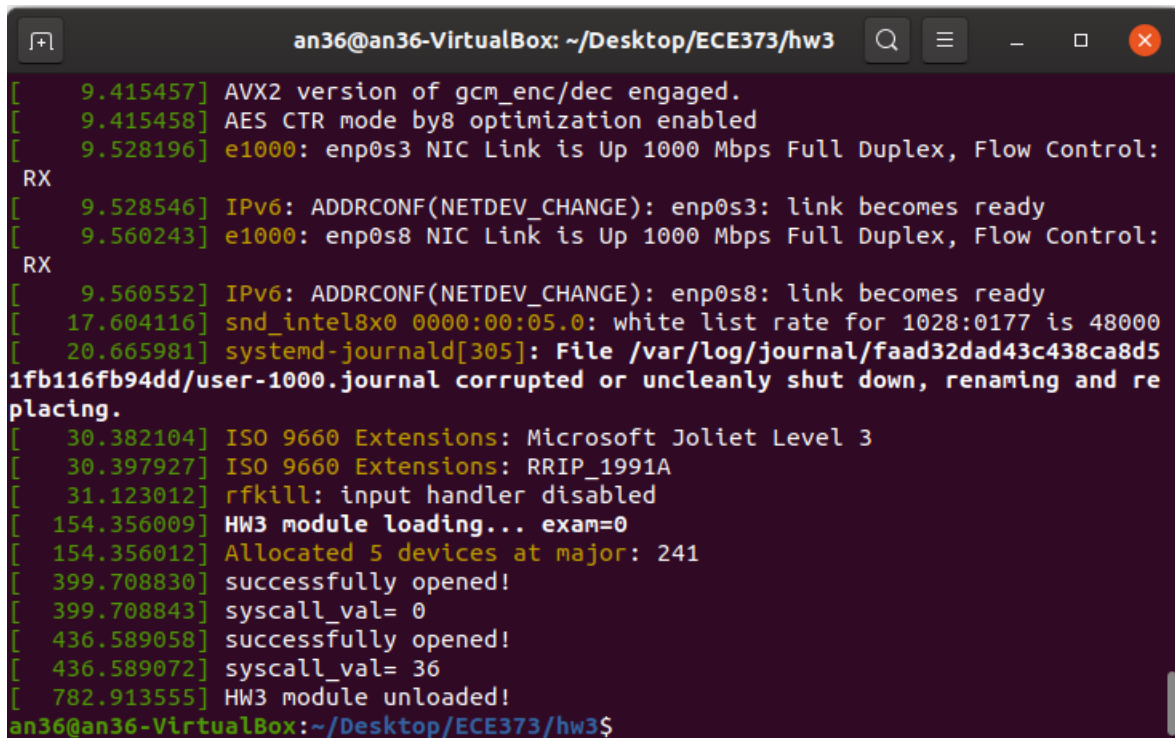


Simple PCI Dev Driver

Making it Blink:

1- Confirmation the char device works:



```
an36@an36-VirtualBox: ~/Desktop/ECE373/hw3
[ 9.415457] AVX2 version of gcm_enc/dec engaged.
[ 9.415458] AES CTR mode by8 optimization enabled
[ 9.528196] e1000: enp0s3 NIC Link is Up 1000 Mbps Full Duplex, Flow Control:
RX
[ 9.528546] IPv6: ADDRCONF(NETDEV_CHANGE): enp0s3: link becomes ready
[ 9.560243] e1000: enp0s8 NIC Link is Up 1000 Mbps Full Duplex, Flow Control:
RX
[ 9.560552] IPv6: ADDRCONF(NETDEV_CHANGE): enp0s8: link becomes ready
[ 17.604116] snd_intel8x0 0000:00:05.0: white list rate for 1028:0177 is 48000
[ 20.665981] systemd-journald[305]: File /var/log/journal/faad32dad43c438ca8d5
1fb116fb94dd/user-1000.journal corrupted or uncleanly shut down, renaming and re
placing.
[ 30.382104] ISO 9660 Extensions: Microsoft Joliet Level 3
[ 30.397927] ISO 9660 Extensions: RRIP_1991A
[ 31.123012] rfkill: input handler disabled
[ 154.356009] HW3 module loading... exam=0
[ 154.356012] Allocated 5 devices at major: 241
[ 399.708830] successfully opened!
[ 399.708843] syscall_val= 0
[ 436.589058] successfully opened!
[ 436.589072] syscall_val= 36
[ 782.913555] HW3 module unloaded!
an36@an36-VirtualBox:~/Desktop/ECE373/hw3$
```

