ACPI from scratch: U-Boot implementation

Andy Shevchenko <andriy.shevchenko@intel.com>

October 5th, 2019.

Agenda

- Introduction to ACPI
- ACPI initializing phase
- U-Boot as a Firmware
- Upstreaming effort
- Demo
- References
- Q&A

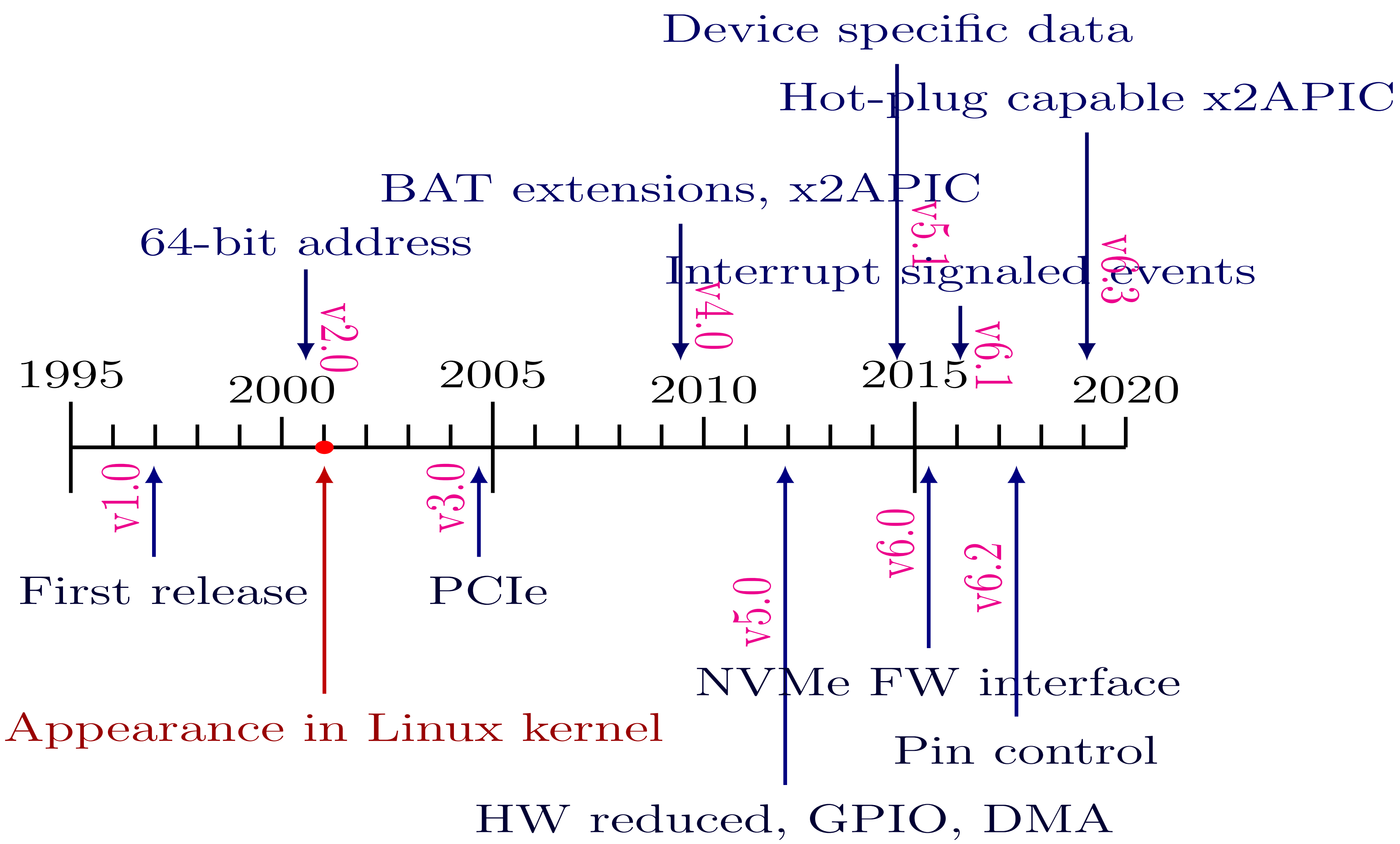
Introduction to ACPI

Quick introduction to ACPI

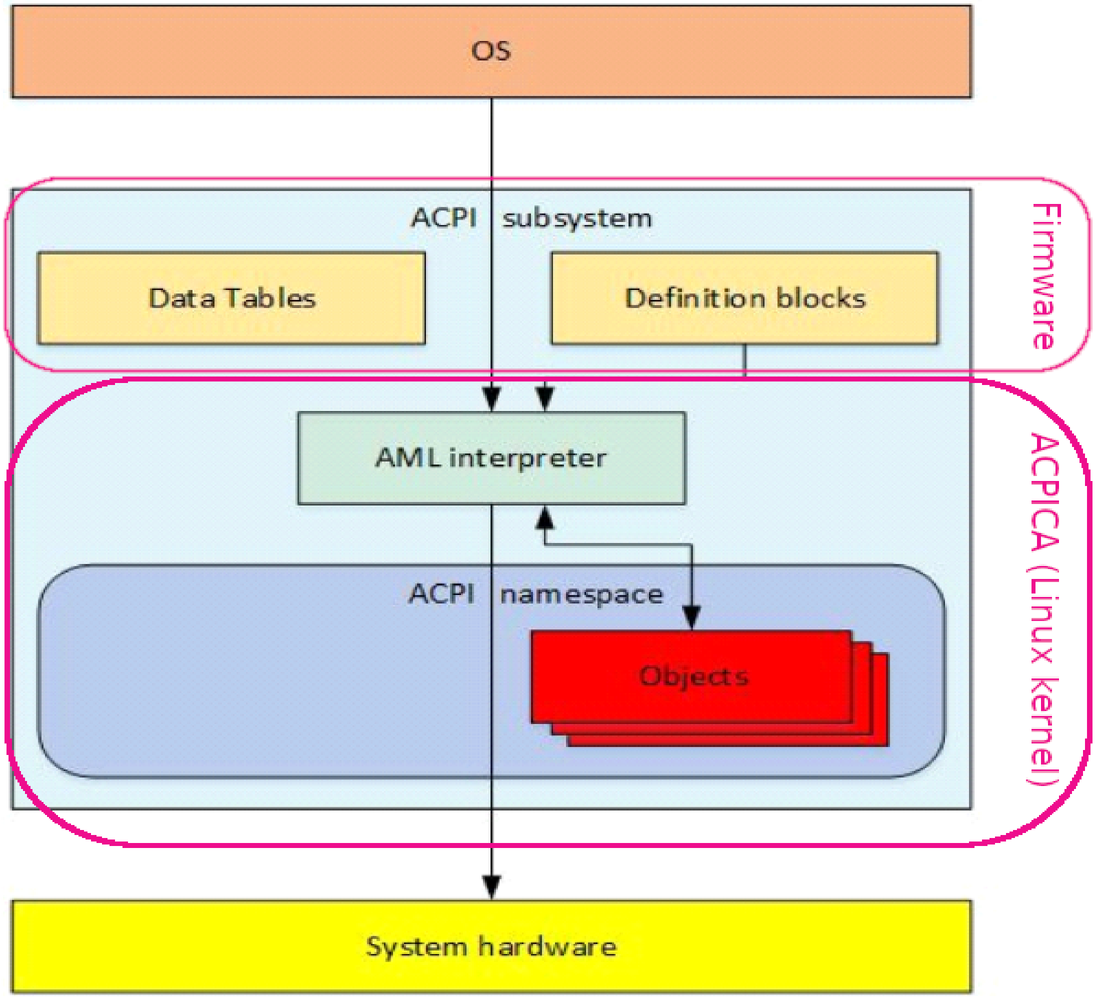
Introduction to ACPI: What is this for?

- Advanced Configuration and Power Interface
  - Platform independent way to suspend and resume the devices and entire system
  - Interface, agnostic to power management hardware
  - Common way to discover and configure platform devices
- Evolving along with the corresponding power management hardware
