ACPI on ARM64: Challenges Ahead

Sudeep Holla - ARM<sudeep.holla@arm.com>

LinuxCon Europe - Dublin 2015

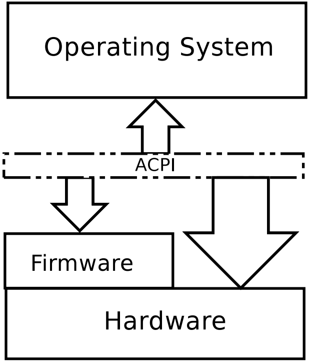
Agenda

- Introduction
- Comparison with Device Tree (DT)
- ACPI : Why on ARM64 ?
- ACPI on ARM64: Status Quo
- ACPI on ARM64: Issues faced
- ACPI on ARM64: Challenges Ahead

ACPI (Advanced Configuration and Power Interface): Introduction

- First released in December 1996
- Originally developed by Intel, Microsoft and Toshiba with HP and Phoenix joining later
- Slowly gained wider adoption with many OS and processor architectures
- In October 2013, ACPI standards were transferred to the UEFI Forum
- ACPI v5.1 was the first release enabling ARM systems
- ACPI v6.0 was published by the UEFI Forum in April 2015

Platform Description + High-Level Interface

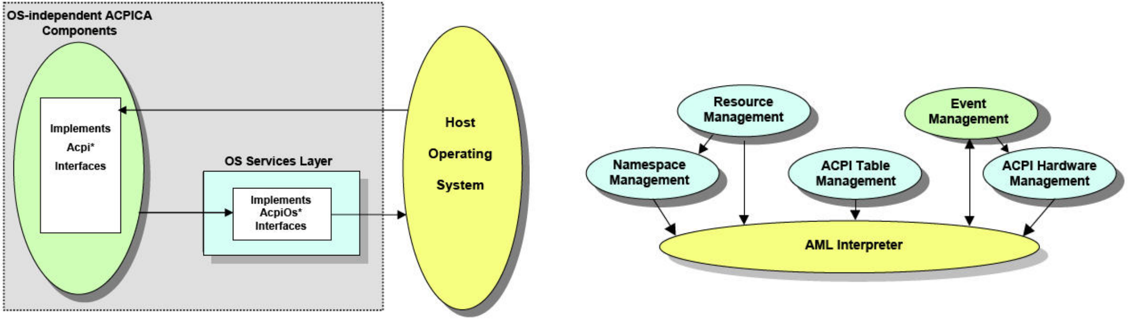


ACPI: Few main functional areas

- Processor Power, Performance and Thermal Management
- System and Device Power Management
- System Event Management
- Configuration of Devices
- Enumeration and Plug 'n' Play
- Flexible Platform Architecture Support

ARM

ACPI: Component Architecture(ACPICA)



Reference: http://gauss.ececs.uc.edu/Courses/c4029/doc/acpica-reference.doc

ARM

ACPI: Motivations

- SoC/Platform Vendors, OS distributions
  - Platform-agnostic OS/kernel images
  - No platform specific code for every platform
  - Need not worry about shipping and maintaining separate binaries
- Linux Kernel Ecosystem
  - No platform specific code to maintain
- Platform Designers
  - Reduce effort required to port to new platform
- Other software developers
  - Unification of description/reduced duplication across multiple layers of software stack

ACPI: an enhanced + dynamic DT ?

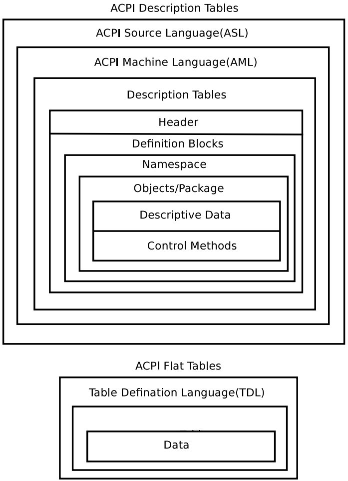
Device Tree

- Data only, more in kernel/drivers
- Suitable in embedded and mobile
- Scope for more optimization in kernel
- Primarily for Linux(though OS agnostic)
- More flexible for changes
- Community driven

