Лекция 4.2 Прерывания

Разработал: Максимов А.Н.

Версия 1.7 2024/06

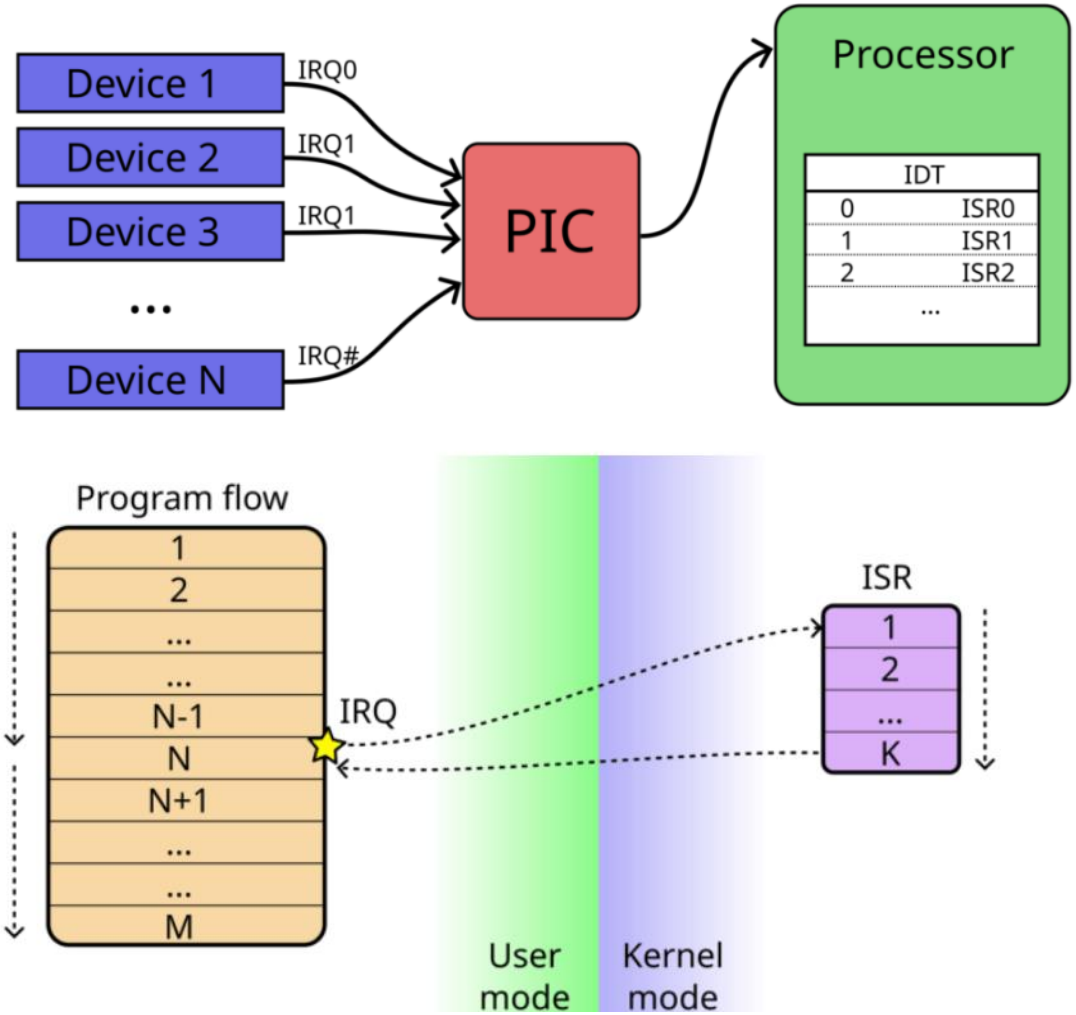
Содержание

- Обзор
- Функции для работы с прерываниями
- Нахождение прерывания
- Пример PCI
- Пример beegle bone

