sensors grouped with cooling devices	Balanced thermal load across servers	NVMe drive zones, CPU zones
NUMA & Chiplet Reporting	SRAT/SLIT describe latency/affinity	Efficient scheduling, VM placement	Threads placed near memory controllers
Accelerator Management	Custom tables describe GPUs/AI units	Independent but coordinated domains	IORT for DMA remapping, HBM reporting

Table 4. ACPI in Datacenter Power Management [7, 11, 15]

8.1 ACPI as a Datacenter-Scale Control Plane

An orchestration system managing a heterogeneous fleet cannot encode specific knowledge about every server model it will encounter. Processor counts, accelerator configurations, memory topologies, and thermal headroom vary widely across hardware generations and vendor configurations, and maintaining custom logic for each combination does not scale. ACPI provides the common vocabulary that makes fleet-wide management tractable. The MADT reports processor count and layout. The SRAT reports memory configuration. Custom tables enumerate accelerators. The orchestration layer reads this information once during server provisioning and relies on it for the server's entire operational life. Current power states, thermal readings, and active performance caps flow through the same interfaces, giving the orchestration layer real-time visibility into conditions across the fleet and the means to enforce datacenter-level power budgets as those conditions change [15].

8.2 Rack-Level Power Budgeting

A rack's power supply capacity sets a hard ceiling, typically in the range of ten to twenty kilowatts, and allowing aggregate server demand to reach that ceiling without prior intervention forces reactive responses that are worse than the alternatives. Waiting for the power distribution unit to act means headroom is already gone before any management decision is made. ACPI interfaces to power management hardware allow the orchestration layer to act ahead of that point. Through running average power limit interfaces, the OS on each server reads current consumption and active limits. Through Platform Power Limit mechanisms, the orchestration layer writes caps that the OS enforces by adjusting processor frequency and memory bandwidth [7, 15]. When rack-level telemetry shows consumption trending toward the ceiling, individual server caps can be tightened before any supply is stressed, keeping the fleet within its power envelope without forced shutdowns.

8.3 ACPI-Driven Power Capping

Cloud infrastructure is not sized for every server running at simultaneous peak demand. Building power and cooling capacity to that level would make the economics of shared infrastructure unworkable. Power capping is the mechanism that allows infrastructure to be right-sized while still accommodating the range of workloads a shared fleet encounters. When the orchestration layer assigns a power budget to a server, it writes the cap through ACPI and the OS kernel enforces it by throttling processor frequency and memory bandwidth to keep measured consumption within the specified limit [1, 7]. The workload operates within a narrower performance envelope than it would have unconstrained, but the datacenter maintains predictable aggregate consumption, and cooling headroom stays manageable. The alternative would require infrastructure investment that most cloud deployments cannot justify.

8.4 Thermal Zones as a Distributed Sensor Network

Responding only to processor package temperature leaves a substantial portion of the thermal picture unmonitored. Power supplies, storage devices, and memory modules all generate heat and can develop conditions that affect system reliability independent of what the processor die temperature shows at any given moment. ACPI thermal zones group sensors with the cooling mechanisms that serve them, allowing each part of the platform to be managed on its own terms while still participating in coordinated system-level response. A processor zone combines sensors across multiple cores with fan and frequency controls. A storage zone tracks NVMe drive temperatures. The

orchestration layer aggregates thermal data across the fleet and can shift workloads away from servers trending toward throttling before any performance impact occurs, distributing thermal load rather than concentrating it [11, 15].

8.5 NUMA and Chiplet Topology Reporting

Chiplet-based server processors distribute cores and memory controllers across multiple dies, and memory access latency depends on the physical relationship between the core issuing the access and the memory controller serving it. That topology is not visible to the OS without a structured description, and scheduling decisions made without it produce cross-die memory traffic that increases latency and reduces throughput unnecessarily. The SRAT and SLIT tables communicate the topology in a form the OS can act on. The scheduler places threads on the cores with the lowest latency path to the memory those threads use. Hypervisors apply the same information when assigning virtual machines to physical resources, ensuring VMs land on processor cores that have memory affinity to their allocated regions. Applications with explicit NUMA awareness can go further, allocating memory directly from the NUMA node closest to their execution context [9, 15].

