ASPM on Linux

ASPM is a PCI-E enhancement. It allows for a device to go completely into electrically idle state, meaning it will not send or receive electrical signals for a while. While ASPM brings a reduction in power consumption, it can also result in increased latency as the serial bus needs to be 'woken up' from low-power mode, possibly reconfigured and the host-to-device link re-established. This is known as ASPM exit latency and takes up valuable time which can be annoying to the end user if it is too obvious when it occurs. This may be acceptable for mobile computing, however, when battery life is critical.

Power Mode L0 and L1

Currently, two low power modes are specified by the PCIe 2.0 specification; L0s and L1 mode. The first mode (L0s) concerns setting low power mode for one direction of the serial link only, usually downstream of the PHY controller. The second mode (L1) is bidirectional and results in greater power reductions though with the penalty of greater exit latency.

PCIE cards should always support ASPM, what the ASPM requirements says today is that L1 is mandatory and L0s is optional unless the formfactor specifications explicitly requies it. Not sure which form factors explicitly require L0s (anyone?). Additionally software must not enable L0s in either direction on a given Link unless components on both sides of the Link each support L0s.

The way it typically works internally on endpoints (devices) is that there are idle timers (counters) in the chipset. There is a set point at which the PCIe link is idle enough to enter L0s, and a second point at which we're idle enough to enter L1. A device could potentially 'support' L0s but internally the timers could be set such that L0s and L1 happen at the same time or L0s happens after L1, so the link will essentially never enter L0s. ASPM compliance may vary by device, ASPM specification has varied as new releases have been

Check current ASPM status

You can verify by using lspci -vvv as root and check the LnkCtl attribute of the devices.

```
# lspci -vvv

In case of enabled:
      LnkCtl:   ASPM L0s L1 Enabled
In case of disabled:
      LnkCtl: ASPM Disabled; RCB 128 bytes Disabled-
```

Why is ASPM disabled for my device? ASPM should automatically be negotiated by the BIOS based on all the endpoints connected on a root complex. IF your device has ASPM disabled it is likely because:

- the BIOS determined that needed to happen
- PCIE requires ASPM but L0s is optional so you might have L0s disabled and only L1 enabled
- you have a buggy BIOS
- you have no BIOS and your systems programmers didn't address ASPM yet

Enforce Linux kernel ASPM support

An Operating System should not need to muck with ASPM, the BIOS would have dealt with the capability exchanges between the root complex and the different endpoints. Of course, BIOSes are buggy though – so the Linux kernel does have the capability to oversee and review the capabilities by itself and overrule the BIOS. ASPM support in the Linux kernel is also used to expose ASPM capabilities for PCIE devices to userspace (need confirmation, I see this being done in the code, but makes no sense).

```
# nano /etc/default/grub

Add pcie_aspm=force to the kernel parameter list:
GRUB_CMDLINE_LINUX_DEFAULT="quiet pcie_aspm=force"

# update-grub
# reboot
```

Enabling ASPM with setpci

The PCIE Link Control Register is properly parsed with lspci -vvv but you might want to know exactly how to determine if your device has ASPM support manually. This is required to know exactly where to poke a PCIE device to force enable ASPM manually for a specific root complex or endpoint device. This section covers how to do this.

How to read the Link Control Register for ASPM

The Link Control Register on the PCI device tells us if ASPM is enabled and what ASPM settings will be used. How to find the register for the Link Control Register for any PCIE device is explained below – but first lets review what to look out for on the register. Look at the last byte of the Link Control Register on the PCIE device the values to decode for ASPM are as follows:

```
0b00 = L0 only
0b01 = L0s only
0b10 = L1 only
0b11 = L1 and L0s
```

We have to enable ASPM on two levels:

1. The PCIE controller (root complex)
2. The device itself

Change setting on PCIE controller level

First find the bus address for the device you want to check for. On a box with Atheros you might get:

```
# lspci | grep -i atheros
```

```
03:00.0 Network controller: Atheros Communications Inc. Device 0030 (rev 01)
```

The 03:00.0 is the bus address. Now first check on which root complex this device sits on, by using lspci -t. You do this to first find the Link Control Register settings of your root complex. Only after you've tuned that should you tune the card. You should unload the module or turn the device off prior to tweaking with it. For the root complex this should not be needed.

```
# lspci -t

-[0000:00]-+-00.0
           +-02.0
           +-02.1
           +-03.0
           +-1b.0
           +-1c.0-[0000:02]--
           +-1c.1-[0000:03]----00.0
           +-1c.2-[0000:04]--
           +-1c.3-[0000:05-0c]--
           +-1c.4-[0000:0d-14]--
           +-1d.0
            \-1f.3
```

In this case we see 03:00.0 sits on 00:1c.1 so you can now do lspci -s 00:1c.1 -xxx on that root complex and to get the PCI config space of that device. The PCIE spec has a fun little algorithm to find the Link Control Register out of the PCI config space. The logic is as follows:

- Read 0x34 and read the register that points to
- If that value is not 0x10 then read the next byte (0x35) and go read that register
- If that register is not 0x10 then read the next byte and go read that register
- Repeat this until you find a register that has 0x10
- Once you find the register with 0x10 then add 0x10 to the final register you were reading
- The Link Control Register is this final register + 0x10 Lets analyze a real world example of a root complex, specifically the one of the root complex above.

```
# lspci -s 00:1c.1 -xxx

00:1c.1 PCI bridge: Intel Corporation 82801H (ICH8 Family) PCI Express Port
00: 86 80 41 28 07 05 10 00 03 00 04 06 10 00 81 00
10: 00 00 00 00 00 00 00 00 00 03 03 00 30 30 00 00
20: 00 dc 30 df e1 df e1 df 00 00 00 00 00 00 00 00
30: 00 00 00 00 40 00 00 00 00 00 00 00 0b 02 04 00
40: 10 80 41 01 c0 8f 00 00 00 00 10 00 11 2c 11 02
50: 40 00 11 30 e0 a0 18 00 00 00 48 01 00 00 00 00
60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
80: 05 90 01 00 0c 30 e0 fe 69 41 00 00 00 00 00 00
....
....
```

So first read 0x34, and we see it is 0x40 (do not hop to the next byte here). We read 0x40 and see it is 0x10. Now we add 0x40 + 0x10 = 0x50. We read 0x50. 0x50 is the value of the Link Control

Register. 0x50 has a value of 0x40. This means only L0 is enabled so ASPM is completely disabled. To tweak ASPM for this root complex then we we would do have to first keep the original value and then OR it with our new ASPM settings. Note: as it turns out 0x50 is also used for the Link Control Register for ICH6, ICH7, ICH8, ICH9.

```
# Disables ASPM, enables only L0 (this was the existing setting)
sudo setpci -s 00:1c.1 0x50.B=0x40

# Enable L0s only
sudo setpci -s 00:1c.1 0x50.B=0x41

# Enable L1 only
sudo setpci -s 00:1c.1 0x50.B=0x42

# Enable L1 and L0s
sudo setpci -s 00:1c.1 0x50.B=0x43
```

Change setting on Device level

Now – lets get on to tweaking your device. Since you have the bus address of your 802.11 device you can now use this to get its respective PCI config space in hex.

```
# lspci -s 03:00.0 -xxx

03:00.0 Network controller: Atheros Communications Inc. Device 0030 (rev 01)
00: 8c 16 30 00 03 01 10 40 01 00 80 02 10 00 00 00
10: 04 00 3e df 00 00 00 00 00 00 00 00 00 00 00 00
20: 00 00 00 00 00 00 00 00 00 00 00 00 8c 16 16 31
30: 00 00 00 00 40 00 00 00 00 00 00 00 0b 01 00 00
40: 01 50 c3 5b 00 00 00 00 00 00 00 00 00 00 00 00
50: 05 70 84 01 00 00 00 00 00 00 00 00 00 00 00 00
60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
70: 10 00 02 00 00 87 04 05 10 20 0b 00 11 5c 03 00
80: 41 00 11 10 00 00 00 00 00 00 00 00 00 00 00 00
...
...
```

This example is a little more complicated so we'll analyze it line by line:

```
00: 8c 16 30 00 03 01 10 40 01 00 80 02 10 00 00 00
10: 04 00 3e df 00 00 00 00 00 00 00 00 00 00 00 00
20: 00 00 00 00 00 00 00 00 00 00 00 00 8c 16 16 31
30: 00 00 00 00 40 00 00 00 00 00 00 00 0b 01 00 00
    ^            ^
    |            |
   0x30         0x34

So 0x34 = 0x40. 0x40 is not 0x10 so we go read 0x40 now
```

```
40: 01 50 c3 5b 00 00 00 00 00 00 00 00 00 00 00 00
    ^
    |
   0x40 = 0x01, this is not 0x10 so read the next byte

40: 01 50 c3 5b 00 00 00 00 00 00 00 00 00 00 00 00
       ^
       |
      0x41 = 0x50, so go read that register next

50: 05 70 84 01 00 00 00 00 00 00 00 00 00 00 00 00
    ^
    |
   0x50 = 0x05, this is not 0x10, so go read the next byte.
   The next byte 0x51 = 0x70 so we go read that register next.

70: 10 00 02 00 00 87 04 05 10 20 0b 00 11 5c 03 00
    ^
    |
    At last, 0x70 = 0x10. So now we do 0x70 + 0x10 = 0x80 and go read 0x80.

80: 41 00 11 10 00 00 00 00 00 00 00 00 00 00 00 00
    ^
    |
    0x80 = 0x41
    0x41 = 0b1000001 so this has ASPM L0s on only.
```

Assuming your root complex has L1 enabled as well already you can force ASPM L1 or tune ASPM off if it had it on. To change the ASPM settings of the device you would change only the last two bits of the byte 0x80 here, so keep the other values (OR the values) as follows:

```
# Disables ASPM, enables only L0
sudo setpci -s 03:00.0 0x80.B=0x40

# Enable L0s only (this was the existing setting)
sudo setpci -s 03:00.0 0x80.B=0x41

# Enable L1 only
sudo setpci -s 03:00.0 0x80.B=0x42

# Enable L1 and L0s
sudo setpci -s 03:00.0 0x80.B=0x43
```