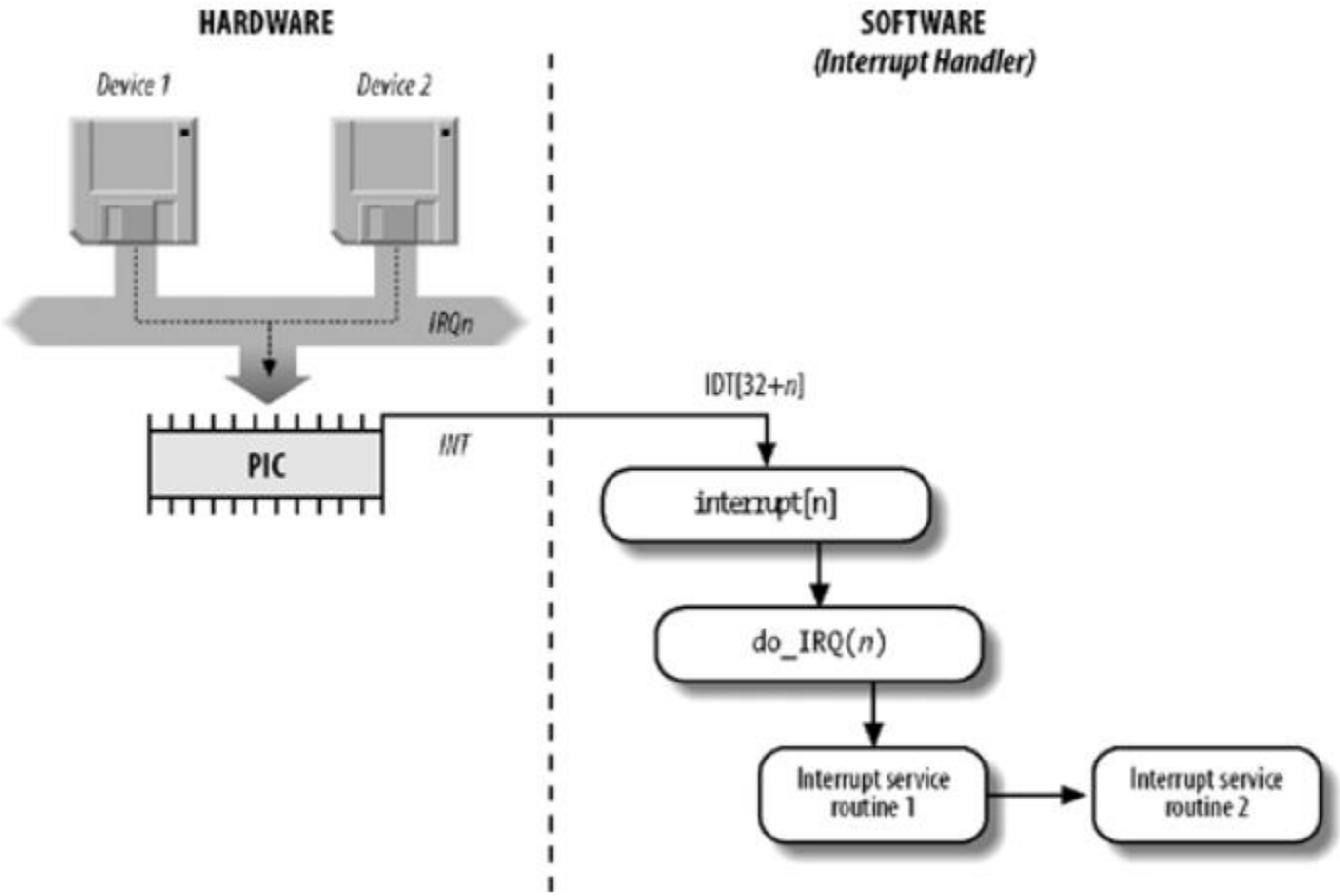
Прерывание

- Сигнал, сообщающий процессору о наступлении какого-либо события.
- При этом выполнение текущей последовательности команд приостанавливается, и управление передаётся обработчику прерывания, который выполняет работу по обработке события и возвращает управление в прерванный код.

Прерывание


Device 1
Device 2
Device 3
...
Device N
IRQ0
IRQ1
IRQ1
IRQ#
PIC
Processor
IDT
0 ISR0
1 ISR1
2 ISR2
...
Program flow
1
2
...
...
N-1
N
N+1
...
...
M
IRQ
ISR
1
2
...
K
User mode
Kernel mode

Прерывание обработка в Linux



Типы прерываний

В зависимости от источника возникновения сигнала прерывания делятся на:

- Аппаратные - события от периферийных устройств (например, нажатия клавиш клавиатуры, движение мыши, сигнал от таймера, сетевой карты или дискового накопителя) - внешние прерывания, или события в микропроцессоре - (например, деление на ноль) - внутренние прерывания;
- Программные — генерируются инициируются выполняемой программой явным исполнением специальных инструкций, то есть синхронно, а не асинхронно. Программные прерывания могут служить для вызова сервисов операционной системы.

Маскируемые и немаскируемые прерывания

- Маскируемые прерывания – все запросы на прерывания (IRQ) от внешнийх устройств приводят к возникновению маскируемых прерываний.
- Немаскируемые прерывания – критические события, например, аппаратные сбои приводят к появлению немаскируемых прерываний.

Маскируемые прерывания

Можно выделить три типа маскируемых прерываний:

- Прерываний ввода-вывода;
- Прерывания таймера
- Межпроцессорные прерывания

Обработка прерываний ввода-вывода

- Прерывание ввода-вывода генерируется внешним устройством, чтобы информирования CPU о состоянии устройства.
- Прерывание может рассматриваться как ресурс и ядро хранит список используемых прерываний.
- Модуль может зарегистрировать в ядре функцию обработчик прерывания.
- После окончания работы драйвер должен освободить прерывание.

Регистрация обработчика прерывания

Заголовочный файл <linux/interrupt.h>

Регистрация обработчика

int request_irq(unsigned int irq, irqreturn_t (*handler)(int irq, void *, struct pt_regs *), unsigned long flags, const char *dev_name,

void *dev_id);

Возвращаемое значение: 0 — success, Ошибка -EBUSY

irq — номер прерывания

handler — обработчик прерывания

dev_name — имя устройства

flags — тип прерывани (IRQF_DISABLED, IRQF_SHARED, IRQF_SAMPLE_RANDOM)

dev_id — параметр используется для идентификации устройства при разделяемых прерываниях. Если прерывание не разделяется, то может быть NULL.ned int irq, void *dev_id);

Flag -типы прерываний

IRQF_DISABLED

"Quick" interrupt handler. Выполняется при с отключение всех остальных прерываний на текущем процессоре(instead of just the current line). Необходимо использовать только при необходимости т.к. влияет на время реакции

IRQF_SHARED выполняется с запретом только текущего прерывания на локальном процессоре. Прерывание может быть разделяться несколькими устройствами. Необходимо проверять апаратный регистр статуса для определения произошло ли прерывание

IRQF_SAMPLE_RANDOM — используется дле генерации /dev/random and /dev/urandom.