  - Handling hot plug for traditional buses like PCI (v1.0)
  - Handling hardware reduced platforms like EC-less (v5.0)
- Direction towards SoC
  - Improvement in device discovery and configuration (I^2C, SPI, …) (v5.0)
  - Support device properties (v5.1)
  - Pin control and configuration (v6.2)

Introduction to ACPI: Evolution of the specification



Introduction to ACPI: Interaction with system hardware



The ACPI subsystem consists of two types of data structures: *data tables* and *definition blocks*.

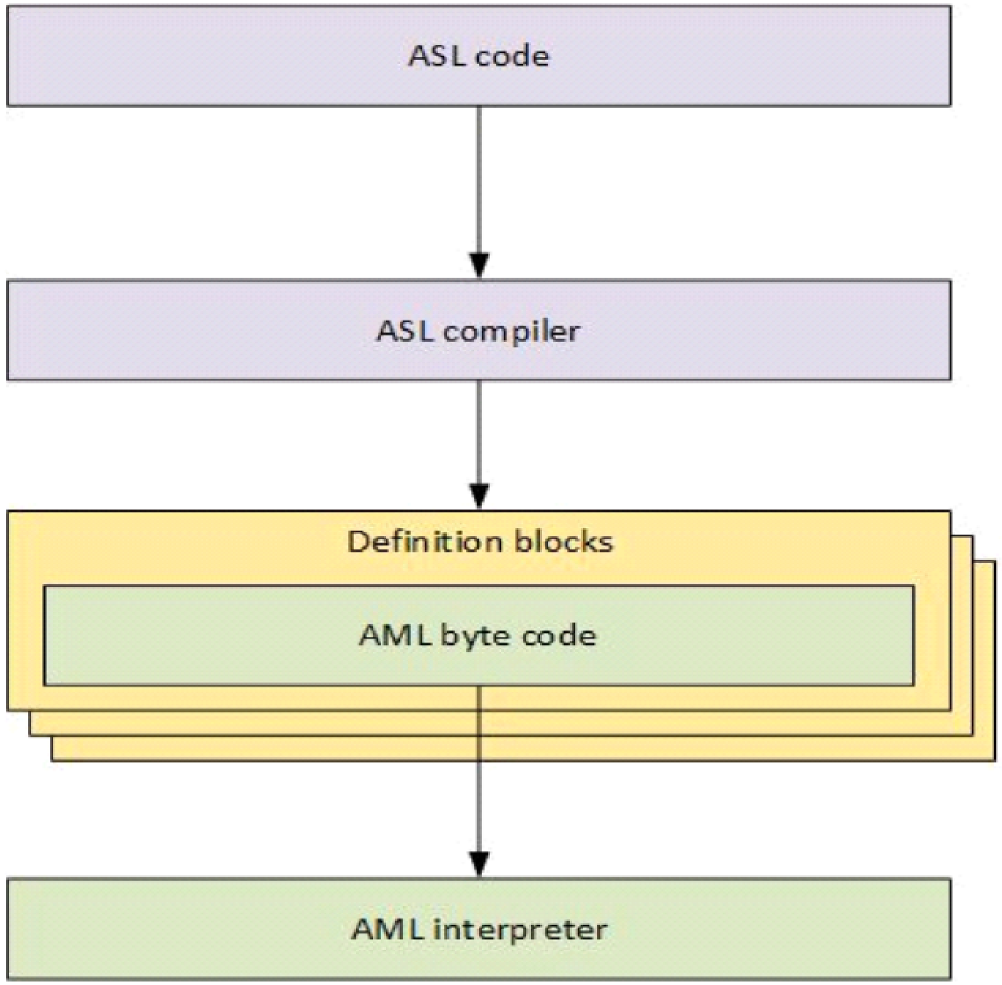
Upon initialization, the *AML interpreter* extracts the byte code in the definition blocks as enumerable objects.

