WTF is ACPI?

Ivan Delalande

colona@lse.epita.fr
http://lse.epita.fr/

July 18, 2012

History

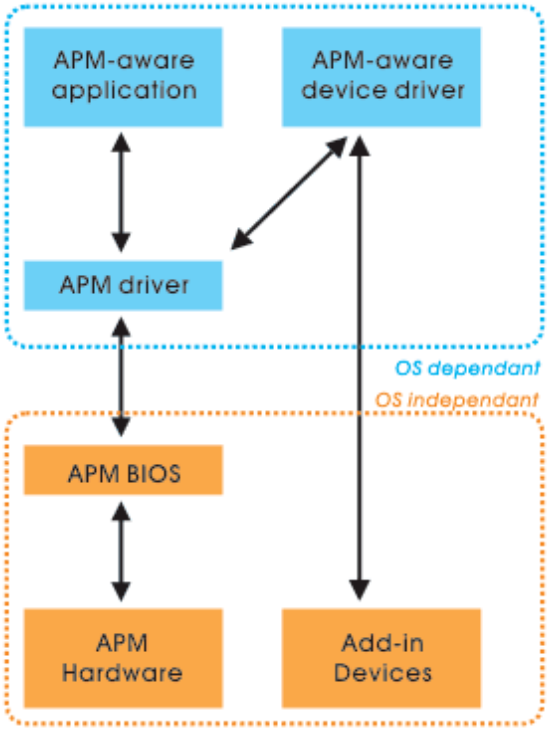


Figure : APM: Advanced Power Management, 1992

- executed by the BIOS,
- quick and dirty.

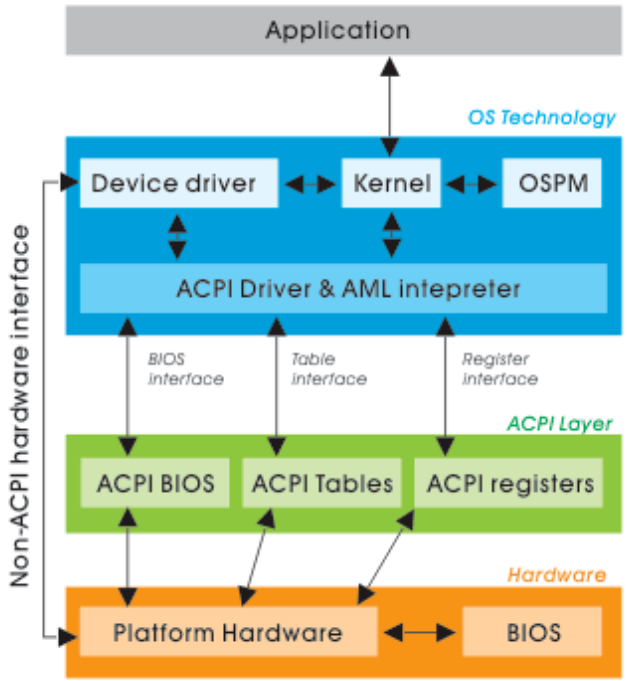


Figure : ACPI: Advanced Configuration and Power Interface, 1996

- executed by the OS,
- abstract and complex.

What is ACPI?

- Advanced Control Power Interface,
- An interface between the PC hardware and software (kernel) to manage power,
- But can also be used to get hardware description,
- And Plug-And-Play devices discovery,
- And as an event handler,
- And...