ACPI

- More platform abstraction, more in firmware
- Suitable for enterprise
- No micro-optimization for ease of long maintenance
- OS agnostic standard
- Less flexible for changes
- UEFI ASWG (ACPI Specification Working Group) driven

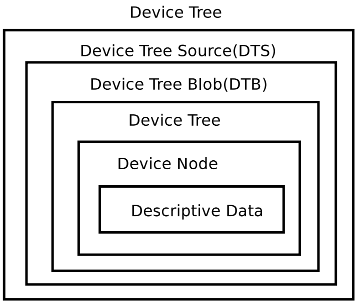
ACPI: Format + Example




```
        ACPI Source Language(ASL) Example
DefinitionBlock (
"forbook.aml", // Output Filename
"DSDT", // Signature
0x02, // DSDT Compliance Revision
"OEM", // OEMID
"forbook", // TABLE ID
0x1000 // OEM Revision)
{ // start of definition block
  OperationRegion(\GIO, SystemIO, 0x125, 0x1)
  Field(\GIO, ByteAcc, NoLock, Preserve) {CT01, 1,}
  Scope(\_SB) {
   Device(PCI0) {
    PowerResource(FET0, 0, 0) {
      Method (_ON)
      {
        Store (Ones, CT01)
        Sleep (30)
      }
      Method (_OFF) {
        Store (Zero, CT01)
      }
      Method (_STA) {
        Return (CT01)
      }
    }
  } // end of device
 } // end of scope
} // end of definition block
```

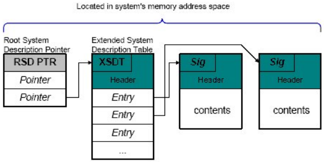

ARM

DT: Format + Example




```
        Device Tree Source(DTS) Example
/ {
  node1 {
    a-string-property = "A string";
    a-string-list-property = "first string", "second string";
    /* each number (cell) is a uint32 */
    a-cell-property = <1 2 3 4>;
    child-node1 {
      first-child-property;
      second-child-property = <1>;
      a-string-property = "Hello, world";
    };
    child-node2 {
    };
  };
  node2 {
    an-empty-property;
    a-byte-data-property = [0x01 0x23 0x34 0x56];
    child-node1 {
    };
  };
};
```

ACPI: System Description Table



DSDT	Differentiated System Description Table
FADT	Fixed ACPI Description Table
GTDT	Generic Timer Description Table
MADT	Multiple APIC Description Table
MCFG	Memory-mapped ConFiGuration space
RSDP	Root System Description Pointer
SRAT	System Resource Affinity Table
SSDT	Secondary System Description Table
XSDT	eXtended System Description Table

ACPI: Why on ARM ?

- ACPI allows the platform to encode run-time behavior while DT is just static data
- ACPI defines a OSPM model that has constraints but allows for flexibility in hardware
- ACPI has already proven bindings such as for RAS, NUMA,..etc
- Support multiple OSes, including Linux and Windows
- ASWG provides both the HW vendors and OS vendors a common platform for discussion

Compliance to ARM SBSA (Server Base System Architecture) and SBBR (Server Base Boot Requirements) is mandated on ARM64 servers to support ACPI

ARM

ACPI and Linux Kernel

- Initially, ACPI was not an obvious win on other architectures (i.e x86)
- The standard was new and actual implementations were unreliable
- Booting with ACPI disabled was the first response for any problems
- Basic ACPI support for ARM64 was merged in v4.1
- Multiple instances of ARMv8-A HW claims to support ACPI - are they really ready?
- We need to avoid repeating mistakes and leverage from x86 learnings

ACPI on ARM64: Current Status(1)

