How Not to Write an x86 Platform Driver

Core-kernel dev plays with device drivers....

October 24, 2013

Darren Hart <darren.hart@intel.com>

Agenda

- Platform
- MinnowBoard Examples
- Lessons Learned
- MinnowBoard Take 2
- Next Steps

Agenda

- Platform
- MinnowBoard Examples
- Lessons Learned
- MinnowBoard Take 2
- Next Steps

Platform

- Computer architecture (PC)
- Software frameworks (OS, application framework, etc.)
- Intel-ese: A CPU and Chipset... or SoC... or...
- Linux Platform drivers
  - Pseudo-bus drivers
  - Board-file drivers
  - (Drivers without hardware enumeration and description)
- Examples
  - PC BIOS
  - UEFI
  - Android
- There's just "little bit" of room for confusion here...

Agenda

- Platform
- MinnowBoard Examples
- Lessons Learned
- MinnowBoard Take 2
- Next Steps

MinnowBoard: Overview

- Intel Atom E6xx CPU (TunnelCreek)
- Intel EG20T PCH (Topcliff)
- TunnelCreek + Topcliff = Queensbay (Platform!)

- 32bit UEFI Firmware
- One of the first designs to make full use of all the GPIO
  - Buttons
  - LEDs
- UART
  - 50MHz clock not the default for the driver (pch_uart)
- Ethernet
  - Low-cost PHY with no EPROM for the Ethernet Address

MinnowBoard: Dynamic Baseboard

- Typical x86 designs have fixed baseboard
- Expansion via self-enumerating buses

- MinnowBoard supports daughter cards called “Lures”
  - USB
  - PCI
  - I2C
  - SPI
  - CAN
  - GPIO
- Requires an in-field-defined hardware description mechanism

MinnowBoard: GPIO

- Three sources of GPIO
  - MFD -> LPC -> GPIO Core (5) and Suspend (8) Wells
  - PCH (12)
  - Both PCI enumerated
- Uses
  - 4 User Buttons
  - 2 User LEDs
  - PHY Reset
  - Expansion GPIO

MinnowBoard: Board-Files

- A board-file is a self-describing non-enumerated driver
- Several examples in the kernel to follow
- Simple to write
  - Reserve GPIO
    - Buttons, LEDs, PHY Reset
  - Define and export platform functions
    - PHY wakeup, board detection
  - Create the Pseudo-bus drivers
    - gpio_keys
    - leds-gpio
  - Export expansion GPIO to sysfs

