

Лекция 3.1

Работа с РСІ

Разработал: Максимов А.Н.

Версия 1.2. 06.2024

Содержание

- Найти сетевую карту на компьютере и определить ее vendor id, device id
- Разработать символьный драйвер для PCI устройства, который возвращает через IOCTL MAC адрес сетевой карты

PCI обзор

PCI (Peripheral component interconnect) - системная шина для подключения периферийных устройств.

Стандарт на шину PCI определяет:

- физические параметры (например, разъёмы и разводку сигнальных линий);
- электрические параметры (например, напряжения);
- логическую модель (типы циклов шины, адресацию на шине и т.д.);

Развитием стандарта PCI занимается организация PCI Special Interest Group. Дополнительную информацию о PCI SIG можно найти на www.pcisig.com

Шины семейства PCI.

PCI это высокоскоростная шина для обмена между CPU и устройствами в.в. Спецификация PCI позволяет передавать 32 бита данных по параллельной шине с частотой 33 или 66 МГц. Пиковая пропускная способность шины 266 Mbs.

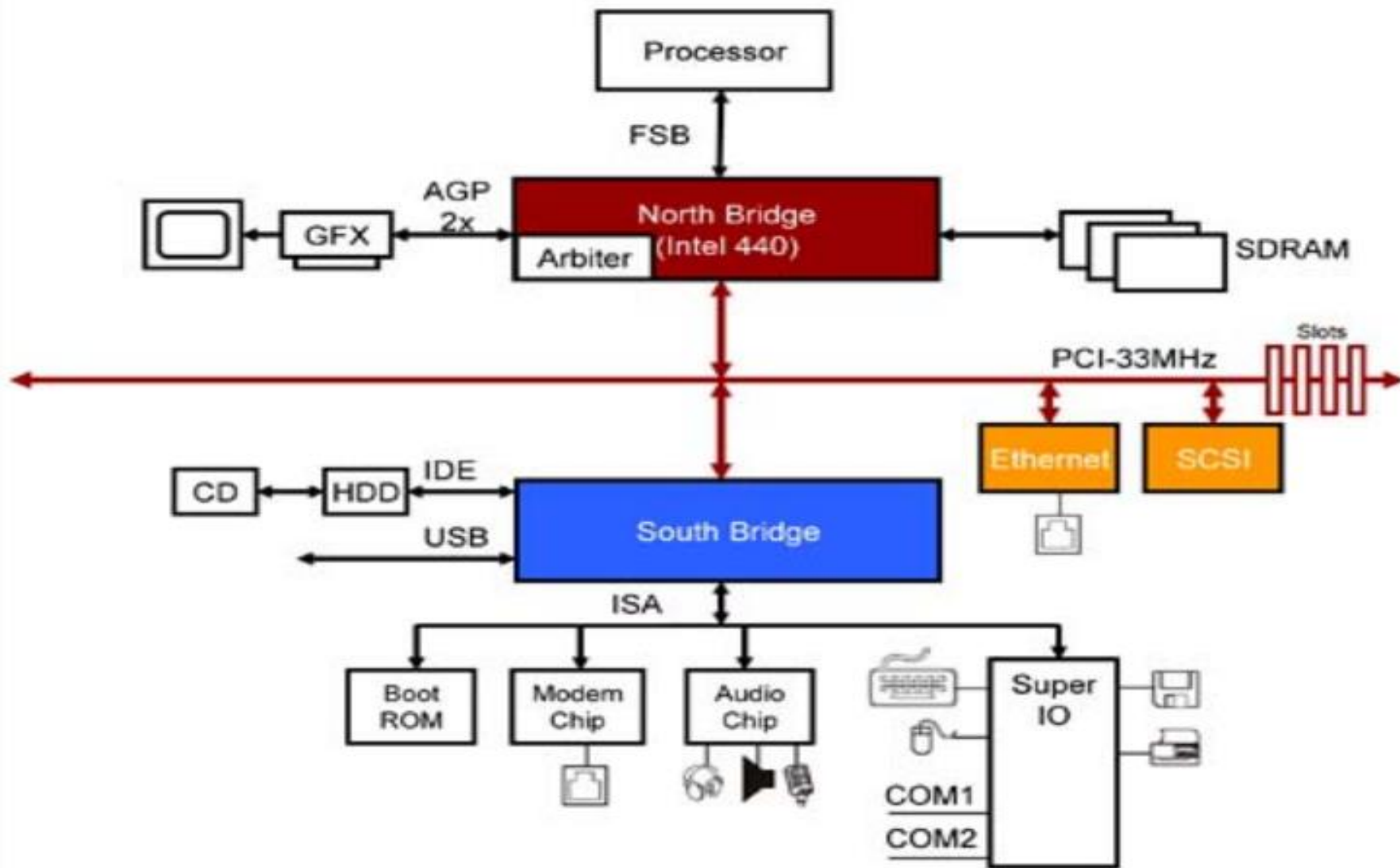
CardBus логически и электрически представляет собой полноценную 32-разрядную шину PCI, работающую на частоте 33 МГц, механические размеры и разъёмы позаимствованы у шины PCMCIA. Устройства CardBus могут поддерживать DMA.