- ACPI Specification
  - New to ARM Linux world at-least
  - ACPI v5.0 lacked support for quite a lot in SBSA
  - GICv2m, virtualization on GIC, generic timer, watchdog, GICv3
  - ACPI v5.1 fixed the above
  - ACPI v6.0 added GICv3, IO topology + SMMU (companion IORT spec) and Lower power idle(LPI) states
- ARM Ecosystem
  - Lack of experience with ACPI within the ARM community
  - Includes ARM Linux kernel developers, firmware authors, hardware designers

ACPI on ARM64: Current Status(2)

- Core ARM64 support upstreamed in v4.1
  - SMP boot (PSCI only)
  - Generic Timer and GIC support
- Minor updates and bug fixes in v4.2
- Work in Progress
  - GICv2m/GICv3 support
  - PCI support
  - cpufreq(CPPC) and cpuidle(_LPI)

ACPI: General Challenges

- ACPI is a complicated specification
- ACPI is "trying" to define an entire interface for abstracting hardware details to enable booting on diverse platforms
- Simply impossible to define the complete behavior
- Doesn't explicitly state the behavior if the spec is violated
- On x86, ACPI systems shipped with Windows helped to ensure the firmware is validated well enough

ARM

ACPI: Challenges on ARM64

- ACPI development itself is mostly x86 driven and more aligned to it
- ACPI firmware testing is the main challenge
  - Firmware development by ARM vendors who are completely new to it
  - LUV (Linux UEFI Validation) bu Intel and luv0S (Yocto Project)
- Growing interest in ACPI within ARM ecosystem
  - Need to take (more strict?) steps to enforce standardization

In order to avoid regressions especially in the variety of ARM systems

- Commit to never modifying the ACPI behavior of Linux (impractical)
- Interface indicating ACPI behavior a specific kernel implements

Real examples(1): Static ACPI Table + DT

```
[Multiple APIC Description Table (MADT)]
            Signature : "APIC"
         Table Length : 00000224
             Revision : 03
             Checksum : BD
            [Other header entries]

        Subtable Type : 0B [Generic Interrupt Controller]
               Length : 4C
             Reserved : 0000
Local GIC Hardware ID : 00000002
        Processor UID : 00000000
Flags (decoded below) : 00000001
    Processor Enabled : 1
 Parking Protocol Ver : 00000000
Performance Interrupt : 00000032
       Parked Address : 0000000000000000
         Base Address : 000000002C02F000

     [Above entry for all the cpus follows]

        Subtable Type : 0C [Generic Interrupt Distributor]
               Length : 18
             Reserved : 0000
Local GIC Hardware ID : 00000000
         Base Address : 000000002C010000
       Interrupt Base : 00000000
             Reserved : 00000000
```

```
gic: interrupt-controller@2c010000 {
  compatible = "arm,gic-400", "arm,cortex-a15-gic";
  reg = <0x0 0x2c010000 0 0x1000>,
        <0x0 0x2c02f000 0 0x2000>,
        <0x0 0x2c04f000 0 0x2000>,
        <0x0 0x2c06f000 0 0x2000>;
  #address-cells = <2>;
  #interrupt-cells = <3>;
  #size-cells = <2>;
  interrupt-controller;
  interrupts = <GIC_PPI 9 (GIC_CPU_MASK_SIMPLE(6)
                | IRQ_TYPE_LEVEL_HIGH)>;
  ranges = <0 0 0 0x2c1c0000 0 0x40000>;
  v2m_0: v2m@0 {
    compatible = "arm,gic-v2m-frame";
    msi-controller;
    reg = <0 0 0 0x1000>;
  };
};
```


ARM

Real examples(2): ACPI AML Table + DT

```
Device(ETH0) {
  Name(_HID, "ARMH9118")
  Name(_UID, Zero)
  Name(_CRS, ResourceTemplate() {
    Memory32Fixed(ReadWrite, 0x1A000000, 0x1000)
    Interrupt(ResourceConsumer, Level, ActiveHigh,
                    Exclusive) { 192 }
  }) // _CRS()
 }) // _DSD()
}  // Device()
```

```
ethernet@2,00000000 {
        compatible = "smsc,lan9118", "smsc,lan9115";
        reg = <2 0x00000000 0x10000>;
        interrupts = <3>;
        phy-mode = "mii";
        reg-io-width = <4>;
        smsc,irq-active-high;
        smsc,irq-push-pull;
        clocks = <&mb_clk25mhz>;
        vdd33a-supply = <&mb_fixed_3v3>;
        vddvario-supply = <&mb_fixed_3v3>;
};
```

