# ACPI considerations for PCI host bridges

The general rule is that the ACPI namespace should describe everything the OS might use unless there's another way for the OS to find it [1, 2].

For example, there's no standard hardware mechanism for enumerating PCI host bridges, so the ACPI namespace must describe each host bridge, the method for accessing PCI config space below it, the address space windows the host bridge forwards to PCI (using _CRS), and the routing of legacy INTx interrupts (using _PRT).

PCI devices, which are below the host bridge, generally do not need to be described via ACPI. The OS can discover them via the standard PCI enumeration mechanism, using config accesses to discover and identify devices and read and size their BARs. However, ACPI may describe PCI devices if it provides power management or hotplug functionality for them or if the device has INTx interrupts connected by platform interrupt controllers and a _PRT is needed to describe those connections.

ACPI resource description is done via _CRS objects of devices in the ACPI namespace [2]. The _CRS is like a generalized PCI BAR: the OS can read _CRS and figure out what resource is being consumed even if it doesn't have a driver for the device [3]. That's important because it means an old OS can work correctly even on a system with new devices unknown to the OS. The new devices might not do anything, but the OS can at least make sure no resources conflict with them.

Static tables like MCFG, HPET, ECDT, etc., are *not* mechanisms for reserving address space. The static tables are for things the OS needs to know early in boot, before it can parse the ACPI namespace. If a new table is defined, an old OS needs to operate correctly even though it ignores the table. _CRS allows that because it is generic and understood by the old OS; a static table does not.

If the OS is expected to manage a non-discoverable device described via ACPI, that device will have a specific _HID/_CID that tells the OS what driver to bind to it, and the _CRS tells the OS and the driver where the device's registers are.

PCI host bridges are PNP0A03 or PNP0A08 devices. Their _CRS should describe all the address space they consume. This includes all the windows they forward down to the PCI bus, as well as registers of the host bridge itself that are not forwarded to PCI. The host bridge registers include things like secondary/subordinate bus registers that determine the bus range below the bridge, window registers that describe the apertures, etc. These are all device-specific, non-architected things, so the only way a PNP0A03/PNP0A08 driver can manage them is via _PRS/_CRS/_SRS, which contain the device-specific details. The host bridge registers also include ECAM space, since it is consumed by the host bridge.

ACPI defines a Consumer/Producer bit to distinguish the bridge registers ("Consumer") from the bridge apertures ("Producer") [4, 5], but early BIOSes didn't use that bit correctly. The result is that the current ACPI spec defines Consumer/Producer only for the Extended Address Space descriptors; the bit should be ignored in the older QWord/DWord/Word Address Space descriptors. Consequently, OSes have to assume all QWord/DWord/Word descriptors are windows.

Prior to the addition of Extended Address Space descriptors, the failure of Consumer/Producer meant there was no way to describe bridge registers in the PNP0A03/PNP0A08 device itself. The workaround was to describe the bridge registers (including ECAM space) in PNP0C02 catch-all devices [6]. With the exception of ECAM, the bridge register space is device-specific anyway, so the generic PNP0A03/PNP0A08 driver (pci_root.c) has no need to know about it.

New architectures should be able to use "Consumer" Extended Address Space descriptors in the PNP0A03 device for bridge registers, including ECAM, although a strict interpretation of [6] might prohibit this. Old x86 and ia64 kernels assume all address space descriptors, including "Consumer" Extended Address Space ones, are windows, so it would not be safe to describe bridge registers this way on those architectures.

PNP0C02 "motherboard" devices are basically a catch-all. There's no programming model for them other than "don't use these resources for anything else." So a PNP0C02 _CRS should claim any address space that is (1) not claimed by _CRS under any other device object in the ACPI namespace and (2) should not be assigned by the OS to something else.

The PCIe spec requires the Enhanced Configuration Access Method (ECAM) unless there's a standard firmware interface for config access, e.g., the ia64 SAL interface [7]. A host bridge consumes ECAM memory address space and converts memory accesses into PCI configuration accesses. The spec defines the ECAM address space layout and functionality; only the base of the address space is device-specific. An ACPI OS learns the base address from either the static MCFG table or a _CBA method in the PNP0A03 device.

