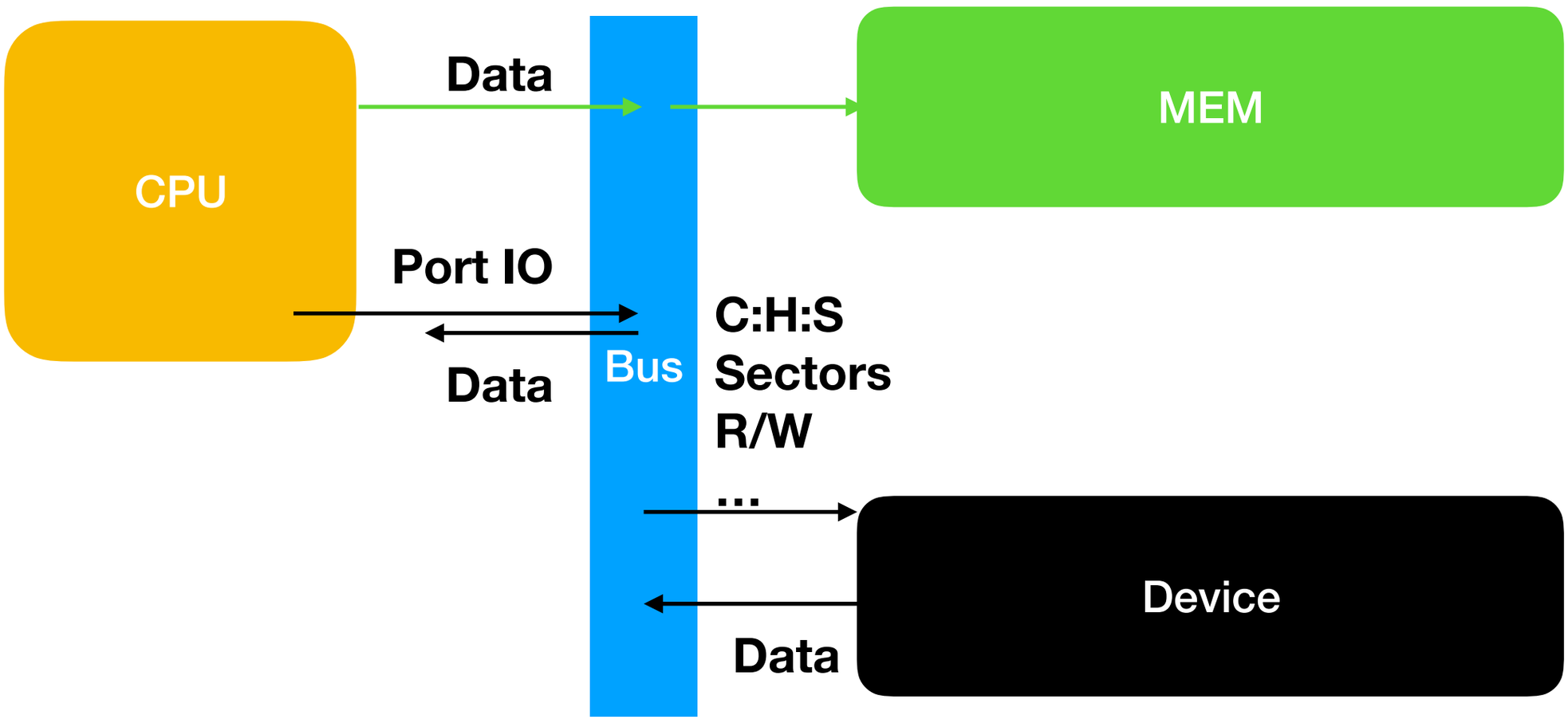
Porting DMA on NCTUOS

B071525—邱韜 4/17, 2019

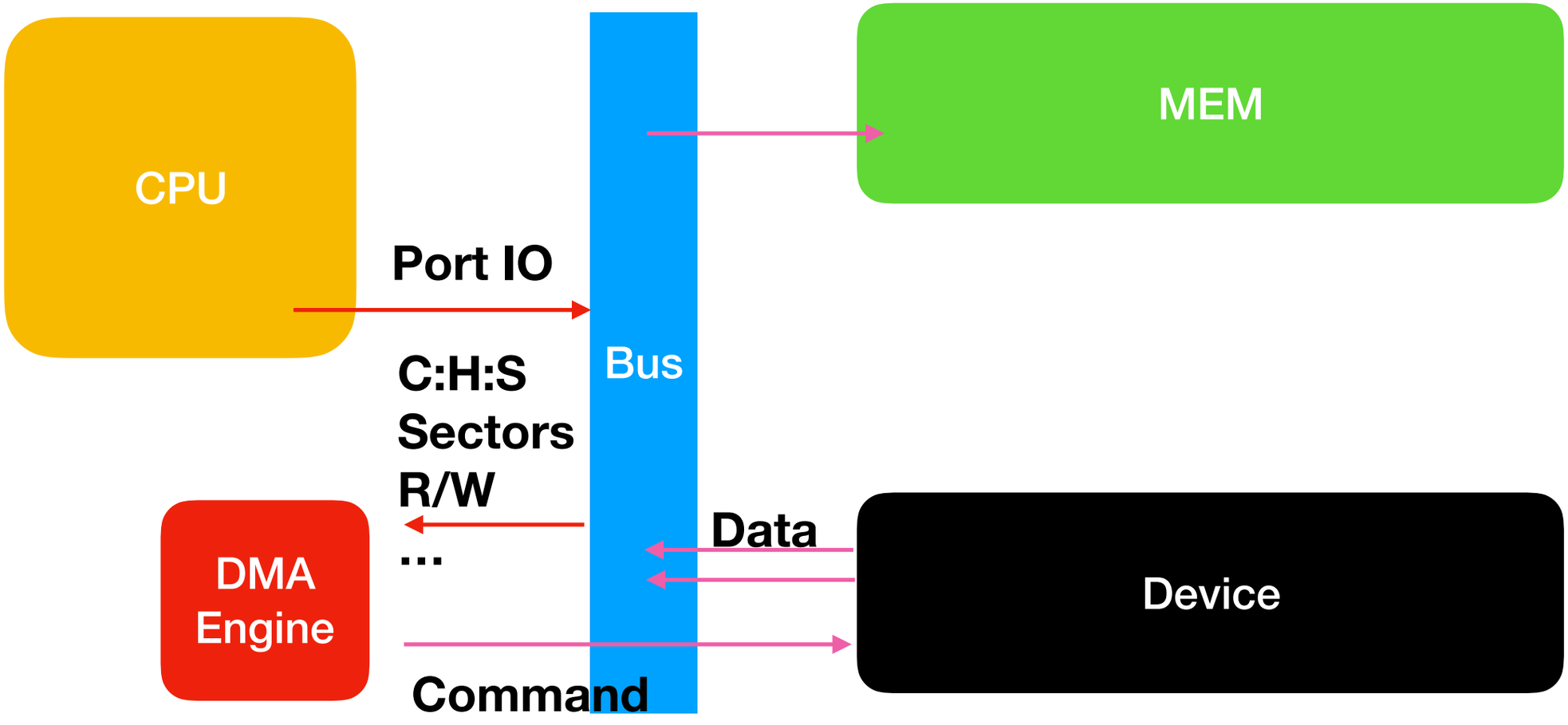
Traditional Data Transfer

- Programmed-I/O (PIO)



