SFO15-TR9: ACPI, PSCI (and UEFI to boot)

Bill Fletcher
Linaro Field Engineering
v2.0
Overview

● An introductory look at ACPI infrastructure
● How a platform (qemu), a bootloader (uefi) and the kernel work together to set up the ACPI configuration
● There's an explanation of what’s been upstreamed in Linux kernel 4.1
● A simple aarch64 demo based on qemu
● A brief look at what's new
Caveats. This is not ...

- An exhaustive tutorial on ACPI
- A reference implementation for PSCI
- “Why to use ACPI instead of device trees”
- “Why to favour UEFI instead of UBoot”
- ...

Linaro connect
San Francisco 2015
ACPI and PSCI (briefly)
"Advanced Configuration and Power Interface” Specification (Currently v6.0: http://www.uefi.org/acpi/specs)

"Industry-standard interfaces enabling OS-directed configuration, power management, and thermal management (since 1996)"

ACPI is important because hardware and OS vendors have already worked out the mechanisms for supporting a general purpose computing ecosystem

Hence it’s non-negotiable in the Enterprise environment, standardized with UEFI

Has three main components: Tables, Drivers and Registers.

There’s an interpreted element in ACPI Machine Language (AML) bytecode stored in the ACPI tables.
What ACPI looks like - a chain of tables

Extended System Description Table
- RSD Ptr
- XSDT
- FACP
  - Table entry
  - Table entry
  - ...
- “FACP”
  - Header
  - Static Info
  - ...
  - DSDT
  - BLKs
- Fixed ACPI Description Table
- Differentiated System Description Table
  - DSDT
  - RSD Ptr
  - Differentiated Definition Block
  - Devices: Interrupts
  - Registers
  - Driver entry points

OEM Specifics
- Hardware
  - Register
  - Accesses
- Device I/O
  - Configuration
- Device Memory
- Controller Space

ACPI drivers

Linaro connect
San Francisco 2015
"Power State Coordination Interface"

- It’s an ARM standard
- The official/only way for power management on ARM64
- A generic interface that low-level supervisory software can use to manage power
- Lives under CPU Ops, which itself lives under CPU Idle and the CPU Governors
- Covers: Core idle management, dynamic addition and removal of cores, and secondary core boot, core migration, system shutdown and reset
- Has both Device Tree and ACPI bindings
Booting
Basic Boot and ACPI/PSCI Discovery

1. ACPI-aware boot
   - Root system description (pointer)
   - ACPI tables

2. Described in the ACPI 5.1 Spec and available in the 4.1 kernel
   - FACP Table
   - DSDT Table
   - APIC Table
   - ARM-specific PSCI information
Practical boot and ACPI/PSCI discovery

"The generic interface for discovering ACPI tables is to boot as an EFI application and then to query the tables from UEFI. That is the interface others are likely to follow, and ACPI without UEFI is unlikely to be of much use to anyone else (ARM)"

Kernel with ACPI patches
EDK2 UEFI bootloader
Root system description (pointer)
ACPI=force
InstallQemuFwCfgTables
ACPI tables
PSCI information
qemu

Linaro connect
San Francisco 2015
What about dtb?

- The EFI stub loader makes the Linux kernel image into a UEFI application.
- This application uses the Flattened Device-Tree (FDT) format to pass information about how to access UEFI to the Linux kernel.
- If no device-tree blob (using dtb= ) is available as in our case when using ACPI for hardware description the loader stub will create a new dtb containing only this information.

See: https://wiki.linaro.org/LEG/Engineering/Kernel/UEFI/Architecture and drivers/firmware/efi/libstub/fdt.c (leading to efi.h)
Basic Feature Set

Having booted successfully and loaded the ACPI tables … what functionality do we have?
4.1 ARM ACPI Kernel functionality

- Basic support to run ACPI on ARM64
- ARM SMP and GICv2 init using MADT
- PSCI announced in the FADT table
- ARM timer init using GTDT
- Kernel documentation why/how to use ACPI on ARM64
Why talk about this now?

