Peripheral Component Interconnect (PCI) Drivers

What to Expect?

- PCI Subsystem
  - Coding Details of a PCI Device Driver
  - Browsing Sample Code

PCI Prologue

- Developed by Intel as a replacement for ISA
  - More of a processor (specific) bus for x86 architectures
- Replacing Features
  - Plug n Play
  - Platform independent
  - Fast
- Hardware Addressing
  - Through South Bridge (PCI Controller)
  - Buses (upto 256) $\rightarrow$ Devices (upto 32) $\rightarrow$ Functions (upto 8)

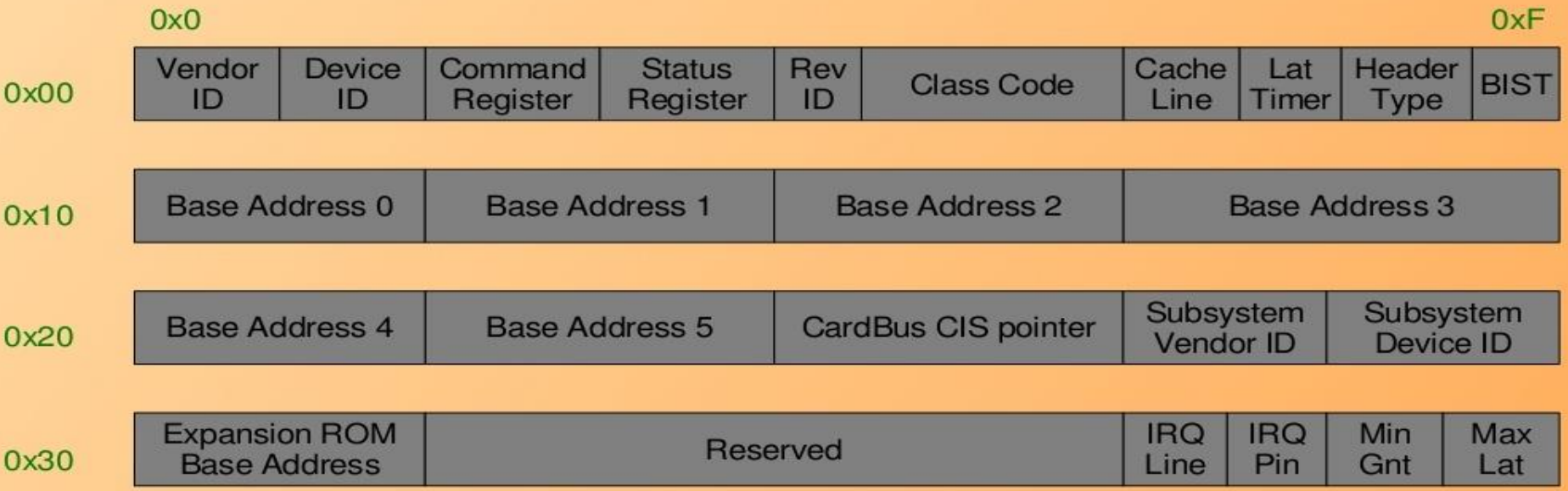
Kernel Aspect

- Organized as
  - PCI Core (driver of the PCI Controller)
  - PCI Drivers (device drivers)
- Kernel Windows for PCI
  - /proc/bus/pci/devices (Try cat)
  - /sys/bus/pci/devices (Try tree)
- Command: lspci

Dynamic Device Configuration

- What Configuration?
  - Device Base Address(es)
    - Register Space(s)
    - Memory(s)
  - Interrupt Vectors
  - ...
- Done during Bootup by
  - BIOS, Or
  - PCI Core (Bootloader or Kernel)
- Configured by the time, driver comes up
- But, where is this configuration stored?

PCI Device Configuration Space



PCI Device Config Space Access

- Header: <linux/pci.h>
- APIs
  - int pci_read_config_byte(struct pci_dev *dev, int where, u8 *val);
  - int pci_read_config_word(struct pci_dev *dev, int where, u16 *val);
  - int pci_read_config_dword(struct pci_dev *dev, int where, u32 *val);
  - int pci_write_config_byte(struct pci_dev *dev, int where, u8 *val);
  - int pci_write_config_word(struct pci_dev *dev, int where, u16 *val);
  - int pci_write_config_dword(struct pci_dev *dev, int where, u32 *val);

NB struct pci_dev *dev would be available as parameter to probe()