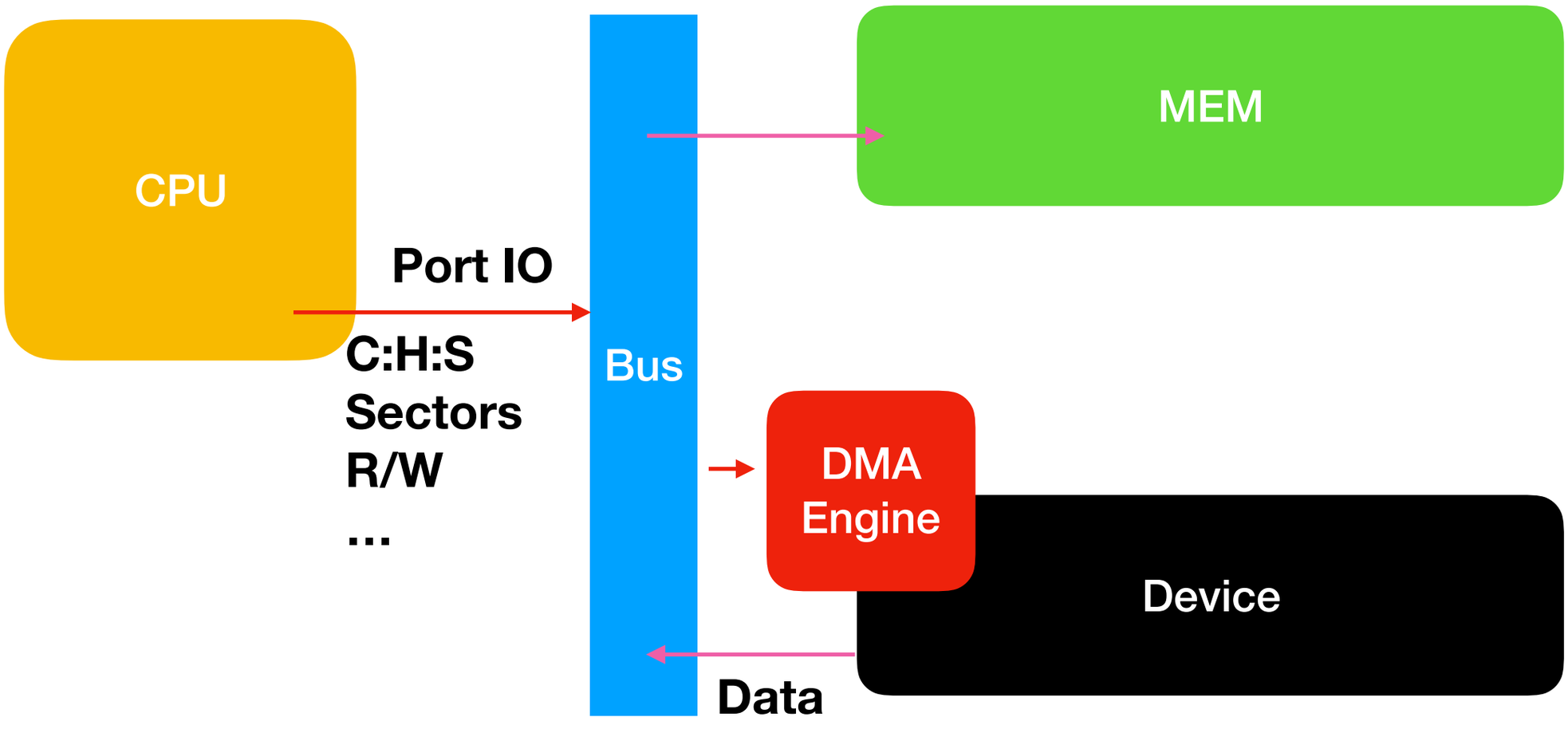
Types of DMA

- Third-Party DMA utilizes a system DMA engine resident on the main system board.



Types of DMA

- First-Party DMA drives its own DMA bus cycles using a channel from the system's DMA engine.



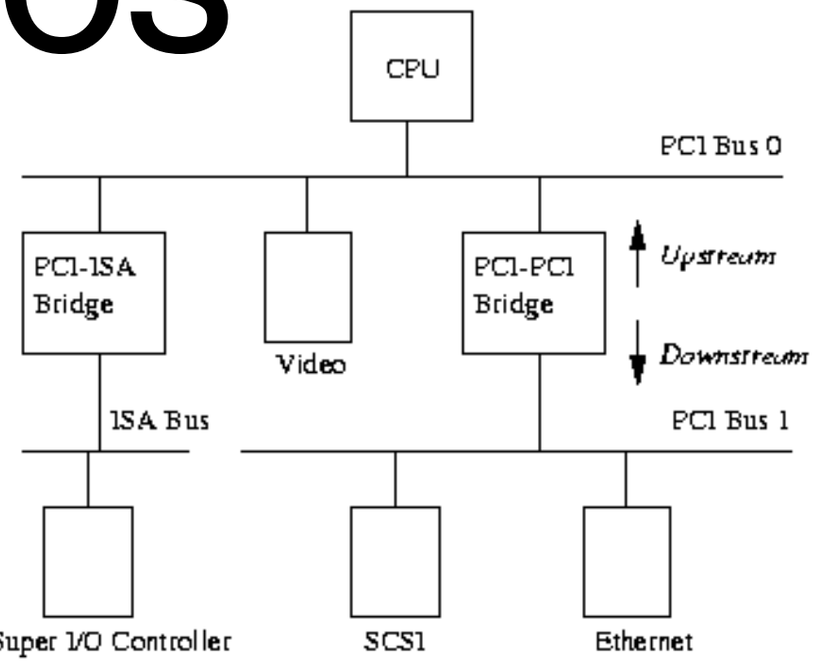
Some DMA examples

- ISA DMA
  - Third-party DMA
    - 4.77 MB/sec