Mini PCI версия шины PCI для использования в ноутбуках стандартизированная в рамках стандарта PCI версии 2.2. Использует 32-разрядную шину PCI, работающую на частоте 33 МГц. PCI устройства могут быть подключены к шине mini PCI через переходник.

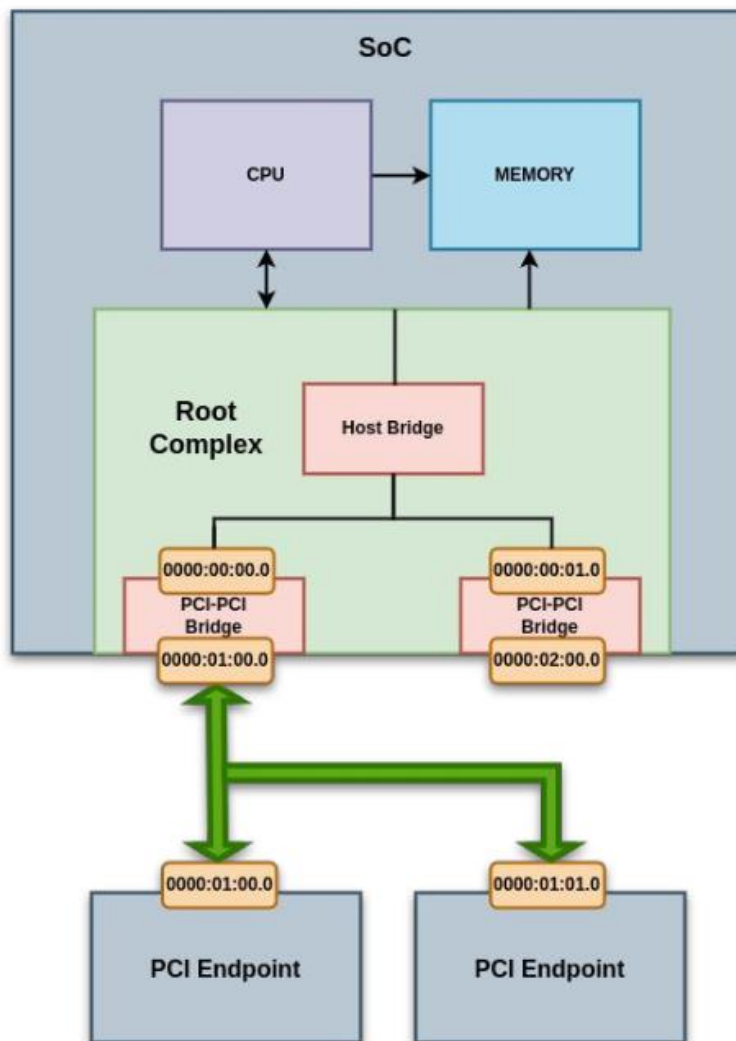
PCI-X (PCI eXtended) 64 разрядное расширение шины PCI для использования в серверных приложениях. Разработана в 1998 IBM, HP и Compaq. Частота шины может быть различной от 66 (PCI-X версия 1.0) до 533 МГц (PCI-X версия 2.0). Пропускная способность от 1.06 Гб/с в первоначальной реализации до 4.3 Гб/с у версии 2.0.