The MCFG table must describe the ECAM space of non-hot pluggable host bridges [8]. Since MCFG is a static table and can't be updated by hotplug, a _CBA method in the PNP0A03 device describes the ECAM space of a hot-pluggable host bridge [9]. Note that for both MCFG and _CBA, the base address always corresponds to bus 0, even if the bus range below the bridge (which is reported via _CRS) doesn't start at 0.

[1] ACPI 6.2, sec 6.1:

> For any device that is on a non-enumerable type of bus (for example, an ISA bus), OSPM enumerates the devices' identifier(s) and the ACPI system firmware must supply an _HID object ... for each device to enable OSPM to do that.

[2] ACPI 6.2, sec 3.7:

> The OS enumerates motherboard devices simply by reading through the ACPI Namespace looking for devices with hardware IDs.

Each device enumerated by ACPI includes ACPI-defined objects in the ACPI Namespace that report the hardware resources the device could occupy [_PRS], an object that reports the resources that are currently used by the device [_CRS], and objects for configuring those resources [_SRS]. The information is used by the Plug and Play OS (OSPM) to configure the devices.

[3] ACPI 6.2, sec 6.2:

> OSPM uses device configuration objects to configure hardware resources for devices enumerated via ACPI. Device configuration objects provide information about current and possible resource requirements, the relationship between shared resources, and methods for configuring hardware resources.
>
> When OSPM enumerates a device, it calls _PRS to determine the resource requirements of the device. It may also call _CRS to find the current resource settings for the device. Using this information, the Plug and Play system determines what resources the device should consume and sets those resources by calling the deviceâ€™s _SRS control method.
>
> In ACPI, devices can consume resources (for example, legacy keyboards), provide resources (for example, a proprietary PCI bridge), or do both. Unless otherwise specified, resources for a device are assumed to be taken from the nearest matching resource above the device in the device hierarchy.

[4] ACPI 6.2, sec 6.4.3.5.1, 2, 3, 4:

> QWord/DWord/Word Address Space Descriptor (.1, .2, .3)
>
> > General Flags: Bit [0] Ignored
>
> Extended Address Space Descriptor (.4)
>
> > General Flags: Bit [0] Consumer/Producer:
> >
> > - 1 â€“ This device consumes this resource
> > - 0 â€“ This device produces and consumes this resource

[5] ACPI 6.2, sec 19.6.43:

> ResourceUsage specifies whether the Memory range is consumed by this device (ResourceConsumer) or passed on to child devices (ResourceProducer). If nothing is specified, then ResourceConsumer is assumed.

[6] PCI Firmware 3.2, sec 4.1.2:

> If the operating system does not natively comprehend reserving the MMCFG region, the MMCFG region must be reserved by firmware. The address range reported in the MCFG table or by _CBA method (see Section 4.1.3) must be reserved by declaring a motherboard resource. For most systems, the motherboard resource would appear at the root of the ACPI namespace (under _SB) in a node with a _HID of EISAID (PNP0C02), and the resources in this case should not be claimed in the root PCI busâ€™s _CRS. The resources can optionally be returned in Int15 E820 or EFIGetMemoryMap as reserved memory but must always be reported through ACPI as a motherboard resource.

[7] PCI Express 4.0, sec 7.2.2:

> For systems that are PC-compatible, or that do not implement a processor-architecture-specific firmware interface standard that allows access to the Configuration Space, the ECAM is required as defined in this section.

[8] PCI Firmware 3.2, sec 4.1.2:

> The MCFG table is an ACPI table that is used to communicate the base addresses corresponding to the non-hot removable PCI Segment Groups range within a PCI Segment Group available to the operating system at boot. This is required for the PC-compatible systems.
>
> The MCFG table is only used to communicate the base addresses corresponding to the PCI Segment Groups available to the system at boot.

[9] PCI Firmware 3.2, sec 4.1.3:

> The _CBA (Memory mapped Configuration Base Address) control method is an optional ACPI object that returns the 64-bit memory mapped configuration base address for the hot plug capable host bridge. The base address returned by _CBA is processor-relative address. The _CBA control method evaluates to an Integer.
>
> This control method appears under a host bridge object. When the _CBA method appears under an active host bridge object, the operating system evaluates this structure to identify the memory mapped configuration base address corresponding to the PCI Segment Group for the bus number range specified in _CRS method. An ACPI name space object that contains the _CBA method must also contain a corresponding _SEG method.