Taming the mess at hardware-software boundary

Lin Zhong

http://recg.org
Civilization is about building ever more complex things
Complexity by scale-out
Complexity by scale-up
Our mission

**more complex computers**

Purdue ECE

UC Irvine CS

Arizona State Arts & Media

http://www.recg.org
Argos: world first massive MIMO system prototype
Computational solution to spectrum scarcity
Omni-directional base station

Poor spatial reuse; poor power efficiency; high inter-cell interference
Massive MIMO => Data center on top of buildings
Small systems are frontiers facing mess of HW/SW boundary
Hardware/software boundary
where code meets silicon

<table>
<thead>
<tr>
<th>Code</th>
<th>Silicon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applications</td>
<td></td>
</tr>
<tr>
<td>High-level OS</td>
<td></td>
</tr>
<tr>
<td>Low-level OS</td>
<td></td>
</tr>
<tr>
<td>Firmware</td>
<td>Registers</td>
</tr>
<tr>
<td>Bootloader</td>
<td>Memory map</td>
</tr>
<tr>
<td>Device drivers</td>
<td>Instruction set</td>
</tr>
<tr>
<td>Exception handling</td>
<td>Exceptions</td>
</tr>
<tr>
<td>Memory management</td>
<td></td>
</tr>
<tr>
<td>VLSI</td>
<td></td>
</tr>
<tr>
<td>Circuits &amp; Devices</td>
<td></td>
</tr>
<tr>
<td>Physics</td>
<td></td>
</tr>
</tbody>
</table>
Why so?

• Semantic gap
• Primitive programming models
• No protection
• Not many eye balls looking at them
Computer in the eyes of a CS major
Computer in the eyes of a CE major
Moore’s Law has blurred the HW/SW boundary
Cortex M cores hidden behind hardware IPs
Software fixes for hardware bugs

- **Architectural Errata**
  - ARM, Intel

- **Silicon Errata**
  - Texas Instruments
Blurred HW/SW boundary challenges **reliability** and **security**
“Moore’s Law is dead!”
Dying Moore’s Law leads to **wider** HW/SW boundary

- Hardware accelerators for efficiency and performance
Mobile/Embedded Systems are frontier facing this widening boundary
Die photo of OMAP4430, red rectangle shows Cortex A9 cores

Widening boundary —> Explosion of drivers

- ~50% Linux kernel and >50% code of Nexus 6
- Notorious for their bugs
How to tame drivers

- Tolerate their faults
- Move them out of kernel
- Verify their correctness
- Catch their bugs
- Synthesize them
Redesign driver
(and hardware)
Runtime power management

Idle devices should sleep

- Driver tracks device usage
- Driver saves/restores device context
- Hardware transits device into low-power state
  - Clock gated, power gated
Driver tracks device usage

- Drivers call runtime framework API to keep a reference count:
  \[\text{pm\_get()}/\text{pm\_put()}\]

- Drivers provide \texttt{suspend()} / \texttt{resume()} to disable/enable devices.
Driver developers don’t seem to care

<table>
<thead>
<tr>
<th>SoC</th>
<th>Device</th>
<th>Delay (month)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TI OMAP</td>
<td>UART</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>WDT</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>USB</td>
<td>45</td>
</tr>
<tr>
<td>Samsung Exynos</td>
<td>Keypad</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>USB</td>
<td>14</td>
</tr>
<tr>
<td>Freescale i.MX</td>
<td>SD</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>I2C</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>SPI</td>
<td>29</td>
</tr>
<tr>
<td>Nvidia Tegra</td>
<td>SD*</td>
<td>56</td>
</tr>
</tbody>
</table>

*: still no PM at the time of this presentation
When they care, they don’t get it right

Example: GPIO controller driver on OMAP

Ideal, fine-grained PM

High power
Low power

GPIO example

Time

\(+\) : call `pm_get()` to increase reference count
\(-\) : call `pm_put()` to decrease reference count
When they care, they don’t get it right

Example: GPIO controller driver on OMAP

Implemented, coarse-grained PM

High power
Low power

System Initialization

GPIO example

Time

![Diagram of GPIO controller driver on OMAP]

It never sleeps!