    - Limited to 16 MB physical memory
    - Adds extremely heavy load to the memory bus
- PIIX3 PCI Bus Mastering IDE
  - First-party DMA
    - 22 MB/sec
    - Can access memory with 32 bit addressing

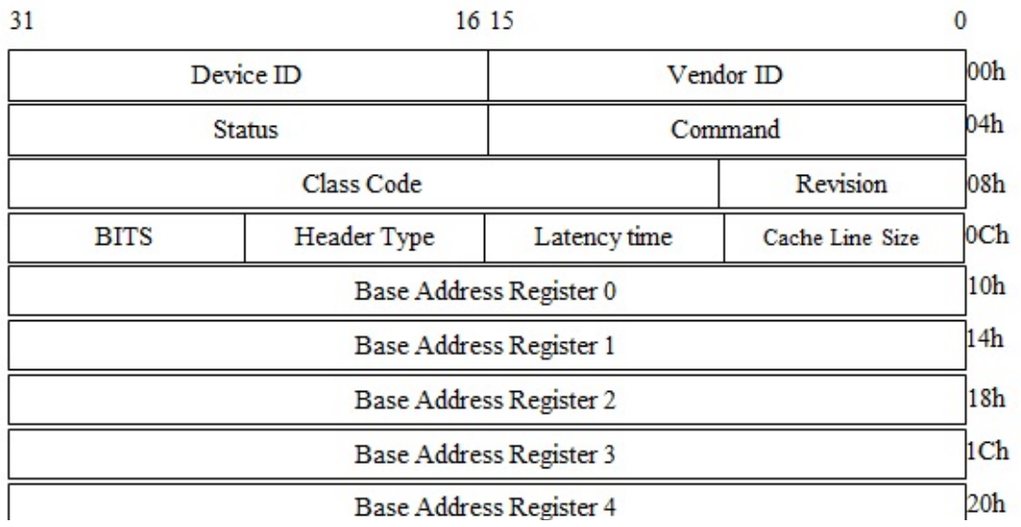
Finding a DMA Controller on NCTUOS

- PCI enumeration:

- We can obtain devices info through port IO defined by PCI:



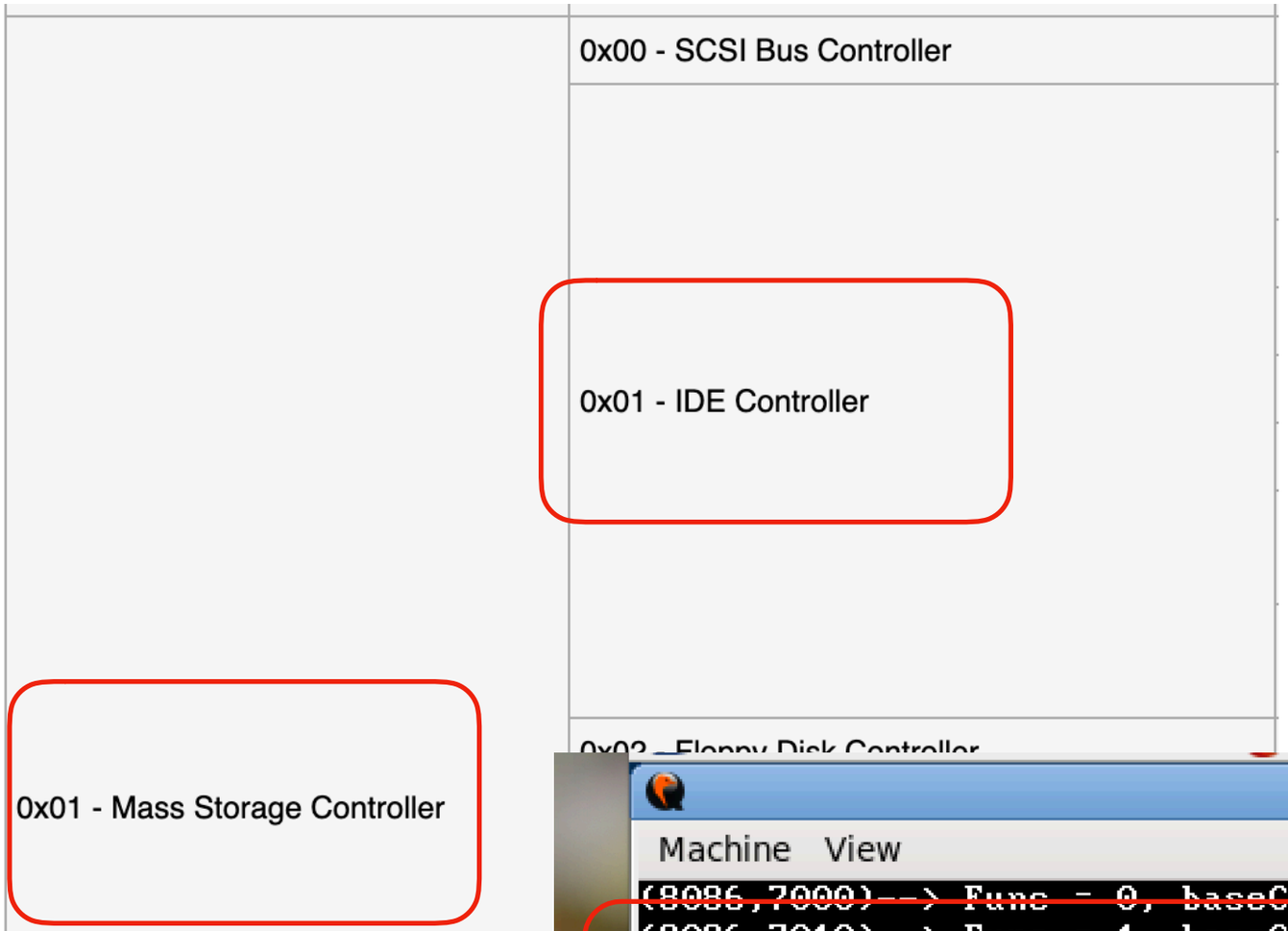
  - Address [B/D/F] sends to port 0xCF8
  - Data R/W to port 0xCFC
  - Refer to the respective data sheet for available registers.