Standard PCI Device Access API

- Upto 6 Memory or I/O regions
- Header: <linux/pci.h>
- APIs
  - unsigned long pci_resource_start(dev, bar);
  - unsigned long pci_resource_len(dev, bar);
  - unsigned long pci_resource_flags(dev, bar);
- Flags
  - IORESOURCE_IO
  - IORESOURCE_MEM
  - IORESOURCE_PREFETCH

PCI Driver Registration

- int pci_register_driver(struct pci_driver *drv);
- int pci_unregister_driver(struct pci_driver *drv);
- struct pci_driver
  - const char *name
  - const struct pci_dev_id *id_table;
    - PCI_DEVICE(vendor, device);
    - PCI_DEVICE_CLASS(dev_class, dev_class_mask);
  - int (*probe)(pci_dev, id_table);
  - void (*remove)(pci_dev);
- Header: <linux/pci.h>

Typical 'probe' Function

```
int probe(struct pci_dev *dev, struct pci_dev_id *id)
{
    /* Enable the PCI Device */
    pci_enable_device(dev);
    ...
    /* Acquire PCI Device's regions */
    pci_request_regions(dev, “label”);
    ...
    /* Possibly map I/Os */
    ...
    pci_set_drvdata(dev, <pvt_data_ptr>); // Optional
    ...
    /* Register the vertical */
    ...
    return 0; /* Claimed. Negative for not Claimed */
}
```

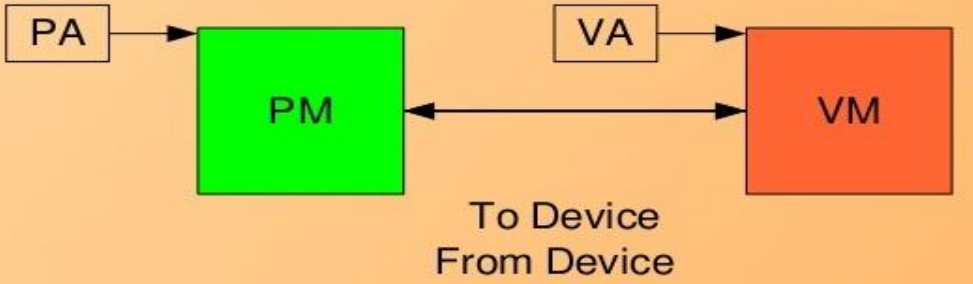
Typical 'remove' Function

```
void remove(struct pci_dev *dev)
{
    /* Unregister the vertical */
    ...
    pci_set_drvdata(dev, NULL); // Optional
    ...
    /* Possibly unmap I/Os */
    ...
    /* Release PCI Device's regions */
    pci_release_regions(dev);
    ...
    /* Disable the PCI Device */
    pci_disable_device(dev);
}
```

Old-style PCI Probing & Getting

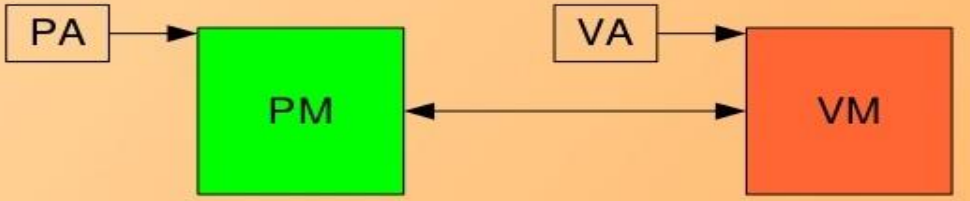
- struct pci_dev *pci_find_*(vendor, device, from);
- struct pci_dev *pci_get_device(v, d, from);
- struct pci_dev *pci_get_subsys(v, d, ssv, ssd, f);
- struct pci_dev *pci_get_slot(bus, devfn);
- pci_dev_put(pci_dev);

DMA Mapping



- APIs
  - dma_addr_t pci_map_single(struct pci_dev *, void *, size_t, int dir);
  - void pci_unmap_single(struct pci_dev *, dma_addr_t, size_t, int dir);
  - Directions
    - PCI_DMA_BIDIRECTIONAL
    - PCI_DMA_TODEVICE
    - PCI_DMA_FROMDEVICE
- Header: <linux/pci.h>

DMA Allocation



- APIs
  - void *pci_alloc_consistent(struct pci_dev *, size_t, dma_addr_t *);
  - void pci_free_consistent(struct pci_dev *, size_t, void *, dma_addr_t);
  - int pci_set_dma_mask(struct pci_dev *, u64 mask);
- Header: <linux/pci.h>

What all have we learnt?

- PCI Subsystem
  - Coding Details of a PCI Device Driver
  - Browsing Sample Code
- DMA with PCI

Any Queries?