# PCI kernel APIs: -

1. **pcim\_enable\_device( )**

Initialize device before it's used by a driver.

Wakes up the device if it is suspended and provides access to I/O and memory.

Note: we don't actually enable the device many times if we call this function repeatedly (we just increment the count).

1. **pcim\_enable\_device(struct pci\_dev \* pdev** **)**

pcim\_enable\_device() is a managed version of pci\_enable\_device(). Means that if we call pci\_enable\_device(), we also need to call pci\_disable\_device() in the end. In case of pcim\_enable\_device(), managed framework will take care of disable operation for us.

1. **pci\_resource\_flags(dev, bar)**

This function returns the flags associated with this resource.

if ((pci\_resource\_flags(pdev, 0) & IORESOURCE\_IO) == 0) {

dev\_err(endpoint->dev, "Incorrect BAR configuration. Aborting.\n");

return -ENODEV;

}

if (!(pci\_resource\_flags(pdev, 0) & IORESOURCE\_MEM)) {

dev\_err(endpoint->dev, "Incorrect BAR configuration. Aborting.\n");

return -ENODEV;

}

Macro values: -

File - include/linux/ioport.h

#define IORESOURCE\_IO 0x00000100 /\* PCI/ISA I/O ports \*/

#define IORESOURCE\_MEM 0x00000200

1. [**pcim\_iomap\_regions**](https://elixir.bootlin.com/linux/latest/C/ident/pcim_iomap_regions)**()**

This function Request and iomap PCI BARs.

Function signature and arguments: -

int [pcim\_iomap\_regions](https://elixir.bootlin.com/linux/latest/C/ident/pcim_iomap_regions)(struct [pci\_dev](https://elixir.bootlin.com/linux/latest/C/ident/pci_dev) \*pdev, int mask, const char \*name)

Arguments:

@pdev: PCI device to map IO resources for

@mask: Mask of BARs to request and iomap

@name: Name used when requesting regions

Example Use:

ret = pcim\_iomap\_regions(dev, BIT(0), KBUILD\_MODNAME);

#define [BIT](https://elixir.bootlin.com/linux/latest/C/ident/BIT)([nr](https://elixir.bootlin.com/linux/latest/C/ident/nr)) ([UL](https://elixir.bootlin.com/linux/latest/C/ident/UL)(1) << ([nr](https://elixir.bootlin.com/linux/latest/C/ident/nr)))

So, BIT(0) means (1 << 0) = 1.

Using this second argument “mask”, we can tell which all BARs to map or avoid.

For example, if we give mask as 1 or BIT(0), then it maps only BAR-0.

Similarly, if we use mask as BIT(0) | BIT(1), then it maps both BAR-0 and BAR-1.

In above function, we can use bit mask as below: -

ret = pcim\_iomap\_regions(dev, 0b0011, KBUILD\_MODNAME);

Here 0b0011 is bit mask in binary.

It can best be understood by the above function’s definition defined in kernel code “lib/devres.c”.

1. **pci\_select\_bars()**

This API returns BAR mask, whose bits set to 1 represents BAR numer and type of BAR is specified by the flag passed in second argument.

Ex. - bar\_mask = pci\_select\_bars(pdev, IORESOURCE\_MEM);

If bar\_mask = 0b0101, then it means MMIO BAR - 0 and BAR – 2 are available.

1. [**pcim\_iomap\_table**](https://elixir.bootlin.com/linux/latest/C/ident/pcim_iomap_table)**()**

/\*\*

\* pcim\_iomap\_table - access iomap allocation table

\* @pdev: PCI device to access iomap table for

\*

\* Access iomap allocation table for @dev. If iomap table doesn't

\* exist and @pdev is managed, it will be allocated. All iomaps

\* recorded in the iomap table are automatically unmapped on driver

\* detach.

\*

\* This function might sleep when the table is first allocated but can

\* be safely called without context and guaranteed to succeed once

\* allocated.

\*/

void [\_\_iomem](https://elixir.bootlin.com/linux/latest/C/ident/__iomem) \* const \*[pcim\_iomap\_table](https://elixir.bootlin.com/linux/latest/C/ident/pcim_iomap_table)(struct [pci\_dev](https://elixir.bootlin.com/linux/latest/C/ident/pci_dev) \*pdev)

{

struct [pcim\_iomap\_devres](https://elixir.bootlin.com/linux/latest/C/ident/pcim_iomap_devres) \*[dr](https://elixir.bootlin.com/linux/latest/C/ident/dr), \*new\_dr;

[dr](https://elixir.bootlin.com/linux/latest/C/ident/dr) = [devres\_find](https://elixir.bootlin.com/linux/latest/C/ident/devres_find)(&pdev->dev, [pcim\_iomap\_release](https://elixir.bootlin.com/linux/latest/C/ident/pcim_iomap_release), NULL, NULL);

if ([dr](https://elixir.bootlin.com/linux/latest/C/ident/dr))

return [dr](https://elixir.bootlin.com/linux/latest/C/ident/dr)->[table](https://elixir.bootlin.com/linux/latest/C/ident/table);

new\_dr = [devres\_alloc\_node](https://elixir.bootlin.com/linux/latest/C/ident/devres_alloc_node)([pcim\_iomap\_release](https://elixir.bootlin.com/linux/latest/C/ident/pcim_iomap_release), sizeof(\*new\_dr), [GFP\_KERNEL](https://elixir.bootlin.com/linux/latest/C/ident/GFP_KERNEL),

[dev\_to\_node](https://elixir.bootlin.com/linux/latest/C/ident/dev_to_node)(&pdev->dev));

if (!new\_dr)

return NULL;

[dr](https://elixir.bootlin.com/linux/latest/C/ident/dr) = [devres\_get](https://elixir.bootlin.com/linux/latest/C/ident/devres_get)(&pdev->dev, new\_dr, NULL, NULL);

return [dr](https://elixir.bootlin.com/linux/latest/C/ident/dr)->[table](https://elixir.bootlin.com/linux/latest/C/ident/table);

}

1. **pci\_set\_master()**

Enable DMA by setting the bus master bit in the PCI\_COMMAND register. The device will then be able to act as a master on the address bus.

PCI DMA: -

PCI DMA is implemented with bus mastering. Bus mastering is the capability of devices on the PCI bus to take control of the bus and perform transfers to the mapped memory directly.

Bus mastering (if supported) can be enabled and disabled with the following functions:

void pci\_set\_master(struct pci\_dev \*dev);

void pci\_clear\_master(struct pci\_dev \*dev);