- The ACPI Spec (5.1) supported ARM specifically for the first time this year
- ACPI on ARM is a major and continuing LEG collaboration between ARM and Linaro
- “There is no longer any reason to feel that ACPI only belongs to Windows or that Linux is in any way secondary to Microsoft in this arena” (kernel arm-acpi.txt)
ACPI Boot Demo

Comprises:

- qemu built from sources
- kernel with ACPI 5.1 built from source
- UEFI built from EDK2 source
- uefi-tools
- iasl, the ACPI table assembler/disassembler

Allows to:

- Boot UEFI as a guest in aarch64 qemu
- Boot the Linux image from the UEFI shell
- Confirm ACPI is enabled
- Dump and read (via iasl) the ACPI tables
- Modify the ACPI data in qemu, rinse and repeat …
ACPI in the booting/running system
UEFI Booting and reading the ACPI tables

- UEFI has platform-specific functionality to load the tables - in this case to interface with qemu
- Notice that PCI enumeration blocks ACPI table loading
Linux Boot

- Boot from UEFI with `ACPI=force`
- ACPI entry in `/proc`
- ACPI tables in `/sys/firmware/acpi/tables`

```
root@localhost:~# ls /proc
buddyinfo  config.gz  crypto  devices  filesystems  kmsg  softirqs

cryptd  crypto  driver  disks  driver  exedoms  fb  locksys
```

```
root@localhost:~# ls /sys/firmware/acpi/tables
APIC  DSDT  FACP  GTDT  MCFG  DYNAMIC
```
Some more about the tables

Devices are reported - CPUs, Comms (DSDT table)
PSCI is present (FADT/FACP table)
Way to access PSCI is stated as HVC (FADT/FACP table)
Interrupt structures are reported (APIC table)
All tables in AML bytecode - disassembled with iasl
AML data generated by qemu - see ./hw/arm/virt-acpi-build.c
Differentiated System Description Table
System & peripherals, memory mapping, IRQs, driver entry points

DefinitionBlock ("DSDT.aml", "DSDT", 1, "BOCHS ", "BXPCDSDT", 0x00000001)
{
  Scope (\_SB)
  {
    Device (CPU0)
    {
      ...
    }
  }

  Device (COM0)
  {
    Name (_HID, "ARMH0011") // _HID: Hardware ID
    ...
    {
      Memory32Fixed (ReadWrite,
      0x09000000, // Address Base
      0x00001000, // Address Length
    }
    Interrupt (ResourceConsumer, Level, ActiveHigh, Exclusive, , , )
    {
      0x00000021,
**FADT**

Fixed ACPI Description Table ("FACP")
Part of the chain that gets to the DSDT
Register blocks relating to power management … and PSCI

```
[000h 0000 4]                      Signature : "FACP"    [Fixed ACPI Description Table (FADT)]

<snip>
```

```
[074h 0116 12]                     Reset Register : [Generic Address Structure]
[074h 0116 1]                      Space ID : 00 [SystemMemory]
[075h 0117 1]                      Bit Width : 00
[076h 0118 1]                      Bit Offset : 00
[077h 0119 1]                     Encoded Access Width : 00 [Undefined/Legacy]
[078h 0120 8]                      Address : 0000000000000000

[080h 0128 1]  Value to cause reset : 00
[081h 0129 2]  ARM Flags (decoded below) : 0003
                  PSCI Compliant : 1
                  Must use HVC for PSCI : 1
```

The full table dumps are reproduced here:
http://people.linaro.org/~bill.fletcher/SFO15-TR9_ACPI_PSCI_and_UEFI_To_Boot_supporting_material/
**MADT**

Multiple APIC Description Table (“APIC”)

APIC = Advanced Programmable Interrupt Controller (8259+)

Interrupt controllers and CPU affinity

[000h 0000 4]  Signature : "APIC" [Multiple APIC Description Table (MADT)]

<snip>

[02Ch 0044 1]  Subtable Type : 0B [Generic Interrupt Controller]

...  

[038h 0056 4]  Flags (decoded below) : 00000001

  Process Enabled : 1

  Performance Interrupt Trigger Mode : 0

  Virtual GIC Interrupt Trigger Mode : 0


[040h 0064 4]  Performance Interrupt : 00000000

...  