ACPI Specification

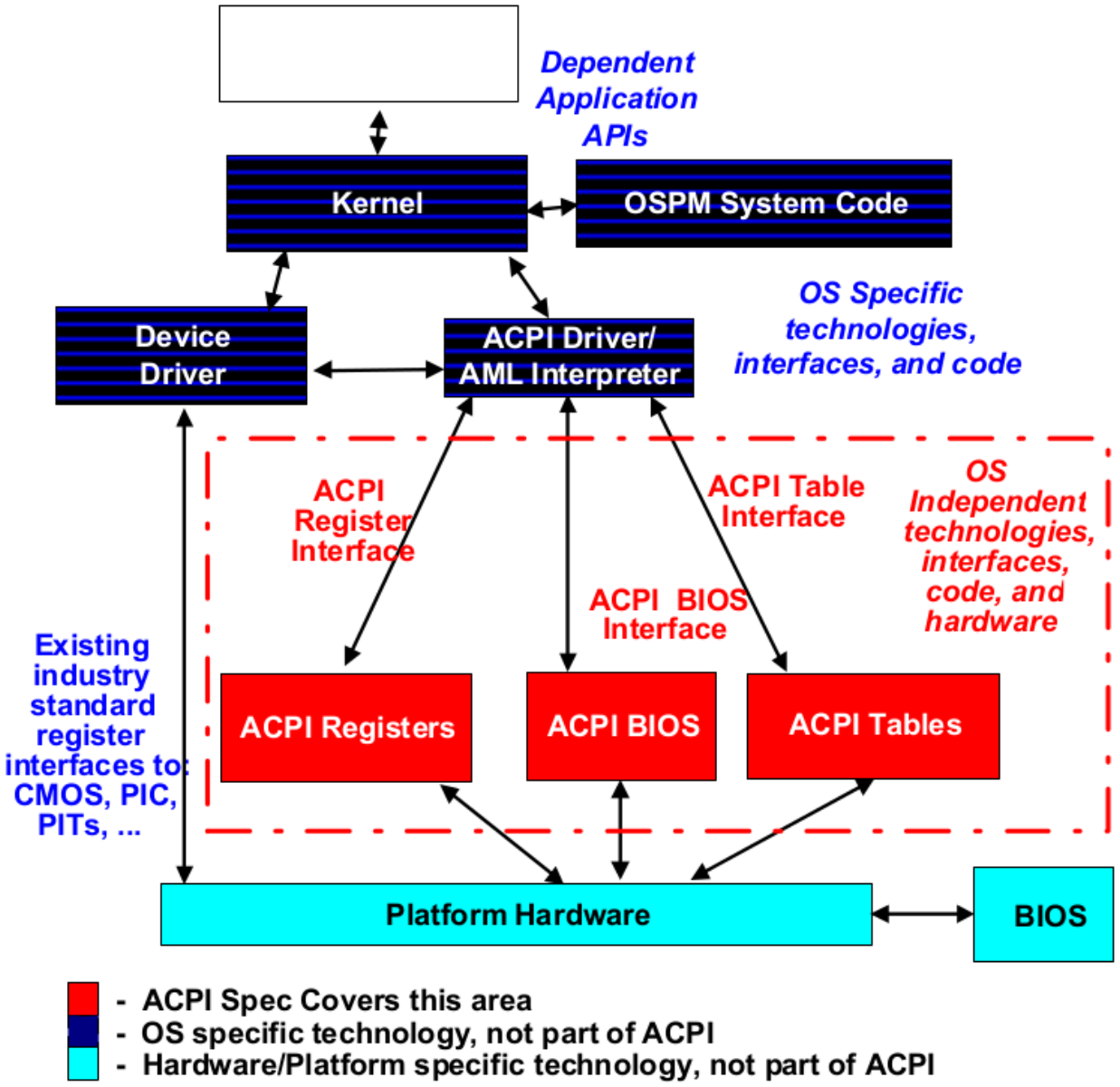
- Document written by Intel, HP, Microsoft, Phoenix and Toshiba,
- 12 versions, *Since 1996*,
- 5th major version, published on December 6, 2011,
- 958 pages.

Example of details in the specification

Example of the requirements for a light sensor device:

- _ALI: illuminance,
- _ALT: light temperature (color),
- _ALC: chromaticity,
- _ALR: light response,
- _ALP: polling frequency,
- _BCL: brightness control.

