

NAME

pci, **pci_read_config**, **pci_write_config**, **pci_enable_busmaster**, **pci_disable_busmaster**, **pci_enable_io**, **pci_disable_io**, **pci_set_powerstate**, **pci_get_powerstate**, **pci_find_bsf**, **pci_find_device** – kernel interface to PCI bus

SYNOPSIS

```
#include <sys/bus.h>
#include <pci/pcivar.h>
#include <pci/pciereg.h>
#include <machine/pci_cfgreg.h>

u_int32_t
pci_read_config(device_t dev, int reg, int width);

void
pci_write_config(device_t dev, int reg, u_int32_t val, int width);

int
pci_enable_busmaster(device_t dev);

int
pci_disable_busmaster(device_t dev);

int
pci_enable_io(device_t dev, int space);

int
pci_disable_io(device_t dev, int space);

int
pci_set_powerstate(device_t dev, int state);

int
pci_get_powerstate(device_t dev);

device_t
pci_find_bsf(u_int8_t, u_int8_t, u_int8_t);

device_t
pci_find_device(u_int16_t, u_int16_t);
```

DESCRIPTION

The **pci** set of functions are used for managing PCI devices.

The **pci_read_config()** function is used to read data from the PCI configuration space of the device *dev*, at offset *reg*, with *width* specifying the size of the access.

The **pci_write_config()** function is used to write the value *val* to the PCI configuration space of the device *dev*, at offset *reg*, with *width* specifying the size of the access.

The **pci_enable_busmaster()** function enables PCI bus mastering for the device *dev*, by setting the PCIM_CMD_BUSMASTEREN bit in the PCIR_COMMAND register. The **pci_disable_busmaster()** function clears this bit.

The **pci_enable_io()** function enables memory or I/O port address decoding for the device *dev*, by setting the PCIM_CMD_MEMEN or PCIM_CMD_PORTEN bit in the PCIR_COMMAND register appropriately. The **pci_disable_io()** function clears the appropriate bit. The *state* argument specifies which resource is affected; this can be either `SYS_RES_MEMORY` or `SYS_RES_IOPORT` as appropriate.

NOTE: These functions should be used in preference to manually manipulating the configuration space.

The `pci_get_powerstate()` function returns the current ACPI power state of the device *dev*. If the device does not support power management capabilities, then the default state of `PCI_POWERSTATE_D0` is returned. The following power states are defined by ACPI:

<code>PCI_POWERSTATE_D0</code>	State in which device is on and running. It is receiving full power from the system and delivering full functionality to the user.
<code>PCI_POWERSTATE_D1</code>	Class-specific low-power state in which device context may or may not be lost. Buses in this state cannot do anything to the bus, to force devices to loose context.
<code>PCI_POWERSTATE_D2</code>	Class-specific low-power state in which device context may or may not be lost. Attains greater power savings than <code>PCI_POWERSTATE_D1</code> . Buses in this state can cause devices to loose some context. Devices <i>must</i> be prepared for the bus to be in this state or higher.
<code>PCI_POWERSTATE_D3</code>	State in which the device is off and not running. Device context is lost, and power from the device can be removed.

The `pci_set_powerstate()` function is used to transition the device *dev* to the ACPI power state *state*. It checks to see if the device is PCI 2.2 compliant. If so, it checks the capabilities pointer to determine which power states the device supports. If the device does not have power management capabilities, the default state of `PCI_POWERSTATE_D0` is set.

The `pci_find_bsf()` function looks up the *device_t* of a PCI device, given its *bus*, *slot*, and *function*.

The `pci_find_device()` function looks up the *device_t* of a PCI device, given its *vendor* and *device* IDs. Note that there can be multiple matches for this search; this function only returns the first matching device.

IMPLEMENTATION NOTES

The `pci_addr_t` type is defined according to the size of the PCI bus address space on the target system.

SEE ALSO

`bus_alloc_resource(9)`, `bus_dma(9)`, `bus_release_resource(9)`, `bus_setup_intr(9)`, `bus_tear_down_intr(9)`, `devclass(9)`, `device(9)`, `driver(9)`, `rman(9)`

"NewBus", *FreeBSD Developers' Handbook*,

http://www.FreeBSD.org/doc/en_US.ISO8859-1/books/developers-handbook/.

Shanley and Anderson, *PCI System Architecture*, Addison-Wesley, 2nd Edition, ISBN 0-201-30974-2.

AUTHORS

This man page was written by Bruce M Simpson (bms@spc.org).

BUGS

This manual page does not yet document PAE and how it affects memory-space mapping of PCI devices.