Linux Power Management Experiences on Moorestown

Kristen Accardi
Len Brown

Intel Open Source Technology Center

Linux Plumbers Conference
Sept. 8, 2011 Santa Rosa, CA
Intel® Atom™ Processor Z6XX Series
(Previously Codenamed Lincroft SOC)

- Highly Integrated SoC
- Combines Atom core, 3D Graphics, and Video Encode/Decode
- Compact Package: 13.8 x 13.8 x 1.1 mm³ (0.5mm pitch)
- High Performance at Low Power
  - 19 power islands w/ on-die clock and power gating
  - New ultra low power states (S0i1, S0i3)
  - Hardware Accelerated Video Encode/Decode
  - Optimized 3D graphics
  - Based on Intel Hi-k 45nm LP SoC process

**ULP ATOM CPU CORE**
- 24K Data Cache, 32K Instruction Cache
- 512K L2 Cache

**Scalable Bus Interface and Coherency engine**

**3D GRAPHICS**
- 2X Menlow performance
- OpenGL ES 2.0, OpenVG 1.0, DirectX 9.1
- PowerVR® graphics

**VIDEO ENCODE/DECODE**
- HW Accelerated

**DISPLAY**
- Up to 1366x768 LVDS, 1024x600 MIPI

**MEMORY**
- DDR2-800 MT/s
- LPDDR1-400 MT/s
Intel Platform Controller Hub MP20
(Previously Codenamed Langwell)

- Highly Integrated PCH reduces power, footprint, and cost
- Incorporates Handheld I/Os and Media Accelerators
- Compact Package: 14 x 14 x 1.3 mm³ (0.5mm pitch)
- Ultra Low Power
  - Low leakage 65nm LP process
  - Fine-grained power management
  - Enables Low active/idle platform power

**SYSTEM CONTROLLER**
- Low Power 32-bit RISC core

**IMAGE PROCESSING** (Camera support)
- 5MP & VGA Camera
- Dedicated Image Signal Processor

**AUDIO ACCELERATION**
- Support for audio codecs
- I2S interface

**CRYPTO ACCELERATION**
- Secure Boot

**IO**
- USB 2.0 EHCI Controller, USB OTG 1 Host/Device port
- HDMI Support

**Solid State Drive**
Mixed Signal IC (MSIC) (Previously Codenamed Briertown)

- Highly Integrated Design
- Efficient Power Delivery
- Integrates Components → Lower Power and Cost
- Multi-source availability: Freescale*, Maxim*, Renesas*

Diagram:
- USB/Adapter
- Battery Charger & 5V OTG Boost
- Battery Pack
- Touch Screen
- Sensor
- Lincroft
- L-langwell
- RGB
- Display
- Comm SPI
- Comm
- SPI Register Map
- DDR2
- Power Button
- GPIO
- Voltage Regulators
- Audio Class A, B, D
- RTC
- Crystal
- MIC
- Speaker
- Headphone
PCI on MRST

- Lincroft Graphics is real PCIe
- All Langwell PCI is emulated by “PCI shim”
  - Config space is simply RAM
    - Requires platform specific config space accessors
    - PMCSR does nothing
    - No #PME
- PCI Device Driver enumeration
- Not real HW, OS exposed to FW bugs
SFI – Simple Firmware Interface
SFI – an 80% solution

✓ Simple
  ✓ Difficult for FW to get it wrong

✓ Open and Freely Available
  ✓ http://simplefirmware.org
  ✓ Upstream Linux since about 2.6.32

✓ Extensible
  ✓ Vendor specific tables permitted

✗ Not Comprehensive
  ✗ Particularly for device enumeration
MRST Processor
Performance States (P-states)

- Linux sfi-cpufreq cpufreq driver
  - Consumes SFI “FREQ” table
- Linux ondemand cpufreq governor
What is S0i3 state?

Lincroft SoC

- GFX
- Video Decode
- Display
- Video Encode
- CPU in Deep Power State
- Memory controller + coherency logic
- DDR IOs (SR)
- DDR IOs
- PL
- Power Manager (PMU)
- PMU (state)
- Wakeup
- cDMI
- Display Links

Ref Clk (Off)
Wakeup (low)
S0i3 “Active Idle” System State

- S0 (C-state) latency
- S3 (system wide) low power

- S0 Active
- S0i3
- S3 Suspend
S0i3: C-state or S-state?

- S0i3 can be used for either/both use-cases.
  - Android used S0i3 only for S3 (STR).
  - MeeGo used S0i3 only as a C-state.
<table>
<thead>
<tr>
<th>Feature</th>
<th>C0 HFM</th>
<th>C0 LFM</th>
<th>C1/C2</th>
<th>C4</th>
<th>C6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core voltage</td>
<td>![Image]</td>
<td>![Image]</td>
<td>![Image]</td>
<td>![Image]</td>
<td>![Image]</td>
</tr>
<tr>
<td>Core clock</td>
<td>![Image]</td>
<td>![Image]</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>PLL</td>
<td>![Image]</td>
<td>![Image]</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>L1 caches</td>
<td>![Image]</td>
<td>![Image]</td>
<td>flushed</td>
<td>flushed</td>
<td>off</td>
</tr>
<tr>
<td>L2 caches</td>
<td>![Image]</td>
<td>![Image]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wakeup time</td>
<td>active</td>
<td>active</td>
<td>Partial flush</td>
<td>![Image]</td>
<td>![Image]</td>
</tr>
</tbody>
</table>
# S0i3 as a C-state