This collection of enumerable objects forms the OS construct called the *ACPI namespace*.

Objects can either have a directly defined value or must be evaluated and interpreted by the AML interpreter.

The AML interpreter, directed by the *OS*, evaluates objects and interfaces with *system hardware* to perform necessary operations.

Introduction to ACPI: From ASL to AML



ACPI Source Langauge (ASL) code is used to define objects and control methods.

The ASL compiler translates ASL into ACPI Machine Language (AML) byte code contained within the ACPI definition blocks.

Definition blocks consist of an identifying table header and byte code that is executable by an AML interpreter.

Introduction to ACPI: Example of ASL and AML

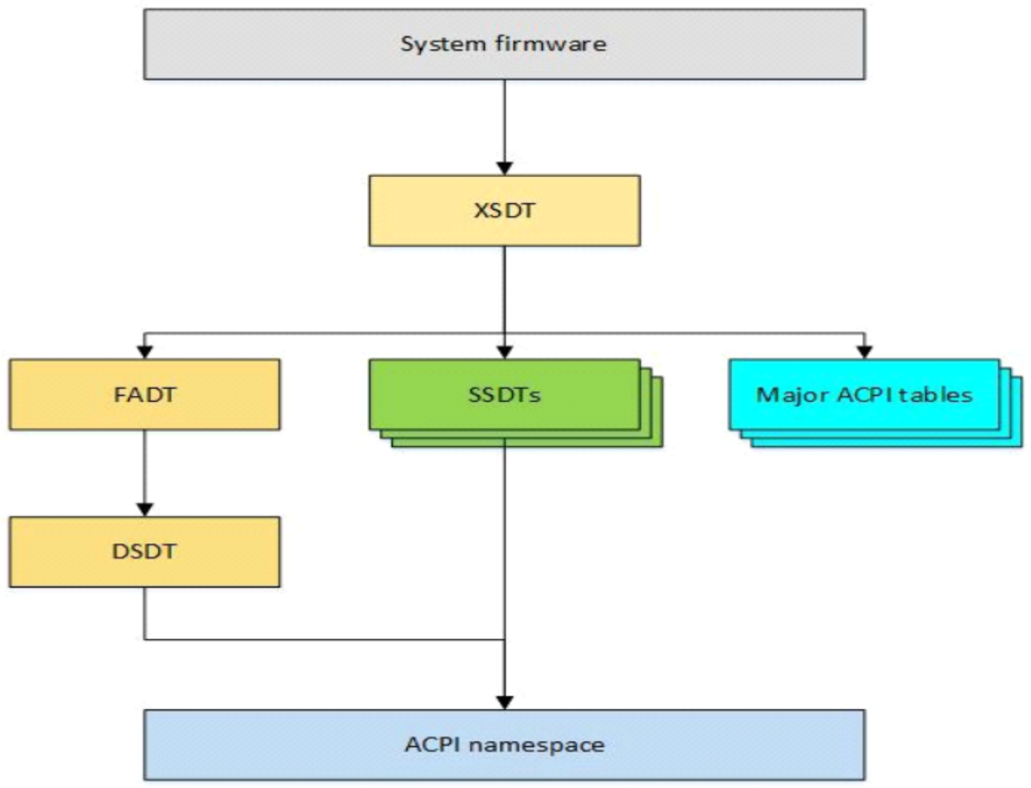
```
Device (GPO1) {
    Name (_HID, "PRP0001")
    Name (_DDN, "74X164 8-bits shift register GPIO expander")
    Name (_CRS, ResourceTemplate () {
        SpiSerialBus (
            1,                      // Chip select
            PolarityLow,            // Chip select is active low
            FourWireMode,           // Full duplex
            8,                      // Bits per word is 8 (byte)
            ControllerInitiated,    // Don't care
            1000000,                // 1 MHz
            ClockPolarityLow,       // SPI mode 0
            ClockPhaseFirst,        // SPI mode 0
            "\\_SB.PCI0.SPI5",      // SPI host controller
            0                       // Must be 0
        )
    })
    Name (_DSD, Package () {
        ToUUID("daffd814-6eba-4d8c-8a91-bc9bbf4aa301"),
        Package () {
            Package () { "compatible", Package () { "fairchild,74hc595" } },
            Package () { "registers-number", 1 },
        }
    })
}
```

```
0000 53 53 44 54 1c 01 00 00 05 48 00 00 SSDT.....H..
000c 00 00 00 00 37 34 58 31 36 34 00 00 ....74X164..
0018 01 00 00 00 49 4e 54 4c 03 07 19 20 ....INTL...
0024 a0 14 00 15 5c 2f 03 5f 53 42 5f 50 ....\/._SB_P
0030 43 49 30 53 50 49 35 06 00 10 42 0e CI0SPI5...B.
003c 5c 2f 03 5f 53 42 5f 50 43 49 30 53 \/._SB_PCI0S
0048 50 49 35 5b 82 4f 0c 47 50 4f 31 08 PI5[.O.GPO1.
0054 5f 48 49 44 0d 50 52 50 30 30 30 31 _HID.PRP0001
0060 00 08 5f 44 44 4e 0d 37 34 58 31 36 .._DDN.74X16
006c 34 20 38 2d 62 69 74 73 20 73 68 69 4 8-bits shi
0078 66 74 20 72 65 67 69 73 74 65 72 20 ft register
0084 47 50 49 4f 20 65 78 70 61 6e 64 65 GPIO expande
0090 72 00 08 5f 43 52 53 11 29 0a 26 8e r.._CRS.).&.
009c 21 00 01 00 02 02 00 00 01 09 00 40 !..........@
00a8 42 0f 00 08 00 00 01 00 5c 5f 53 42 B.......\_SB
00b4 2e 50 43 49 30 2e 53 50 49 35 00 79 .PCI0.SPI5.y
00c0 00 08 5f 44 53 44 12 45 05 02 11 13 .._DSD.E....
00cc 0a 10 14 d8 ff da ba 6e 8c 4d 8a 91 .......n.M..
00d8 bc 9b bf 4a a3 01 12 3d 02 12 24 02 ...J...=..$.
00e4 0d 63 6f 6d 70 61 74 69 62 6c 65 00 .compatible.
00f0 12 15 01 0d 66 61 69 72 63 68 69 6c ....fairchil
00fc 64 2c 37 34 68 63 35 39 35 00 12 15 d,74hc595...
0108 02 0d 72 65 67 69 73 74 65 72 73 2d ..registers-
0114 6e 75 6d 62 65 72 00 01             number..
```