From the figure above, we can see that we can load, read, modify and unload our kernel module character device safely. This confirmation has been done by using the kernel module program and userspace program which can be found under *Simple Kernel Module* folder. Moreover, “syscall_val” can be modified by writing a value to the parameter “exam” under “/sys/module/HW3/parameters/exam”.

2- Register a PCI driver:

```
an36@an36-VirtualBox: ~/Desktop/ECE373
an36@an36-VirtualBox:~/Desktop/ECE373/hw3$ lspci -n
00:00.0 0600: 8086:1237 (rev 02)
00:01.0 0601: 8086:7000
00:01.1 0101: 8086:7111 (rev 01)
00:02.0 0300: 15ad:0405
00:03.0 0200: 8086:100e (rev 02)
00:04.0 0880: 80ee:cafe
00:05.0 0401: 8086:2415 (rev 01)
00:07.0 0680: 8086:7113 (rev 08)
00:08.0 0200: 8086:100e (rev 02)
00:0c.0 0c03: 8086:1e31
00:0d.0 0106: 8086:2829 (rev 02)
an36@an36-VirtualBox:~/Desktop/ECE373/hw3$
```

As shown in the figure above, using the command “lspci -n” lists all of the devices’ IDs. The device ID that we are looking for is ‘100e’.

After figuring out the device’s ID and vendor, I proceeded to attaching the device by implementing the .probe() function and mapping the BAR for register access. I’ve also written the .remove() function and added both pci_request and pci_release to both __init and __exit respectively.

Then, after compiling the kernel module, I unbinded e1000 and loaded the kernel module. After that, I unloaded the kernel module and binded e1000 again. This step can be seen in the figure below and in the typescript file called ‘bindscript’.

```
an36@an36-VirtualBox: ~/Desktop/ECE373/hw3
an36@an36-VirtualBox:~/Desktop/ECE373/hw3$ script
Script started, file is typescript
an36@an36-VirtualBox:~/Desktop/ECE373/hw3$ sudo su
[sudo] password for an36:
root@an36-VirtualBox:/home/an36/Desktop/ECE373/hw3# echo 0000:00:03.0 > /sys/module/e1000/drivers
pci\:e1000/unbind
root@an36-VirtualBox:/home/an36/Desktop/ECE373/hw3# exit
an36@an36-VirtualBox:~/Desktop/ECE373/hw3$ sudo insmod HW3.ko
an36@an36-VirtualBox:~/Desktop/ECE373/hw3$ lspci -s 00:03.0 -vv
00:03.0 Ethernet controller: Intel Corporation 82540EM Gigabit Ethernet Controller (rev 02)
Subsystem: Intel Corporation PRO/1000 MT Desktop Adapter
Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr- Stepping- SERR- FastB
2B- DisINTx-
Status: Cap+ 66MHz+ UDF- FastB2B- ParErr- DEVSEL=medium >TAbort- <TAbort- <MAbort- >SERR-
<PERR- INTx-
Latency: 64 (63750ns min)
Interrupt: pin A routed to IRQ 19
Region 0: Memory at f8200000 (32-bit, non-prefetchable) [size=128K]
Region 2: I/O ports at d020 [size=8]
Capabilities: <access denied>
Kernel driver in use: HW3_driver
Kernel modules: e1000

an36@an36-VirtualBox:~/Desktop/ECE373/hw3$ sudo rmmod HW3.ko
an36@an36-VirtualBox:~/Desktop/ECE373/hw3$ sudo su
root@an36-VirtualBox:/home/an36/Desktop/ECE373/hw3# echo 0000:00:03.0 > /sys/module/e1000/drivers
pci\:e1000/bind
root@an36-VirtualBox:/home/an36/Desktop/ECE373/hw3# exit
an36@an36-VirtualBox:~/Desktop/ECE373/hw3$ exit
Script done, file is typescript
an36@an36-VirtualBox:~/Desktop/ECE373/hw3$
```

