Sam Salin

ECE 373

HW 4

**Kernel Module**

/\*

\* Sam Salin

\* 4/11/16

\* ECE 373

\*

\* HW 6: blink lights on packet reciept

\*/

#include <linux/module.h>

#include <linux/types.h>

#include <linux/kdev\_t.h>

#include <linux/fs.h>

#include <linux/cdev.h>

#include <linux/usb.h>

#include <linux/slab.h>

#include <linux/pci.h>

#include <linux/time.h>

#include <linux/dma-mapping.h>

#define GET\_DESC(R, i) (&(((struct rx\_desc \*)((R).desc))[i]))//shamelessly copied from lxr e1000.h

#define NUMDESC 16 //number of descriptors in ring line 459, modified for struct type

#define DEVCNT 1

#define DEVNAME "hw\_4\_driver"

#define LEDCTL 0x00e00

#define RECCTL 0x00100

#define RDBAL 0x02800 //receive base address low

#define RDBAH 0x02804 //receive base address high

#define RDH 0x02810

#define RDT 0x02818

#define CTLEXT 0x00018

#define IMS 0x000D0

#define IMCR 0x000D8 //IMC

static struct mydev\_dev {

struct cdev cdev;

/\* more stuff will go in here later... \*/

} mydev;

static dev\_t mydev\_node;

static char \*pe\_driver\_name = "packet\_blinker";

static DEFINE\_PCI\_DEVICE\_TABLE(pci\_tbl) = {

{PCI\_DEVICE(0x8086, 0x150c) }, /\*numbers from lspci, lspci -n. could use 0x1501 instead of 0x150c\*/

{}, /\*needed for some reason\*/

};

/\*PCI STUFF\*/

struct pes{

struct pci\_dev \*pdev;

void \*hw\_addr; //add offsets to get places

struct work\_struct queue;

};

//static int b\_r = 2;

//module\_param(b\_r, int, S\_IRUSR | S\_IWUSR);

static struct pes \*usr;

/\*Descriptors\*/

struct rx\_desc{

\_\_le64 buffer\_addr;

\_\_le16 length;

\_\_le16 csum;

u8 status;

u8 errors;

\_\_le16 special;

}\*rsd;

// descriptor ring

//could be done with struct rx\_desc ring[16]?

struct ring {//pretty heavily based on E1000.h

void \*desc;

dma\_addr\_t dma;

unsigned int size;

unsigned int count;

struct buffer \*buf;

void \*head;

void \*tail;

}\*rx\_ring;

//BUFFER

struct buffer {

void \*buf;

dma\_addr\_t dma;

};

/\*

buffer\_info->buf = databuf;

buffer\_info->length = len;

buffer\_info->dma = dma\_map\_single(mydev, databuf, len, DMA\_FROM\_DEVICE)

rx\_desc =

\*/

/\*workque hdlr\*/

static void wrkq\_hdlr(struct work\_struct \*work)

{

struct pes \* adptr = container\_of(work, struct pes, queue);

u32 tmp, tmp2;

printk(KERN\_INFO "ENTERED WRKQ HDLR\n");

mdelay (500); //sleep .5 sec

//turn off LEDs

tmp = readl(adptr->hw\_addr + LEDCTL); //read it

tmp = tmp & 0x000000; //clear

tmp = tmp | 0x0f0f0f ;//turn lights off

writel(tmp, adptr->hw\_addr + LEDCTL); //store it

// do{

// }while(--count > 0);

//bump tail

tmp = readl(rx\_ring->tail);

tmp2 = readl(rx\_ring->head);

printk(KERN\_INFO"TAIL = %d, HEAD = %d\n",tmp,tmp2);

if (tmp == 15)//last entry

tmp = 0;

else//not last entry

++tmp;

writel(tmp, rx\_ring->tail);

}

/\* interrupt handler \*/

static irqreturn\_t irq\_hdlr (int irq, void \*data)

