#include <linux/kernel.h>

#include <linux/module.h>

#include <linux/pci.h>

#include <linux/init.h>

#include <linux/fs.h>

#include <asm/uaccess.h>

MODULE\_LICENSE("Dual BSD/GPL");

MODULE\_DESCRIPTION("PCIHello");

MODULE\_AUTHOR("HUE");

typedef struct VECTOR

{

int option;

unsigned data;

}Vect;

//-- Hardware Handles

static void \*hexport;

static void \*inport;

static void \*ledverde;

static void \*ledvermelho;

static void \*button0;

static void \*ledextra;

static void \*cool;

//-- Char Driver Interface

static int access\_count = 0;

static int MAJOR\_NUMBER = 91;

static int char\_device\_open ( struct inode \* , struct file \*);

static int char\_device\_release ( struct inode \* , struct file \*);

static ssize\_t char\_device\_read ( struct file \* , char \*, size\_t , loff\_t \*);

static ssize\_t char\_device\_write ( struct file \* , const char \*, size\_t , loff\_t \*);

char dados[5];

static struct file\_operations file\_opts = {

.read = char\_device\_read,

.open = char\_device\_open,

.write = char\_device\_write,

.release = char\_device\_release

};

static int char\_device\_open(struct inode \*inodep, struct file \*filep) {

access\_count++;

printk(KERN\_ALERT "altera\_driver: opened %d time(s)\n", access\_count);

return 0;

}

static int char\_device\_release(struct inode \*inodep, struct file \*filep) {

printk(KERN\_ALERT "altera\_driver: device closed.\n");

return 0;

}

static ssize\_t char\_device\_read(struct file \*filep, char \*buf, size\_t len, loff\_t \*off) {

/////////////////////////////////////

size\_t count = len;

short switches;

while (len > 0) {

switches = ioread16(inport);

put\_user(switches & 0xFF, buf++);

put\_user((switches >> 8) & 0xFF, buf++);

len -= 2;

}

return count;

}

static ssize\_t char\_device\_write(struct file \*filep, const char \*buf, size\_t len, loff\_t \*off)

{

Vect\* ptr = (Vect\*) buf;

size\_t count = len;

unsigned k = ((unsigned)ptr->data);

switch(ptr->option)

{

case 1:

iowrite32(k,hexport);

break;

case 2:

iowrite32(k,ledverde);

break;

case 3:

iowrite32(k,ledvermelho);

break;

case 4:

iowrite32(k,ledextra);

break;

case 5:

iowrite32(k,cool);

}

return count;

}

//-- PCI Device Interface

static struct pci\_device\_id pci\_ids[] = {

{ PCI\_DEVICE(0x1172, 0x0004), },

{ 0, }

};

MODULE\_DEVICE\_TABLE(pci, pci\_ids);

static int pci\_probe(struct pci\_dev \*dev, const struct pci\_device\_id \*id);

static void pci\_remove(struct pci\_dev \*dev);

static struct pci\_driver pci\_driver = {

.name = "alterahello",

.id\_table = pci\_ids,

.probe = pci\_probe,

.remove = pci\_remove,

};

static unsigned char pci\_get\_revision(struct pci\_dev \*dev) {

u8 revision;

pci\_read\_config\_byte(dev, PCI\_REVISION\_ID, &revision);

return revision;

}

static int pci\_probe(struct pci\_dev \*dev, const struct pci\_device\_id \*id) {

int vendor;

int retval;

unsigned long resource;

retval = pci\_enable\_device(dev);

if (pci\_get\_revision(dev) != 0x01) {

printk(KERN\_ALERT "altera\_driver: cannot find pci device\n");

return -ENODEV;

}

pci\_read\_config\_dword(dev, 0, &vendor);

printk(KERN\_ALERT "altera\_driver: Found Vendor id: %x\n", vendor);

resource = pci\_resource\_start(dev, 0);

printk(KERN\_ALERT "altera\_driver: Resource start at bar 0: %lx\n", resource);

hexport = ioremap\_nocache(resource + 0XC000, 0x20);//Mapeando memória

inport = ioremap\_nocache(resource + 0XC020, 0x20);//Mapeando memória

ledverde = ioremap\_nocache(resource + 0XC040, 0x20);//Mapeando memória

ledvermelho = ioremap\_nocache(resource + 0XC060, 0x20);//Mapeando memória

button0 = ioremap\_nocache(resource + 0XC080, 0x20);//Mapeando memória

ledextra = ioremap\_nocache(resource + 0XC100, 0x20);

cool = ioremap\_nocache(resource + 0XC120, 0x20);

return 0;

}

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

iounmap(hexport);

iounmap(inport);

iounmap(ledverde);

iounmap(ledvermelho);

iounmap(button0);

iounmap(ledextra);

iounmap(cool);

}

//-- Global module registration

static int \_\_init altera\_driver\_init(void) {

int t = register\_chrdev(MAJOR\_NUMBER, "de2i150\_altera", &file\_opts);

t = t | pci\_register\_driver(&pci\_driver);

if(t<0)

printk(KERN\_ALERT "altera\_driver: error: cannot register char or pci.\n");

else

printk(KERN\_ALERT "altera\_driver: char+pci drivers registered.\n");

return t;

}

static void \_\_exit altera\_driver\_exit(void) {

printk(KERN\_ALERT "Goodbye from de2i150\_altera.\n");

unregister\_chrdev(MAJOR\_NUMBER, "de2i150\_altera");

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

}

module\_init(altera\_driver\_init);

module\_exit(altera\_driver\_exit);