

# ECE373 Linux Drivers

Aaron Chan  
Assignment #3

Aaron Chan  
ECE373 (Spring 2017)  
Assignment #3

## **PART 1: Digging for Details**

### **Computer Box: AIMB-212 from Advantech**

#### **[1] Atomic Motherboards (go boom)**

##### **1a. What is the audio device?**

Realtek ALC888

##### **1b. What device is the GPIO connected to?**

8-bit GPIO is connected to Super IO Winbond W83627DHG-P  
(User Manual ed.1 pg7)

##### **1c. How many network (LAN) devices are on the motherboard and what are they?**

2 Ethernet LAN controllers: (User Manual ed.1 pg3)

LAN1: Intel 82567v

LAN2: Intel 82583v

##### **1d. How many total serial ports does the box support, inside and out?**

6 Serial ports in total

Rear I/O: RS-232 (x2), RS232/422/485

Internal: RS-232 (x3)

#### **[2] Network Noodling**

##### **2a. What pins control the LEDs?**

LED0: pin 31

LED1: pin 30

LED2: pin 33

##### **2b. What address offset is the Device Control Register?**

Device Control Register offset 0x00000 and 0x00004 (datasheet pg 197)

##### **2c. What bit in the Device Control Register will force a reset of the network chip?**

Device Reset bit 26, write 1b initiates reset, reset of MAC function of device (pg 202)

### [3] Winken, Blinken, and Nod

#### 3a. What register (name and address) controls the LEDs?

\*\*Default activity can be modified in NVM. Functionality can be modified in LEDCTL.

LED Control (offset 0x00E00)

#### 3b. What bit pattern should you use to turn off LED1?

LED1\_MODE bits 11:8

LED1\_IVRT bit 14

|          | ... | 14 | ... | 11 | 10 | 9 | 8 | ... |
|----------|-----|----|-----|----|----|---|---|-----|
| LED1 off | X   | 0  | X   | 1  | 1  | 1 | 1 | X   |
| LED1 on  | X   | 1  | X   | 1  | 1  | 1 | 0 | X   |

#### 3c. What bit pattern should you use to make LED2 blink?

LED2\_MODE bits 19:16

LED2\_BLINK bit 23

|            | ... | 23 | ... | 19 | 18 | 17 | 16 | ... |
|------------|-----|----|-----|----|----|----|----|-----|
| LED2 blink | X   | 1  | X   | 1  | 1  | 1  | 0  | X   |

### [4] EIEIO

#### 4a. What company makes the Super I/O chip in this box?

Winbond

#### 4b. Can you find the datasheet on the web?

Yes

#### 4c. Where/how can you find this chip's datasheet if you aren't able to find it with a standard web search?

Go to the company's website.