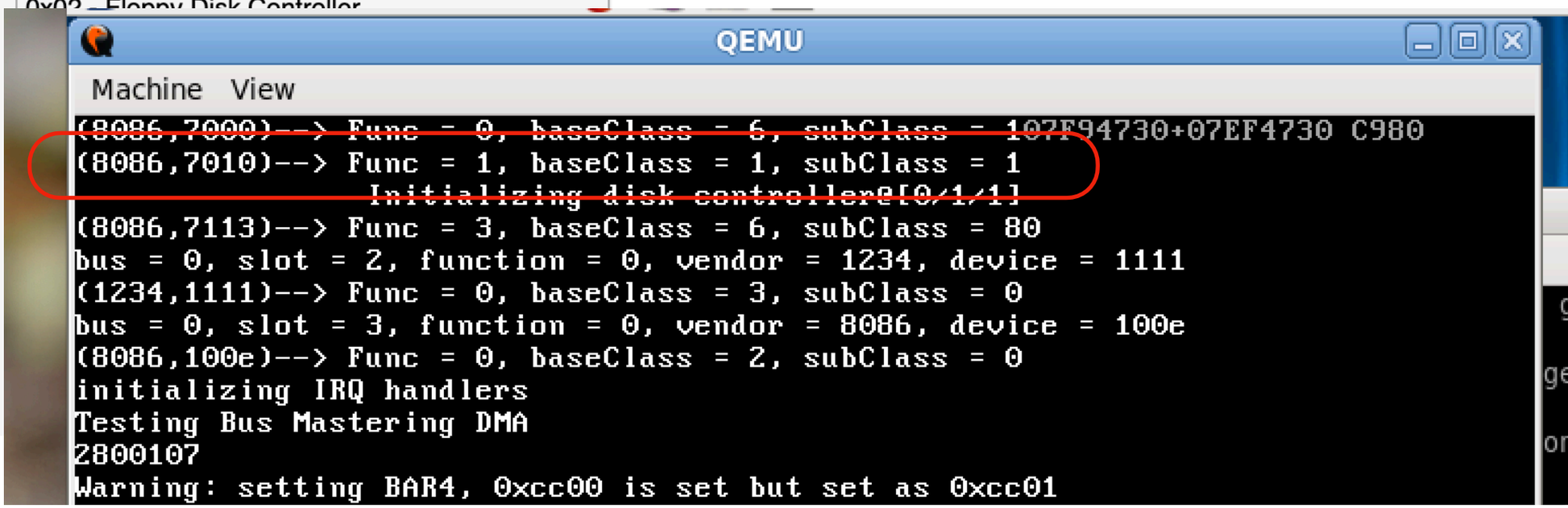


Finding a DMA Controller on NCTUOS

```
15  struct pci_struct ide_bus_master;
16
17
18 › uint16_t pciConfigReadWord (uint8_t bus, uint8_t slot, uint8_t func, uint8_t offset) {⋯}
37
38 › void pciConfigWriteDWord (uint8_t bus, uint8_t slot, uint8_t func, uint8_t offset, uint32_t data) {⋯}
55
56 › uint8_t  getHeaderType(uint8_t bus, uint8_t slot, uint8_t func){⋯}
59 › uint8_t  getBaseClass(uint8_t bus, uint8_t slot, uint8_t func){⋯}
62 › uint8_t  getSubClass(uint8_t bus, uint8_t slot, uint8_t func){⋯}
65 › uint8_t  getSecondaryBus(uint8_t bus, uint8_t slot, uint8_t func){⋯}
68 › uint16_t getVendorID(uint8_t bus, uint8_t slot, uint8_t func){⋯}
71 › uint16_t getDeviceID(uint8_t bus, uint8_t slot, uint8_t func){⋯}
74 › uint32_t setBAR4(uint8_t bus, uint8_t slot, uint8_t func, uint32_t data){⋯}
80 › uint32_t getBAR4(uint8_t bus, uint8_t slot, uint8_t func){⋯}
86
87
88 › void checkFunction(uint8_t bus, uint8_t device, uint8_t function) {⋯}
114
115 › void checkDevice(uint8_t bus, uint8_t device) {⋯}
137
138 › void checkBus(uint8_t bus) {⋯}
145
146 › void checkAllBuses(void) {⋯}
164
165  void pci_init(void){
166      checkAllBuses();
167  }
168
```

Finding a DMA Controller on NCTUOS





Finding a DMA Controller on NCTUOS

PCILookup

Vendor | Device | Submit

Show 10 entries | Search:

Vendor	Vendor ID	Description	Device ID
Intel Corporation	8086	82371SB PIIX3 IDE [Natoma/Triton II]	7010

Showing 1 to 1 of 1 entries | Previous | 1 | Next

```
(8086,7000)--> Func = 0, baseClass = 6, subClass = 1071A4730+07EF4730 C980
(8086,7010)--> Func = 1, baseClass = 1, subClass = 1
               Initializing disk controller@[0/1/1]
(8086,7113)--> Func = 3, baseClass = 6, subClass = 80
bus = 0, slot = 2, function = 0, vendor = 1234, device = 1111
(1234,1111)--> Func = 0, baseClass = 3, subClass = 0
bus = 0, slot = 3, function = 0, vendor = 8086, device = 100e
(8086,100e)--> Func = 0, baseClass = 2, subClass = 0
initializing IRQ handlers
Testing Bus Mastering DMA
2800107
Warning: setting BAR4, 0xcc00 is set but set as 0xcc01
```

Intel PIIX3

- It is an intel southbridge multi-function integrated circuit for PCI devices
  - a PCI-to-ISA bridge
  - PCI IDE function
  - a Universal Serial Bus host/hub
  - Consists of one ISA DMA pair and Bus-Mastering DMA on ATA controller.