ACPI Initialization phase

What tables ACPI is using during initialization phase

ACPI initialization phase: Path to ACPI namespace



System firmware updates the ACPI tables as necessary with information only available at runtime before handing off control to the boostrap loader.

The *XSDT* is the first table used by the OS's ACPI subsystem and contains the addresses of most of the other ACPI tables on the system.

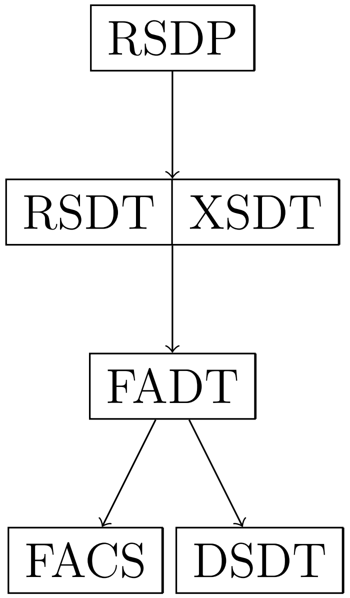
The *XSDT* points to the *FADT*, the *SSDTs*, and other *major ACPI tables*.

The FADT directs the ACPI subsystem to the *DSDT*, which is the beginning of the namespace by virtue of being the first table that contains a definition block.

The ACPI subsystem then consumes the DSDT and begins building the *ACPI namespace* from the definition blocks. The XSDT also points to the SSDTs and adds them to the namespace.