MinnowBoard: Board-Files

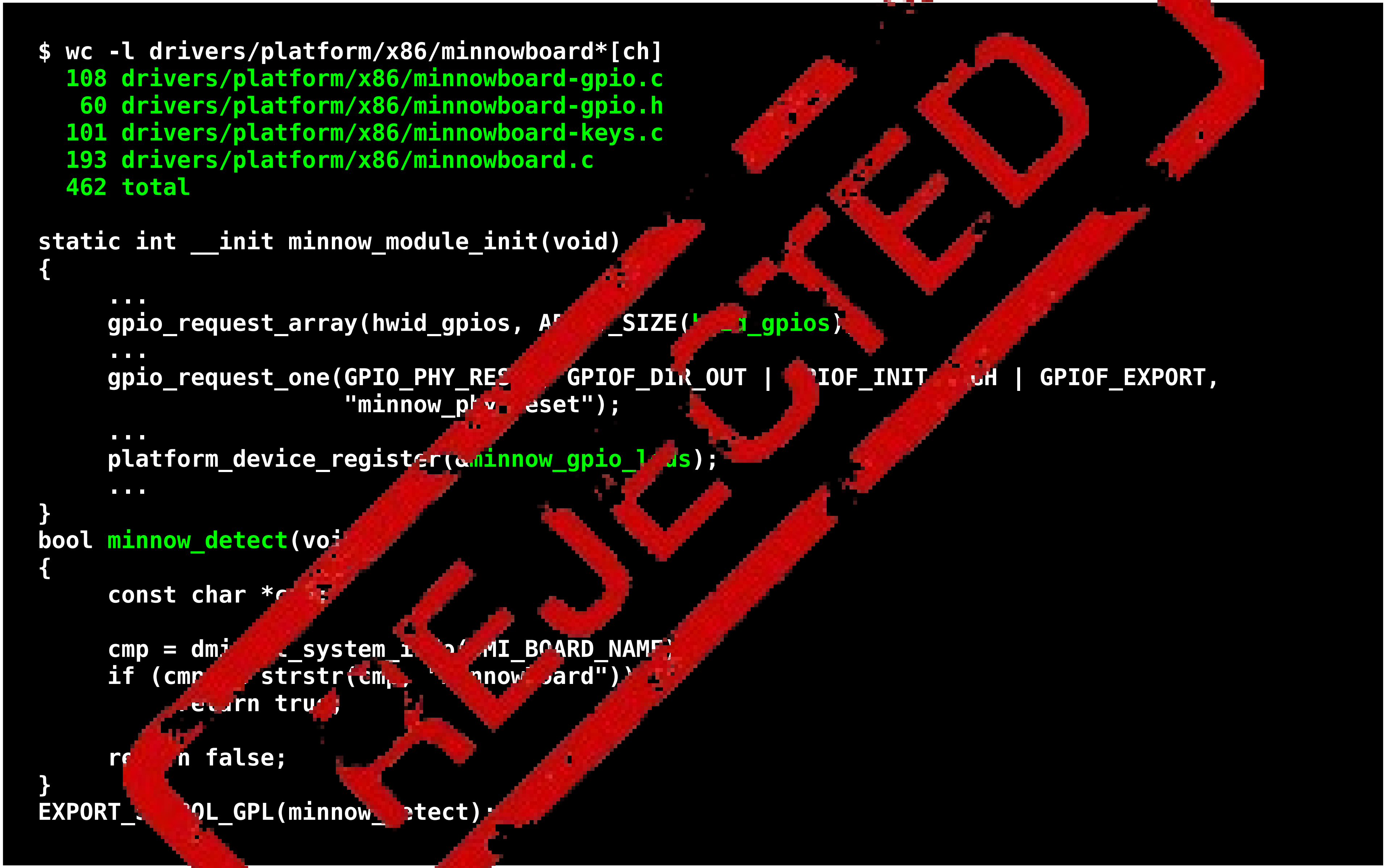
```
$ wc -l drivers/platform/x86/minnowboard*[ch]
  108 drivers/platform/x86/minnowboard-gpio.c
   60 drivers/platform/x86/minnowboard-gpio.h
  101 drivers/platform/x86/minnowboard-keys.c
  193 drivers/platform/x86/minnowboard.c
  462 total

static int __init minnow_module_init(void)
{
     ...
     gpio_request_array(hwid_gpios, ARRAY_SIZE(hwid_gpios));
     ...
     gpio_request_one(GPIO_PHY_RESET, GPIOF_DIR_OUT | GPIOF_INIT_HIGH | GPIOF_EXPORT,
                      "minnow_phy_reset");
     ...
     platform_device_register(&minnow_gpio_leds);
     ...
}
bool minnow_detect(void)
{
     const char *cmp;

     cmp = dmi_get_system_info(DMI_BOARD_NAME);
     if (cmp && strstr(cmp, "MinnowBoard"))
          return true;

     return false;
}
EXPORT_SYMBOL_GPL(minnow_detect);
```

MinnowBoard: Board Files



```
$ wc -l drivers/platform/x86/minnowboard*[ch]
  108 drivers/platform/x86/minnowboard-gpio.c
   60 drivers/platform/x86/minnowboard-gpio.h
  101 drivers/platform/x86/minnowboard-keys.c
  193 drivers/platform/x86/minnowboard.c
  462 total

static int __init minnow_module_init(void)
{
     ...
     gpio_request_array(hwid_gpios, ARRAY_SIZE(hwid_gpios));
     ...
     gpio_request_one(GPIO_PHY_RESET, GPIOF_DIR_OUT | GPIOF_INIT_HIGH | GPIOF_EXPORT,
                      "minnow_phy_reset");
     ...
     platform_device_register(&minnow_gpio_leds);
     ...
}
bool minnow_detect(void)
{
     const char *cmp;

     cmp = dmi_get_system_info(DMI_BOARD_NAME);
     if (cmp && strstr(cmp, "Minnowboard"))
          return true;

     return false;
}
EXPORT_SYMBOL_GPL(minnow_detect);
```

REJECTED

MinnowBoard: Board-Files Bad

- Not automatically enumerated and loaded
  - Leads to evil vendor trees
  - Make assumptions about hardware layout
  - Dangerous
  - Reduces image reuse
  - Fragment the platform
- Don’t leverage code reuse
- Code bloat, added maintenance
- Add unnecessary dependency to independent drivers
  - pch_uart, pch_gbe, lpc_sch, gpio_sch, gpio_pch