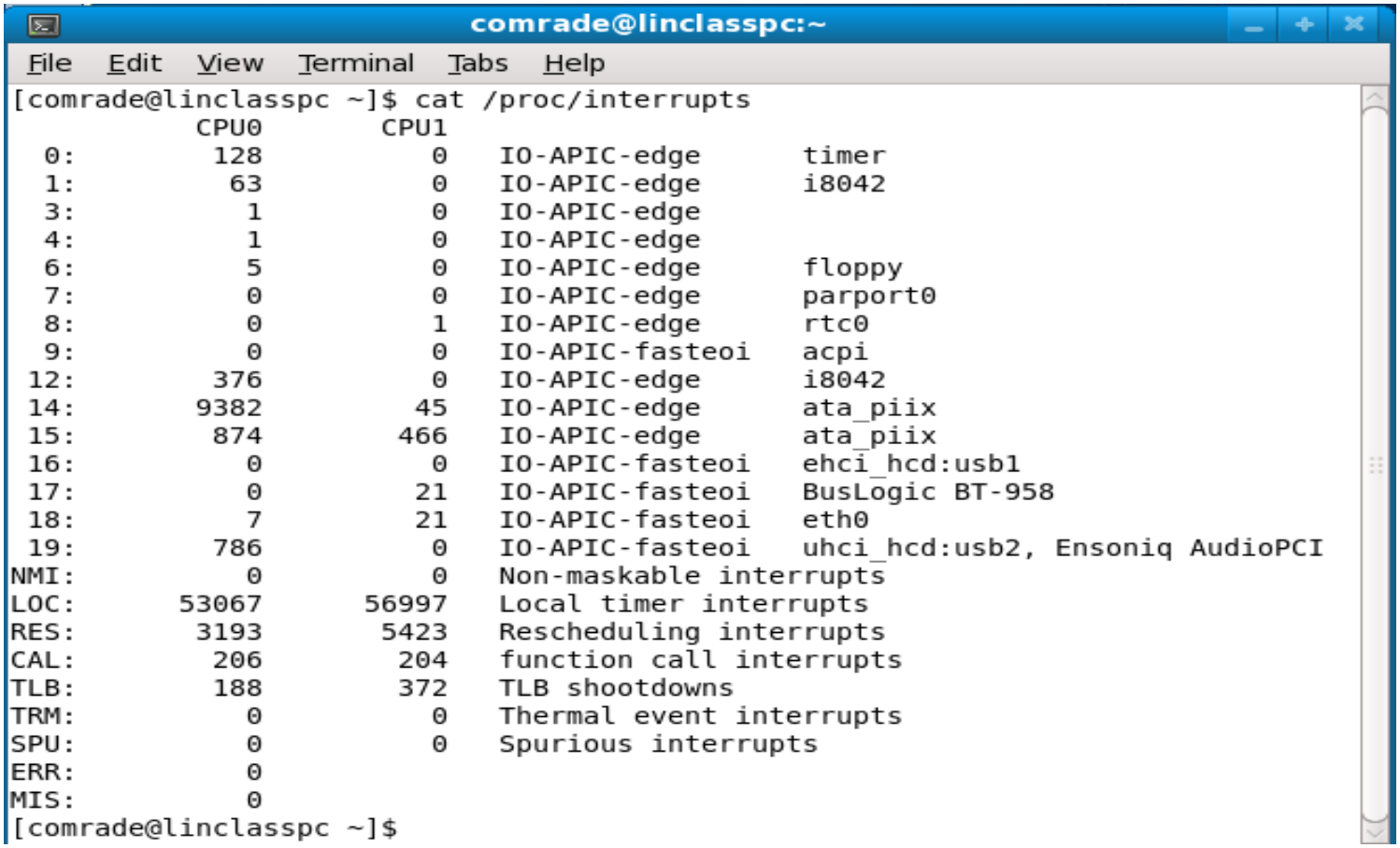
Удаление обработчика прерывания

Заголовочный файл <linux/interrupt.h>

```
void free_irq(unsigned int irq, void *dev_id);
```

Зарегистрированые прерывания

/proc/interrupts



Как узнать номер прерывания

- Передать в качестве параметра в модуль
- Запросить у подситемы шины PCI
- Запросить у подситемы платформы

Пример определения вектора прерывания

```
int probe(struct pci_dev *pdev, …) {
   int res = 0;
   res = pci_enable_device_mem(pdev);

…..

  rtl8139_dev->irq = pdev->irq;  // Номер прерывания

…..

}
```

Прототип обработчика прерываний

```
irqreturn_t (*handler) (
int irq,            // irq number of the current interrupt
void *dev_id, // Pointer used to keep track of the corresponding device.
// Useful when several devices are managed by the same module
);
```

Возвращаемое значение:

IRQ_HANDLED: прерывание наше и обреботано

IRQ_NONE: не наше прерывание. Полезно при работе с разделяемыми обработчиками и при странные и для информирования ядра о spurious interruptsl.

Когда устанавливать обработчик

init_module:

- много каналов IRQ используется в холостую!

При открытии устройства (первый вызов файловой операции open):

- эффективно используются IRQ каналы;
- необходимо отслеживать число открытий устройства, чтобы освободить IRQ, когда оно больше не используется;
- Возможна частичная потеря информации.