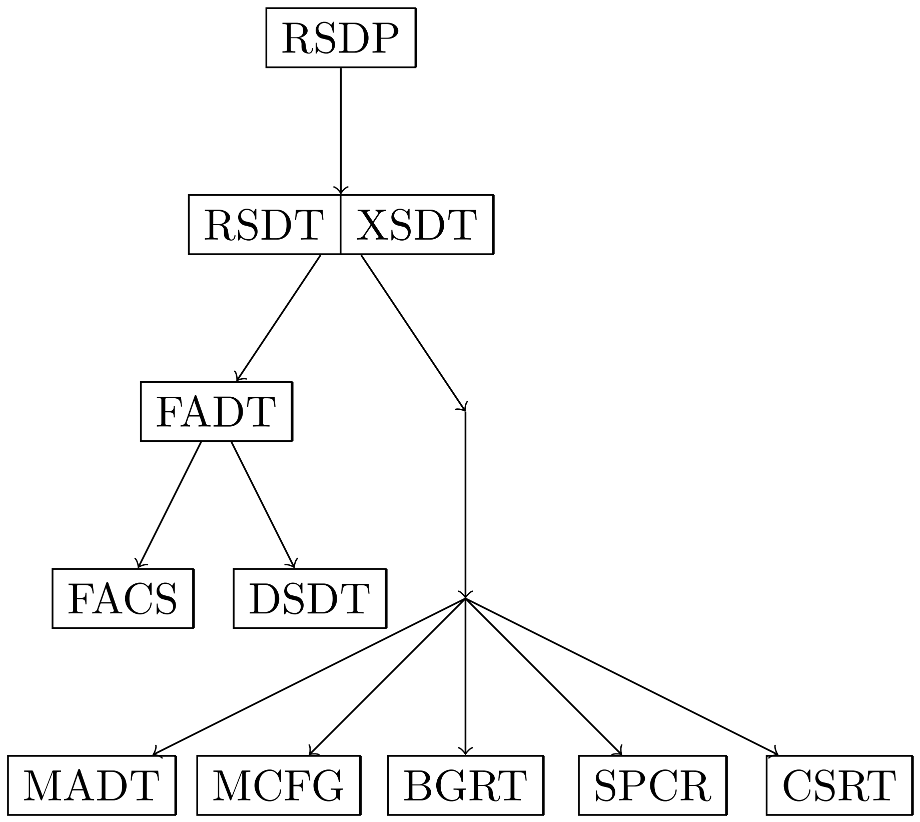
ACPI initialization phase: Mandatory tables

- Step 1: RSDP
  - Root System Device Pointer
- Step 2 (32-bit): RSDT
  - Root System Description Table
- Step 2 (64-bit): XSDT
  - Extended System Description Table
- Step 3: FADT (FACP, FACS, DSDT)
  - Fixed ACPI Description Table
  - Firmware ACPI Control Structure
  - Differentiated System Description Table



ACPI initialization phase: Optional tables (example)

- Platforms with Interrupt Controller
  - Step 4: Multiple APIC Description Table (MADT)
- Platform with PCI bus
  - Step 5: Memory mapped Configuration space base address description table (MCFG)
- Platform agnostic
  - Boot Graphics Resource Table (BGRT)
  - ...
- UEFI ACPI Support (extensions)
  - Step 6: Serial Port Console Redirection (SPCR)
  - Step 7: Core System Resource Table (CSRT)



U-Boot as a Firmware

U-Boot as a Firmware in order to provide ACPI

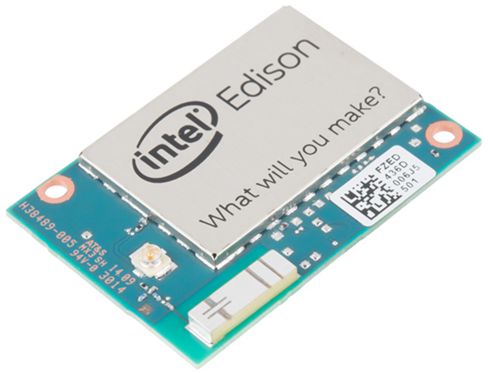
UaaF: Intel Merrifield and Intel Tangier

Platform

- Intel Merrifield
  - PMIC (no ACPI PM hardware)
  - System Firmware Interface
  - U-Boot
  - Mobile phone and tablet market

SoC

- Intel Tangier
  - x86_64 Dual-Core Atom CPU
  - PowerVR
  - Intel SST
  - I²C, SPI, HS UART, GPDMA, ...

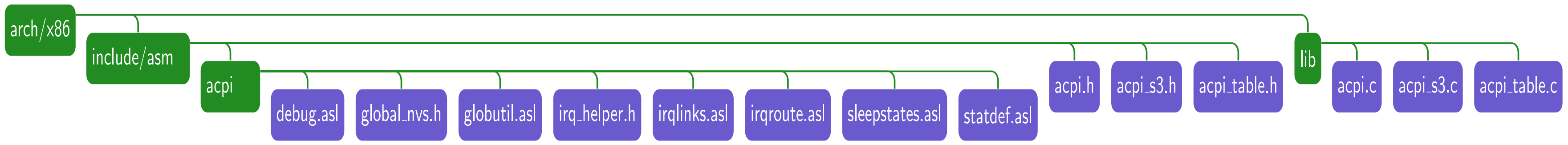


UaaF: The Universal Boot loader

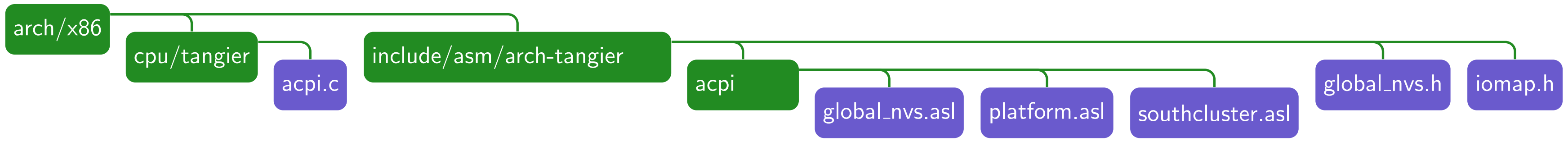
- About
  - Small executable to initialize minimum hardware to load Linux
  - Can run on bare metal or as classical variant (invoked by firmware)
- Features
  - x86 support
  - ACPI structure from scratch (generating tables)
  - Well maintained
- Limitations
  - ACPI support only for x86