ACPI overview

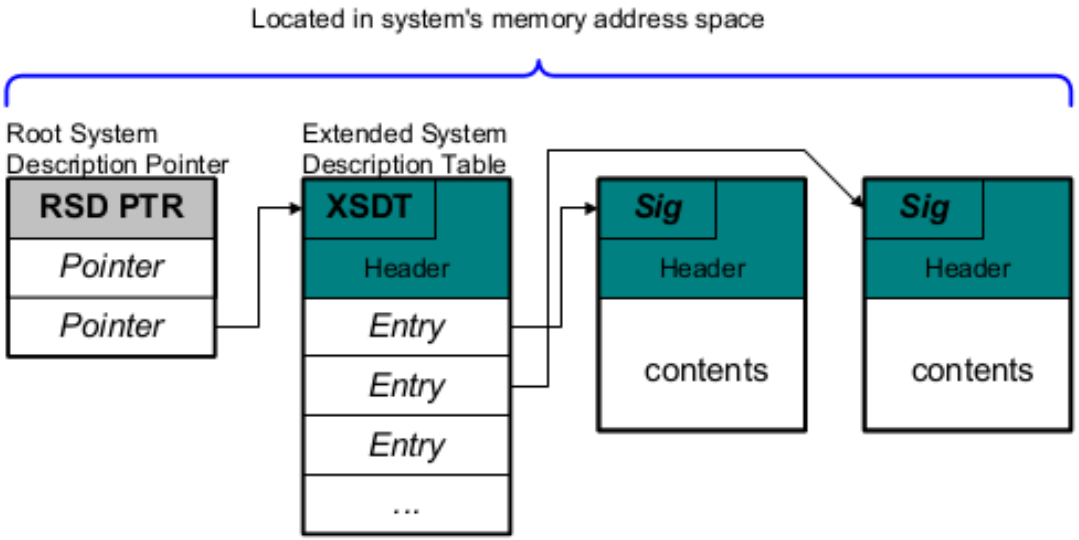


At the OS level

To support ACPI, a kernel has to:

- find the RSDP,
- load and parse ACPI tables,
- find, parse and load the DSDT, which contains the ACPI methods,
- execute ACPI methods,
- handle ACPI events,
- expose features to the user (power and thermal status, control of the computer cooling. . .).