Ограничения обработчиков прерываний

- Обмениваться данными с пространством пользовательских задач т.к выполняется не в контексте процесса;
- Выполнять оперции приводящие к засыпанию, например вызов wait_event
- Выделять память с другими квалификаторами, кроме GFP_ATOMIC
- Захватывать семафоры
- Обращаться к диспетчеру

Типичная функциональность обработчика прерывания

- Подтвердить прием прерывания устройству, чтобы не было последующих
- Произвести обмен с устройством
- Разбудить ожидающие окончания операции процессы wake_up_interruptible(&module_queue);

Пример установки обработчика

```
int foo_probe(struct pci_dev *devp, const struct pci_device_id *id) {
  int res = 0;
  res = pci_enable_device_mem(devp);

  if ( !devp ) return -ENODEV;
  irq = devp->irq;

  if ( request_irq( irq, my_isr, IRQF_SHARED, devname, devp ) )
     return -EBUSY;
....
}
```

(lect_usfca1\receiver.c)

Реализация обработчика прерывания

```
irqreturn_t my_isr(int irq, void *dev_id ) {
   static int     rep = 0;
   int       intstatus = inw( iobase + 0x3E );
   if ( intstatus == 0 ) return IRQ_NONE; // it wasn't for us
   ++rep;                              // count the interrupt
   if ( intstatus & 0x3 )  {
      wake_up_interruptible( &wq );   // wake up any readers
      outw( 0x3, iobase + 0x3E );       // clear the interrupt
      return      IRQ_HANDLED;
   }
   printk( "unexpected RTL-8139 interrupt! " );
   printk( " (intstatus=%04X) \n", intstatus );
   outw( intstatus, iobase + 0x3E );
   return  IRQ_HANDLED;
}
```

Запрет прерываний.

В некоторых случаях в качестве средства синхронизации может использоваться запрет прерываний.

Позволяет гарантировать, что обработчик прерывания не произойдет в процессе выполнения кода ядра (including kernel preemption)

Запрет прерываний для локального CPU:

/*Устаревшиие*/

unsigned long flags;

local_irq_save(flags); // Прерывание запрещены,

local_irq_restore(flags); // Прерывания разрешены в предыдущее состояние.

spinlock_t device_lock;

spin_lock_init (&device_lock); /*Инициализация структуры*/

spin_lock_irqsave(&device_lock, irq_flags); // Прерывание запрещены,

spin_unlock_irqrestore(&device_lock,irq_flags); // Прерывания разрешены в предыдущее состояние.

Оба вызова должны быть в одной функции!

Пример использования запрета прерываний.

```
my_ioctl()
{       spin_lock_irq(&my_lock);
        /* критическая секция */
        spin_unlock_irq(&my_lock);
}
/*Для обработчика прерываний*/
my_irq_handler()
{
        spin_lock(&lock);
        /* критическая секция */
        spin_unlock(&lock);
}
```

Маскирование прерываний

`void disable_irq (unsigned int irq);`

Отключает прерывания для всех процессоров в системе. Ждет все выполняющиеся в настоящее время обработчики.

`void disable_irq_nosync (unsigned int irq);`

Не ждет, пока обработчики выполняться.

`void enable_irq (unsigned int irq);`

Восстанавливает прерывания.

`void synchronize_irq (unsigned int irq);`

Ожидает, пока все обработчики выполняться.

Проверка статуса прерываний

Можно проверить выполняется ли текущий код в контексте прерывания. Например для того, чтобы вызвать функция приводящую к засыпанию.

irqs_disabled()

Проверяет запрещена ли генерация локальных прерываний.

in_interrupt()

Проверяем находимся ли мы в контексте прерывания

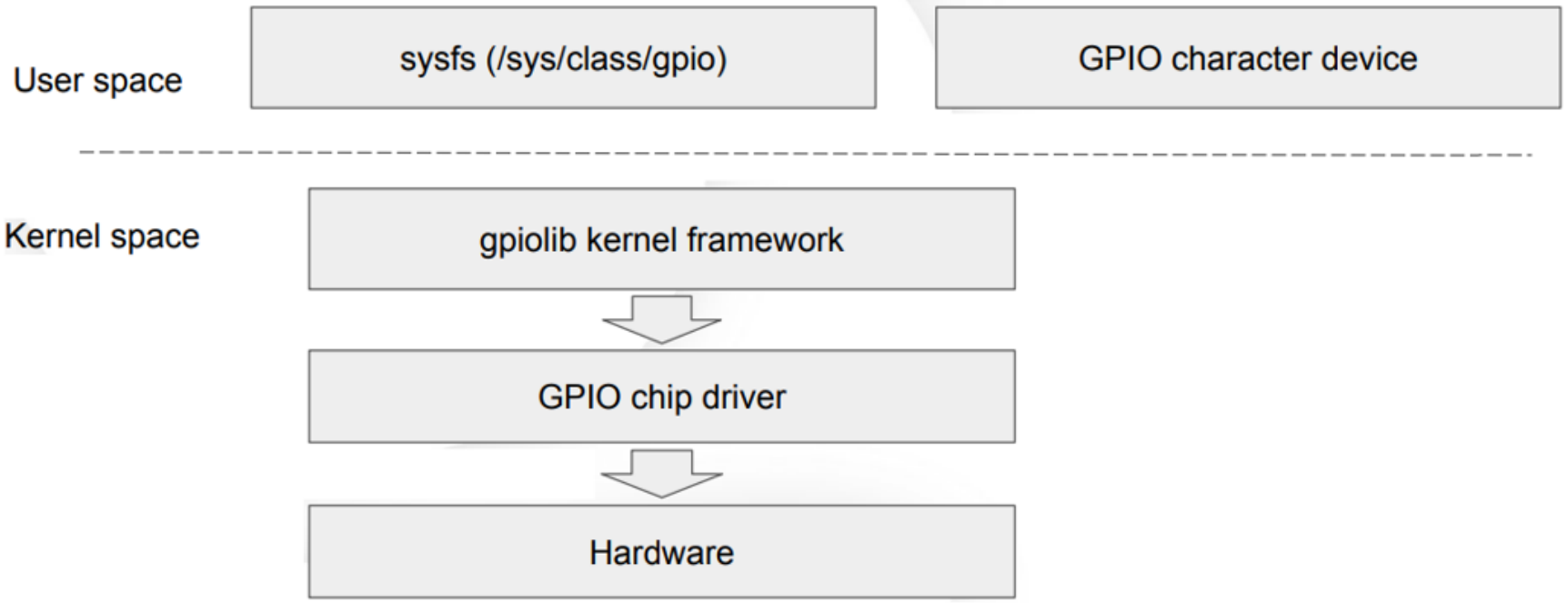
in_irq()