UaaF: Source tree structure

Platform agnostic (x86)

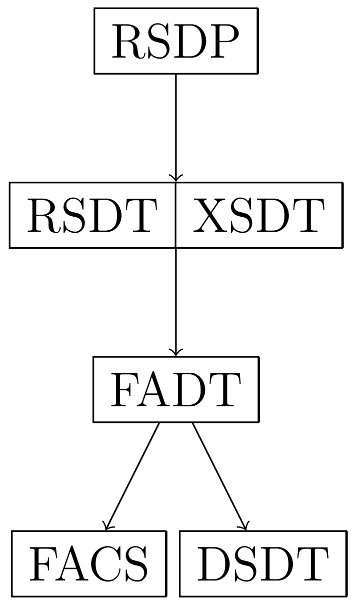


Platform dependent (Intel Tangier)



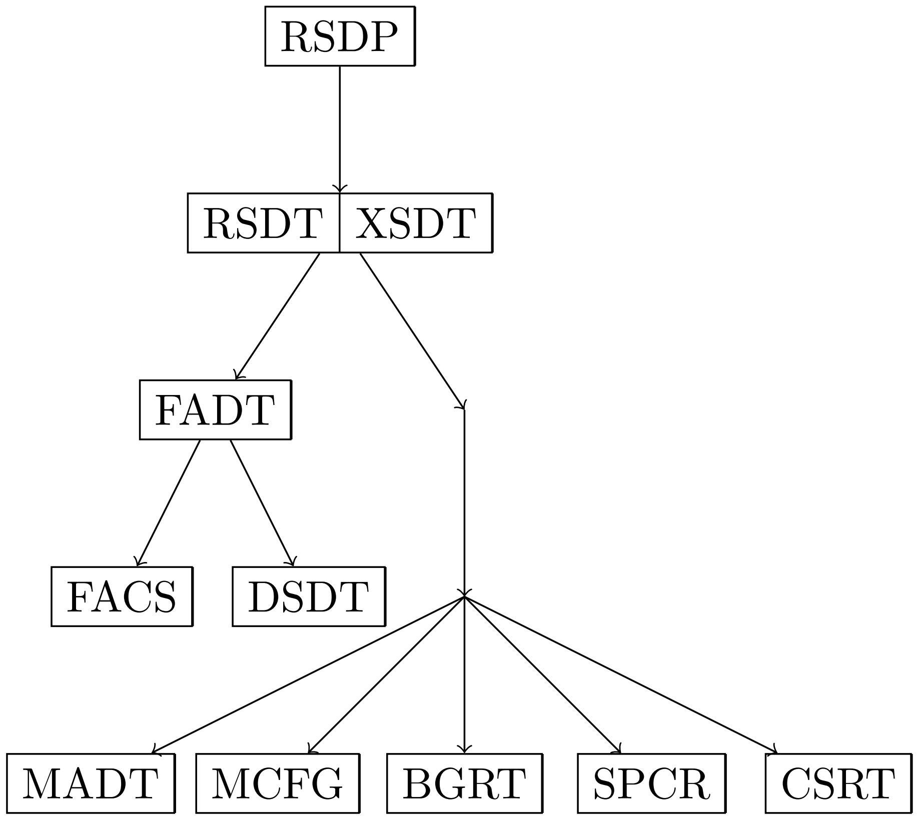
Mandatory tables

- Step 1: RSDP
  - Root System Device Pointer
- Step 2 (32-bit): RSDT
  - Root System Description Table
- Step 2 (64-bit): XSDT
  - Extended System Description Table
- Step 3: FADT (FACP, FACS, DSDT)
  - Fixed ACPI Description Table
  - Firmware ACPI Control Structure
  - Differentiated System Description Table



Optional tables (example)

- Platforms with Interrupt Controller
  - Step 4: Multiple APIC Description Table (MADT)
- Platform with PCI bus
  - Step 5: Memory mapped Configuration space base address description table (MCFG)
- Platform agnostic
  - Boot Graphics Resource Table (BGRT)
  - ...
- UEFI ACPI Support (extensions)
  - Step 6: Serial Port Console Redirection (SPCR)
  - Step 7: Core System Resource Table (CSRT)



UaaF: In platform dependent code (C code)

- arch/x86/cpu/<$CPU>/acpi.c
  - Step 3 (FADT, Global NVS, DSDT): acpi_create_fadt() and acpi_create_gnvs()
  - Step 4 (MADT): __weak acpi_fill_madt()
  - Step 5 (MCFG): __weak acpi_fill_mcfg()
  - Step 7 (CSRT): __weak acpi_fill_csrt()
- arch/x86/include/asm/arch-<$CPU>/
  - iomap.h
  - global_nvs.h (Global NVS for C side of interface)

UaaF: In platform dependent code (C code)