{

unsigned int tmp;

u32 led\_reg;

printk(KERN\_INFO "ENTERED IRQ HDLR\n");

tmp = readl(usr->hw\_addr + 0x000C0); //irq cause reg

if(tmp){/\*our IRQ\*/

writel(0x0, usr->hw\_addr+ IMCR);//clear interrupts

/\* turn on green LEDs \*/

led\_reg = readl(usr->hw\_addr + LEDCTL);

led\_reg = led\_reg & 0x000000;

led\_reg = led\_reg | 0x0E0E00;

writel(led\_reg, usr->hw\_addr + LEDCTL);

//workqueue thread call

schedule\_work(&(usr->queue));

return IRQ\_HANDLED;

}

/\*not our IRQ, probably shouldnt happen\*/

printk(KERN\_ERR"NOT OUR IRQ\n");

tmp = 0x100000;

writel(tmp, usr->hw\_addr + IMS); //reenable IRQ

return IRQ\_NONE;

}

static int pci\_probe(struct pci\_dev \*pdev, const struct pci\_device\_id \*ent) {

struct pes \* pe;

u32 ioremap\_len;

u32 led\_reg;

void \* addr;

unsigned int tmp;

u64 tmp1;

int err;

err = pci\_enable\_device\_mem(pdev);

if(err)

return err; /\*nonzero implies error\*/

/\*hi/lo DMA prep\*/

err = dma\_set\_mask(&pdev->dev, DMA\_BIT\_MASK(64));

if (err) {

dev\_err(&pdev->dev, "DMA configuration failed: 0x%x\n", err);

goto err\_dma;

}

/\*connect pci\*/

err = pci\_request\_selected\_regions(pdev, pci\_select\_bars(pdev, IORESOURCE\_MEM), pe\_driver\_name);

if (err) {

dev\_info(&pdev->dev, "pci\_request\_selected\_regions failed %d\n", err);

goto err\_pci\_reg;

}

pci\_set\_master(pdev);

pe = kzalloc(sizeof(\*pe), GFP\_KERNEL);

if (!pe) {

err = -ENOMEM;

goto err\_pe\_alloc;

}

pe->pdev = pdev;

pci\_set\_drvdata(pdev, pe);

/\*map dev mem\*/

ioremap\_len = min\_t(int, pci\_resource\_len(pdev, 0), 131072);//2^17

pe->hw\_addr = ioremap(pci\_resource\_start(pdev, 0), ioremap\_len);

if (!pe->hw\_addr) {

err = -EIO;

dev\_info(&pdev->dev, "ioremap(0x%04x, 0x%04x) failed: 0x%x\n",

(unsigned int)pci\_resource\_start(pdev, 0),

(unsigned int)pci\_resource\_len(pdev, 0), err);

goto err\_ioremap;

}

//initailze device

addr = pe->hw\_addr; //fewer derefrences

writel(0xFFFFFFFF, addr + IMCR); //disable IRQ

writel(0x84000000, addr); //reset

udelay(1); //manual said to wait 1 us after resetting before reads or writes p. 199

writel(0xFFFFFFF, addr + IMCR); //disable IRQ again (4.6.1 datasheet)

//clear LEDs from reset

led\_reg = readl(addr + LEDCTL);

led\_reg = led\_reg & 0x0f0f0f;//set all LEDs to OFF, set invert bits

writel (led\_reg,addr + LEDCTL);

/\* PHY \*/

writel(0x400000, addr + 0x05B00); //set bit 22 GCR

writel(0x1, addr + 0x05B64); // set bit 0 GCR2

writel(0x1831AF08, addr + 0x00020); // write to MDIC

//promiscuous mode

tmp = readl(addr + RECCTL);

tmp = tmp | 0x800C; // set bit 3,4 and 15

writel(tmp, addr + RECCTL);

//set up ring

rx\_ring = kmalloc (sizeof (struct ring), GFP\_KERNEL);

if(!rx\_ring)

printk(KERN\_INFO "RING FAIL");

//set up descriptors

rx\_ring->count = 16;