### [6] Register a PCI driver

Device ID: 0x8086, 0x150c

3:00.0 Ethernet Controller: Intel Corporation 82583V Gigabit Network Connection

```
1  /*
2  * Aaron Chan
3  * ECE 373 (Spring 2017)
4  * Assignment #3
5  *
6  * This module will initialize a PCI driver
7  * so that we can write to the Atom Box's
8  * LED control register to turn it on and off.
9  */
10
11 #include <linux/module.h>
12 #include <linux/types.h>
13 #include <linux/kdev_t.h>
14 #include <linux/fs.h>
15 #include <linux/cdev.h>
16 #include <linux/slab.h>
17 #include <linux/uaccess.h>
18 #include <linux/pci.h>
19
20 #define DEVCNT 5
21 #define DEVNAME "my_pci_dev"
22
23 #define VENDOR_ID 0x8086
24 #define DEVICE_ID 0x150c
25 #define LED_REG 0x0E00
26
27 static struct mydev_dev {
28     struct cdev cdev;
29     int syscall_val; // not used in this assignment
30     int led_val; // store value for LED register
31     void *hw_addr; // base address of driver
32
33 } mydev;
34
35 static dev_t mydev_node;
36 static char *device_name = "my_pci_dev";
37
38 static DEFINE_PCI_DEVICE_TABLE(pci_test_tbl) = {
39     { PCI_DEVICE(VENDOR_ID, DEVICE_ID) },
40     { }, /* must have an empty at the end! */
41 };
42
43 static int my_pci_probe(struct pci_dev *pdev, const struct pci_device_id *ent)
44 {
45     resource_size_t mmio_start, mmio_len;
46     int bars, err;
47
48     /* this is where I'd map BAR's for access, save stuff off, etc. */
49     printk(KERN_INFO "It's dangerous to go alone, take this with you.\n");
50
51     err = pci_enable_device_mem(pdev);
52
53     // set up pci pci connections
54     bars = pci_select_bars(pdev, IORESOURCE_MEM);
55     err = pci_request_selected_regions(pdev, bars, device_name);
56
57     pci_set_master(pdev);
58
59     // map memory
60     mmio_start = pci_resource_start(pdev, 0);
61     mmio_len = pci_resource_len(pdev, 0);
62     mydev.hw_addr = ioremap(mmio_start, mmio_len);
63
64     /* 0 means success */
65     return 0;
66 }
```

```

67 }
68 // Assignment #3 pci remove function
69 static void my_pci_remove(struct pci_dev *pdev)
70 {
71     iounmap(mydev.hw_addr);
72     pci_release_selected_regions(pdev, pci_select_bars(pdev, IORESOURCE_MEM));
73     pci_disable_device(pdev);
74
75     printk(KERN_INFO "So long!!\n");
76 }
77
78 // Information about my PCI driver
79 static struct pci_driver my_pci_driver = {
80     .name = DEVNAME,
81     .id_table = pci_test_tbl,
82     .probe = my_pci_probe,
83     .remove = my_pci_remove,
84 };
85
86 // Open function
87 static int pci_hw3_open(struct inode *inode, struct file *file)
88 {
89     printk(KERN_INFO "(my_pci_driver)successfully opened!\n");
90     return 0;
91 }
92
93 // Read function
94 static ssize_t pci_hw3_read(struct file *file, char __user *buf,
95                             size_t len, loff_t *offset)
96 {
97     /* Get a local kernel buffer set aside */
98     int ret;
99
100     if (*offset >= sizeof(int))
101         return 0;
102
103     /* Make sure our user wasn't bad... */
104     if (!buf) {
105         ret = -EINVAL;
106         goto out;
107     }
108
109     // Read the value in the LED register
110     mydev.led_val = readl(mydev.hw_addr + LED_REG);
111
112     // Pass LED value to userspace
113     if (copy_to_user(buf, &mydev.led_val, sizeof(unsigned int))) {
114         ret = -EFAULT;
115         goto out;
116     }
117     ret = sizeof(unsigned int);
118     *offset += len;
119
120     /* Good to go, so printk the thingy */
121     printk(KERN_INFO "(my_pci_driver:read)User got from us %d\n", mydev.led_val);
122
123 out:
124     return ret;
125 }
126
127 // Write function
128 static ssize_t pci_hw3_write(struct file *file, const char __user *buf,
129                              size_t len, loff_t *offset)
130 {
131     int ret;
132