8.6 Accelerator-Rich Servers

A server combining CPUs, GPUs, AI accelerators, and specialized processors requires ACPI to describe a more diverse set of compute elements than conventional server designs ever needed to express. The IORT describes DMA remapping capabilities required for security isolation in multi-tenant environments. Custom ACPI tables enumerate the accelerator's memory resources, including high-bandwidth memory attached to the accelerator die, and their interrupt routing. Power and thermal domains for accelerators are tracked independently from CPU domains because GPU frequency, memory bandwidth, and power consumption respond to their own control interfaces and must be managed separately while remaining coordinated at the system level [15].

8.7 Virtualization and Cloud-Scale Management

Guest operating systems in cloud environments interact with virtual hardware the hypervisor constructs rather than physical hardware directly. ACPI extends into that abstraction layer, allowing guests to discover their virtual topology through the same table structures they would use on a bare-metal system. The hypervisor constructs guest ACPI tables to reflect the physical resources actually assigned to each virtual machine, and those tables are updated as assignments change. When a guest receives additional virtual CPUs, it discovers them through ACPI hotplug methods and brings them online without disturbing workloads that are already running. Memory additions follow the same path, giving the hypervisor a way to adjust allocations in response to demand without requiring the guest to restart [4, 6].

8.8 ACPI in Datacenter Telemetry and Predictive Failure Analysis

When ACPI telemetry is aggregated across a large server fleet, patterns become visible that monitoring any individual machine would never reveal. Power consumption tracked across millions of servers over time shows which workload classes are energy-intensive and where efficiency improvements are achievable. Thermal trends that develop gradually over hours or days can indicate problems accumulating in cooling infrastructure, such as filter dust loading or fan bearing wear, well before any individual server crosses a threshold that would generate an alert [15, 16].

Correctable memory error rates reported through ACPI events are a known leading indicator of uncorrectable errors that eventually cause data loss. Thermal zone logs capture temperature excursions on servers that are trending toward their operating limits, and because these readings arrive before throttling kicks in, they give operators a workable lead time. Scheduled maintenance and proactive workload migration can then be used to address the developing condition during a planned window, keeping the response organized rather than reactive when an actual failure would otherwise force the issue.

9. Case Studies

9.1 Mobile SoC: LPDDR5 & PMM Hybrid Memory

Mobile platforms use ACPI to coordinate power management across memory technologies that differ significantly in their power draw and access characteristics. Low-power DRAM trades off differently from standard DRAM across the operating cycle: self-refresh draws less, but active operation consumes more than its conventional counterpart. Persistent memory presents a different profile again, sitting at very low idle power while behaving in ways that diverge from DRAM once accesses begin.

The OS coordinates both technologies through ACPI power state definitions, holding all memory active when workload demand is high and stepping down to self-refresh during idle periods, where DRAM consumption falls to the milliwatt range. Over extended idle periods, system state can be written to persistent memory and DRAM powered off entirely [15]. On mobile devices, thermal management is consequential in ways that go beyond performance: heat affects how a device feels to hold and influences industrial design choices about thickness and materials. Multiple thermal trip points allow the system to respond in graduated steps, reducing processor frequency, limiting network activity, or adjusting display brightness as temperature rises, rather than applying abrupt throttling when a single threshold is crossed.

9.2 Server SoC: Multi-Socket EPYC-Class Architecture

Multi-socket EPYC-based servers present ACPI with a topology description challenge that scales with the processor generation. Each socket may carry up to 128 cores with local memory controllers and direct memory access, connected to other sockets through high-bandwidth fabric. Memory latency is not uniform; a read from a memory module local to the accessing socket completes faster than one requiring a cross-socket hop, and that difference is large enough to matter for throughput-sensitive workloads.

The SRAT and SLIT communicate this topology to the OS, and the scheduler uses it to place threads on the cores with the shortest path to the memory those threads access. Power management across sockets requires coordination: a socket carrying no workload can be powered down without affecting running applications, and ACPI gives the OS per-socket utilization visibility to make that decision safely rather than guessing.

