```c
#include <linux/syscalls.h> // included for GPIO/BSC macros
#include <asm/io.h> // included for ioread/iowrite
#include <linux/module.h>    // included for all kernel modules
#include <linux/kernel.h>    // included for KERN_INFO
#include <linux/moduleparam.h> // included for parameters
#include <linux/init.h>      // included for __init and __exit macros
#include <linux/delay.h> // included for udelay
#include <linux/cdev.h> // included for character device
#include <linux/device.h> // included for device files in sys
#include <linux/errno.h> // included for default error messages
#include <linux/uaccess.h>

// definice dedikovanych portu
#define BCM2708_PERI_BASE       0x20000000
#define GPIO_BASE               (BCM2708_PERI_BASE + 0x200000)

#define UR_CS (1 << 18)
#define UR_RD (1 << 17)
#define UR_WR (1 << 23)
#define UR_RESET (1 << 24)
#define UR_A0 (1 << 22)
#define UR_A1 (1 << 27)
#define UR_D0 (1 << 5)
#define UR_D1 (1 << 6)
#define UR_D2 (1 << 12)
#define UR_D3 (1 << 13)
#define UR_D4 (1 << 19)
#define UR_D5 (1 << 16)
#define UR_D6 (1 << 26)
#define UR_D7 (1 << 21)
#define UR_INT (1 << 4)
#define UR_CLKOUT (1 << 25)
#define UR_REMOTE (1 << 20)

#define UR_DATAMASK (UR_D0+UR_D1+UR_D2+UR_D3+UR_D4+UR_D5+UR_D6+UR_D7)
#define UR_ADDRMASK (UR_A0+UR_A1)
#define UR_READMASK (UR_DATAMASK+UR_CLKOUT+UR_INT)
#define UR_SETDIRNEG (UR_RD+UR_WR+UR_CS)

// definice specifickych masek -- plati pouze pro uvedene obsazeni portu
// DODELAT generovani masek v zavislosti na obsazenem pinu

#define GPFSEL_AND0 4292874239LU
#define GPFSEL_AND1 3221483583LU
#define GPFSEL_AND2 4278190080LU
#define GPFSEL_ORR0 0
#define GPFSEL_ORR1 18874368
#define GPFSEL_ORR2 2101824
#define GPFSEL_ORW0 294912
#define GPFSEL_ORW1 153354816
#define GPFSEL_ORW2 2363976

// definice vymaskovani FSEL

#define GPIO_AND(dekada,maska)   *(gpio.addr + (dekada)) &= (maska) // nastavi n
emaskovane na cteni
#define GPIO_OR(dekada,maska)   *(gpio.addr + (dekada)) |= (maska) // nastavi za
pis do maskovanych
#define GPIO_WR1  *(gpio.addr + 7)  // slovo k nastaveni na 1
#define GPIO_WR0  *(gpio.addr + 10) // slovo k nastaveni na 0
#define GPIO_READ(maska)  *(gpio.addr + 13) &= (maska) // nactene a vymaskovane
slovo
```



```c
#define GPIO_EVENT(maska) *(gpio.addr + 16) &= (maska) // udalost na vymaskovane
m GPIO
#define GPIO_SETRIS *(gpio.addr + 19) // slovo k nastaveni detekce nabezne hrany
#define GPIO_SETFAL *(gpio.addr + 22) // slovo k nastaveni detekce odbehove hran
y
#define GPIO_SETHI *(gpio.addr + 25) // slovo k nastaveni detekce log. 1
#define GPIO_SETLO *(gpio.addr + 28) // slovo k nastaveni detekce log. 0
#define GPIO_SETARIS *(gpio.addr + 31) // slovo k nastaveni asynchronni detekce
nabezne hrany
#define GPIO_SETAFAL *(gpio.addr + 34) // slovo k nastaveni asynchronni detekce
odbehove hrany
#define GPIO_PULLREQ *(gpio.addr + 37) // pozadavek na zaveseni portu (1 na LO,
2 na HI)
#define GPIO_PULLMASK *(gpio.addr + 38) // slovo k nastaveni zaveseni (smysl zav
eseni predava GPIO_PULLREQ)

// definice zarizeni

#define MYMODULE_NAME               "urpi"
#define MINOR_COUNT             4

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Zbynek Hubinka");
MODULE_DESCRIPTION("Kernelspace driver for Intel 8255");


struct urpi_t {
        dev_t dev_num;
        struct cdev * cdev_ptr;
        struct class * sys_class_ptr;
        struct device * sys_dev_ptr;
        atomic_t open_cnt;
};
static struct urpi_t urpi = {
        .dev_num = MKDEV(0, 0),
        .cdev_ptr = NULL,
        .sys_class_ptr = NULL,
        .sys_dev_ptr = NULL,
        .open_cnt = ATOMIC_INIT(0),
};

static int urpi_open(struct inode* inode_ptr, struct file* file_ptr);
static int urpi_release(struct inode* inode_ptr, struct file* file_ptr);
static ssize_t urpi_read(struct file * file_ptr, char __user * data_ptr, size_t
data_size, loff_t * pos_ptr);
static ssize_t urpi_write(struct file * file_ptr, const char __user * data_ptr,
size_t data_size, loff_t * pos_ptr);