- DO NOT WRITE X86 BOARD FILES… TO BE CONTINUED…

MinnowBoard: UART

- PCI Enumerated
- Vendor/Device ID insufficient
- Firmware can select the clock
- Existing precedent uses SMBIOS (not for new drivers)

```
static struct dmi_system_id pch_uart_dmi_table[] = {
	...
	{
		.ident = "Fish River Island II",
		{
			DMI_MATCH(DMI_PRODUCT_NAME, "Fish River Island II"),
		},
		(void *)FRI2_48_UARTCLK,
	},
	{
		.ident = "MinnowBoard",
		{
			DMI_MATCH(DMI_BOARD_NAME, "MinnowBoard"),
		},
		(void *)MINNOW_UARTCLK,
	},
};
```

MinnowBoard: Ethernet PHY

- Software configured 2ns TX Clock delay
- Aggressive power saving, must be woken up

```
/* Wake up the PHY */
gpio_set_value(13, 0);
usleep_range(1250, 1500);
gpio_set_value(13, 1);
usleep_range(1250, 1500);

/* Configure 2ns Clock Delay */
pch_gbe_phy_read_reg_miic(hw, PHY_AR8031_DBG_OFF, &mii_reg);
pch_gbe_phy_write_reg_miic(hw, PHY_AR8031_DBG_OFF, PHY_AR8031_SERDES);
pch_gbe_phy_read_reg_miic(hw, PHY_AR8031_DBG_DAT, &mii_reg);
mii_reg |= PHY_AR8031_SERDES_TX_CLK_DLY;
pch_gbe_phy_write_reg_miic(hw, PHY_AR8031_DBG_DAT, mii_reg);


/* Disable Hibernate */
pch_gbe_phy_write_reg_miic(hw, PHY_AR8031_DBG_OFF, PHY_AR8031_HIBERNATE);
pch_gbe_phy_read_reg_miic(hw, PHY_AR8031_DBG_DAT, &mii_reg);
mii_reg &= ~PHY_AR8031_PS_HIB_EN;
pch_gbe_phy_write_reg_miic(hw, PHY_AR8031_DBG_DAT, mii_reg);
```

MinnowBoard: Ethernet PHY

- How do you identify the PHY?
  - RGMII read
  - But you can’t read yet because it’s asleep…
- How do you identify the platform?
  - ~~SMBIOS~~
  - ~~Device Tree~~
  - ~~ACPI~~ (actually, this could work well)
  - PCI Subsystem ID (Already PCI Enumerated)
- How do you describe the hardware?
  - ~~Board-File platform functions~~ (REJECTED)
  - Platform init routine and private driver data per platform
- Next Steps: PHYLIB

MinnowBoard: Ethernet PHY

```
+static struct pch_gbe_privdata pch_gbe_minnow_privdata = {
+       .phy_tx_clk_delay = true,
+       .phy_disable_hibernate = true,
+       .platform_init = pch_gbe_minnow_platform_init,
+};


 static DEFINE_PCI_DEVICE_TABLE(pch_gbe_pcidev_id) = {
+       {.vendor = PCI_VENDOR_ID_INTEL,
+        .device = PCI_DEVICE_ID_INTEL_IOH1_GBE,
+        .subvendor = PCI_VENDOR_ID_CIRCUITCO,
+        .subdevice = PCI_SUBSYSTEM_ID_CIRCUITCO_MINNOWBOARD,
+        .class = (PCI_CLASS_NETWORK_ETHERNET << 8),
+        .class_mask = (0xFFFF00),
+        .driver_data = (kernel_ulong_t)&pch_gbe_minnow_privdata
+        },
        {.vendor = PCI_VENDOR_ID_INTEL,
         .device = PCI_DEVICE_ID_INTEL_IOH1_GBE,
         .subvendor = PCI_ANY_ID,
         .subdevice = PCI_ANY_ID,
         .class = (PCI_CLASS_NETWORK_ETHERNET << 8),


+static int pch_gbe_minnow_platform_init(struct pci_dev *pdev) { ... }
+static int pch_gbe_phy_tx_clk_delay(struct pch_gbe_hw *hw) { ... }
+int pch_gbe_phy_disable_hibernate(struct pch_gbe_hw *hw) { ... }
```

MinnowBoard: Ethernet MAC

- No EEPROM for Ethernet Address
  - Reduced cost
- ~~Use one from the local allocation pool~~
- Where should we store the Ethernet Address?
  - ~~Fixed location in memory~~
  - ~~EFI Var~~
  - PCI registers
- The first implementation used EFI Vars
  - Not quite as horrible as you might think
- Final solution was done in firmware to read a fixed location from the SPI flash and populate the PCI MAC register
  - No driver changes required!