PCI Express (PCIe или PCI-E) дальнейшее развитие технологии PCI. PCIe использует последовательную шину для передачи данных. PCIe поддерживает до 32 последовательных линков. Каждый PCIe линк имеет пропускную способность до 250 Мб/с в каждом направлении обеспечивая до 8 Гб/с в каждом направлении. Текущая версия спецификации - PCIe 2.0.

Типичная архитектура системы с PCI



Типичная архитектура системы с PCI



Работа с PCI

- Linux предоставляет специальные функции для определения куда физически отображается область памяти
- Функции шины PCI (Peripheral Component Interconnect) определены в стандарте на шину
- Устройство на шине идентифицируется парой значений:

```
#define VENDOR_ID    0x1039
```

```
#define DEVICE_ID    0x6325
```

Посмотреть перечень pci устройств

Для получения информации можно воспользоваться утилитой
`apt-get install pciutils`

`lspci -n`

02:00.0 0604: 8086:032c (rev 09)

03:00.0 0c04: 1077:2312 (rev 02)

03:00.1 0c04: 1077:2312 (rev 02)

04:00.0 0200: 14e4:1659 (rev 21)

05:00.0 0200: 14e4:1659 (rev 21)

`lspci -vvv`

Посмотреть перечень pci устройств

Ядро Linux представляет устройства PCI как псевдоустройства в файловой системе sysfs

```
ls -la /sys/bus/pci/devices
```

```
drwxr-xr-x 2 root root 0 2023-08-03 10:38 .
```

```
drwxr-xr-x 5 root root 0 2023-08-03 10:38 ..
```

```
lrwxrwxrwx 1 root root 0 2023-08-03 10:38 0000:00:00.0 -> ../../../../devices/pci0000:00/0000:00:00.0
```

```
lrwxrwxrwx 1 root root 0 2023-08-03 10:38 0000:00:01.0 -> ../../../../devices/pci0000:00/0000:00:01.0
```

```
lrwxrwxrwx 1 root root 0 2023-08-03 10:38 0000:00:01.1 -> ../../../../devices/pci0000:00/0000:00:01.1
```

Информация об устройстве

Более детально:

```
lrwxrwxrwx 1 root root 0 2023-08-03 10:38 0000:04:00.0 -> ../../../../devices/pci0000:00/0000:00:0b.0/0000:04:00.0
```

0000 : PCI domain (each domain can contain up to 256 PCI buses)

04 : the bus number the device is attached to

00 : the device number

.0 : PCI device function

```
cd 0000:04:00.0
```

```
$ ls -la
```

```
-r--r--r-- 1 root root 4096 2023-08-03 10:38 class
```

```
-rw-r--r-- 1 root root 4096 2023-08-03 11:34 config
```

```
-r--r--r-- 1 root root 4096 2023-08-03 10:38 device
```

```
lrwxrwxrwx 1 root root 0 2023-08-03 10:38 driver -> ../../../../bus/pci/drivers/tg3
```

```
...
```

```
-r--r--r-- 1 root root 4096 2023-08-03 10:38 subsystem_device
```

```
-r--r--r-- 1 root root 4096 2023-08-03 10:38 subsystem_vendor
```

```
-r--r--r-- 1 root root 4096 2023-08-03 10:38 vendor
```

