ACPI (Advanced Configuration and Power Interface) is designed to enable features such as power saving, thermal management, device enumeration, and system sleep states dynamically. ACPI works as an interface between firmware (BIOS/UEFI), hardware, and the operating system.

Linux interacts with ACPI primarily through the ACPI driver in the kernel, which reads ACPI tables provided by the BIOS/UEFI during boot and processes ACPI events dynamically at runtime.

1. ACPI Tables & Initialization

* The firmware provides ACPI tables (e.g., DSDT, SSDT, FADT) that define hardware configurations and power management policies.
* The Linux kernel ACPI subsystem parses these tables at boot time (drivers/acpi/).
* The ACPI interpreter (ACPICA) translates ACPI bytecode into kernel actions.

2. Device Enumeration & Power Management

* ACPI helps enumerate devices dynamically and provides hotplugging support for some hardware. Linux ACPI supports power states:
  + G0 (S0) – Running
  + G1 (S1-S4) – Sleep/Standby/Hibernate
  + G2 (S5) – Soft power off
  + G3 – Mechanical off
* ACPI-based CPU power management (P-states and C-states) dynamically adjusts CPU performance to save power.

3. ACPI Events Handling in Linux

* Events like lid close, battery status change, thermal warnings, and key presses are detected by ACPI and passed to user space via the ACPI event daemon (acpid).
* The kernel handles ACPI events in /proc/acpi/event, and acpid can execute scripts based on these events. For systemd-based distros, ACPI events are managed via udev rules and logind.

4. Driver Interactions

* ACPI interacts with Linux drivers through the ACPI bus (/sys/bus/acpi). Devices that support ACPI are controlled through ACPI namespace methods (e.g., \_PS0 for power on, \_PS3 for power off).
* ACPI-based drivers include:
  + acpi\_cpufreq – CPU frequency scaling
  + acpi\_thermal – Thermal zone management
  + acpi\_battery – Battery and power supply monitoring

5. Debugging ACPI in Linux

* View ACPI tables:

cat /sys/firmware/acpi/tables/

* Check ACPI logs:

dmesg | grep -i acpi

* List ACPI-supported devices:

ls /sys/bus/acpi/devices/

ACPI (Advanced Configuration and Power Interface) plays a crucial role in managing PCI Express (PCIe) devices in Linux, particularly for power management, hotplugging, and resource allocation.

1. ACPI's Role in PCIe Initialization

* At boot time, the BIOS/UEFI provides ACPI tables that describe the system’s PCIe topology. The DSDT (Differentiated System Description Table) and SSDT (Secondary System Description Table) contain methods (\_CRS, \_PRT, etc.) that define PCIe bus resources.
* The Linux ACPI subsystem reads these tables to configure PCIe devices.

2. PCIe Bus Enumeration via ACPI

* Linux detects PCIe devices using two mechanisms:
  + ACPI-based enumeration (for servers, laptops with power management)
  + PCIe-native enumeration (for performance-focused systems)
* The **ACPI PCI Root Bridge driver** (drivers/pci/pci-acpi.c) parses the \_CRS (Current Resource Settings) method to assign memory regions, I/O ports, and interrupts to PCIe devices.

3. Power Management (PCIe + ACPI)

ACPI provides power states for PCIe devices:

* D0 – Fully on (active)
* D1/D2 – Intermediate low-power states
* D3 (D3hot & D3cold) – Low-power/sleep state

Linux kernel PCIe drivers use ACPI methods to manage power transitions:

* \_PS0 (Power State 0) → Power on
* \_PS3 (Power State 3) → Power off
* \_PR3 (Power Resource for D3) → Helps with fine-grained power control

Linux implements this in the PCI runtime power management framework, controlled via:

cat /sys/bus/pci/devices/0000:00:1c.0/power/control

Values:

* auto – Kernel manages power dynamically (based on ACPI hints)
* on – Always keep the device powered

4. PCIe Hotplug with ACPI

* ACPI provides \_EJ0 (Eject), \_HPX (Hotplug parameters), and \_DSM (Device-Specific Methods) for handling PCIe hotplug. When a PCIe device is inserted/removed:
  + The ACPI firmware notifies the OS through ACPI events.
  + The Linux kernel ACPI subsystem triggers the appropriate PCIe hotplug driver (pciehp).
  + The kernel configures or powers down the PCIe slot dynamically.