intel®

82371FB (PIIX) AND 82371SB (PIIX3)
PCI ISA IDE XCELERATOR

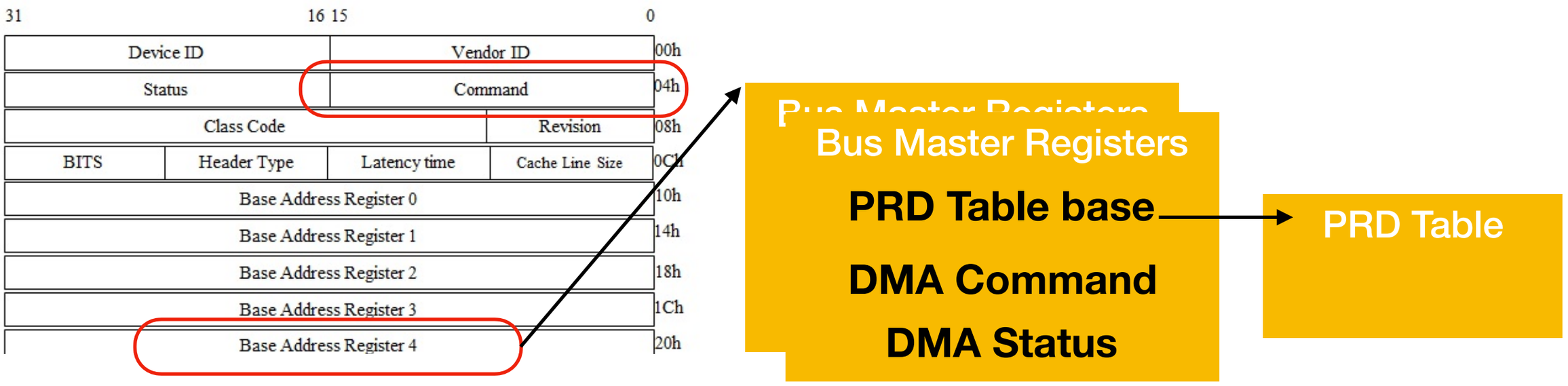
- **Bridge Between the PCI Bus and ISA Bus**
- **PCI and ISA Master/Slave Interface**
  - **PCI from 25–33 MHz**
  - **ISA from 7.5–8.33 MHz**
  - **5 ISA Slots**
- **Fast IDE Interface**
  - **Supports PIO and Bus Master IDE**
  - **Supports up to Mode 4 Timings**
  - **Transfer Rates to 22 MB/Sec**
  - **8 x 32-Bit Buffer for Bus Master IDE PCI Burst Transfers**
- **Enhanced DMA Functions**
  - **Two 8237 DMA Controllers**
  - **Fast Type F DMA**
  - **Compatible DMA Transfers**
  - **7 Independently Programmable Channels**
- **X-Bus Peripheral Support**
  - **Chip Select Decode**
  - **Controls Lower X-Bus Data Byt Transceiver**
- **I/O Advanced Programmable Interr**

Detail of PIIX3 IDE Bus-Mastering DMA

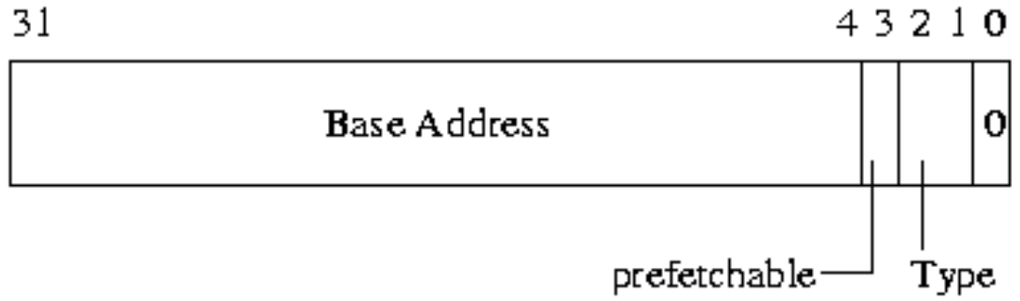
- We implement a DMA functionality on the PCI bus master IDE, which is a first-party DMA.

- According to OSDev, we have to config several registers, including:

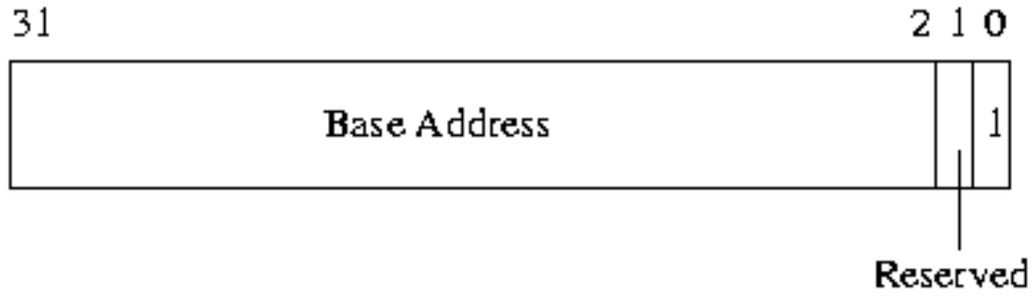
  - Command (pmio): determine whether to enable this DMA, etc.
  - BAR4 (pmio): points to the base address of Bus Master Register.
  - Bus Master Registers (mmio/pmio): consist of command, status of a pair of DMAs, and point to base address of PRD table in physical memory.