Проверяем находимся ли мы в обработчике прерывания.

Пример обработчика прерываний

```
irqreturn_t dm7820_handler (int irq, void *dev_id,)  {
  spinlock_t slock;
  uint32 pci_status;
  spin_lock (&slock);
  if (irq!=dm7820_pci->irq) {spin_unlock (&slock);return;}
  pci_status=IO_MEMORY_READ32 (dm7820_PCIMem.virtual+0x68);
  if (!(pci_status&0x00608000)) {spin_unlock (&slock);return;}
  if ((IO_MEMORY_READ8 (dm7820_PCIMem.virtual+0xA8)&0x10)&&   (IO_MEMORY_READ8
       (dm7820_PCIMem.virtual+0xA9)&0x10))
  {
      dm7820_DMABuffer[dm7820_ptrSendDMABuffer].busy=0;
      if (dm7820_ptrSendDMABuffer>=(DM7820_DMABUFFERCOUNT-1)) dm7820_ptrSendDMABuffer=0; else
        dm7820_ptrSendDMABuffer++;
      IO_MEMORY_WRITE16 (0x0000,dm7820_FPGAMem.virtual+0x0288);
       IO_MEMORY_WRITE32
        (dm7820_DMABuffer[dm7820_ptrSendDMABuffer].bus+DM7820_DMABUFFERSIZE,dm7820_PCIMem.virtual+0x0098);
       IO_MEMORY_WRITE8 (0x0B,dm7820_PCIMem.virtual+0xA9);
  }
  spin_unlock (&slock);
  return IRQ_HANDLED;
}
```