[070h 0112 8]  ARM MPIDR : 0000000000000000

There can be a system MADT referenced from the XSDT

Peripherals with their own interrupt controllers can have a method in their DSDT entry for you to get a local MADT
Usage outside qemu
ATF PSCI functionality  (it’s the reference)

Handles SMCs (Secure Monitor Calls) conforming to the SMC Calling Convention PDD

There’s extensive design documentation available.

Source is published under a BSD license.
Calling PSCI from an ACPI context

PSCI implementation assumes either a Hypervisor running at EL2 or a Secure Monitor running at EL3.

Requires a platform running the Secure Monitor at EL3, which implements PSCI.

Alternatively if there's no EL3 but a hypervisor instead you can emulate this with a HVC call.

This HVC mechanism is also emulated in qemu.

either ARM trusted firmware, platform-specific implementation in hypervisor, or qemu emulation

ACPI-aware application

ACPI tables

HVC or SMC trap

PSCI driver calls

PSCI information

1 2 3
What’s New and What’s Next?
What arrived in the latest ACPI 6.0 spec?

Enough in ACPI "to boot a server, support ARM’s latest IP (GICv2m, GICv3, SMMUv2) and do decent idle management."

What’s next? DVFS performance management ...

… for an ARM server to implement DVFS in an architecture agnostic way via “CPPC” (CPPC - Collaborative Processor Performance Control)
ARM Low Power Idle (LPI) states

ACPI 6.0 introduces Low Power Idle states, which allows an operating system to manage the power states of the processor power domain hierarchy.

The ACPI FFH* mechanism is used in ARM-based systems to allow the operating system to discover:  the entry method into a low power state;  how to collect power state residency, and statistics This ACPI register-based interface is defined in ARM [DEN0048A](http://www.arm.com) available on ARM Infocentre portal

*FFH = *Functional Fixed Hardware Interface* - an ACPI name for the register address space used for power management*
Back to DVFS → CPPC & PCC (1)

CPPC (Collaborative Processor Performance Control)
Provides more autonomy to the CPU subsystem to control power/perf
An abstract interface where the OS specifies performance bounds and leaves the underlying subsystem some latitude in how it achieves them.
"CPPC implementation designed as shim which allows Cpufreq drivers to plug into existing governors or alternately implement inbuilt ones"

PCC (Platform Communication Channel)
PCC (Platform Communication Channel) is a generic means for PCC Clients such as CPPC, to talk to the firmware.
CPPC & PCC (2)

- CPPC has been used on ARM64 servers successfully. Some ARM vendors may not have feedback counters (workaroundable). Missing in CPPC at the moment is the power-to-performance level mapping. This is to be addressed in the near future once EAS is upstream.

- This patchwork is part of LEG activity. See https://git.linaro.org/people/ashwin.chaugule/leg-kernel.git

- V9 patches picked up for v4.3 - hoping to upstream by v4.4
Wrap Up
Wrap-Up

- UEFI is a requirement for ACPI
- Upstream qemu can emulate an aarch64 ACPI platform (with elementary PSCI)
- You can build a simple ARM64 ACPI-aware system with 4.1
- ARM idle management arrived in the ACPI 6.0 spec
- Interfaces to ARM DVFS (via CPPC) are still "works in progress".
- Once these arrive, a demonstrator will need more than qemu’s emulation of PSCI
- ARM trusted firmware is the reference implementation for PSCI
References

- ACPI standard: http://www.uefi.org/sites/default/files/resources/ACPI_5_1release.pdf
- Basic ACPI info: http://www.acpi.info/over.htm
Resources

- The full table dumps are reproduced here: http://people.linaro.org/~bill.fletcher/SFO15-TR9_ACPI_PSCI_and_UEFI_To_Boot_supporting_material
- LEG wiki: https://wiki.linaro.org/LEG/Engineering/Kernel/ACPI
- For a comparison of ACPI and FDT, google the presentation done by Graeme Gregory at Linux Plumbers 2013.
- See http://article.gmane.org/gmane.linux.ports.arm.kernel/382864-match=cppc for an explanation of how to use ACPI
- Ashwin’s V8 patch set including CPPC via PCC http://thread.gmane.org/gmane.linux.power-management.general/63697
Q&A?
More about Linaro: www.linaro.org/about/
Linaro members: www.linaro.org/members