Работа с PCI. Области памяти

- CPU и устройство на шине PCI обмениваются информацией через общую память. Обычно разделяемая память содержит регистры команд и статусные регистры. Периферийные устройства обладают собственной памятью. CPU может обращаться к этой памяти. Доступ устройства к памяти системы осуществляется при помощи механизма DMA
- Для шины PCI есть три области адресов:
 - PCI Configuration
 - PCI I/O
 - PCI Memory

Работа с PCI

- Linux предоставляет специальные функции для определения куда физически отображается область памяти
- Функции шины PCI (Peripheral Component Interconnect) определены в стандарте на шину
- Устройство на шине идентифицируется парой значений:

```
#define VENDOR_ID    0x1039
```

```
#define DEVICE_ID 0x6325
```

Работа с PCI. Области памяти

- CPU и устройство на шине PCI обмениваются информацией через общую память. Обычно разделяемая память содержит регистры команд и статусные регистры. Периферийные устройства обладают собственной памятью. CPU может обращаться к этой памяти. Доступ устройства к памяти системы осуществляется при помощи механизма DMA
- Для шины PCI есть три области адресов:
 - PCI Configuration
 - PCI I/O
 - PCI Memory

Работа с PCI. Конфигурационная область

PCI Configuration

32 байта

31				0		Dwords
Status Register		Command Register		Device ID	Vendor ID	1 - 0
BIST	Header Type	Latency Timer	Cache Line Size	Class Code Class/SubClass/ProgIF		Revision ID 3 - 2
Base Address 1				Base Address 0		5 - 4
Base Address 3				Base Address 2		7 - 6
Base Address 5				Base Address 4		9 - 8
Subsystem Device ID		Subsystem Vendor ID		CardBus CIS Pointer		11 - 10
reserved			capabilities pointer	Expansion ROM Base Address		13 - 12
Maximum Latency	Minimum Grant	Interrupt Pin	Interrupt Line	reserved		15 - 14

Модель устройств. Bus PCI

Шина PCI в каталоге /sys/bus/pci/ имеет две директории : 'devices' and 'drivers'.

PCI bus сопоставляет устройства сравнивая PCI Device ID для всех устройств и драйверов.

```
struct bus_type pci_bus_type = {  
    .name      = "pci",  
    .match     = pci_bus_match,  
    .uevent    = pci_uevent,  
    .probe     = pci_device_probe,  
    .remove    = pci_device_remove,  
    .shutdown  = pci_device_shutdown,  
    .dev_attrs = pci_dev_attrs,  
    .pm        = PCI_PM_OPS_PTR,  
};
```

(*linux-2.6.28/drivers/pci/pci-driver.c*)

Драйвер PCI

pci driver должен выполнить следующие задачи в `init_module`:

- Определить `struct pci_driver`
- Инициализировать `struct pci_driver` structure
- Зарегистрировать `struct pci_driver` `pci_register_driver`

Структура `struct pci_driver` включает структуру `device_driver` для регистрации в модели устройств .

```
struct pci_driver {
    struct list_head node;
    char *name;
    const struct pci_device_id *id_table;

    int (*probe) (struct pci_dev *dev, const struct
pci_device_id *id);
    void (*remove) (struct pci_dev *dev);
    int (*suspend) (struct pci_dev *dev,
pm_message_t state);
    int (*suspend_late) (struct pci_dev *dev,
pm_message_t state);
    int (*resume_early) (struct pci_dev *dev);
    int (*resume) (struct pci_dev *dev);
    void (*shutdown) (struct pci_dev *dev);

    struct pm_ext_ops *pm;
    struct pci_error_handlers *err_handler;
    struct device_driver driver;
    struct pci_dynids dynids;
};
(linux-XXX\include\linux\pci.h)
```