static struct file_operations urpi_file_ops = { .owner = THIS_MODULE, .open = ur
pi_open, .release = urpi_release, .read = urpi_read, .write = urpi_write };

int out_8255(unsigned char port, unsigned char value,int chdir);
unsigned char in_8255(unsigned char port,int chdir);

static int urpi_open(struct inode* inode_ptr, struct file* file_ptr) {
//      if (down_interruptible(&sem)<0) {
//              printk(KERN_ALERT "/dev/urpi%i: the device has been opened by so
me other device, unable to open lock\n",iminor(inode_ptr));
//              return -1;
//      }
        if (atomic_inc_and_test(&urpi.open_cnt)>1) {
                atomic_dec(&urpi.open_cnt);
```

```
                printk(KERN_INFO MYMODULE_NAME ": nelze otevrit");
                return -1;
        }
        if (nonseekable_open(inode_ptr, file_ptr)<0) {
                printk(KERN_INFO MYMODULE_NAME ": nepodporuje lseek");
                return -1;
        }
        printk(KERN_INFO MYMODULE_NAME ": UR-Pi otevreno %u.%u\n",imajor(inode_ptr),i
minor(inode_ptr));
        return 0;
}
static int urpi_release(struct inode* inode_ptr, struct file* file_ptr) {
        atomic_dec(&urpi.open_cnt);
        printk(KERN_INFO MYMODULE_NAME ": UR-Pi uzavreno\n");
        return 0;
}
static ssize_t urpi_read(struct file * file_ptr, char __user * data_ptr, size_t
data_size, loff_t * pos_ptr) {
        unsigned char port=iminor(file_ptr->f_path.dentry->d_inode);
        unsigned char data=in_8255(port,1);
        if (copy_to_user(data_ptr,&data,1)<1) {
                printk(KERN_INFO "Z portu %u nelze cist\n",port);
                return -1;
        }
        printk(KERN_INFO "Z portu %u prislo cislo %u\n",port,data);
        return 0;
}
static ssize_t urpi_write(struct file * file_ptr, const char __user * data_ptr,
size_t data_size, loff_t * pos_ptr) {
        unsigned char port=iminor(file_ptr->f_path.dentry->d_inode);
        unsigned char data=*data_ptr;
        out_8255(port,data,1);
        return data_size;
}

struct bcm2835_peripheral {
    unsigned long addr_p;
    void *map;
    volatile unsigned int *addr;
};

struct bcm2835_peripheral gpio = {GPIO_BASE};
int UR_DATA[8]={UR_D0,UR_D1,UR_D2,UR_D3,UR_D4,UR_D5,UR_D6,UR_D7};
int UR_ADDR[2]={UR_A0,UR_A1};
struct resource * mem;

static int map_peripheral(struct bcm2835_peripheral *p) {
        mem=request_mem_region(GPIO_BASE, 4096, "mygpio");
        p->map=(uint32_t *)ioremap(GPIO_BASE, 4096);
        if (p->map == NULL) { return -1; }
        else { p->addr = (volatile unsigned int *)p->map; return 0; }
}
static void unmap_peripheral(struct bcm2835_peripheral *p) {
    iounmap(p->map);
    release_mem_region(GPIO_BASE,4096);
}
unsigned char fromurpi(unsigned int vstup) {
        int vystup=0; int i;
        for (i=0;i<8;i++) {
                if ((UR_DATA[i]&vstup)>0) vystup+=(1 << i);
        }
        return vystup;
```



```
}
static unsigned int tourpi(unsigned char vstup,int vector[]) {
        int vystup=0; int i;
        for (i=0;i<8;i++) {
                int podil=vstup/(1 << i);
                vystup+=podil%2*vector[i];
        }
        return vystup;
}
unsigned char in_8255(unsigned char port,int chdir) {
        int vystup=0;
        if (chdir==1) {
                GPIO_AND(0,GPFSEL_AND0);
                GPIO_AND(1,GPFSEL_AND1);
                GPIO_AND(2,GPFSEL_AND2);
                GPIO_OR(0,GPFSEL_ORR0);
                GPIO_OR(1,GPFSEL_ORR1);
                GPIO_OR(2,GPFSEL_ORR2);
        }
        udelay(100);
        GPIO_WR1=tourpi(port,UR_ADDR);
        GPIO_WR0=tourpi(3-port,UR_ADDR);
        GPIO_WR0=UR_CS;
        GPIO_WR0=UR_RD;
        udelay(700);
        vystup=fromurpi(GPIO_READ(UR_DATAMASK));
        GPIO_WR1=UR_RD;
        GPIO_WR1=UR_CS;
        return vystup;
}
int out_8255(unsigned char port, unsigned char value,int chdir) {
        if (chdir==1) {
                GPIO_AND(0,GPFSEL_AND0);
                GPIO_AND(1,GPFSEL_AND1);
                GPIO_AND(2,GPFSEL_AND2);