Пример драйвера в ядре - GPIO

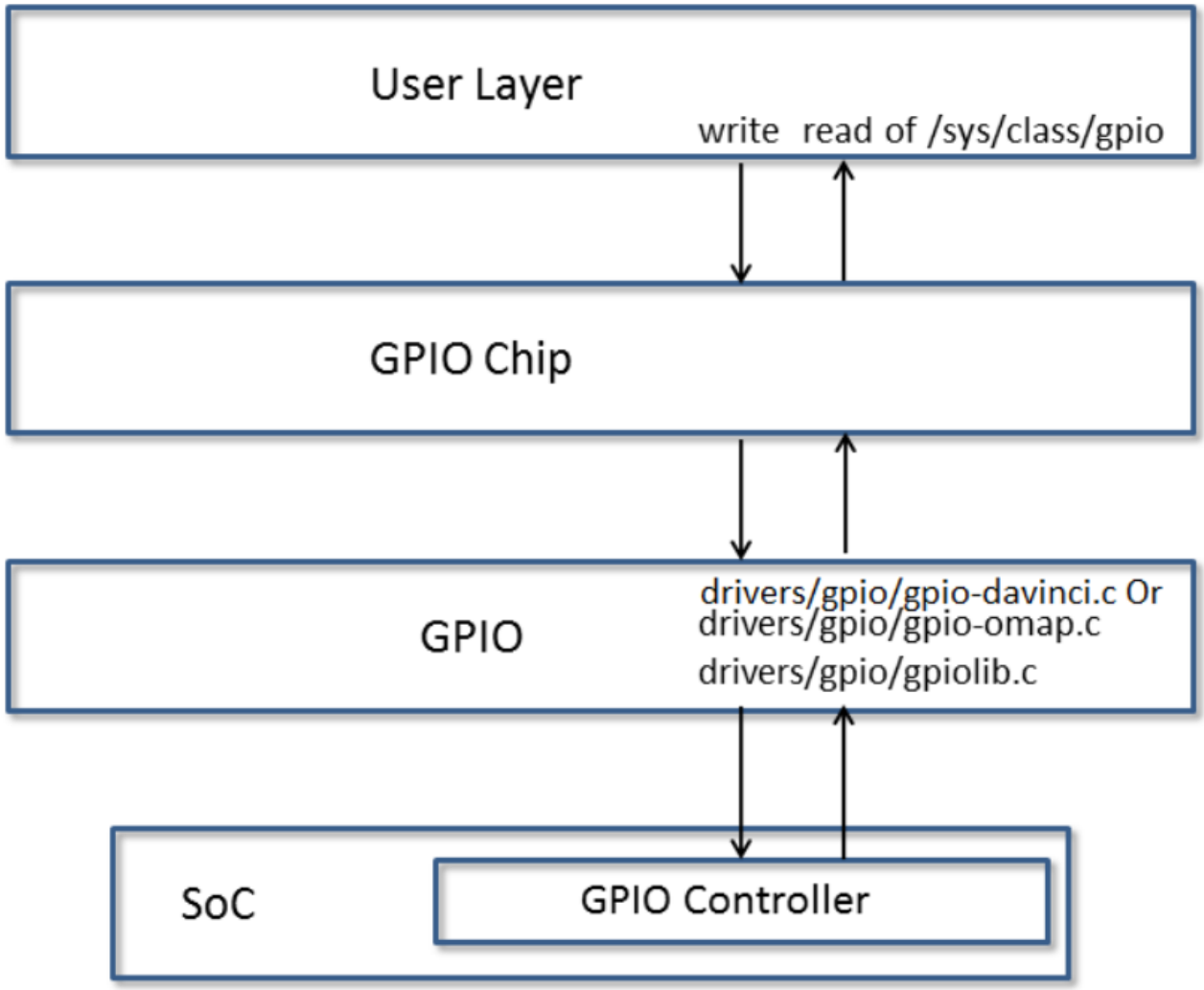


Beegle bone dts

Device tree дял BeagleBone's определена в:

- am335x-boneblack.dts
- am33xx.dtsi
- am335x-bone-common.dtsi

Beegle bone архитектура gpio



https://software-dl.ti.com/processor-sdk-linux/esd/docs/latest/linux/Foundational_Components/Kernel/Kernel_Drivers/GPIO.html

RISoft

Пример интерфейса GPIO через sysfs

экспортируем пин, без этого шага им не получится управлять:
echo 2 > /sys/class/gpio/export

Делаем его out-пином, то есть, он будет либо подавать, либо не подавать напряжение в 3.3 вольта:
echo out > /sys/class/gpio/gpio2/direction

Подаем напряжение:
echo 1 > /sys/class/gpio/gpio2/value

Перестаем подавать напряжение:
echo 0 > /sys/class/gpio/gpio2/value

Узнаем, подается ли сейчас напряжение:
cat /sys/class/gpio/gpio2/value

По завершении работы пину можно сделать unexport:
echo 2 > /sys/class/gpio/unexport

Gpio в device tree

Раздел относящий к контроллеру прерываний в am33xx.dtsi

```
gpio1: gpio@44e07000 {
  compatible = "ti,omap4-gpio";
  ti,hwmods = "gpio1";
  gpio-controller;
  interrupt-controller;
  reg = <0x44e07000 0x1000>;
  interrupts = <96>;
};
```

https://www.righto.com/2016/08/the-beaglebones-io-pins-inside-software.html

Gpio в device tree

Раздел относящий к контроллеру прерываний в am33xx.dtsi

```
gpio1: gpio@44e07000 {
  compatible = "ti,omap4-gpio";
  ti,hwmods = "gpio1";
  gpio-controller;
  interrupt-controller;
  reg = <0x44e07000 0x1000>;
  interrupts = <96>;
};
```

https://www.righto.com/2016/08/the-beaglebones-io-pins-inside-software.html

Драйвер gpio для beegle bone

```
static struct platform_driver omap_gpio_driver = {
        .probe          = omap_gpio_probe,
        .remove         = omap_gpio_remove,
        .driver         = {
                .name       = "omap_gpio",
                .pm   = &gpio_pm_ops,
                .of_match_table = omap_gpio_match,
        },
};
static int __init omap_gpio_drv_reg(void)
{
        return platform_driver_register(&omap_gpio_driver);
}
postcore_initcall(omap_gpio_drv_reg);

static void __exit omap_gpio_exit(void)
{
        platform_driver_unregister(&omap_gpio_driver);
}
module_exit(omap_gpio_exit);

MODULE_DESCRIPTION("omap gpio driver");
MODULE_ALIAS("platform:gpio-omap");
MODULE_LICENSE("GPL v2");
```

https://elixir.bootlin.com/linux/v5.10.209/source/drivers/gpio/gpio-omap.cl

Драйвер gpio для beegle bone

```
static const struct of_device_id omap_gpio_match[] = {
	{
		.compatible = "ti,omap4-gpio",
		.data = &omap4_pdata,
	},
	{
		.compatible = "ti,omap3-gpio",
		.data = &omap3_pdata,
	},
	{
		.compatible = "ti,omap2-gpio",
		.data = &omap2_pdata,
	},
	{ },
};
MODULE_DEVICE_TABLE(of, omap_gpio_match);
```

https://elixir.bootlin.com/linux/v5.10.209/source/drivers/gpio/gpio-omap.cl

Драйвер gpio ресурсы и прерывания

```
static int omap_gpio_probe(struct platform_device *pdev) {
        struct device *dev = &pdev->dev;
        struct device_node *node = dev->of_node;
        const struct of_device_id *match;
        const struct omap_gpio_platform_data *pdata;
        struct gpio_bank *bank;
        struct irq_chip *irqc;
        int ret;

        match = of_match_device(of_match_ptr(omap_gpio_match), dev);
        pdata = match ? match->data : dev_get_platdata(dev);
…
      bank->irq = platform_get_irq(pdev, 0);
…
      bank->base = devm_platform_ioremap_resource(pdev, 0);
…
        ret = devm_request_irq(bank->chip.parent, bank->irq,
                        omap_gpio_irq_handler,
                        0, dev_name(bank->chip.parent), bank);
```

https://elixir.bootlin.com/linux/v5.10.209/source/drivers/gpio/gpio-omap.cl

Драйвер gpio обработчик прерывания

```
static irqreturn_t omap_gpio_irq_handler(int irq, void *gpiobank)
{
    ...

    return IRQ_HANDLED;
}
```

https://www.righto.com/2016/08/the-beaglebones-io-pins-inside-software.html

Практическое задание. Вариант 0.