<table>
<thead>
<tr>
<th>Package</th>
<th>CPU 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATM-C1</td>
<td>ATM-C1 0.5%</td>
</tr>
<tr>
<td>ATM-C1</td>
<td>0.5% 5.1 ms</td>
</tr>
<tr>
<td>ATM-C2</td>
<td>ATM-C2 1.2%</td>
</tr>
<tr>
<td>ATM-C2</td>
<td>1.2% 7.7 ms</td>
</tr>
<tr>
<td>ATM-C4</td>
<td>ATM-C4 0.6%</td>
</tr>
<tr>
<td>ATM-C4</td>
<td>0.6% 5.4 ms</td>
</tr>
<tr>
<td>ATM-C6</td>
<td>ATM-C6 39.0%</td>
</tr>
<tr>
<td>ATM-C6</td>
<td>39.0% 24.9 ms</td>
</tr>
<tr>
<td>MRST-S0i3</td>
<td>MRST-S0i3 56.6%</td>
</tr>
<tr>
<td>MRST-S0i3</td>
<td>56.6% 70.9 ms</td>
</tr>
</tbody>
</table>
MRST pmu driver

- Track all PCI D-state requests
  - Translate PCI device to MRST Logical Subsystem
  - Translate PCI D-States to MRST D-states
- Facilitate S0i3 transitions
  - Track system-wide entry criteria
  - Invoked by intel_idle's S0i3 entry point

- Upstream: arch/x86/platform/mrst/pmu.c
  - See initial commit-id for contributors
Linux/MRST Power Mgmt. Architecture

- TRM: CPU1 on/offline
- sysfs CPU online
- cpuidle subsystem
  - menu governor
  - intel_idle driver
- S0i3
- Xorg: screen on/off
- PCI subsystem
  - I/O Device Drivers
  - graphics
  - mrst_pmu driver

User
Kernel

Software
Hardware

Lincroft North Complex
CPU
GMA 600

Langwell South Complex
IO Devices
SCU
PCI devices ->
MRST Logical Subsystems (LSS)

- PCI device has a driver
- MRST LSS is a HW clock/power domain

✔ Usually a simple 1:1 mapping
✗ Sometimes LSS shared
### PCI -> MRST Device-States: Imperfect Mapping

<table>
<thead>
<tr>
<th>PCI</th>
<th>Lincroft</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0</td>
<td>D0 – full on</td>
</tr>
<tr>
<td></td>
<td>D0i1 – auto clock gate</td>
</tr>
<tr>
<td>D1, D2</td>
<td></td>
</tr>
<tr>
<td>D3hot, D3cold</td>
<td>D0i2 – hard clock gate</td>
</tr>
<tr>
<td></td>
<td>D0i3 – hard clock gate</td>
</tr>
<tr>
<td>Address</td>
<td>Module</td>
</tr>
<tr>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>0800</td>
<td>dw_spi_pci</td>
</tr>
<tr>
<td>0802</td>
<td>i2c-designware-p</td>
</tr>
<tr>
<td>0803</td>
<td>i2c-designware-p</td>
</tr>
<tr>
<td>0804</td>
<td>i2c-designware-p</td>
</tr>
<tr>
<td>0806</td>
<td>ehci_hcd</td>
</tr>
<tr>
<td>0807</td>
<td>sdhci-pci</td>
</tr>
<tr>
<td>0808</td>
<td>sdhci-pci</td>
</tr>
<tr>
<td>0809</td>
<td>pci</td>
</tr>
<tr>
<td>080A</td>
<td>intel_sst_driver</td>
</tr>
<tr>
<td>080B</td>
<td>mrstisp</td>
</tr>
<tr>
<td>080C</td>
<td>pci</td>
</tr>
<tr>
<td>080D</td>
<td>pci</td>
</tr>
<tr>
<td>080E</td>
<td>intel_scu_ipc</td>
</tr>
<tr>
<td>4102</td>
<td>pvrsrvkm</td>
</tr>
<tr>
<td>0814</td>
<td>Intel MID DMA</td>
</tr>
<tr>
<td>0815</td>
<td>pci</td>
</tr>
<tr>
<td>084F</td>
<td>sdhci-pci</td>
</tr>
</tbody>
</table>
Linux Run Time Device PM

- 10 observations from deployment on MRST
Run Time Device PM #1

Understand the concept of runtime power management.

** you sleep when you are not busy, not when someone tells you to! **

Transparent to user space. Initiated by kernel.
Know what your subsystem does for you.

Review your subsystem power management code - what does it really do?

For example, on PCI, the amount of setup work you have to do in the driver is limited to allowing runtime pm and then decrementing the usage count. Similarly, in the pm_runtime_suspend/resume code paths in the driver, you will not need to duplicate core functionality like sending power state transitions.
Run Time Device PM #3

Understand all the possible **entry points** to your driver.

sysfs counts!
Run Time Device PM #4

Understand when you are actually touching hardware.
Run Time Device PM #5

Understand the context in which you are being called during these entry points.

Are you called in interrupt context?
Run Time Device PM #6

For interrupt context, ensure the hardware is available without sleeping, otherwise use a workqueue.
Run Time Device PM #7

Refcounting during the irq handler does cost cycles. this may be ok depending on your device.

On mrst, we had no #PME, so wakeups would come as regular interrupts.

This required that we do ref counting at irq time, and that we use threaded irqs as much as possible.
Run Time Device PM #8

Have the correct granularity for your ref counting. Depending on your hw, it likely has a performance penalty for entering a suspended state. You may even lose device state. Restoring can take time, so you want to make sure you are appropriately idle before allowing a suspend.
Run Time Device PM #9

Unbalanced ref counting occurs when you do not pay attention to error paths.
Run Time Device PM #10

Unbalanced ref counting can occur when you try to do something tricky with the refcounting to work around overly complex code paths.

eg. setting your pm usage count to zero without decrementing.

This will cost you hours and hours of debug time

Don't do it! Clean up your driver and stop trying to hack your way out of it.
Thank you!