Building Blocks for Embedded Power Management

Kevin Hilman
khilman@mvista.com
The mission

• Brand new embedded platform
• Fresh baseport of linux
• Your job, add power management

• Where to begin?

• Focus: Linux kernel
Overview

• No one-size fits all
• PM as building blocks
  – Strong base: HW features + kernel support
  – Suspend and Resume
  – managing idle
  – DVFS
  – application frameworks

• With each added block
  – improved power management
  – increasing level of effort, complexity
Baby steps: Getting started

• Static tweaking
  - disable unused features, drivers
  - timeouts
  - screen blanking
  - scale back clocks, voltage
Building a Strong Base

• Know your hardware

• HW features
  - clock hierarchy
  - voltage/current regulators
  - voltage domains, clock domains, power domains
  - platform-specific PM hardware

• Kernel internals
  - clock framework
  - NEW: voltage/current regulator framework (today @ 1:30)
Clock framework

- model HW clock tree
- track dependencies
- maintain usage counts
- disable when unused
- propagate changes

- drivers must use it !!
Suspend and Resume

- Hardware features
  - suspend state
  - retention: memory, registers
  - configurable wake-up sources
- Kernel features
  - Suspend/resume infrastructure
    - driver notifications
    - platform hooks: prepare, entry, finish
- Side Benefits
  - Fast “boot”
Managing idle time

• Customize idle loop
• minimize power when idle
• nothing to do? take a nap

• Kernel features
  – Tickless idle (a.k.a dynamic tick)
  – CPUIdle

montavista™
Different kinds of sleep

- Sleep states, idle states
  - wakeup latency
  - increase power savings

- How deep can I sleep?
- How long can I sleep?
  - use dynamic tick...
Waking up... just to press snooze

• Dynamic tick: tickless idle
  - No more periodic tick when idle
  - Only wake for next “event”
  - Sleep-when-idle can be smarter

• Tools: PowerTOP
  - who is preventing sleep
  - /proc/timer_stats
CPUidle: when to be idle

• Platform-specific “driver”:
  – defines processor idle states by
    • power consumption
    • wakeup latency
  – hooks for entering idle states

• Platform independent “governor”
  – transition decisions

• in-kernel as of 2.6.24 (x86, ACPI only)
**DVFS**

- Dynamic Voltage and Frequency Scaling (DVFS)
- Analyze available “operating points”
  - unique set of frequencies, voltages
- Kernel features
  - clock framework
  - CPUfreq
  - policies, governors
  - notification framework
CPUfreq: managing operating points

- Framework for defining, and managing operating points
- Standardized interface to applications
- Platform-specific “driver”
- Platform-independent “governors”
- Notification framework
- Existing set of open-source utilities
Available operating points

• Platform specific code
  - defines hardware operating points
  - registers them with CPUfreq
  - provides hook for setting operating point

• CPUfreq policy: which hardware OPs are “available”
  - define available set of hardware OPs
  - dynamic
Governors: when to change

• When to change OP, and to which one?
• free to pick from available OPs
  – performance: always pick highest OP
  – powersave: always pick lowest OP
  – on-demand: based on CPU utilization
    • e.g. if average CPU utilization >= 80%, increase OP
  – userspace: any user application
    • sysfs
    • cpufrequtils
Power Management QoS

• Applications know constraints, requirements
  - latency, throughput, ...
• Pass them to PM subsystem
• PM can be smarter by using constraints

• Examples:
  - wake-up latency constraints may constrain sleep depth
  - WiFi: transmitter power can be defined by latency/bandwidth requirements
  - network: packets/interrupts could be buffered to reduce power at the expense of latency
Application Frameworks

• Open Hardware Manager (OHM)
  - addresses embedded
  - small is beautiful
  - http://ohm.freedesktop.org/

• Intel PPM (Power Policy Manager)
  - Mobile & Internet Linux Project (moblin.org)
  - http://www.lesswatts.org/
The End