Добавить обработчик прерывания в функцию open сетевого драйвера, а также удаление обработчика из функции stop.

Проверить:

sudo ifconfig eth0 up -> обработчик установливается см. .cat /proc/interrupt

sudo ifconfig eth0 down -> обработчик установливается см. .cat /proc/interrupt

Информацию о прерывании передавать через поле в структуре net_device !!!

Практическое задание. Вариант 1.

Установить разделяемый обработчик прерывания на прерывания от клавиатуры. Отобразить через proc число пришедших прерываний.

Практическое задание. Вариант 2.

Разработать символьный драйвер для приема пакетов от сетевого адаптера RTL8139.

TODO – phy link ip interrupt !!!

Общая информация о RTL8139

Принятые пакеты в адаптере помещаются в циклический буфер из аппаратного FIFO.

После записи пакета в циклический буфер происходит следующее:

- Перед пакетом помещается заголовок (статус и длина пакета). Формат статуса в заголовке приведен на рис.1.
- После ппередачи обновляется регистр CBA = конец пакета (текущая позиция записи).
- Обновляются CMD(BufferEmpty) = 1 and ISR(TOK)=1
- Вызывается обработчик прерывания, который должен очистить ISR(TOK)=0 и обновить CAPR.

Регистр CBA хранит адрес данных помещенных в буфер. CAPR хранит текущую позицию чтения.

Receive staus register

Receive Status Register in Rx packet header

Bit	R/W	Symbol	Description
15	R	MAR	**Multicast Address Received:** This bit set to 1 indicates that a multicast packet is received.
14	R	PAM	**Physical Address Matched:** This bit set to 1 indicates that the destination address of this packet matches the value written in ID registers.
13	R	BAR	**Broadcast Address Received:** This bit set to 1 indicates that a broadcast packet is received. BAR, MAR bit will not be set simultaneously.
12-6	-	-	**Reserved**
5	R	ISE	**Invalid Symbol Error:** (100BASE-TX only) This bit set to 1 indicates that an invalid symbol was encountered during the reception of this packet.
4	R	RUNT	**Runt Packet Received:** This bit set to 1 indicates that the received packet length is smaller than 64 bytes (i.e. media header + data + CRC < 64 bytes)
3	R	LONG	**Long Packet:** This bit set to 1 indicates that the size of the received packet exceeds 4k bytes.
2	R	CRC	**CRC Error:** When set, indicates that a CRC error occurred on the received packet.
1	R	FAE	**Frame Alignment Error:** When set, indicates that a frame alignment error occurred on this received packet.
0	R	ROK	**Receive OK:** When set, indicates that a good packet is received.

Инициализация

```
int init_module( void ) {
  printk(KERN_INFO "\nInstalling %s (major=%d) \n", modname,my_major );
  devp = pci_find_device( VENDOR_ID, DEVICE_ID, devp );
  if ( !devp ) return -ENODEV;
  irq = devp->irq;
  iobase = pci_resource_start( devp, 0 ); // Используем BAR0
  outb( 0x10, iobase + 0x37 );                // Сбросить сетевой интерфейс
  printk( "RealTek 8139 Network Interface: iobase=%04X \n", iobase );
  while ( inb( iobase + 0x37 ) & 0x10 );

  init_waitqueue_head( &wq );
  if ( request_irq( irq, my_isr, IRQF_SHARED, devname, &devname ) )
       return -EBUSY;
  kmem = kmalloc( KBUFSIZ, GFP_KERNEL );
  if ( !kmem ) return -ENOMEM;
  kmem_base = __pa( kmem );
  kmem_size = KBUFSIZ;
  printk( "Physical address-range of kernel buffer: " );
  printk( "%08lX-%08lX \n", kmem_base, kmem_base+kmem_size );
  outl( kmem_base, iobase + 0x30 );  // Передать адрес приемного буфера в  RBStart
  return register_chrdev( my_major, devname, &my_fops );
}
```

Открытие устройства.

Для инициализации приема данных необходимо выполнить следующие действия:

Разрешить прием пакетов (регистр CR).

Проинициализировать регистр (RCR)

Проинициализировать текущий адрес чтения пакета (CAPR)

Подтвердить любые предыдущие прерывания (ISR = 0xffff)

Разрешить генерацию прерываний на приход пакета (IMR=0x73)

Разрешение приема пакетов производится путем установки соответсвующего бита (маска 0x8) в регистре управления CR (смещение 0x37).

