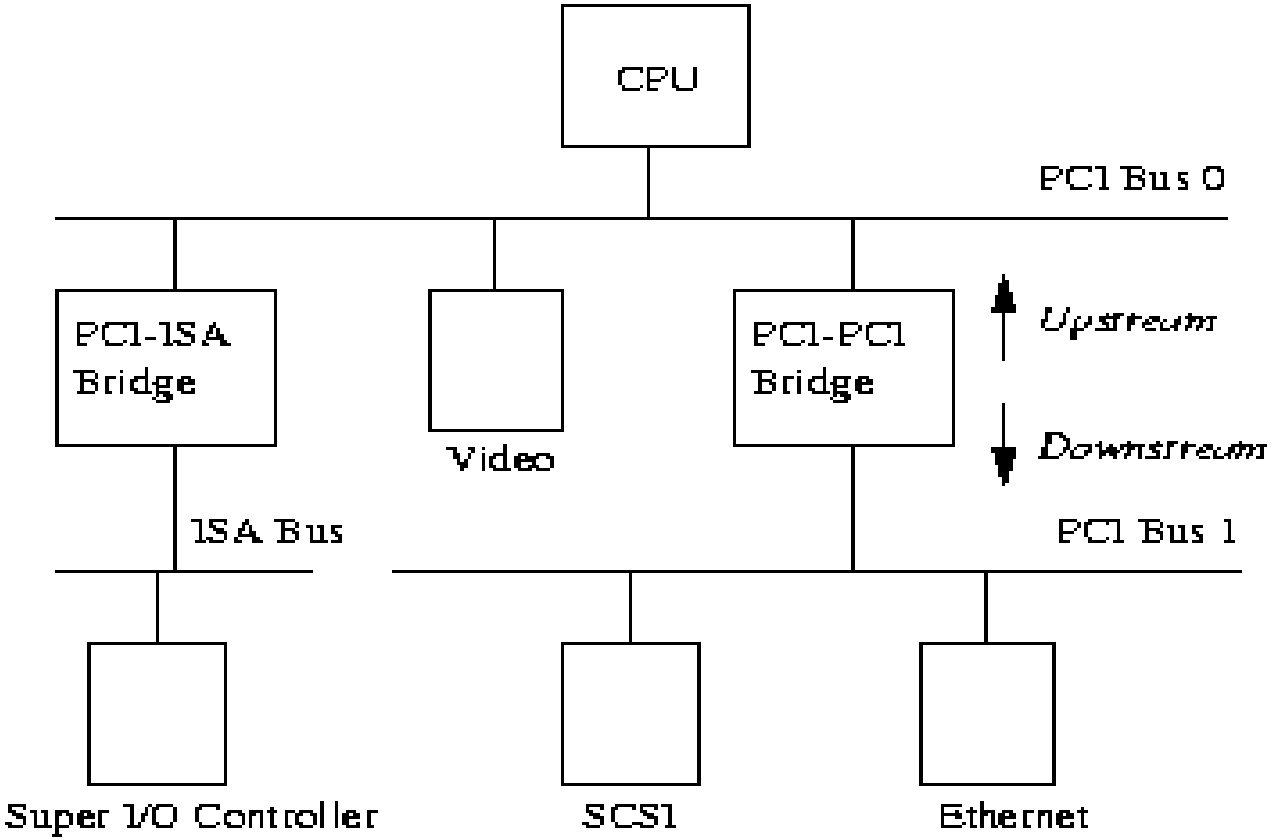
Лекция 3.1 Операции ввода-вывода

Разработали: Максимов А.Н., Крапивный А.В.

Содержание

- Работа с вводом выводом
- Обзор
- Получение доступа к портам ввода-вывода
- Использование памяти ввода-вывода
- Барьеры.
- Шина PCI

Пример архитектуры системы



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/ioports

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 C, если у процессора есть специальные команды

Получение адреса области портов ввода-вывода.

Адрес области портов ввода вывода может быть получен:

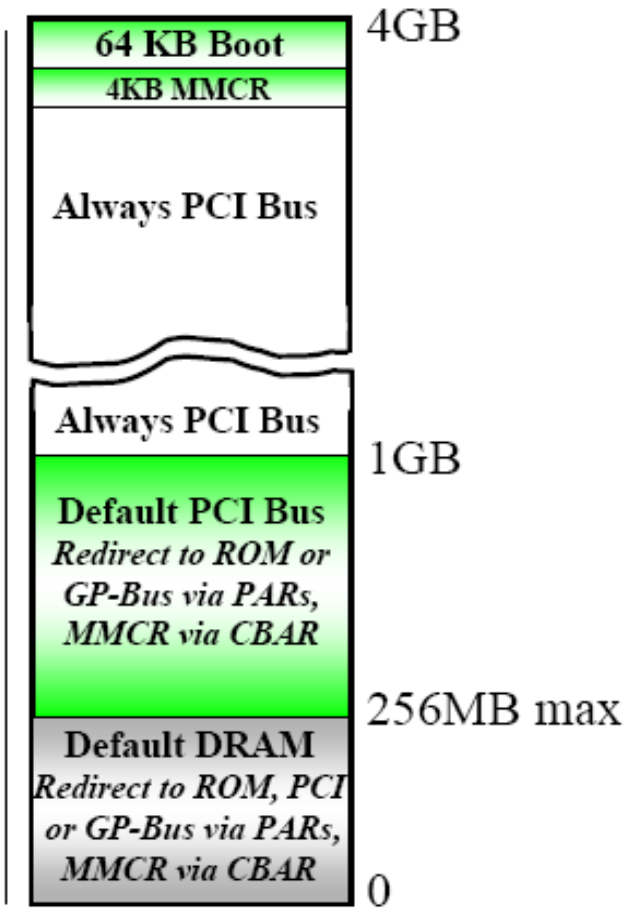
- Через параметры модуля;
- Известен заранее и закодирован в драйвере
- Запрошен у платформы
- Прочитан из области конфигурирования шины PCI

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_nocash 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 не 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 PC 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_nocache (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,SA_SHIRQ,CAN527_NAME,device);
  if (status!=0) {printk ("failed.\n"); return -1;} else  printk
     ("%i...OK.\n",device->irq);
```