```

```

133     /* Make sure our user isn't bad... */
134     if (!buf) {
135         ret = -EINVAL;
136         goto out;
137     }
138
139     /* Copy from the user-provided buffer */
140     if (copy_from_user(&mydev.led_val, buf, len)) {
141         /* uh-oh... */
142         ret = -EFAULT;
143         goto out;
144     }
145     ret = len;
146
147     // Display value we are going to write to LED control register
148     printk(KERN_INFO "(my_pci_driver:write)Value to write to LED_CTRL register:
%d\n",mydev.led_val);
149
150     // Write to LED control register
151     writel((unsigned int)mydev.led_val,mydev.hw_addr + LED_REG);
152
153 out:
154     return ret;
155 }
156
157 /* File operations for our device */
158 static struct file_operations mydev_fops = {
159     .owner = THIS_MODULE,
160     .open = pci_hw3_open,
161     .read = pci_hw3_read,
162     .write = pci_hw3_write,
163 };
164
165 // Initialization
166 static int __init pci_hw3_init(void)
167 {
168     int ret;
169     printk(KERN_INFO "(my_pci_driver) Registering PCI Driver...\n");
170     ret = pci_register_driver(&my_pci_driver);
171
172     printk(KERN_INFO "(my_pci_driver) module loading...\n");
173
174     if (alloc_chrdev_region(&mydev_node, 0, DEVCNT, DEVNAME)) {
175         printk(KERN_ERR "alloc_chrdev_region() failed!\n");
176         return -1;
177     }
178
179     printk(KERN_INFO "Allocated %d devices at major: %d\n", DEVCNT,
180             MAJOR(mydev_node));
181
182     /* Initialize the character device and add it to the kernel */
183     cdev_init(&mydev.cdev, &mydev_fops);
184     mydev.cdev.owner = THIS_MODULE;
185
186     if (cdev_add(&mydev.cdev, mydev_node, DEVCNT)) {
187         printk(KERN_ERR "cdev_add() failed!\n");
188         /* clean up chrdev allocation */
189         unregister_chrdev_region(mydev_node, DEVCNT);
190
191         return -1;
192     }
193     printk(KERN_INFO "(my_pci_driver)Tried to register pci driver. Return =
%d\n",ret);
194     return ret;
195 }
196

```

```
197 // Clean up when removing driver
198 static void __exit pci_hw3_exit(void)
199 {
200     /* destroy the cdev */
201     cdev_del(&mydev.cdev);
202
203     /* Unregister PCI Driver*/
204     pci_unregister_driver(&my_pci_driver);
205
206     /* clean up the devices */
207     unregister_chrdev_region(mydev_node, DEVCNT);
208     printk(KERN_INFO "(my_pci_driver) module unloaded!\n");
209 }
210
211 MODULE_AUTHOR("Aaron Chan");
212 MODULE_LICENSE("GPL");
213 MODULE_VERSION("0.2");
214 module_init(pci_hw3_init);
215 module_exit(pci_hw3_exit);
216
```

```
1  # Aaron Chan
2  # ECE373 (Spring 2017)
3  # Assignment #3 module Makefile
4
5  obj-m = blinkLEDv3.o
6  KVERSION = $(shell uname -r)
7
8  all:
9      make -C /lib/modules/$(KVERSION)/build SUBDIRS=$(shell pwd) modules
10 clean:
11     make -C /lib/modules/$(KVERSION)/build SUBDIRS=$(shell pwd) clean
12
```



```
1  /*
2  Aaron Chan
3  ECE373 (Spring 2017)
4  Assignment #3 Userspace Program
5
6  This C program writes to a node file to change the
7  state of LED0 on the Atom Box
8
9  We will imitate a LED blink by turning it on, pause
10 for 2 seconds, then turn it off. The program will also
11 read the value stored at the LED control register.
12 */
13 #include <stdio.h>
14 #include <stdlib.h>
15 #include <stddef.h>
16 #include <fcntl.h>
17 #include <string.h>
18 #include <unistd.h>
19
20 int main(void)
21 {
22     int ret, fd;
23     int readval;
24     int turn_on = 78, turn_off = 15;
25
26     // Open file made with mknod
27     fd = open("/dev/hw3_pci", O_RDWR);
28     if (fd < 0)
29     {
30         fprintf(stderr, "Error opening file.\n");
31         return -1;
32     }
33
34     // Read value from LED register
35     ret = read(fd, &readval, 0);
36     if (ret < 0)
37         fprintf(stderr, "Error reading led_val\n");
38     printf("Current led_val: %d\n", readval);
39
40     // Write to LED and turn on
41     ret = write(fd, &turn_on, sizeof(turn_on));
42     if (ret < 0)
43     {
44         fprintf(stderr, "Error writing to file.\n");
45         return -1;
46     }
47
48     // Read LED register again.
49     ret = read(fd, &readval, 0);
50     printf("Current led_val: %d\n", readval);
51     sleep(2);
52
53     // Write to LED and turn off
54     ret = write(fd, &turn_off, sizeof(turn_off));
55
56     // Done with file. Close.
57     close(fd);
58     printf("End of program.\n\n");
59     return 0;
60 }
61
62
```