9.3 Datacenter: AI/ML Accelerator Racks

Accelerator racks combining GPUs or dedicated AI processors with general-purpose CPUs present a management challenge that conventional server power management was not designed to handle. Each accelerator carries independent power consumption, thermal limits, and management interfaces, and the orchestration layer must track and respond to all of them in coordination with processor-level state [15].

ACPI provides the framework that makes unified management possible. Custom tables enumerate GPUs and their memory resources. Separate power and thermal domains are reported for processors and accelerators. When a machine learning training job specifies a particular GPU allocation and memory configuration, the orchestration layer consults ACPI to find servers whose current resource state matches those requirements. Once the job is placed, ACPI continues feeding thermal and power readings to the orchestration layer throughout execution, and those readings determine whether conditions have degraded to the point where the job must be throttled or moved to a server running at a lower temperature.

10. Future Directions

10.1 AI-Driven Power Management

Current ACPI power management rests on heuristic algorithms calibrated for workload patterns that were observed and characterized before the algorithms were written. As workload diversity increases and the gap between average and peak utilization widens, fixed heuristics become less effective at recovering the energy that adaptive policies could capture [10].

Machine learning models trained on historical power and performance data will increasingly drive voltage and frequency decisions by predicting near-term demand rather than reacting to current utilization. Predictive thermal management will act on temperature trends before trip points are reached, activating cooling proactively and avoiding the performance degradation that reactive throttling produces [10]. Renewable-energy-aware scheduling will shift computationally intensive work toward periods of high renewable generation, reducing dependence on carbon-intensive grid supply.

10.2 Chiplet-Native ACPI Models

Today's ACPI model describes a topology that is established at boot and treated as fixed throughout operation. Chiplet-based designs that allow individual dies to be powered down or reconfigured at runtime does not fit within that assumption, and ACPI will need to evolve to express topologies that change while the system is running [9].

CXL-attached memory accessible from multiple physical locations will displace purely local memory in many deployment contexts, and ACPI must describe those resources and their access characteristics in a way the OS can use to manage them as a coherent pool rather than separate islands. Rack-scale memory and accelerator resources disaggregated

from individual servers extend that requirement further, pointing toward a model where ACPI describes and mediates access to resources regardless of their physical location in the infrastructure [9].

10.3 Cloud-Native ACP I

Cloud providers are moving toward ACPI table designs that carry no instance-specific information, allowing identical firmware images to deploy across heterogeneous hardware and simplifying provisioning at scale. Hot-plug-first architectures that treat every resource as potentially dynamic will reduce firmware complexity by eliminating the distinction between resources that are fixed at boot and those that can change afterward. Virtualization-optimized specifications tailored for hypervisor deployment will remove features that add no value in virtual environments while adding capabilities that cloud operations specifically require. Direct integration with the Redfish management API will allow management interfaces to query and modify ACPI parameters through a single unified path rather than through parallel and sometimes inconsistent management surfaces [15].

Conclusion

ACPI originated as a response to a practical problem: without a standard interface between hardware and OS for power management, operating system portability across platforms was not achievable. The two decades since have layered new coordination requirements onto that foundation at every major technological transition, and each time, ACPI has extended rather than been replaced. For SoC verification teams, the specification serves as the behavioral contract against which power management correctness is measured before silicon is committed. As SoCs incorporate neural processing units, graphics processors, data plane processors, and CXL-attached memory resources alongside conventional compute, the role of that contract in defining correct interaction between those elements becomes more important, not less. For firmware engineers, the precision of table authoring and AML method implementation is what determines whether the hardware's capabilities reach the OS intact. Dynamic chiplet compositions, where topology can shift at boot or during operation, demand firmware that keeps ACPI representations synchronized with physical reality as that reality changes. For datacenter operators, ACPI has become the abstraction layer that makes fleet-scale management tractable across hardware that varies widely in generation, vendor, and configuration. Consistent topology reporting, standardized power and thermal interfaces, and telemetry aggregated across millions of servers together support the predictive management that modern datacenter economics depend on. Disaggregated architectures, renewable-energy-aware scheduling, and AI-optimized operations all build on that foundation rather than requiring a replacement for it. Three forces will drive how ACPI develops from here. Heterogeneous compute will demand richer topology and power domain descriptions to cover the accelerators and specialized processors joining mainstream platforms. Disaggregated and CXL-based architectures will push resource pooling beyond the server boundary, requiring ACPI to describe resources shared across physical machines. AI-driven management will use ACPI telemetry as input to predictive models that treat performance and energy as jointly optimized objectives rather than separate concerns. Each of these directions extends what ACPI already provides, which is why the standard remains an active engineering effort rather than a compatibility layer frozen at a past point in time.

