```c
// hex_cl.c
// - include ioctl() : NOFILL, BLINK

#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/interrupt.h>
#include <asm/io.h>

#include <linux/fs.h>
#include <linux/uaccess.h>
#include <linux/types.h>
#include <linux/ioport.h>
#include <linux/cdev.h>
#include <linux/device.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("SangKyun Yun");
MODULE_DESCRIPTION("Seven Segment LEDs");

void init_add_timer(void);
void remove_timer(void);
void hex_timer_function(unsigned long ptr);

#define base_lwFPGA 0xFF200000
#define len_lwFPGA  0x200000

#define addr_LED    0
#define addr_HEX0   0x20
#define addr_HEX1   0x30
#define addr_SW     0x40
#define addr_KEY    0x50

static void  *mem_base;
static void  *hex0_addr;    // HEX3-HEX0
static void  *hex1_addr;    // HEX5-HEX4
static unsigned int data = -1;

static unsigned int mode = 0;
#define NOFILL  4   // bit 2
#define BLINK   8   // bit 3

unsigned int hex0, hex1;    // HEX LED output data

int hex_conversion[16] = {
    0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07,
    0x7F, 0x67, 0x77, 0x7C, 0x39, 0x5E, 0x79, 0x71,
};

static ssize_t hex_write (struct file *file, const char __user *buf, size_t count, loff_t *f_pos){
    unsigned int hex_data = 0;
    unsigned int nofill = 0;

    get_user(hex_data, (unsigned int *)buf);
//  copy_from_user(&hex_data, buf, count);
    hex_data = hex_data & 0xFFFFFF;     // 24-bit mask
    data = hex_data;    // save for read

    if (mode & NOFILL) nofill = 1;

    hex1 = 0;
    hex0 = hex_conversion[hex_data &0xf];      // digit 0

    do {    // use do-while to use break
        hex_data >>= 4;
        if (nofill && hex_data==0) break;
        hex0 |= hex_conversion[hex_data &0xf]<<8;  // digit 1

```

```
        hex_data >>= 4;
        if (nofill && hex_data==0) break;
        hex0 |= hex_conversion[hex_data &0xf]<<16; // digit 2

        hex_data >>= 4;
        if (nofill && hex_data==0) break;
        hex0 |= hex_conversion[hex_data &0xf]<<24; // digit 3

        hex_data >>= 4;
        if (nofill && hex_data==0) break;
        hex1 = hex_conversion[hex_data &0xf];      // digit 4

        hex_data >>= 4;
        if (nofill && hex_data==0) break;
        hex1 |= hex_conversion[hex_data &0xf]<<8;  // digit 5
    } while (0);

    iowrite32(hex0, hex0_addr);
    iowrite32(hex1, hex1_addr);

    return 4;
}

static ssize_t hex_read (struct file *file, char __user *buf, size_t count, loff_t *f_pos){
    put_user(data, (unsigned int *)buf);
    return 4;
}

static int hex_open(struct inode *minode, struct file *mfile)
{
    return 0;
}

static int hex_release(struct inode *minode, struct file *mfile)
{
//  if (mode & BLINK)
//      remove_timer();
    return 0;
}

static long hex_ioctl(struct file *file, unsigned int cmd, unsigned long arg)
{
    unsigned int newcmd;

    newcmd = cmd;
    if ( (mode & BLINK) && !(newcmd & BLINK))
        remove_timer();
    else if ( !(mode & BLINK) && (newcmd & BLINK))
        init_add_timer();
    mode = newcmd;

    return 0;
}

static struct file_operations hex_fops = {
    .read       =   hex_read,
    .write      =   hex_write,
    .open       =   hex_open,
    .release    =   hex_release,
    .unlocked_ioctl =   hex_ioctl,
};

static struct cdev hex_cdev;
static struct class *cl;
static dev_t dev_no;

#define DEVICE_NAME "hex"

```

```
137 static int __init hex_init(void)
138 {
139     // allocate char device
140     if (alloc_chrdev_region(&dev_no,0, 1, DEVICE_NAME) < 0) {
141         printk(KERN_ERR "alloc_chrdev_region() error\n");
142         return -1;
143     }
144     // init cdev
145     cdev_init(&hex_cdev, &hex_fops);
146 
147     // add cdev
148     if (cdev_add(&hex_cdev, dev_no, 1) < 0) {
149         printk(KERN_ERR "cdev_add() error\n");
150         goto unreg_chrdev;
151     }
152     // create class struct
153     cl = class_create (THIS_MODULE, DEVICE_NAME);
154     if (cl == NULL) {
155         printk(KERN_ALERT "class_create() error\n");
156         goto unreg_chrdev;
157     }
158     // create device
159     if (device_create(cl, NULL, dev_no, NULL, DEVICE_NAME) == NULL) {
160         printk(KERN_ALERT "device_create error\n");
161         goto unreg_class;
162     }
163 
164     mem_base = ioremap_nocache(base_lwFPGA, len_lwFPGA);
165     if(mem_base == NULL) {
166         printk(KERN_ERR "ioremap_nocache() error\n");
167         goto un_device;
168     }
169     printk("Device: %s MAJOR: %d %x\n", DEVICE_NAME, MAJOR(dev_no), dev_no);
170     hex0_addr = mem_base + addr_HEX0;
171     hex1_addr = mem_base + addr_HEX1;
172 
173     return 0;
174 
175 // error
176 un_device:
177     device_destroy(cl, dev_no);
178 unreg_class:
179     class_destroy(cl);
180 unreg_chrdev:
181     unregister_chrdev_region(dev_no,1);
182     return -1;
183 }
184 
185 static void __exit hex_exit(void){
186     iowrite32(0, hex0_addr);    // turn off all HEX LEDs
187     iowrite32(0, hex1_addr);
188     if (mode & BLINK)           // if blink mode, remove timer
189         remove_timer();
190 
191     iounmap(mem_base);
192     device_destroy(cl, dev_no);
193     class_destroy(cl);
194     unregister_chrdev_region(dev_no,1);
195     printk(" %s unregisterd.\n", DEVICE_NAME);
196 }
197 
198 module_init(hex_init);
199 module_exit(hex_exit);
200 
201 
202 static int turnoff = 0;     // for blinking mode
203 static struct timer_list hex_timer;
204 
```

```
205 void init_add_timer(void)
206 {
207     init_timer(&hex_timer);
208 
209     hex_timer.function = hex_timer_function;
210     hex_timer.expires = jiffies + HZ;  // after 1 sec
211     hex_timer.data = 0;
212 
213     add_timer(&hex_timer);
214 }
215 
216 void remove_timer(void)
217 {
218     del_timer(&hex_timer);
219 }
220 
221 void hex_timer_function(unsigned long ptr)
222 {
223     if ( !(mode & BLINK) ) return;
224     turnoff = !turnoff;
225     if (turnoff) {
226         iowrite32(0, hex0_addr);
227         iowrite32(0, hex1_addr);
228     } else {
229         iowrite32(hex0, hex0_addr);
230         iowrite32(hex1, hex1_addr);
231     }
232 
233     init_add_timer();
234 }
235 
```