Agenda

- Platform
- MinnowBoard Examples
- Lessons Learned
- MinnowBoard Take 2
- Next Steps

Lessons: Platform

- A “Platform” is a reusable building block
- The less it changes, the more reusable it is
- Reduces time-to-market
- x86 has a well established platform
  - Consider the number of systems current Linux Distributions support on the same Linux kernel
  - This must be preserved and upheld

Lessons: Complexity

- Core kernel
  - Simple primitives
  - Complex algorithms
- Drivers
  - Simple algorithms
  - Complex set of primitives

Lessons: The Front End

- Many IA products closely follow a reference design
  - High confidence in reference design
  - Can lead to inflexible driver implementations
- If you have input into the design phase
  - Consider existing device driver support
  - Component selection
    - Which PHY to use with a given MAC?
  - Layout and configuration

Lessons: Identification and Description

- The problem can be reduced to:
  - Identification
  - Description
- Identification
  - Vendor/Product IDs
    - PCI Subsystem ID
  - Firmware (ACPI, DT)
  - SMBIOS
- Description
  - PCI Config or Registers (USB?)
  - Hardcoded by ID
  - Firmware (ACPI, DT)

Lessons: It's Not About You!

- Reduce long-term maintenance
  - Avoid board-files, reuse existing platform drivers
  - Avoid creating evil vendor trees
- Don't lock your customers to a specific kernel version
- Avoid creating unnecessary driver dependencies
  - pch_gbe and minnowboard board-files
- Use device meta-data rather than a new device ID to distinguish between functionally equivalent devices
  - Such as UART-to-USB devices
- Simplify existing Linux distribution support
  - Stable trees and distributions will readily pull in device IDs
  - Reusable device IDs are even better

Lessons: It’s Not About You!

“If you’re a company that thinks your tiny change to the kernel is what gives you a competitive edge, you’ll probably be facing economic problems. You’d be much better off worrying about making the best damn hardware for the lowest price.”

-- Linus Torvalds, LinuxCon Europe, 2013

Agenda

- Platform
- MinnowBoard Examples
- Lessons Learned
- MinnowBoard Take 2
- Next Steps

Take 2: GPIO Revisited

- New approach based on Lessons Learned
- No Board-Files
  - No new files at all
- Reuse existing code
  - Or at least set the stage to do so in the future
- Support the platform
  - ACPI device identification and description

Take 2: Identification and Description

- **ACPI 5.0 does:**
  - Assign device IDs to pseudo devices
  - Adding ACPI enumeration to PCI devices is trivial and links the pseudo device with the underlying physical device in the in-kernel device tree
  - Identify GPIO resources (pins, address, interrupt lines)
- **ACPI 5.0 does not (currently):**
  - Provide a standard mechanism to describe arbitrary device attributes
    - Keybinding, default trigger, number of queues, etc.
    - Some vendors currently invent their own
- **Acknowledgements:**
  - Rafael Wysocki, Mika Westerberg, Mathias Nyman, Robert Moore
  - H. Peter Anvin, Len Brown, Mark Doran
  - Linus Walleij, many more....

Take 2: ACPI DSDT Example

- Define a pseudo device LEDS below the LPC device
- Create the hardware ID “MNW0003”
- Add GPIO 10 and 11 (relative to the LPC device) to LEDS
- The kernel can identify, but has no mapping information

```
Scope (\_SB.PCI0.LPC) {
    Device (LEDS) {
        Name (_HID, "MNW0003")
        Method (_CRS, 0, Serialized) {
            Name (RBUF, ResourceTemplate () {
                GpioIo (Exclusive, PullDown, 0, 0, IoRestrictionOutputOnly,
                        "\\_SB.PCI0.LPC", 0, ResourceConsumer,,)
                {
                    10 // SUS 5
                }
                GpioIo (Exclusive, PullDown, 0, 0, IoRestrictionInputOnly,
                        "\\_SB.PCI0.LPC", 0, ResourceConsumer,,)
                {
                   11 // SUS6
                }
            })
            Return (RBUF)
}}}
```