References

[1] Improve efficiency with server energy consumption tools https://www.techtarget.com/searchdatacenter/tip/Improve-efficiency-with-server-energy-consumption-tools

[2] Advanced Configuration and Power Interface (ACPI) Specification https://uefi.org/sites/default/files/resources/ACPI_Spec_6.6.pdf

[3] Advanced Configuration and Power Interface Specification https://web.archive.org/web/20151128143452/http://www.acpi.info/DOWNLOADS/ACPIspec30.pdf

[4] ACPI BIOS https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/acpi-bios?redirectedfrom=MSDN

[5] Andrew Grover, "Modern system power management," ACM, vol. 1, no. 7, Dec. 5, 2023. https://queue.acm.org/detail.cfm?id=957774

[6] Rajeev Muralidhar, Renata Borovica-Gajic, and Rajkumar Buyya, "Energy efficient computing

systems: architectures, abstractions, and modeling to techniques and standards," ACM Comput. Surv., vol. 54, no. 11s, p. 236, Sept. 2022. https://dl.acm.org/doi/full/10.1145/3511094

[7] Antonio del Vecchio et al., "Performance characterization of hardware/software communication interfaces in end-to-end power management solutions of high-performance computing processors," Energies, vol. 17, p. 5778, Nov. 2024. https://www.researchgate.net/publication/385952147

[8] Weichao He et al., "Active thermal management method for improving current capability of power devices under influence of random convection," Energies, vol. 17, no. 13, p. 3249, July 2024. https://www.mdpi.com/1996-1073/17/13/3249

[9] Darpan Virmani and Baibhab Chatterjee, "Thermal management challenges in 2.5D and 3D chiplet integration: a review on architecture–cooling co-design," Eng., vol. 6, no. 12, p. 373, Dec. 2025. https://www.mdpi.com/2673-4117/6/12/373

[10] Zichen Du and Renhao Lu, "Physics-informed neural networks for advanced thermal management in electronics and battery systems: a review of recent developments and future prospects," Batteries, vol. 11, no. 6, p. 204, May 2025. https://www.mdpi.com/2313-0105/11/6/204

[11] Zi-Qiang Zhu and Dawei Liang, "Perspective of thermal analysis and management for permanent magnet machines, with particular reference to hotspot temperatures," Energies, vol. 15, no. 21, p. 8189, Nov. 2022. https://www.mdpi.com/1996-1073/15/21/8189

[12] Rens Baeyens, Joachim Denil, Jan Steckel, Dennis Laurijssen, and Walter Daems, "Automated firmware generation for compressive sensing on heterogeneous hardware," Sensors, vol. 22, no. 21, p. 8147, Oct. 2022. https://www.mdpi.com/1424-8220/22/21/8147

[13] Rens Baeyens, Joachim Denil, Jan Steckel, and Walter Daems, "Model-based firmware generation for acquisition systems using heterogeneous hardware," Autom., vol. 3, no. 3, pp. 471–485, Aug. 2022. https://www.mdpi.com/2673-4052/3/3/471

[14] Ryan Aalund and Vincent Philip Paglioni, "Enhancing reliability in embedded systems hardware: a literature survey," IEEE Access, vol. 13, Jan. 2025. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10852318

[15] Ashkan Safari, Hoda Sorouri, Afshin Rahimi, and Arman Oshnoei, "A systematic review of energy efficiency metrics for optimizing cloud data center operations and management," Electronics, vol. 14, no. 11, p. 2214, May 2025. https://www.mdpi.com/2079-9292/14/11/2214

[16] Ioannis Makris et al., "A comprehensive survey of federated intrusion detection systems: techniques, challenges and solutions," Comput. Sci. Rev., vol. 56, p. 100717, May 2025. https://www.sciencedirect.com/science/article/pii/S1574013724001000X