ACPI tables

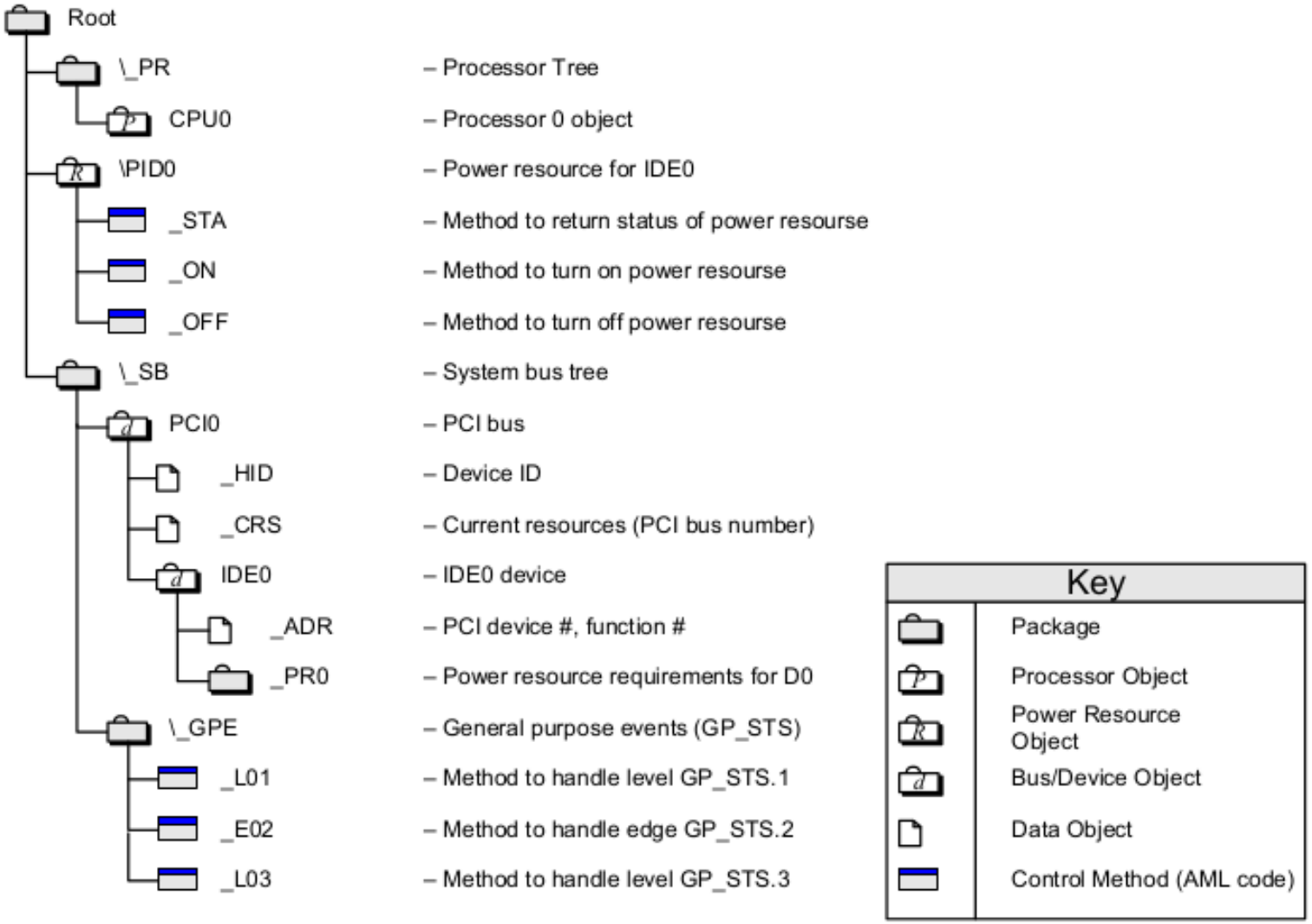


- RSDP might be found between 0xA0000 and 0xA0400 or between 0xE0000 and 0xFFFFF in th Extended BIOS Data
- RSDT and XSDT are an array of pointers to other tables,
- DSDT and SSDT are for ACPI method definition,
- FACP and FACS for various static configuration,
- and 41 other tables types (firware, APIC, Smart Battery, . . .).

ASL: the ACPI source language

- Turing-complete,
- hierarchical source-code,
- 110 opcodes, 35 macros.
- compiled to AML bytecode,
- executed by the kernel.

ASL namespaces



ASL namespace

- Every object has a 4-letter name,
- objects can be referenced by their absolute or relative path,
- relative objects name look-up is recursive,
- various types of objects: names, methods, devices, thermal zones. . .

ASL data types

20 data types:

- *Uninitialized*,
- Integer,
- String,
- Buffer, Buffer Field,
- Object Reference.
- Operation Region,
- Mutex.
- Event,
- ...

ASL: Opcodes

- Usual opcodes: If, While, Add, Sleep, ...
- Namespace-related: Name, Device, Processor, ThermalZone, ...
- Type-related: ToBuffer, ToString, ...
- Memory-related: RefOf, DeRefOf, CreateField, OperationRegion, ...

Example: strcmp(3)

```
1  Method (SCMP, 2, NotSerialized)
2  {
3      Store (Zero, Local0)
4      Store (SizeOf(Arg0), Local1)
5      Store (SizeOf(Arg1), Local2)
6      If (LLess(Local2, Local1))
7      {
8          Store (Local2, Local1)
9      }
10
11     While (LAnd (LLess (Local0, Local1),
12         LEqual (Index (DerefOf (Arg0), Local0),
13                 Index (DerefOf (Arg1), Local0))))
14     {
15         Add(One, Local0, Local0)
16     }
17
18     If (LEqual (Local0, Local1))
19         { Return (Zero) }
20     Else
21         { Return (One) }
22 }
```

ASL is high level

- Logical comparison operators works with integer, string or raw data buffer.
  - So the previous example could be simplified as: `Return (LEqual (Arg0, Arg1))`
- Implicit cast between data types, acting as an `atoi`, `sprintf`, depending on what type is needed.
- `SizeOf` act as classic C `sizeof` on Integers, and as `strlen` on Strings.
- `Match (Src, Op1, Arg1, Op2, Arg2, Ind)` $\Longleftrightarrow$ `(Src[i] Op1 Arg1) and (Src[i] Op2 Arg2)`