Проинициализировать регист конфигурации приема пакетов RCR (смещение 0x44). В качестве маски можно взять следующую:

0xE70F

(Принимать все пакеты, включая broadcast и multicast, Max DMA Burst Size per Rx DMA Burst

= не огнаничена, размер RX буфера 8k+16 байт, минимальное количество принятых данных после которого данные из аппаратного FIFO передаются в циклический буфер = после приема всего пакета)

Открытие устройства.

```
int my_open( struct inode *inode, struct file *file )
{
   outb( 0x08, iobase + 0x37 );                 // Разрешить прием пакетов
   outl( 0xE70F, iobase + 0x44 );               // Установить Rx Configuration
   outw( KBUFSIZ-16, iobase + 0x38 );           // Инициализировать CAPR в KBUFSIZ
   outw( 0xFFFF, iobase + 0x3E );               // Подтвердить все прерывания
   outw( 0x0073, iobase + 0x3C );               // Разрещить прерывания по вх. пакетам
  return       0;  // SUCCESS
}
```

Обработка прерывания.

```
static irqreturn_t my_isr(int irq,void *dev_id) {
  static int     rep = 0;
  int    intstatus = inw( iobase + 0x3E );        // Прочесть регистр статусв
  if ( intstatus == 0 ) return IRQ_NONE;       // Не наше прерывание
      ++rep;                                                  // Увеличение счетчика прерываний
  printk( "#%d: intstatus=%04X \n", rep, intstatus );
  if(intstatus & 0x3) {                    // Прерывание на входящий пакет
      wake_up_interruptible(&wq); // Пробудить ожидающие процессы
      outw( 0x3, iobase + 0x3E );    // Подтвердить прерывание
      return    IRQ_HANDLED;
  }
  printk( "unexpected RTL-8139 interrupt! " );
  printk( " (intstatus=%04X) \n", intstatus );
  outw( intstatus, iobase + 0x3E );
  return IRQ_HANDLED;
}
```

Чтение данных.

При чтении данных выполняются следущие операции:

- Ожидаени до тех пор пока в буфере данных не появятся данные (CR&0x1);
- Из CARP читаем текущее смещени в буфере данных;

Чтение данных.

```
ssize_t my_read( struct file *file, char *buf, size_t len, loff_t *pos ) {
   unsigned int         offset, nbytes, header, status, length, nextrx;
   char           *where = (char*)kmem; // Указатель на начало буфера
    // Спать до прихода данных в Receive Buffer( // 0-й байт CR - признак Rx-буфер пуст)
   wait_event_interruptible( wq, (inb( iobase + 0x37 )&1) == 0 );
   //  Готовимся читать пакет и обновлять CAPR
   offset = ( inw( iobase + 0x38 ) + 16 ) % KBUFSIZ;
   where += offset;
   header = *(unsigned int*)where;
  status = header & 0xFFFF;
  length = header >> 16;
  nextrx = offset + 4 + ((length + 3 )&~3);
  nextrx -= 16;
  nextrx %= KBUFSIZ;
  where += 4;                           // skip packet's header
  nbytes = length - 4;                          // omit the 4-bytes CRC
  if ( nbytes > len ) nbytes = len;   // don't exceed request
  if ( copy_to_user( buf, where, nbytes ) ) return -EFAULT;
  // update the Receive Buffer's 'guard' register
  outw( nextrx, iobase + 0x38 );              // update CAPR register
  // advance the file position and report count of bytes copied
  *pos += nbytes;
   printk("My read end.\n");
  return         nbytes;
}
```

Регистры.

0030h-0033h	RBSTART	Receive (Rx) Buffer Start Address
0037h	CR	Регистр команд.
0038h-0039h	CAPR	Текущий адрес чтения пакета (Current Address of Packet Read)
003Ah-003Bh	CBA	Current Buffer Address: The initial value is 0000h. It reflects total received byte-count in the rx buffer
003Ch-003Dh	IMR	Регист маски прерываний (IMR)
003Eh-003Fh	ISR	Регистр статуса прерываниий. (Interrupt Status Register)
0044h-0047h	RCR	Регистр конфигурации приема пакетов (Receive (Rx) Configuration Register)

Ethernet пакет.

802.3 MAC Frame

Preamble	Start-of-Frame-Delimiter	MAC destination	MAC source	802.1Q header (optional)	Ethertype/Length	Payload (Data and padding)	CRC32	Interframe gap
7 octets of 10101010	1 octet of 10101011	6 octets	6 octets	(4 octets)	2 octets	46–1500 octets	4 octets	12 octets
		64–1522 octets						
72–1530 octets								
84–1542 octets								

Заголовок Ethernet пакета.

bit offset	0–3	4–7	8–15	16–18	19–31
0	Version	Header length	Differentiated Services	Total Length	
32	Identification			Flags	Fragment Offset
64	Time to Live		Protocol	Header Checksum	
96	Source Address				
128	Destination Address				
160	Options (if Header Length > 5)				
160 or 192+	Data				

Пример пользовательского приложения.

```
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>
int main( int argc, char **argv ) {
int i,j;
int       fd = open( "/dev/nic", O_RDONLY );
if ( fd < 0 ) { printf("Error \n"); exit(1); }
int       count = 0;
for(;;)  {
   char buffer[ 2048 ] = {0};
   int    nbytes = read( fd, buffer, sizeof( buffer ) );
   if ( nbytes < 0 ) { exit(1); }
   printf( "\npacket #%d (%d bytes)\n", ++count, nbytes );
   if ( nbytes > 64 ) nbytes = 64;
   for (i = 0; i < nbytes; i+=16)   {
      printf( "\n%04X: ", i );
      for (j = 0; j < 16; j++)      {
      unsigned char              ch = buffer[i+j];
      if ( i+j < nbytes ) printf( "%02X ", ch );
         else             printf( "   " );
    }
   for (j = 0; j < 16; j++){
     char  ch = buffer[i+j];
     if (( ch < 0x20 )||( ch > 0x7E )) ch = '.';
     if ( i+j < nbytes ) printf( "%c", ch );
        else             printf( " " );
    }
   }
   printf( "\n\n" );
}
}
```