Функции pci_driver

```
int (*probe) (struct pci_dev *dev, const struct pci_device_id *id); // Вставить устройство.  
void (*remove) (struct pci_dev *dev); // Удалить устройство (NULL если устройство  
    не HOT plug)  
int (*suspend) (struct pci_dev *dev, pm_message_t state); // Усыпить  
int (*suspend_late) (struct pci_dev *dev, pm_message_t state);  
int (*resume_early) (struct pci_dev *dev);  
int (*resume) (struct pci_dev *dev); //Пробудить  
void (*shutdown) (struct pci_dev *dev);
```

Пример

```
#include <linux/pci.h>

static struct pci_device_id rtl8139_pci_tbl[] = {
    {0x10ec, 0x8139, PCI_ANY_ID, PCI_ANY_ID, 0, 0, RTL8139 },
    {0x10ec, 0x8138, PCI_ANY_ID, PCI_ANY_ID, 0, 0, RTL8139 },
    {0,}
};

MODULE_DEVICE_TABLE (pci, rtl8139_pci_tbl)

static struct pci_driver rtl8139_pci_driver = {
    .name      = "foo8139",
    .id_table  = rtl8139_pci_tbl,
    .probe     = foo_probe,
    .remove    = foo_remove,
};

static int init_module (void) {
    return pci_register_driver(&rtl8139_pci_driver);
}
```

Пример

```
int foo_probe(struct pci_dev *dev, const struct pci_device_id *id) // Реализация probe
{
    port_addr = pci_resource_start(dev,0);
    major = register_chrdev(0,"MyPCI",&fops);

    printk(KERN_INFO "Load driver PCI %d\n",major);
    return 0;
}

void foo_remove(struct pci_dev *dev)
{
    unregister_chrdev(major,"MyPCI");
}

void cleanup_module(void)
{
    pci_unregister_driver(&rtl8139_pci_driver);
}
```

Нахождение устройства на шине PCI

```
int fooprobe (struct pci_dev *dm7820_pci, const struct pci_device_id *id) {  
  
    printk ("Get physics BAR0...");  
    dm7820_PCIMem.real = pci_resource_start (dm7820_pci,0);  
    dm7820_PCIMem.size = pci_resource_len (dm7820_pci,0);  
    if ((dm7820_PCIMem.real==0)|| (dm7820_PCIMem.size==0))  
        {printk ("failed.\n"); return -1;}  
    else  
        printk ("%u...OK.\n", (uint32)dm7820_PCIMem.real);  
    printk ("Checks physics BAR0...");  
    if (pci_resource_flags (dm7820_pci,0)&IORESOURCE_MEM)  
        printk ("OK.\n");  
    else {printk ("failed.\n"); return -1;}  
    printk ("Get virtual BAR0...");  
    dm7820_PCIMem.virtual=ioremap(dm7820_PCIMem.real,dm7820_PCIMem.size);  
    if (dm7820_PCIMem.virtual==0) {printk ("failed.\n"); return -1;}  
    else printk ("%u...OK.\n", (uint32)dm7820_PCIMem.virtual);  
    printk ("Request region BAR0...\t\t");  
    if (request_mem_region (dm7820_PCIMem.real,dm7820_PCIMem.size,DM7820_NAME))  
        printk ("OK.\n");  
    else {printk ("failed.\n"); return -1;}  
}
```

ioremap_nocache
исключена и
зменана на
ioremap

Задание

Разработать минимальный драйвер для pci адаптера сетевой карты.

Драйвер должен:

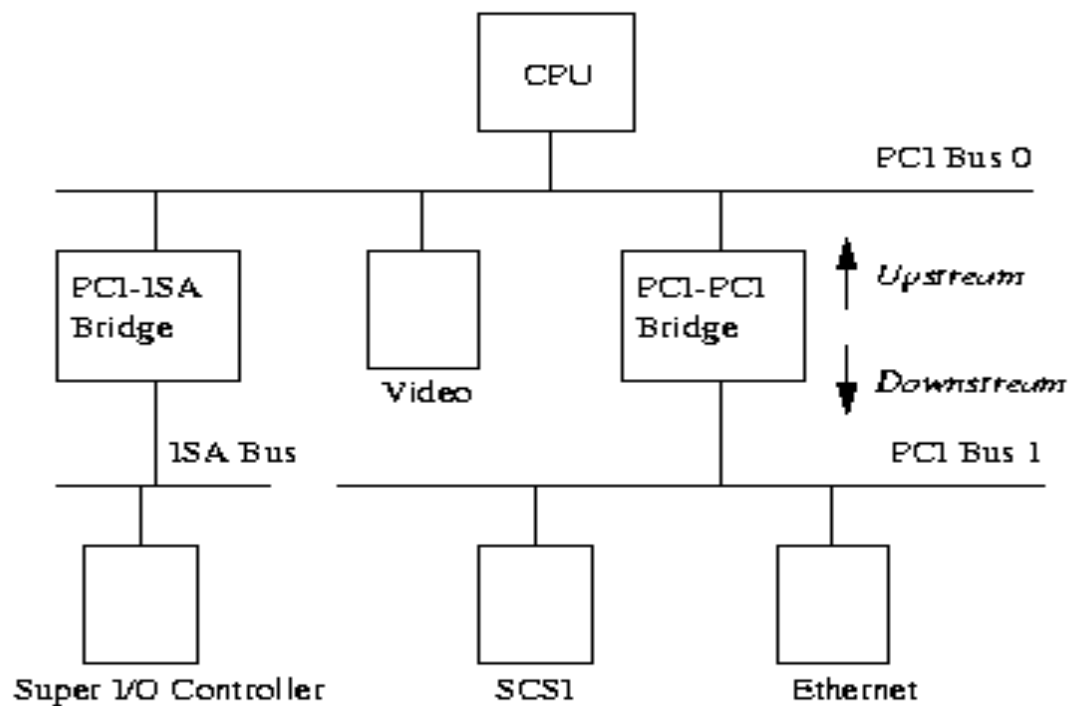
- Иметь функции probe, remove
- Регистрироваться в системе

Предварительно необходимо:

- 1.Посмотреть при помощи `lspci vendor_id, device_id` сетевого адаптера realtek
- 2.Посмотреть и записать MAC адрес сетевого адаптера realtek
- 3.Выключить сетевой адаптер при помощи `sudo ifconfig eth0 down` (или `sudo ifdown eth0`)
- 4.Выгрузить модуль сетевого драйвера realtek

Пример архитектуры системы

Работа с
памятью



Взаимодействие с устройством

Устройства взаимодействуют в CPU через специальные регистры:

- управляющий регистр;
- регистр состояния;
- входной регистр;
- выходной регистр.

Регистры могут:

- расположены в специальном адресном пространстве (в пространстве портов ввода/вывода);
- отображаться в память.

Операции чтения обмена с устройствами могут дать неожиданный результат

Получение доступа к портам ввода-вывода

```
#include <linux/ioport.h>
```

```
struct resource *request_region(unsigned long first, unsigned long n,  
const char *name);
```

```
void release_region(unsigned long start, unsigned long n);
```

```
int check_region(unsigned long first, unsigned long n); /*Устарела с версии 2.4.xx*/
```

Посмотреть распределение портов можно:

```
/proc/iports
```


Команды для работы взаимодействия через порты В.В.

- *Bytes*

`unsigned inb(unsigned port);`

`void outb(unsigned char byte, unsigned port);`

- *Words*

`unsigned inw(unsigned port);`

`void outw(unsigned char byte, unsigned port);`

- *"Long" integers*

`unsigned inl(unsigned port);`

`void outl(unsigned char byte, unsigned port);`

Реализация функций может отличаться на разных платформах.

Команды для работы взаимодействия через порты В.В.