AML: ACPI machine language

```
1  Method (SCMP, 2, NotSerialized) {
2      Store (Zero, Local0)
3      Store (SizeOf (Arg0), Local1)
4      Store (SizeOf (Arg1), Local2)
5      If (LGreater(Local2, Local1)) { Store (Local2, Local1) }
6      While (LAnd (LLess (Local0, Local1),
7              LEqual (Index (DerefOf (Arg0), Local0),
8                      Index (DerefOf (Arg1), Local0))))
9          { Add (One, Local0, Local0) }
10     If (LEqual (Local0, Local1)) { Return (Zero) }
11     Else { Return (One) }
12 }
13 Method (METH, 1, Serialized) {
14     Return (SCMP (Arg0, "foo"))
15 }
```

```
73747263 70000000 01B74C4F 4C544F50   strcp.....LOLTOP
53455249 414C0000 10020000 494E544C   SERIAL......INTL
20061220 14395343 4D500270 00607087    .. .9SCMP.p.‘p.
68617087 6962A007 94626170 6261A214   hap.ib...bapba..
90956061 93888368 60008883 69600072   ..‘a...h‘...i‘.r
016060A0 06936061 A400A103 A4011411   .‘‘...‘a........
4D455448 09A45343 4D50680D 666F6F00   METH..SCMPh.foo.
```

Compilation

- Substitute macros,
- fill optional parameters of opcodes,
- serialize opcodes and their parameters.

AML: ACPI machine language

- Bytecode interpreted by the OS,
- poorly defined: uncomplete grammer, obscure casting,
- many opcodes, some of them are high level.
- Thus it is not trivial to parse.
- But it is quite compact.

AML: grammar

```
DefMethod     := MethodOp PkgLength NameString MethodFlags TermList

MethodOp      := 0x14

PkgLength     := PkgLeadByte |
                 <PkgLeadByte ByteData> |
                 <PkgLeadByte ByteData ByteData> |
                 <PkgLeadByte ByteData ByteData ByteData>

TermList      := Nothing | <TermObj TermList>

TermObj       := NameSpaceModifierObj | NamedObj |
                 Type1Opcode | Type2Opcode

Type2Opcode   := DefAcquire | DefAdd | DefAnd | DefBuffer
                 DefConcatRes | DefCondRefOf | DefCopyObject
                 DefDerefOf | DefDivide | DefFindSetLeftBit |
                 DefFromBCD | DefIncrement | DefIndex | DefLAnd |
                 [...]
                 DefToDecimalString | DefToHexString | DefToInteger |
                 DefWait | DefXOr | UserTermObj

UserTermObj   := NameString TermArgList
```

ACPI VM

- Stack frame: 9 arguments and 9 local objects,
- supports recursivity,
- has access to the whole physical memory space and every devices,
- can load and unload new methods at run-time (from a Plug-And-Play device?).

From Kernel to Hardware

- Thanks to ACPI tables, the kernel has a precise description of the hardware,
- thus it can write to memory owned by the hardware or to *ACPI registers*,
- by calling ACPI methods, it can safely control hardware, the way the manufacturer intended.

From Hardware to Kernel

- Hardware can retrieve information about the Kernel through ACPI control methods,
- ACPI events:
  - The hardware sets the GPE: General-Purpose Event Registers,
  - and triggers an SCI interrupt.
  - The kernel catches the interruption and calls the right ACPI method.

Power Control

- Enough information is provided to the kernel to let it manage power consumption and cooling of the devices,
  - eventually organized in thermal zones,
- manufacturer can define `Device` namespaces that include standardized methods to control the hardware (Going To Sleep, Back From Sleep, . . .),
- the kernel can control devices directly thanks to states.

Power Control: states

ACPI defines power states of the system:

- Global System State:
  - G3: Mechanical Off,
  - G2: Soft Off,
  - G1: Sleeping,
  - G0: Working.
- Device Power State,
- Sleeping State (S1-S5),
- Processor Power State,
- Performance State.

Working implementations

- ACPICA by Intel: the reference,
- Microsoft ACPI implementation,
- various independant attempts. . .

Description of our project

Initial goal:

- simple ACPI implementation with basic functionalities
- only for the STOS kernel.

Definitive goal:

- simple and complete ACPI implementation,
- OS-independant.

State of our project

- Parsing and AST building: *done*,
- interpretation of generic opcodes: *done*,
- interpretation of low-level opcodes: *mostly done*,
- definition of interfaces between the kernel and the ACPI VM: *ongoing*,
- events handling: *todo*,
- integration in a real kernel: *todo*.

Conclusion

Modern PCs are horrible. ACPI is a complete design disaster in every way. But we're kind of stuck with it. If any Intel people are listening to this and you had anything to do with ACPI, shoot yourself now, before you reproduce.

Linus Torvald, Linux Journal, November 2003