ACPI: _DSD (Device Specific Data)

- Name(KEY0, "value0")
  - limits names ("KEY0") to 4 characters
  - maintenance issue + minimizing re-use
  - no registry for valid property values
  - backward compatibility as new hardware comes out
- DSD (Device Specific Data) introduced in ACPI 5.1
- Device Properties UUID: daffd814-6eba-4d8c-8a91-bc9bbf4aa301

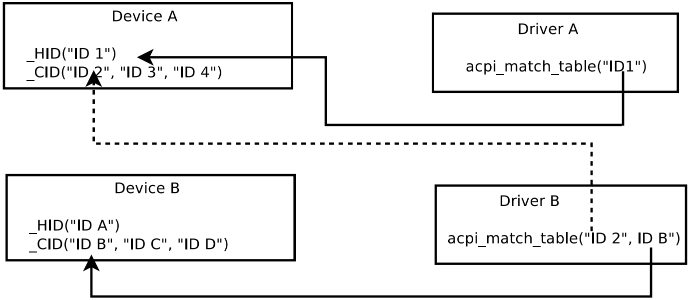
```
Name (_DSD, Package () {
  ToUUID("daffd814-6eba-4d8c-8a91-bc9bbf4aa301"), // Format identifier
  Package () {
    Package {"a-string-property", "A string"},
    Package {"a-string-list-property", Package {"first string", "second string"}};
    Package {"a-cell-property", Package {1, 2, 3, 4}};
  }
}
```

Real example: ACPI AML with _DSD

```
Device(ETH0) {
  Name(_HID, "ARMH9118")
  Name(_UID, Zero)
  Name(_CRS, ResourceTemplate() {
    Memory32Fixed(ReadWrite, 0x1A000000, 0x1000)
    Interrupt(ResourceConsumer, Level, ActiveHigh,
                    Exclusive) { 192 }
  }) // _CRS()
  Name(_DSD, Package() {
     ToUUID("daffd814-6eba-4d8c-8a91-bc9bbf4aa301"),
     Package() {
       Package(2) {"phy-mode", "mii"},
       Package(2) {"reg-io-width", 4 },
       Package(2) {"smsc,irq-active-high",1},
       Package(2) {"smsc,irq-push-pull",1}
     }
 }) // _DSD()
}  // Device()
```

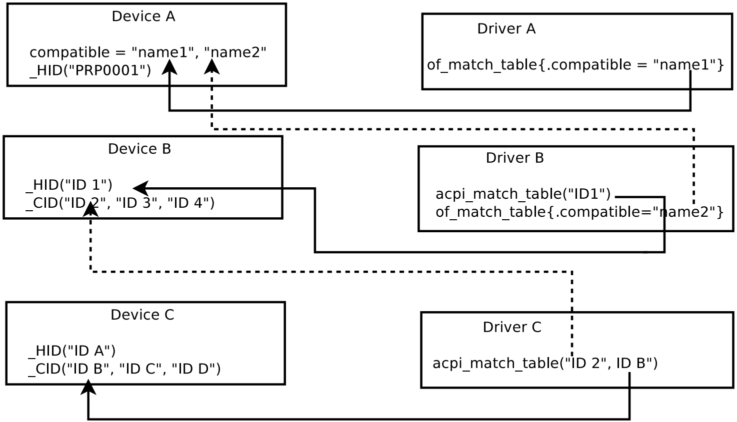
```
ethernet@2,00000000 {
        compatible = "smsc,lan9118", "smsc,lan9115";
        reg = <2 0x00000000 0x10000>;
        interrupts = <3>;
        phy-mode = "mii";
        reg-io-width = <4>;
        smsc,irq-active-high;
        smsc,irq-push-pull;
        clocks = <&mb_clk25mhz>;
        vdd33a-supply = <&mb_fixed_3v3>;
        vddvario-supply = <&mb_fixed_3v3>;
};
```