                GPIO_OR(0,GPFSEL_ORW0);
                GPIO_OR(1,GPFSEL_ORW1);
                GPIO_OR(2,GPFSEL_ORW2);
        }
        GPIO_WR1=tourpi(value,UR_DATA)+tourpi(port,UR_ADDR);
        GPIO_WR0=tourpi(255-value,UR_DATA)+tourpi(3-port,UR_ADDR);
        printk(KERN_INFO "adresa: %u",tourpi(3,UR_ADDR));
        GPIO_WR0=UR_CS; // nejdriv CS, potom WR
        GPIO_WR0=UR_WR;
        udelay(100);
        GPIO_WR1=UR_WR;
        GPIO_WR1=UR_CS;
        return 0;
}
int reset_8255(int direction) {
        GPIO_AND(0,GPFSEL_AND0);
        GPIO_AND(1,GPFSEL_AND1);
        GPIO_AND(2,GPFSEL_AND2);
        if (direction==1) {
                GPIO_OR(0,GPFSEL_ORW0);
                GPIO_OR(1,GPFSEL_ORW1);
                GPIO_OR(2,GPFSEL_ORW2);
        } else {
                GPIO_OR(0,GPFSEL_ORR0);
                GPIO_OR(1,GPFSEL_ORR1);
                GPIO_OR(2,GPFSEL_ORR2);
        }
```

```
        GPIO_PULLREQ=2;
        udelay(100);
        GPIO_PULLMASK=UR_INT;
        GPIO_PULLREQ=2;
        udelay(100);
        GPIO_PULLMASK=UR_CLKOUT;
        GPIO_SETRIS=UR_CLKOUT;
        GPIO_WR1=UR_SETDIRNEG+UR_RESET;
        GPIO_WR0=UR_DATAMASK+UR_ADDRMASK+UR_RESET;
        udelay(100);
        GPIO_WR0=UR_DATAMASK+UR_ADDRMASK+UR_RESET;
        return 0;
}









static int __init urpi_init(void) {
        int i;
        printk(KERN_INFO "UR Pi driver v. 0.1\n");
        if (map_peripheral(&gpio) == -1) {
                printk("Failed to map the physical GPIO registers into the virtual memory space.\n");
                return -1;
        }
        if (alloc_chrdev_region(&urpi.dev_num, MINOR(urpi.dev_num), MINOR_COUNT,
 MYMODULE_NAME)<0) {
                printk(KERN_INFO "nepodarilo se vytvorit node v dev\n");
                return -1;
        }
        urpi.cdev_ptr=cdev_alloc();
        if (urpi.cdev_ptr == NULL) {
                printk(KERN_INFO "nedostatek pameti pro dalsi zarizeni\n");
                unregister_chrdev_region(urpi.dev_num, MINOR_COUNT);
                return -1;
        }
        urpi.cdev_ptr->owner=THIS_MODULE;
        urpi.cdev_ptr->ops = &urpi_file_ops;
        if (cdev_add(urpi.cdev_ptr,urpi.dev_num,MINOR_COUNT)<0) {
                printk(KERN_INFO "nepodarilo se priradit minor/major par\n");
                unregister_chrdev_region(urpi.dev_num, MINOR_COUNT);
                return -1;
        }
        printk(KERN_INFO "urpi: modul inicializovan (major=%u)\n\nDruha faze inicializace...",MAJOR(u
rpi.dev_num));
        urpi.sys_class_ptr = class_create(THIS_MODULE, MYMODULE_NAME);
        if (IS_ERR(urpi.sys_class_ptr)) {
                printk(KERN_INFO "nepodarilo se vytvorit tridu\n");
                cdev_del(urpi.cdev_ptr);
                unregister_chrdev_region(urpi.dev_num, MINOR_COUNT);
                return -1;
        }
        for(i=0;i<4;i++) {
                urpi.sys_dev_ptr = device_create(urpi.sys_class_ptr, NULL, MKDEV
(MAJOR(urpi.dev_num),i), NULL, "urpi%d",i);
                if (IS_ERR(urpi.sys_dev_ptr)) {
                        printk(KERN_INFO "nepodarilo se vytvorit soubor zarizeni\n");
                        class_destroy(urpi.sys_class_ptr);
```



```
                        cdev_del(urpi.cdev_ptr);
                        unregister_chrdev_region(urpi.dev_num, MINOR_COUNT);
                        return -1;
                }
        }
        reset_8255(1);
        printk(KERN_INFO "Initialisation successfull.\n");
        return 0;     // Non-zero return means that the module couldn't be loaded
.
}

static void __exit urpi_cleanup(void) {
        int i;
    printk(KERN_INFO "Cleaning up module... ");
    for (i=0;i<4;i++) { device_destroy(urpi.sys_class_ptr, MKDEV(MAJOR(urpi.dev_
num),i)); }
    class_destroy(urpi.sys_class_ptr);
        cdev_del(urpi.cdev_ptr);
        unregister_chrdev_region(urpi.dev_num, MINOR_COUNT);
    unmap_peripheral(&gpio);
        printk(KERN_INFO "Module released.\n");
}

module_init(urpi_init);
module_exit(urpi_cleanup);
```