#include <linux/init.h>

#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/timer.h>

#include <asm/io.h>

#include <linux/proc\_fs.h> /\* Necessary because we use the proc fs \*/

#include <linux/seq\_file.h>

#include <linux/uaccess.h> /\* for copy\_from\_user \*/

/\* #define GPIO\_BASE 0x20200000 \*/

#define GPIO\_BASE 0x3F200000

#define GPIO\_BLOCK\_SIZE 4096

volatile unsigned int \* gpio;

// GPIO setup macros. Always use INP\_GPIO(x) before using OUT\_GPIO(x) or SET\_GPIO\_ALT(x,y)

#define INP\_GPIO(g) \*(gpio+((g)/10)) &= ~(7<<(((g)%10)\*3))

#define OUT\_GPIO(g) \*(gpio+((g)/10)) |= (1<<(((g)%10)\*3))

#define SET\_GPIO\_ALT(g,a) \*(gpio+(((g)/10))) |= (((a)<=3?(a)+4:(a)==4?3:2)<<(((g)%10)\*3))

#define GPIO\_SET \*(gpio+7) // sets bits which are 1 ignores bits which are 0

#define GPIO\_CLR \*(gpio+10) // clears bits which are 1 ignores bits which are 0

#define GPIO\_SETB(g) GPIO\_SET |= 1 <<(g)

#define GPIO\_CLRB(g) GPIO\_CLR |= 1 <<(g)

#define GPIO\_READ(g) (\*(gpio + 13) &= (1<<(g)))

#define LED 21 // Maps to Pin 40 on the RasPi connector

#define PROCFS\_NAME "led\_blink" // Name of the proc\_fs file (/proc/led\_blink)

#define PROCFS\_MAX\_SIZE 32 // Maximum size of the message that can be sent to the proc fs.

struct timer\_list led\_timer;

int enable = 1, status = 0;

static char msg[PROCFS\_MAX\_SIZE];

/\* main function \*/

static void blink\_timer(unsigned long ptr)

{

printk("jiffies + HZ = %ld %d\n", jiffies, HZ);

if (enable && status == 0) {

GPIO\_SETB(LED);

status = 1;

} else {

GPIO\_CLRB(LED);

status = 0;

}

led\_timer.expires = jiffies + HZ;

add\_timer(&led\_timer);

}

static int blink\_proc\_show(struct seq\_file \*m, void \*v)

{

seq\_printf(m, "%d,%d\n", enable, status);

return 0;

}

static int blink\_proc\_open(struct inode \*inode, struct file \*file)

{

return single\_open(file, blink\_proc\_show, NULL);

}

static int blink\_proc\_write(struct file \*filp,const char \*buf,size\_t count,loff\_t \*offp)

{

if (copy\_from\_user(msg,buf,(count<PROCFS\_MAX\_SIZE)?count:PROCFS\_MAX\_SIZE)) {

printk("copy\_from\_user error");

return count;

}

enable = (msg[0] != '0');

printk(KERN\_INFO "proc\_write: enable=%d", enable);

return count;

}

static const struct file\_operations blink\_proc\_fops = {

.owner = THIS\_MODULE,

.open = blink\_proc\_open,

.read = seq\_read,

.write = blink\_proc\_write,

.llseek = seq\_lseek,

.release = single\_release,

};

static int blink\_init(void)

{

printk("Hello,LED world!\n");

gpio = ioremap(GPIO\_BASE,GPIO\_BLOCK\_SIZE);

INP\_GPIO(LED);

OUT\_GPIO(LED);

init\_timer(&led\_timer);

led\_timer.function = blink\_timer;

led\_timer.data = (unsigned long) status;

led\_timer.expires = jiffies + HZ;

add\_timer(&led\_timer);

proc\_create(PROCFS\_NAME, 0666, NULL, &blink\_proc\_fops);

return 0;

}

static void blink\_exit(void)

{

del\_timer(&led\_timer);

GPIO\_CLRB(LED);

iounmap(gpio);

remove\_proc\_entry(PROCFS\_NAME, NULL);

printk("Good bye!\n");

}

module\_init(blink\_init);

module\_exit(blink\_exit);

MODULE\_LICENSE("GPL");

MODULE\_AUTHOR("WWW");

MODULE\_DESCRIPTION("LED Blinker Driver");

MODULE\_VERSION("0.1");