//round 4k (e1000e netdev 2383)

rx\_ring->size = rx\_ring->count \* sizeof(struct rx\_desc);

rx\_ring->buf = vmalloc(rx\_ring->size);

rx\_ring->size = ALIGN(rx\_ring->size, 4096);

rx\_ring->desc = dma\_alloc\_coherent(&pdev->dev, rx\_ring->size, &rx\_ring->dma, GFP\_KERNEL);

if (!rx\_ring->desc) {

err = -ENOMEM;

return err;

}

//pin buf

for (err = 0; err < NUMDESC; ++err){//using err as a loop counter

rx\_ring->buf[err].buf = kmalloc (2048, GFP\_KERNEL);

rx\_ring->buf[err].dma = dma\_map\_single(&pdev->dev, rx\_ring->buf[err].buf, 2048, DMA\_FROM\_DEVICE);

rsd = GET\_DESC(\*rx\_ring, err);

rsd->buffer\_addr = cpu\_to\_le64(rx\_ring->buf[err].dma);

//i feel like there should be some kind of error checking going on

}

//head/tail

tmp1 = rx\_ring->dma;

tmp1 = tmp1 & 0x00000000FFFFFFFF; //clear upper 32

writel(tmp1, addr + RDBAL); //base low

tmp1 = tmp1 >> 32;

writel(tmp1, addr + RDBAH); //base high

writel(0x800, addr + 0x02808); //set buffer size RDLEN

writel(0x1, addr + RDH); //head desc 0

writel(0x0, addr + RDT); //tail desc 0

rx\_ring->head = addr + RDH; //map head

rx\_ring->tail = addr + RDT; //map tail

//writel(0xF,addr + RDT);//set tail to desc 1-head //0x3f

//set up workqueue

INIT\_WORK(&pe->queue, wrkq\_hdlr);

//set up IRQ

err = request\_irq(pdev->irq, irq\_hdlr, IRQF\_SHARED, "hw6\_intr", addr);//last feild 0 device not shared

if(err)

printk(KERN\_ERR "ERROR %d\n", err);

//set up receive registers

//tmp = readl(hw\_base\_addr + RECCTL);

//tell to autonegotiate speed

//tell to accept bad packets?

/\* setup link in CTRL \*/

tmp = readl(addr);

tmp = tmp | 0x40; /\* set bit 6 \*/

writel(tmp, addr);

//set receive enable

tmp = readl(addr + RECCTL);

tmp = tmp | 0x2; // set bit 1

writel(tmp, addr + RECCTL);

//sets the mask for receive queue 0 interrupt in IMS

tmp = readl(addr + IMS);

tmp = tmp | 0x10; //set RXDMT0 to 1

writel(tmp, addr + IMS);

tmp = readl(addr + CTLEXT);

tmp = tmp | 0x18000000; //tell device driver has loaded

writel(tmp, addr + CTLEXT);

writel(0x100000, addr + IMS); //set bit 20

usr = pe;

return 0;

err\_ioremap:

kfree(pe);

err\_pe\_alloc:

pci\_release\_selected\_regions(pdev,pci\_select\_bars(pdev, IORESOURCE\_MEM));

err\_pci\_reg:

err\_dma:

pci\_disable\_device(pdev);

return err;

}

static void pci\_remove(struct pci\_dev \*pdev) {

struct pes \*pe = pci\_get\_drvdata(pdev);

//turn off interrupts

free\_irq(pdev->irq,usr->hw\_addr);

//cancel workqueue

//free the ring

kfree(rx\_ring);

/\* unmap device from memory \*/

iounmap(pe->hw\_addr);

/\* free any allocated memory \*/

kfree(pe);

pci\_release\_selected\_regions(pdev, pci\_select\_bars(pdev, IORESOURCE\_MEM));

pci\_disable\_device(pdev);

}

static struct pci\_driver pe\_driver = {

.name = "hw\_3\_pci",

.id\_table = pci\_tbl,

.probe = pci\_probe,

.remove = pci\_remove,

};

static int example5\_open(struct inode \*inode, struct file \*file) {

printk(KERN\_INFO "successfully opened!\n");

return 0;

}

static ssize\_t example5\_read(struct file \*file, u16 \_\_user \*buf, size\_t len, loff\_t \*offset) {