5. ASPM (Active State Power Management)

ACPI works with PCIe ASPM (L0s, L1) to reduce power consumption:

* The BIOS/ACPI firmware provides ASPM policy. The Linux kernel applies these settings via:

cat /sys/module/pcie\_aspm/parameters/policy

Possible values:

* + default (BIOS settings)
  + performance (ASPM disabled)
  + powersave (ASPM enabled)

6. PCIe Wake Events (ACPI + PME)

* ACPI allows PCIe devices to wake the system using Power Management Events (PME). \_S0W (Wake capability in S0) defines if a PCIe device can wake the system from suspend states. Linux checks ACPI wake capabilities using:

cat /proc/acpi/wakeup

Unlike x86, where ACPI is the standard for hardware discovery and power management, ARM traditionally used Device Tree (DT). However, ACPI is gaining traction in ARM servers (like ARMv8 and ARMv9) due to its structured power management and runtime configuration benefits.

1. BootROM (BL1)

* Role: BootROM (BL1) is the first-stage bootloader stored in ROM, responsible for initializing low-level hardware (e.g., secure world, memory controllers).
* ACPI Dependency:
  + No direct interaction with ACPI.
  + It initializes basic peripherals and hands off control to Trusted Firmware-A (TFA).
  + ARM systems mostly rely on Device Tree, so ACPI tables are not handled at this stage.

2. Trusted Firmware-A (TFA) (BL2 & BL3)

* Role: TFA (BL2/BL3) initializes security features, ATF (ARM Trusted Firmware), and hands off control to UEFI. It sets up EL3 (Exception Level 3) and initializes Secure World.
* ACPI Dependency:
  + TFA does not directly interact with ACPI.
  + It ensures that secure world services (e.g., PSCI – Power State Coordination Interface) are available to the OS, which ACPI may later use for power state management (P-states, C-states, etc.).

3. UEFI (EDK2/Tianocore) (BL3-2)

* Role: UEFI is the firmware interface that loads and executes the OS bootloader (GRUB, Linux, etc.).
* ACPI Dependency:
  + UEFI is responsible for providing ACPI tables (DSDT, SSDT, FADT, etc.) to the Linux kernel. UEFI implementations on ARM (like Tianocore/EDK2) can dynamically create ACPI tables or load them from firmware.
  + ACPI vs. Device Tree in ARM:
    - UEFI can pass either Device Tree (DTB) or ACPI tables to the OS.
    - For ARM servers (SBSA-compliant systems), ACPI is preferred over Device Tree.

4. Linux Kernel

* Role: The Linux kernel initializes hardware, manages drivers, and controls system power states.
* ACPI Dependency:
  + The kernel ACPI subsystem reads ACPI tables provided by UEFI.
  + Uses ACPI-based power management (C-states, P-states, S-states) instead of relying on PSCI directly.
  + Manages PCIe enumeration, thermal management, and device power control via ACPI \_DSM, \_PS0/\_PS3, etc.
  + Reads ACPI tables from /sys/firmware/acpi/tables/.

Key Takeaways

1. BootROM & TFA: No direct ACPI role; they focus on initializing secure/normal worlds.
2. UEFI: Supplies ACPI tables to Linux, replacing Device Tree in SBSA-compliant ARM servers.
3. Linux Kernel: Uses ACPI tables for power management, PCIe handling, and runtime configurations.

**On consumer ARM devices (Raspberry Pi, Android, embedded systems), ACPI is rarely used—they still rely on Device Tree. However, for ARM servers (Ampere, AWS Graviton, etc.), ACPI is becoming the standard for firmware abstraction.**

Detailed ARMv8-A Boot Flow

1. BootROM (BL1 - First Bootloader Stage)

* Location: Stored in Read-Only Memory (ROM)
* Execution Level: Runs at EL3 (Highest Privilege - **Secure World**) **Remember, it runs in secure world.**
* Main Tasks:
  + Initializes the CPU core, internal memory, and secure boot verification (if enabled).
  + Initializes low-level hardware components (SoC, UART, DDR controller, etc.).
  + Checks for a valid firmware image (Trusted Firmware-A and UEFI).
  + Loads and verifies the next bootloader stage (BL2) from non-volatile storage (eMMC, NAND, SPI, etc.).
* Dependency on ACPI:
  + None. BootROM does not interact with ACPI or Device Tree.