- arch/x86/cpu/<$CPU>/acpi.c
  - Step 3 (FADT, Global NVS, DSDT): acpi_create_fadt() and acpi_create_gnvs()
  - Step 4 (MADT): __weak acpi_fill_madt()
  - Step 5 (MCFG): __weak acpi_fill_mcfg()
  - Step 7 (CSRT): __weak acpi_fill_csrt()
- arch/x86/include/asm/arch-<$CPU>/
  - iomap.h
  - global_nvs.h (Global NVS for C side of interface)

UaaF: In platform dependent code (ASL code)

- arch/x86/include/asm/arch-<$CPU>/acpi/global_nvs.asl
  - Global NVS for ACPI side of interface
- arch/x86/include/asm/arch-<$CPU>/acpi/southcluster.asl
  - Defines PCI host bridge
  - Defines South Cluster devices
  - Defines other devices behind PCI bus
- arch/x86/include/asm/arch-<$CPU>/acpi/platform.asl
  - Includes Global NVS
  - Includes South Cluster devices
  - Defines hooks, such as _PTS() and _WAK() for system sleep states
  - Defines platform devices which are not falling to the above category
  - Container for the rest of ASL methods and objects in _SB scope

UaaF: In platform dependent code (ASL code)

- arch/x86/include/asm/arch-<$CPU>/acpi/global_nvs.asl
  - Global NVS for ACPI side of interface
- arch/x86/include/asm/arch-<$CPU>/acpi/southcluster.asl
  - Defines PCI host bridge
  - Defines South Cluster devices
  - Defines other devices behind PCI bus
- arch/x86/include/asm/arch-<$CPU>/acpi/platform.asl
  - Includes Global NVS
  - Includes South Cluster devices
  - Defines hooks, such as _PTS() and _WAK() for system sleep states
  - Defines platform devices which are not falling to the above category
  - Container for the rest of ASL methods and objects in _SB scope

UaaF: In platform dependent code (ASL code)

- arch/x86/include/asm/arch-<$CPU>/acpi/global_nvs.asl
  - Global NVS for ACPI side of interface
- arch/x86/include/asm/arch-<$CPU>/acpi/southcluster.asl
  - Defines PCI host bridge
  - Defines South Cluster devices
  - Defines other devices behind PCI bus
- arch/x86/include/asm/arch-<$CPU>/acpi/platform.asl
  - Includes Global NVS
  - Includes South Cluster devices
  - Defines hooks, such as _PTS() and _WAK() for system sleep states
  - Defines platform devices which are not falling to the above category
  - Container for the rest of ASL methods and objects in _SB scope

Upstreaming effort

Upstreaming effort of U-Boot changes to support ACPI on new platform

Upstreaming effort: Platform bring up

- Intel Merrifield platform bring up (from 2017-02-05 till 2017-07-06)
  - 446d4e048ee3 (“x86: make LOAD_FROM_32_BIT visible for platforms”)
  - 7a96fd8ef002 (“x86: Introduce INTEL_MID quirk option”)
  - 308c75e08dea (“x86: Intel MID platforms has no microcode update”)
  - c5f8dd482b41 (“serial: Add serial driver for Intel MID”)
  - ca0d29e4f060 (“x86: Introduce minimal PMU driver for Intel MID platforms”)
  - 495f3774be68 (“x86: Add Intel Edison board files”)
  - …
- 664 insertions(+), 11 deletions(-)
- U-Boot v2017.09

Upstreaming effort: ACPI preparations

- ACPI related preparations (from 2017-07-21 till 2017-07-28)
  - 06054b1a623d (”Makefile: Don't shadow actual error when compile ASL”)
  - e7aa9c294759 (“Makefile: Export build date as integer”)
  - 684c4cd01125 (“x86: acpi: Fill OEM revision”)
  - 2dcbef6f6c43 (“x86: acpi: Name fields in FADT in accordance with specification”)
  - b156da91fb1c (“x86: acpi: Deduplicate acpi_fill_madt() implementation”)
  - ace7762b2854 (“x86: acpi: Export acpi_fill_mcfg() with __weak attribute”)
  - 382fabb2974a (“x86: acpi: Don't touch hardware on HW reduced platforms”)
  - c3df28f6e2fc (“x86: Make table address selectable”)
- 61 insertions(+), 72 deletions(-)
- U-Boot v2017.09

Upstreaming effort: Basic ACPI support

- Basic ACPI support for Intel Edison (from 2017-10-03 till 2018-01-10)
  - 39665beed6f7 (“x86: tangier: Enable ACPI support for Intel Tangier”)
  - 256df1e1c666 (“x86: edison: Bring minimal ACPI support to the board”)
  - 1602d215b595 (“x86: tangier: Use official ACPI HID for FLIS IP”)
  - d08953e04596 (“x86: tangier: Use actual GPIO hardware numbers”)
  - 5d8c4ebd95e2 (“x86: tangier: Add Bluetooth to ACPI table”)
  - 3ffb33d6362b (“x86: tangier: Make _CRS for BTH0 Serialized to avoid warning”)
  - 378960d8c2c7 (“x86: zImage: Move subarch assignment out of cmd_line check”)
- 544 insertions(+), 18 deletions(-)
- U-Boot v2018.03