- *byte strings*

`void insb(unsigned port, void *addr, unsigned long count);`

`void outsb(unsigned port, void *addr, unsigned long count);`

- *word strings*

`void insw(unsigned port, void *addr, unsigned long count);`

`void outsw(unsigned port, void *addr, unsigned long count);`

- *long strings*

`void insl(unsigned port, void *addr, unsigned long count);`

`void outsl(unsigned port, void *addr, unsigned long count);`

Передача нескольких значений. Может быть более эффективно, чем соответствующий цикл на С, если у процессора есть специальные команды

Получение адреса области портов ввода-вывода.

Адрес области портов ввода вывода может быть получен:

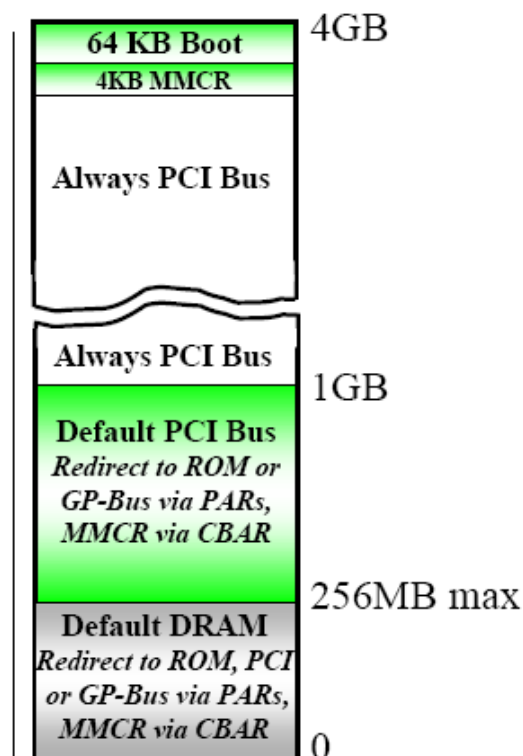
- Через параметры модуля;
- Известен заранее и закодирован в драйвере
- Запрошен у платформы
- Прочитан из области конфигурирования шины PCI

Практический пример.

Чтение портов ввода-вывода. Реализовать
макросы изменения, установки и очистки
бит в регистрах.

Обмен через память

- Обмен через область разделяемой памяти - наиболее распространенный способ взаимодействия.
- При взаимодействии важен порядок операций.
- Иногда возникает неожиданный результат



Резервирование области памяти

Область память должна быть выделена до использования. Функции определены в `<linux/ioports.h>`

```
struct resource * request_mem_region(unsigned long start,unsigned long len, char *name);
```

- возвращает NULL, если ошибка

```
void release_mem_region(unsigned long start,unsigned long len);
```

Уточнить назначение регионов памяти можно в **`/proc/iomem`**

Отображение памяти ввода-вывода в виртуальную память ядра

Для обращения к разделяемой памяти драйверу необходим виртуальный адрес области памяти.

Для реализации этой задачи используется функция `ioremap`:

```
#include <asm/io.h>;
```

```
void *ioremap(unsigned long phys_addr,unsigned long size);
```

(реализация `ioremap_nocache` аналогична `ioremap` на большинстве платформ)

Функция возвращает виртуальный адрес, соответствующий заданному физическому или NULL в случае неудачи.

```
void iounmap(void *address);
```

Отличие от обычной памяти

- Запросы чтения и записи могут быть кэшированы
- Компилятор может произвести оптимизацию и использовать регистры вместо памяти
- Компилятор может произвести изменение порядка команд
- CPU может переупорядочить команды

Решение проблем с памятью

- Кэширование I/O портов и памяти в.в. запрещается аппаратно или ядром Linux
- Можно использовать квалификатор `volatile`
- Для указания компилятору того, что нельзя использовать регистры вместо записи в память.
- Для предотвращения изменения порядка можно использовать барьеры памяти (Memory barriers).

Memory barriers. Логика работы

Барьер вставляется между операциями, которые должны быть видны аппаратуре в определенном порядке.