ACPI Driver lookup



ARM

Unified (ACPI + DT) Driver lookup



ARM

_DSD: Few Do's and Don'ts

Examples where _DSD can be used:

- MAC address or PHY for a networking device

Examples where _DSD must not be used include:

- dynamic device configurations(including hotplug)
- hardware abstraction through control methods
- power, performance and thermal management
- RAS interfaces

```
Device (FOO) {
  Name (_CRS, ResourceTemplate () {
    GpioIo (Exclusive, ..., IoRestrictionOutputOnly,
      "\\_SB.GPIO") {15} // red
    GpioIo (Exclusive, ..., IoRestrictionOutputOnly,
      "\\_SB.GPIO") {16} // green
    GpioIo (Exclusive, ..., IoRestrictionOutputOnly,
      "\\_SB.GPIO") {17} // blue
    GpioIo (Exclusive, ..., IoRestrictionOutputOnly,
      "\\_SB.GPIO") {1} // power
  })

  Name (_DSD, Package () {
    ToUUID("daffd814-6eba-4d8c-8a91-bc9bbf4aa301"),
    Package () {
      Package () {
        "led-gpios",
        Package () {
          ^FOO, 0, 0, 1,
          ^FOO, 1, 0, 1,
          ^FOO, 2, 0, 1,
        }
      },
      Package () {
        "power-gpios",
        Package () {^FOO, 3, 0, 0},
      },
    }
  })
}
```

ARM

_DSD: Don'ts (contd..)

- ACPI must not use kernel clock and regulator framework
- _DSD must not be used to represent data when they can provided using existing ACPI objects

Device Control Methods for PM

- _PSx method for entry to power state Dx and _ON/_OFF methods
- _PRx specifies which power resources a device needs in Dx

```
Device (BTKL)
{
  Name (_HID, "INT3420")
  Method (_PS0, 0, Serialized) // Power State 0
  {
      GL0A &= 0x7F
  }
  Method (_PS3, 0, Serialized) // Power State 3
  {
      GL0A |= 0x80
  }
  Method (_PSW, 1, NotSerialized) //Power State Wake
  {
      PSW (Arg0, 0x02)
  }
}
```

```
ethernet@2,00000000 {
        compatible = "smsc,lan9118", "smsc,lan9115";
        reg = <2 0x00000000 0x10000>;
        interrupts = <3>;
        phy-mode = "mii";
        reg-io-width = <4>;
        smsc,irq-active-high;
        smsc,irq-push-pull;
        clocks = <&mb_clk25mhz>;
        vdd33a-supply = <&mb_fixed_3v3>;
        vddvario-supply = <&mb_fixed_3v3>;
};
```

Conclusions / Recommendations

- Formalise _DSD proposal, review and maintenance process
  - New mailing list is setup : `dsd@acpica.org`
  - `https://lists.acpica.org/pipermail/dsd/2015-September/000026.html`
- Reporting, discussion and resolution of firmware issues
- SBSA/SBBR compliance check ?
- Limiting platform quirks
- More involvement of platform designers, firmware authors and kernel developers in specification

References

- Advanced Configuration and Power Interface Specification, Version 6.0
- R. J. Wysocki, ACPI 6 and Linux
- Device Tree Wiki
- LWN Article "ACPI for ARM?"
- Server Base System Architecture
- Server Base Boot Requirements, System Software on ARM Platforms
- Linux UEFI Validation
- luvOS on ARM64 - Linaro Wiki
- Documentation/arm64/arm-acpi.txt: Kernel Documentation - ACPI on ARMv8 Servers

Thank You

The Architecture for the Digital World®

ARM