Take 2: ACPI Packages and Properties

- Packages are non-typed arrays
- Packages can nested
- Packages can easily implement dictionaries
- A PROPERTIES method could return a dictionary

```
Package() { <VALUE1>, <VALUE2> }

Package() {
    Package() { <VALUE1>, <VALUE2> }
    Package() { <VALUE1>, <VALUE2> }
}

Package() {
    Package() { “String”, “Hello World” }
    Package() { “Number”, 10 }
    Package() { “List”, Package() { 1, 2 } }
}

Method (PROPERTIES, 0, NotSerialized) {
    Return (Package() {
        Package() { "Key", “Value” }
    })
}
```

Take 2: ACPI _PRP Method Proposal

- A standardized mechanism is needed
- Consider an ACPI reserved method _PRP
  - Optionally implemented per device

```
Scope (\_SB.PCI0.LPC) {
    Device (LEDS) {
        Name (_HID, "MNW0003")
        Method (_CRS, 0, Serialized) { ... }

        Method (_PRP, 0, NotSerialized) {
            Return (Package() {
                Package() { "label", Package (2) { "minnow_led0", "minnow_led1" }},
                Package() { "linux,default-trigger", Package (2) { "heartbeat", "mmc0" }},
                Package() {"linux,default-state", Package (2) { "on", "on" }},
                Package() { "linux,retain-state-suspended", Package (2) { 1, 1 }},
            })
        }
    }
}
```

Take 2: ACPI Device Enumeration

- Documentation/acpi/enumeration.txt
- Add ACPI ID to drivers/acpi/acpi_platform.c
- Add ACPI enumeration to pseudo-bus driver

```
drivers/acpi/acpi_platform.c:
 static const struct acpi_device_id acpi_platform_device_ids[] = {
        { "PNP0D40" },
+       { "MNW0002" },
+       { "MNW0003" },
        { }
 };

drivers/leds/leds-gpio.c:
+#ifdef CONFIG_ACPI
+static inline struct gpio_leds_priv *
+gpio_leds_create_acpi(struct platform_device *pdev)
+{ ... }

+static const struct acpi_device_id acpi_gpio_leds_match[] = {
+       { "MNW0003" },
+       {},
+};

static struct platform_driver gpio_led_driver = {
+               .acpi_match_table = ACPI_PTR(acpi_gpio_leds_match),
```

Take 2: ACPI Device Description

- New set of ACPI APIs
- Populate the platform_device with the ACPI Properties

```
int acpi_dev_get_property_<TYPE>(struct acpi_device *adev, const char *name, <TYPE> *value)
int acpi_dev_get_property_array_<TYPE>(struct acpi_device *adev, const char *name,
                                       <TYPE> *values, size_t nvalues)

drivers/leds/leds-gpio.c:
static inline struct gpio_leds_priv *
gpio_leds_create_acpi(struct platform_device *pdev) {
     ...
     trigger = kcalloc(nleds, sizeof(char *), GFP_KERNEL);
     error = acpi_dev_get_property_array_string(adev, "linux,default-trigger",
                                                trigger, nleds);
     ...
     for (i = 0; i < nleds; i++)   {
          struct gpio_led led = {};

          led.gpio = acpi_get_gpio_by_index(dev, i, NULL);
          ...
          led.default_trigger = trigger[i];
          ...
     }
     ...
     return priv;
}
```

Take 2: ACPI MinnowBoard Example

- 3.12.0-rc5 + Minnow ACPI V2

```
cat /sys/kernel/debug/gpio

GPIOs 0-4, platform/sch_gpio.33158, sch_gpio_core:
 gpio-0   (minnow_btn0         ) in  hi
 gpio-1   (minnow_btn1         ) in  hi
 gpio-2   (minnow_btn2         ) in  hi
 gpio-3   (minnow_btn3         ) in  hi

GPIOs 5-13, platform/sch_gpio.33158, sch_gpio_resume:
 gpio-10  (minnow_led0         ) out lo
 gpio-11  (minnow_led1         ) out hi
 gpio-13  (minnow_phy_reset    ) out hi
```

Agenda

- Platform
- MinnowBoard Examples
- Lessons Learned
- MinnowBoard Take 2
- Next Steps

Next Steps

- Formalize and propose the ACPI _PRP method
  - Consider existing implementations
- One more layer of Linux device property abstraction
  - Abstract Open Firmware (DT) and ACPI (_PRP), allowing pseudo-device drivers to have a single firmware device property API
- Layered ACPI SSDT tooling
  - Current mechanisms replace rather than augment the firmware provided DSDT
- Generic ACPI platform device HIDs
  - LNX**** or just more PNP**** IDs
- Opening up firmware
  - At the very least we need to be able to rebuild it with modifications

Comments / Questions

/* */ || ?

(Come see us at the Intel booth for a “Chalk Talk”)