Пример:

```
writel(dev->registers.addr, io_destination_address); // Подготовка данных
writel(dev->registers.size, io_size);                  // Подготовка данных
writel(dev->registers.operation, DEV_READ);            // Подготовка данных
wmb( ); // БАРЬЕР НА ЗАПИСЬ – гарантирует, что все команды записи,
указанные до него выполнены
writel(dev->registers.control, DEV_GO);
```

Memory barriers

Барьер вставляется между операциями, которые должны быть видны аппаратуре в определенном порядке.

Аппаратно независимые. Влияют только на поведение компилятора. Не влияют на переупорядочивание CPU

```
#include <asm/kernel.h>
```

```
void barrier(void);
```

Аппаратно зависимые

```
#include <asm/system.h>
```

```
void rmb(void);
```

```
void wmb(void);
```

```
void mb(void);
```

Безопасны на всех архитектурах.

Еще один пример использования барьеров

```
while (count--) {  
    outb(*(ptr++), port);  
    wmb( );  
}
```

Запись необходимо произвести сейчас, без оптимизации.

Функции обмена через разделяемую память В.В.

Для обеспечения портабельности кода лучше использовать специальные функции:

```
unsigned int ioread8(void *addr);  
unsigned int ioread16(void *addr);  
unsigned int ioread32(void *addr);  
void iowrite8(u8 value, void *addr);  
void iowrite16(u16 value, void *addr);  
void iowrite32(u32 value, void *addr);
```

Чтение или запись нескольких байт:

```
void ioread8_rep(void *addr, void *buf, unsigned long count);  
void ioread16_rep(void *addr, void *buf, unsigned long count);  
void ioread32_rep(void *addr, void *buf, unsigned long count);  
void iowrite8_rep(void *addr, const void *buf, unsigned long count);  
void iowrite16_rep(void *addr, const void *buf, unsigned long count);  
void iowrite32_rep(void *addr, const void *buf, unsigned long count);
```

Дополнительные утилиты:

```
void memset_io(void *addr, u8 value, unsigned int count);  
void memcpy_fromio(void *dest, void *source, unsigned int count);  
void memcpy_toio(void *dest, void *source, unsigned int count);
```

Непосредственное чтение или запись по адресу полученному от ioremap может не работать на некоторых архитектурах.

Как узнать параметры области память в.в

- Современные периферийные устройства для РС не используют фиксированных адресов памяти
- Устройство получает физический адрес неиспользованной области памяти CPU в процессе конфигурирования системы (обычно это обязанность BIOS)
- Расположение и размер области памяти запоминается в специальной non-volatile battery-powered RAM ('configuration memory')

Пример работы с памятью В.В

```
printk ("Get virtual BAR...\t\t");
device->virtual = ioremap (device->real,device->size);
if (device->virtual==0) {
    printk ("failed.\n"); return -1;
}
else printk ("%u...OK.\n",(uint32)device->virtual);

printk ("Request region BAR...\t\t");
if (request_mem_region (device->real,device->size,CAN527_NAME))
    printk ("OK.\n");
else {
    printk ("failed.\n"); return -1;
}
printk ("Set IRQ...");
status=request_irq(device->irq,device->irq_handler, IRQF_SHARED,
    CAN527_NAME,device);
```

Литература

1. Memory management в ядре Linux. Семинар в Яндексе
<https://habr.com/ru/companies/yandex/articles/231957/>
2. Understanding VFIO and UIO Driver Framework - Vandana Salve, Prasme Systems
<https://www.youtube.com/watch?v=uOQ5POP8hCs>
3. <https://bootlin.com/doc/training/linux-kernel/linux-kernel-slides.pdf>
4. Ordering I/O writes to memory-mapped addresses
https://docs.kernel.org/driver-api/io_ordering.html
5. Different I/O Access Methods for Linux, What We Chose for ScyllaDB, and Why
<https://www.scylladb.com/2017/10/05/io-access-methods-scylla/>
6. Linux-Kernel Memory Model
<https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p0124r6.html>