2. Trusted Firmware-A (TFA)

TFA is split into three stages: BL2, BL3-1, and BL3-2.

2.1 BL2 (EL3 - Secure World)

* Location: Stored in Secure Flash or Secure ROM
* Main Tasks:
  + Initializes DRAM (DDR Controller).
  + Configures TrustZone (Secure and Non-Secure worlds).
  + Loads BL31 (EL3 Runtime Firmware) and BL33 (UEFI, if applicable).
  + Verifies and hands control to BL31 (EL3 Runtime Firmware).

2.2 BL31 (EL3 Runtime Firmware - Secure Monitor)

* Role: Provides EL3 runtime services, such as:
  + PSCI (Power State Coordination Interface) → Manages power and CPU sleep states.
  + Handles Exception Handling and Secure Call APIs (SMC - Secure Monitor Calls).
  + Sets up Secure World EL3 Exception Vector Table.
  + Hands control to either:
    - BL32 (Secure OS like OP-TEE, if present)
    - BL33 (UEFI firmware, if no secure OS is used).

2.3 BL32 (Optional Secure OS - Trusted Execution Environment)

* If a TEE (Trusted Execution Environment) like OP-TEE is used, BL32 runs at EL1 (Secure World).
* Provides secure services (e.g., **cryptography, secure key storage**).
* If no secure OS is present, the bootloader proceeds directly to UEFI (BL33).
* Dependency on ACPI:
  + Minimal. TFA does not interact with ACPI directly, but PSCI (used by ACPI for power management) is implemented here.

3. UEFI (BL33 - Non-Secure Bootloader)

* Location: Stored in NOR Flash, SPI Flash, or eMMC.
* Execution Level: Runs at EL2 or EL1 (Non-Secure World).
* Main Tasks:
  + **Initializes PCIe**, IOMMU, USB controllers, networking, and storage.
  + Loads ACPI tables (if used) or Device Tree Blob (DTB).
  + Implements EFI runtime services (variable storage, boot services).
  + Locates the bootloader (GRUB) or directly boots the Linux kernel.
  + Passes control to the Linux Kernel at EL2 or EL1.
* Dependency on ACPI:
  + UEFI is responsible for passing ACPI tables (DSDT, SSDT, FADT) to the Linux kernel.
  + If ACPI is not used, it passes the Device Tree Blob (DTB) instead.

4. Linux Kernel (EL1/EL2)

* Location: Loaded into RAM by UEFI.
* Execution Level:
  + If virtualization is enabled → Runs at EL1 (Hypervisor at EL2).
  + If no hypervisor is used → Runs at EL2.
* Main Tasks:
  + Parses ACPI tables or Device Tree (DTB) for hardware configuration.
  + Initializes the MMU, scheduler, drivers, and memory management.
  + Loads essential kernel modules (PCIe, USB, network drivers, etc.).
  + Starts the init process, which brings up the user space.
* Dependency on ACPI:
  + If ACPI is used, Linux reads ACPI tables from /sys/firmware/acpi/tables/.
  + If Device Tree is used, Linux reads the DTB passed by UEFI.
  + Manages power states via PSCI calls from ACPI.

Key Takeaways

|  |  |  |  |
| --- | --- | --- | --- |
| Stage | Component | Runs At | ACPI Dependency |
| BL1 | BootROM | EL3 (Secure) | No ACPI, only loads BL2 |
| BL2 | TFA (Loader) | EL3 (Secure) | No ACPI, loads BL31 |
| BL31 | TFA Runtime | EL3 (Secure) | Implements PSCI (used by ACPI for power management) |
| BL32 (Optional) | Secure OS (OP-TEE) | EL1 (Secure) | No direct ACPI, used for security services |
| BL33 | UEFI (Tianocore/EDK2) | EL2/EL1 | Passes ACPI tables (or DTB) to Linux |
| Linux Kernel | OS | EL1/EL2 | Uses ACPI tables for power management, PCIe, and hardware config |