PCI Base Address Register



Base Address for PCI Memory Space



Base Address for PCI I/O Space

The BAR4

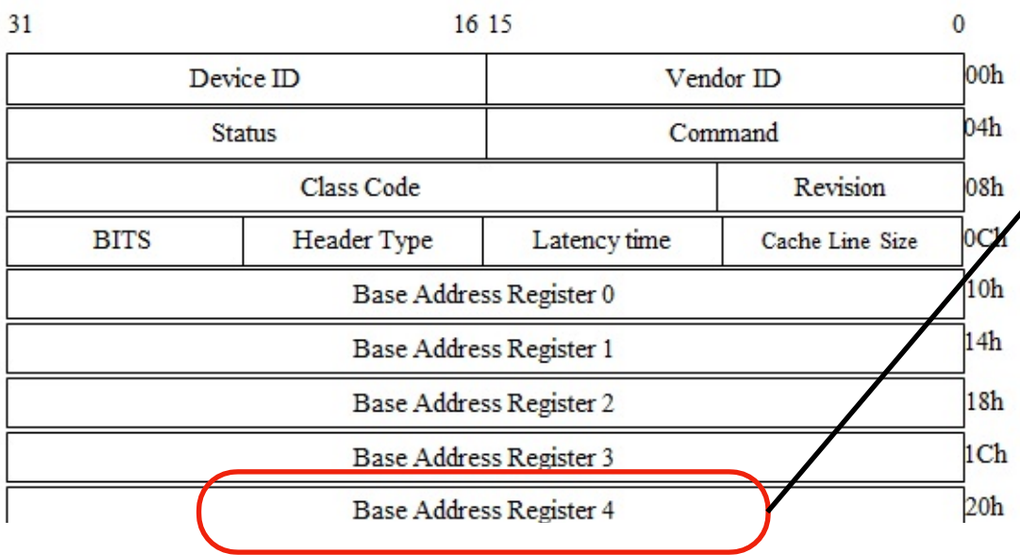
- It points to a port-mapped IO space.

2.3.9. BMIBA—BUS MASTER INTERFACE BASE ADDRESS REGISTER (Function 1)

Address Offset: 20–23h
Default Value: 00000001h
Attribute: Read/Write

This register selects the base address of a 16 byte I/O space to provide a software interface to the Bus Master functions. Only 12 bytes are actually used (6 bytes for primary and 6 bytes for secondary).

Bit	Description
31:16	**Reserved.** Hardwired to 0.
15:4	**Bus Master Interface Base Address.** These bits provide the base address for the Bus Master interface registers and correspond to AD[15:4].
3:2	**Reserved.** Hardwired to 0.
1	**Reserved.**
0	**Resource Type Indicator (RTE)—RO.** This bit is hardwired to 1 indicating that the base address field in this register maps to I/O space.



PRD Table

- PRD table describes the destination of the following DMA transfer.