Upstreaming effort: Serial Port Console Redirection

- SPCR support for x86 (from 2018-11-20 till 2018-11-20)
  - ac7f5db9dc69 (“dm: serial: Add ->getconfig() callback”)
  - d5bb4f862b47 (“dm: serial: Introduce ->getinfo() callback”)
  - 0af761620f2b (“serial: ns16550: Group reg_* members of ns16550_platdata”)
  - 4e7207791c05 (“serial: ns16550: Read reg-io-width from device tree”)
  - 50bf7d03c26b (“serial: ns16550: Provide ->getinfo() implementation”)
  - f3275aa4a17f (“x86: acpi: Add SPCR table description”)
  - b288cd960072 (“x86: acpi: Generate SPCR table”)
- 344 insertions(+), 5 deletions(-)
- U-Boot v2019.01

Upstreaming effort: Core System Resource Table and extra changes

- CSRT support (from 2019-07-14 till 2019-07-14, U-Boot v2019.10)
  - eef30079454c (“x86: acpi: Add CSRT description”)
  - ddd2a4244c26 (“x86: acpi: Introduce a stub to generate CSRT”)
  - e3be52ceac7f (“x86: acpi: Enable ACPI companion for Intel iDMA 32-bit”)
  - 5e99fde34a77 (“x86: tangier: Populate CSRT for shared DMA controller”)
- Miscellaneous changes (from 2018-11-10 till 2019-02-26, U-Boot v2019.04)
  - f1b8641fd48d (“x86: acpi: Enable RTC for Intel Tangier”)
  - 24b56e2bf369 (“x86: tangier: Add initial ACPI support for PMIC device”)
  - 1d2825aa30b6 (“x86: acpi: Add DMA descriptors for SPI5 on Intel Tangier”)
  - c652dd15571e (“x86: acpi: Add DMA descriptors for I2C1 on Intel Tangier”)
- 238 insertions(+), 0 deletions(-)

Demo

Demo slides

Demo: ACPI tables in memory (firmware)

```
RSDP 0x000000008D2FE014 000024 (v02 INTEL )

XSDT 0x000000008D2AA188 0000EC (v01 INTEL  KBL      00000000
01000013)

FACP 0x000000008D2F2000 0000F4 (v05 INTEL  KBL      00000000 MSFT
0000005F)

DSDT 0x000000008D2BC000 032C2A (v02 INTEL  SKL      00000000 INTL
20160422)

APIC 0x000000008D2F0000 0000BC (v03 INTEL  KBL      00000001 MSFT
0000005F)

MCFG 0x000000008D2EF000 00003C (v01 INTEL  KBL      00000001 MSFT
0000005F)

UEFI 0x000000008D28F000 000042 (v01 INTEL  EDK2     00000002
01000013)

HPET 0x000000008D2F1000 000038 (v01 INTEL  KBL      00000001 MSFT
0000005F)
```

Demo: ACPI tables in memory (U-Boot)

```
RSDP 0x00000000000E4500 000024 (v02 U-BOOT)

XSDT 0x00000000000E45E0 00004C (v01 U-BOOT U-BOOTBL 20190913 INTL
00000000)

FACP 0x00000000000E5310 0000F4 (v06 U-BOOT U-BOOTBL 20190913 INTL
00000000)

DSDT 0x00000000000E4780 000A8A (v02 U-BOOT U-BOOTBL 00010000 INTL
20190509)

APIC 0x00000000000E5410 000048 (v04 U-BOOT U-BOOTBL 20190913 INTL
00000000)

MCFG 0x00000000000E5460 00003C (v01 U-BOOT U-BOOTBL 20190913 INTL
00000000)

CSRT 0x00000000000E54A0 000058 (v00 U-BOOT U-BOOTBL 20190913 INTL
00000000)

SPCR 0x00000000000E5500 000050 (v02 U-BOOT U-BOOTBL 20190913 INTL
00000000)
```

Demo: Interrupt controllers initialization (U-Boot)

```
ACPI: Local APIC address 0xfee00000

ACPI: Local APIC address 0xfee00000

ACPI: LAPIC (acpi_id[0x00] lapic_id[0x00] enabled)

ACPI: LAPIC (acpi_id[0x02] lapic_id[0x02] enabled)

ACPI: IOAPIC (id[0x02] address[0xfec00000] gsi_base[0])

Using ACPI (MADT) for SMP configuration information
```

Demo: Dynamic load of SSDT (U-Boot)

```
Host-directed Dynamic ACPI Table Load:

SSDT 0xFFFFA0E0062A0000 001820 (v05        HD44780  00000001
INTL 20190703)
```

Summary

- Current ACPI v6.3 covers System-on-Chips and embedded platforms
- U-Boot can be used as ACPI provider
- Less than three months to bring up basic ACPI support in U-Boot

References

Useful resources to continue with

References

- ACPI Specification v6.3 (released January 2019) or newer
- ACPI support documents: https://www.uefi.org/acpi
- U-Boot support
  - Official Git repository https://gitlab.denx.de/u-boot/u-boot
  - Official mailing list: https://lists.denx.de/listinfo/u-boot
- Intel Edison community support
  - GitHub repositories: https://github.com/edison-fw
  - Wiki (user): https://edison-fw.github.io/meta-intel-edison/
  - Wiki (developer): https://edison.internet-share.com/wiki/Main_Page
  - U-Boot repository: https://github.com/andy-shev/u-boot/tree/edison-acpi

Thank you!

Questions and Answers

Legal Information

intel®

experience
what's inside™