/\* Get a local kernel buffer set aside \*/

u16 tmp;

u16 ret=0;

if (\*offset >= sizeof(int))

return 0;

/\* Make sure our user wasn't bad... \*/

if (!buf) {

ret = -EINVAL;

goto out;

}

//pack head/tail location

ret = readl(rx\_ring->head);

ret = ret << 8;

tmp = readl(rx\_ring-> tail);

ret = ret | tmp;

if (copy\_to\_user(buf, &ret, sizeof(u16))) {

ret = -EFAULT;

goto out;

}

ret = sizeof(int);

\*offset += len;

out:

return ret;

}

static ssize\_t example5\_write(struct file \*file, const int \_\_user \*buf, size\_t len, loff\_t \*offset) {

return 0;

}

/\* File operations for our device \*/

static struct file\_operations mydev\_fops = {

.owner = THIS\_MODULE,

.open = example5\_open,

.read = example5\_read,

.write = example5\_write,

};

static int \_\_init example5\_init(void) {

printk(KERN\_INFO "hw\_3\_driver module loading...");

if (alloc\_chrdev\_region(&mydev\_node, 0, DEVCNT, DEVNAME)) {

printk(KERN\_ERR "alloc\_chrdev\_region() failed!\n");

return -1;

}

printk(KERN\_INFO "Allocated %d devices at major: %d\n", DEVCNT,

MAJOR(mydev\_node));

/\* Initialize the character device and add it to the kernel \*/

cdev\_init(&mydev.cdev, &mydev\_fops);

mydev.cdev.owner = THIS\_MODULE;

if (cdev\_add(&mydev.cdev, mydev\_node, DEVCNT)) {

printk(KERN\_ERR "cdev\_add() failed!\n");

/\* clean up chrdev allocation \*/

unregister\_chrdev\_region(mydev\_node, DEVCNT);

return -1;

}

pci\_register\_driver(&pe\_driver); /\*for PCI\*/

return 0;

}

static void \_\_exit example5\_exit(void) {

//kill the workqueue

cancel\_work\_sync(&usr->queue);

/\* destroy the cdev \*/

cdev\_del(&mydev.cdev);

/\* clean up the devices \*/

pci\_unregister\_driver(&pe\_driver);

unregister\_chrdev\_region(mydev\_node, DEVCNT);

printk(KERN\_INFO "hw\_3\_driver module unloaded!\n");

}

MODULE\_AUTHOR("Sam Salin");

MODULE\_LICENSE("GPL");

MODULE\_VERSION("0.2");

module\_init(example5\_init);

module\_exit(example5\_exit);

**Makefile**

obj-m = hw\_6.o

KVERSION = $(shell uname -r)

all:

make -C /lib/modules/$(KVERSION)/build M=$(shell pwd) modules

clean:

make -C /lib/modules/$(KVERSION)/build M=$(shell pwd) clean

**User Program**

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

int main()

{

int fd;

ssize\_t num\_read, num\_written;

char my\_read\_str[10];

unsigned short tmp,tmp1; /\*to read from driver\*/

unsigned int clr = 0xffffff00; //clr low 4 bts (led0 ctl reg)

unsigned int on = 0x0000004e; //00000000000000000000000000001110

unsigned int off = 0x0000000f; //00000000000000000000000000001111

fd = open("/dev/hw\_6", O\_RDWR);

num\_read = read(fd, &tmp, sizeof(int));//read

tmp1 = tmp & 0xFF00;

tmp1 = tmp1 >> 8; //head

tmp1 = tmp1 & 0xff;

tmp = (tmp & 0xff)-1; //clear higher bits

printf("HEAD = %d, TAIL = %d\n",tmp1 ,tmp);

close(fd);

return 0;

}

**Typescript**

root@ece373:/home/linux/T# ./a.out

HEAD = 7, TAIL = 3

root@ece373:/home/linux/T# ./a.out

HEAD = 8, TAIL = 4

root@ece373:/home/linux/T# ./a.out

HEAD = 9, TAIL = 5

root@ece373:/home/linux/T# ./a.out

HEAD = 10, TAIL = 7

root@ece373:/home/linux/T# ./a.out