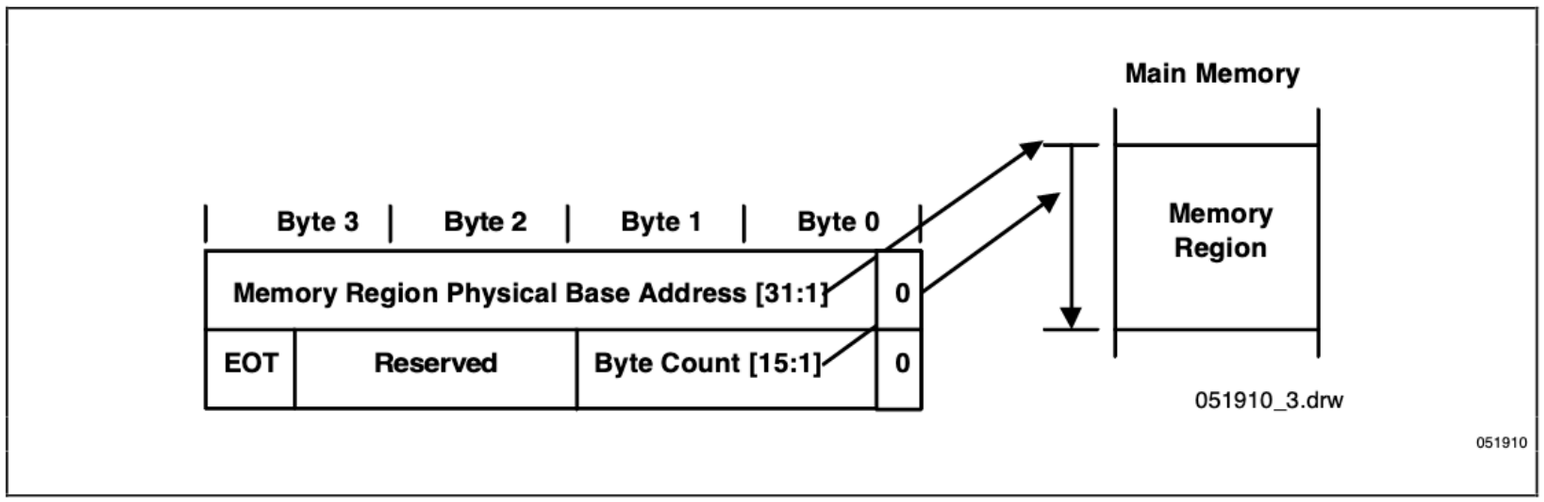
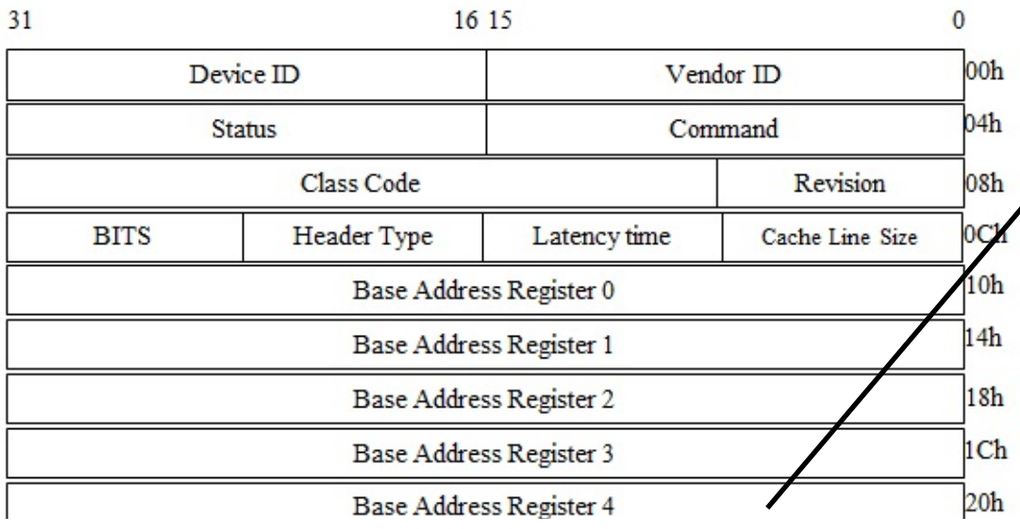
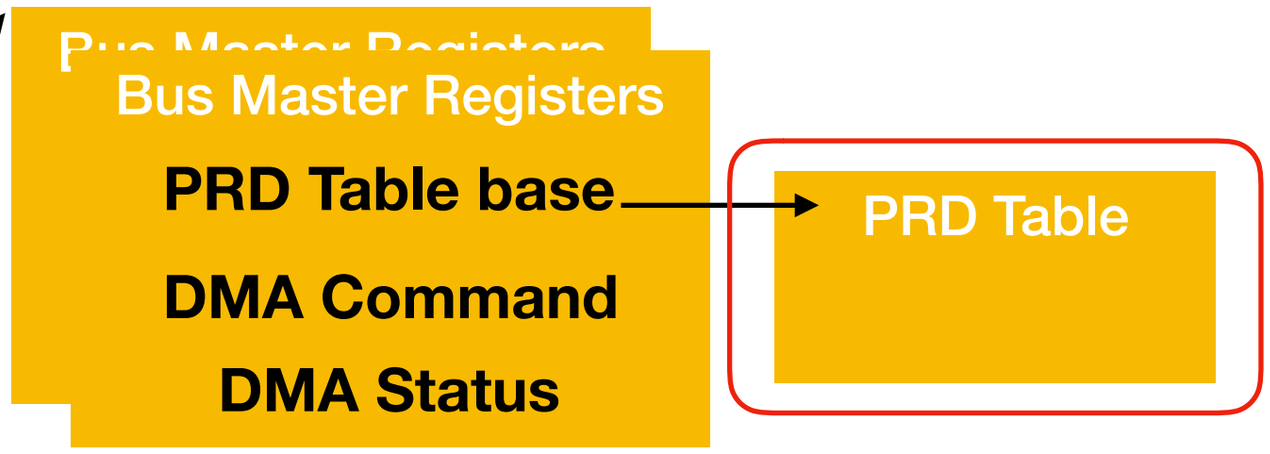


Figure 5. Physical Region Descriptor Table Entry





Steps to initialize a DMA Transfer

1. Software prepares a PRD Table in main memory.

2. Software provides the starting address of the PRD Table by loading the PRD Table Pointer Register.

3. Software issues the appropriate DMA transfer command to the disk device.

4. Engage the bus master function by writing a 1 to the Start bit in the Bus Master IDE Command Register.

5. The controller transfers data.

6. At the end of the transfer, the IDE device signals an interrupt.

Implementation

- The implementation is based on our lab4 demo environment. However, for convenience, `mem_init` is commented out.

- To ease the situation when it comes to debugging and interrupt handling, the DMA get started up after kernel is loaded.

- After interrupts are enabled, we test the DMA facility by loading kernel image again on physical address of 2MB.

```
Andys-MacBook-Pro:nctuos andy$ git status
On branch lab4
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md
        modified:   kernel/Makefile
        modified:   kernel/main.c
        modified:   kernel/trap.c
        modified:   kernel/trap_entry.S

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        inc/pci.h
        kernel/ide.c
        kernel/pci.c
        kernel/pci_dev.h
```

Encountered Surprises

- Data sheet of PIIX3 does not give instructions on how to send command to disk device for ATA neither in bus mastering DMA mode nor PIO mode.

- Data flow direction indicated on the data sheet is opposite to the one on OSDev.

- After new BAR4 is set, old port numbers still reach the mapped registers.

- For bus mastering DMA mode, the 16MB physical limit is not mentioned on the data sheet.

DEMO

–Tao Chiu