Moreover, when loading and unloading the kernel module, the following happens. As shown in the figure below, the BAR gets printed along with the major number and the “HW3 module loading... exam=0” message when the module gets loaded. While, when the module gets unloaded, dmesg shows “HW3 module unloaded” and “**released**” messages. And when binding e1000 again the content under “**released**” in the figure below gets printed

```
[ 5123.343239] HW3 module loading... exam=0
[ 5123.343242] Allocated 5 devices at major: 241
[ 5123.343245] hw_addr=00000000cae4ff57
[ 5203.017946] HW3 module unloaded!
[ 5203.017947] released
[ 5212.318879] e1000 0000:00:03.0 eth0: (PCI:33MHz:32-bit) 08:00:27:e5:3b:68
[ 5212.318883] e1000 0000:00:03.0 eth0: Intel(R) PRO/1000 Network Connection
[ 5212.320736] e1000 0000:00:03.0 enp0s3: renamed from eth0
[ 5214.355301] e1000: enp0s3 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: RX
[ 5214.356151] IPv6: ADDRCONF(NETDEV_CHANGE): enp0s3: link becomes ready
an36@an36-VirtualBox:~/Desktop/ECE373/hw3$
```

3- Hook up file operations:

I modified the read function in my kernel module program to read and print the LED register address using the information found above. As shown in the figure below, the LED register address gets assigned to ‘syscall_val’ and printed under dmesg.

```
[ 5803.257037] HW3 module loading... exam=0
[ 5803.257038] Allocated 5 devices at major: 241
[ 5803.257039] hw_addr=00000000cae4ff57
[ 5905.927792] successfully opened!
[ 5905.927820] LED register: syscall_val= 0x7068302
an36@an36-VirtualBox:~/Desktop/ECE373/hw3$
```

I’ve also modified the write function to write the LED register with a value supplied in the write() system call. The figure below shows the results of writing to the kernel module.

```
[ 8921.351407] HW3 module loading... exam=0
[ 8921.351411] Allocated 5 devices at major: 241
[ 8921.351413] hw_addr=00000000cae4ff57
[ 8934.923630] successfully opened!
[ 8934.923659] LED register: syscall_val= 0x2040603
[ 8960.106698] successfully opened!
[ 8960.106750] Userspace wrote 32343633 to syscall_val
[ 8983.861823] successfully opened!
[ 8983.861867] Userspace wrote 36337830 to syscall_val
an36@an36-VirtualBox:~/Desktop/ECE373/hw3$
```

4- The Home stretch! :

```
an36@an36-VirtualBox:~/Desktop/ECE373/hw3$ sudo su
[sudo] password for an36:
root@an36-VirtualBox:/home/an36/Desktop/ECE373/hw3# echo 0000:00:03.0 > /sys/module/e1000/drivers/pci\:e1000/unbind
root@an36-VirtualBox:/home/an36/Desktop/ECE373/hw3# exit
an36@an36-VirtualBox:~/Desktop/ECE373/hw3$ sudo insmod HW3.ko
an36@an36-VirtualBox:~/Desktop/ECE373/hw3$ sudo mknod /dev/ece c 241 0
an36@an36-VirtualBox:~/Desktop/ECE373/hw3$ sudo ./testing
Current LED register value: 7068302
new value of LED register (LED on): 706830E
new value of LED register (LED off): 706830F
file closed
an36@an36-VirtualBox:~/Desktop/ECE373/hw3$
```

As shown in the figure above, I've modified my userspace program which is called 'testing.c' to open the character device node file, read and print the LED register value. Then, change the LED register value to turn the LED on, print the new value and go to sleep for two seconds. After that, change the LED register value once again to turn the LED off and print the new value. Note: the userspace program (testing.c) open a node file called '/dev/ece'. Also, the process shown in the figure can also be found within a typescript file called 'typescript'