++ : call `pm_get()` to increase reference count

--- : call `pm_put()` to decrease reference count
Drivers can be very complex

- OMAP display subsystem
  - 565 pages in manual
  - 22,000 lines of code
  - Tens of callbacks
  - Several asynchronous executions

Display controller never sleeps when screen is on
Hierarchical PM makes it worse

Bad PM for a module => Bad PM for the domain

17mW waste
The problem:

Driver does a lot of things
Business solution

Hire more driver developers!
Technical solution I

Build a tool to simplify developers’ jobs

PowerAdvisor:
  Collect execution traces about register access
  Integer programming to insert runtime_pm_get/put()
Technical solution II

If you cannot change the developers, change the system.
Systems solution II

If you cannot change the developers, change the system

Can we build the system differently so that bugs are not possible?
Take it out of driver
Reference counting ➔ external monitoring

CPU

Linux runtime PM

Driver

Device

Interconnect

suspend()
resume()

CPU

Monitor

Controller

Driver

Device

Interconnect
A centralized runtime PM architecture

- Driver provides mechanisms (PM routines)
- Central PM agent decides when to call which

```
suspend()
resume()
```
Software only solution

- Observation: *When there is a pending task, there will be frequent register accesses*

- Hypothesis: *When there are no register accesses for $T_{threshold}$, there is no pending task*
Monitor register access with memory exception

- Memory-map registers of devices
  - Supported by all ARM-based SoC

- Periodically remove access permission of mapped memory region
  - $T_{\text{threshold}}$

- Grant permission after first access
Evaluation

- Pandaboard uses TI OMAP4460
- Linaro Android 13.10 release
- Kernel version 3.2
Evaluation setup

• Tested devices
  • MMC controller (used by file system on SD card)
  • I2C, SDIO controllers (used by Wi-Fi NIC)
  • DISPC (Display controller)

• 10 minutes of usage trace
  • Reading emails with Android email application
  • Browsing webpages with Android browser
Central PM equally effective as good drivers

\[ T_{\text{threshold}} = 100 \text{ms} \]
Enable runtime PM for bad drivers

![Graph showing disabled time percentage for different devices with threshold T_{\text{threshold}}=100\text{ms}]

- MMC
- I2C
- SDIO
- DISPC

- Stock Linux Driver
- Central PM Agent
Extend standby time by 3 hours

Estimated based on smartphone user study in LiveLab: http://livelab.recg.rice.edu/traces.html
Key hypothesis revisited

When there are no register accesses for $T_{\text{threshold}}$, there is no pending task

- All I/O controllers
- Some accelerators, e.g., face detection

- Not true for programmable units (GPU & DSP)
Limitations

- Not aggressive: $T_{\text{threshold}}$ is lower bounded
- Overhead: periodic memory exceptions
- Does not work for all SoC modules
The key problem:

Software does not know if a device is idle
Small hardware modifications

- A busy/idle register bit per device
Small hardware modifications

- A busy/idle register bit per device
- Poll it to infer about pending task
Hardware modification is small

• Reusing the existing finite state machine
• A few extra gates
Hardware modification is small

• Reusing the existing finite state machine
  • A few extra gates
  • A few lines of Verilog code

```verilog
reg busyIdle;
always @(posedge clk)
/* state machine in busy */
if(|state != 0)
  busyIdle = 1'b1';
/* busy/idle reg is just read,
and state machine is in idle*/
else if (read)
  busyIdle = 1'b0;
```
Prototype on Zynq SoC
Central PM architecture with Busy/Idle Register (recap)

- Small effort to add a busy/idle register per device
- Enabling aggressive PM; Incurring less overhead
- Work for all SoC devices

![Diagram of Central PM architecture]

- CPU
  - Monitor
  - Controller
  - Driver
  - Device

- Interconnect

suspend() resume()
A centralized runtime PM architecture

All examples, source code & traces available at http://www.recg.org

Ongoing work

Can we move more out of driver?

- Driver provides mechanisms (`suspend()`, `resume()`)
- Central PM agent decides **when** to call **which**

![Diagram showing the transition from driver-centric to more central PM control]
Get `suspend()`/`resume()` out of driver!

- Driver provides mechanisms (`suspend()`/`resume()`)
- Central PM agent decides `when` to call `which`
Lessons learned from redesigning drivers

• Functions in drivers can be moved out
• A little hardware support goes a long way

Project Jenga: rethinking device drivers
Three ways to tame the mess

• Hire more EE/CS majors

• Build a tool to fix them

• Redesign so that mess is not possible
Does the end of Moore’s Law doom our mission?

More complex computers!
Good news: Computing innovation can continue with fixed hardware!!!
Bad news: OS has slowed down years ago

Lines of code (Million)

Windows
Debian Linux
Linux Kernel
Even Linux has slowed down

![Graph showing lines of code growth over years with Windows, Debian Linux, and Linux Kernel categories.]

Our goal

More complex computers!
Build a skyscraper by scaling up a cottage?
Thinking about new DESIGN

Lines of code (Million)

- Windows
- Debian Linux
- Linux Kernel

Our goal

More complex computers!