Attack Your SoC Power Challenges with Virtual Prototyping

Stefan Thiel
Gunnar Braun
Agenda

- Part #1: Power-aware Architecture Definition
- Part #2: Power-aware Software Development
- Questions
Power-aware Architecture Definition

Top three goals for today

Reduce risk of wrong design decision due to late power analysis

Define the right HW/SW architecture to meet power and performance goals

Define the right Power Management strategy
How does architecture affect power?

Hardware Architecture Questions

- Power domain per core or entire CPU?
- Best cache size and organization?
- Use coherent interconnect?
- Separate power domain per cluster?
- HW/SW Partitioning?

Accelerators

- DMA
- SATA
- PCIe
- GMAC

Multicore CPU

- L2

Cache Coherent Interconnect

- DDR Memory
- SRAM Memory
- SW Architecture and Configuration Questions

- Run SW in parallel or power down idle cores?
- How to avoid page misses?
- Run fast and stop or DVFS?
- Turn off when idle?
- Timeout for moving to low power states?
Power and Performance Duality

Platform Processing Element

Workload

Memory

Power and time

consume

utilization

use

© Accellera Systems Initiative
Power and Performance Duality

Impact of power management on power and performance?

Impact of architecture decisions on power and performance?

© Accellera Systems Initiative
Virtual Prototyping Approach

Power Overlay Model

CPU Power Domain
- CPU
  - Core 0
  - Core 1

Bus Power Domain
- Memory

Workload

PMIC, PLL

Power Manager

Load

V_{dd}, f

Triggers

Records

Energy/Power recording

© Accellera Systems Initiative
Example: DVFS What-If Analysis

- **Coarse-grain DVFS**
  - Work load
  - Frequency trace
  - Power stats

- **Fine-grain DVFS**
  - Work load
  - Frequency trace
  - Power stats

- **Total Power Stats**
  - Grouped: HARDWARE.CPU0
  - Core: PowerAnalysis
  - Core: PowerAnalysis
  - Core: PowerAnalysis
  - Core: PowerAnalysis

- **MCU Stats**
  - CPU0.task_manager
  - Utilization
  - No Slicing

- **Frequency Trace**
  - CORE0.PowerAnalysis
  - CORE1.PowerAnalysis
  - CORE2.PowerAnalysis
  - CORE3.PowerAnalysis
  - CPU0.CACHE.PowerAnalysis
  - CPU0.CACHE.PowerAnalysis

- **Task manager**
  - CPU0.task_manager
  - CPU0.task_manager

- **Powerstats**
  - Workload
  - Frequency trace
  - Power stats
What we just learned

• Reduce risk of late power analysis
• The Power and Performance duality
• Virtual Prototype for early power and performance analysis
  – Create workload model to capture processing and communication requirements
  – Create architecture model to represent processing and communication resources
  – Map workload onto architecture
  – Measure resulting system performance and power analysis
IEEE 1801 UPF System Level Power

• New IEEE 1801 Sub-committee on System Level Power

• Participation from EDA, IP providers and users
  – Active participation from AGGIOS, ARM, Broadcom, Cadence, CSR, Intel, Mentor, Microsoft, ST, Synopsys

• Goal:
  – Standardize format for system level power analysis
  – Enable exchange of power models between groups, companies and EDA tools and across abstraction levels

• Visit
  – http://standards.ieee.org/develop/wg/UPF.html
  – http://www.p1801.org/
Power-aware Architecture Definition

*Top three goals for today*

- Reduce risk of wrong design decision due to late power analysis
- Define the right HW/SW architecture to meet power and performance goals
- Define the right Power Management strategy

© Accellera Systems Initiative
Micro Server in Platform Architect MCO

Workload model: ETH processing

Ethernet Bandwidth: 512 MB/s, 1GB/s
CPU-cycles per packet: 1k (avg), 2k (high), 4k (stress)

SoC Platform
Number of CPUs: 1, 2
Number of NPUs: 0, 1
Number of SRAMs: 0, 1

Level 2 cache
Cache size in KB: 32, 64, 128

Models

Memory Map

© Accellera Systems Initiative
Micro Server in Platform Architect MCO

Power Manager
DVFS-levels: 1, 2, 3, 4,
Thresholds: when to scale up and down
Response delays: slow, med, fast
Power Model Instrumentation

```plaintext
set pm [SNPS_PAM create_monitor {Power Analysis} Cpu0Pam CPU0]

$pm set_frequency_input signal CPU0/PLL/clk_in
$pm set_voltage_input signal CPU0/PMIC/vdd_out

#define states
$pm add_state CORE0 SLEEP 5mW 1.1V 1.5GHz
$pm add_state CORE0 IDLE 70mW 1.1V 1.5GHz
$pm add_state CORE0 ACTIVE 120mW 1.1V 1.5GHz

#define state transition triggers: signals
$pm link_signal_to_state CORE0 CPU.CORE0.p_notify 0 IDLE
$pm link_signal_to_state CORE0 CPU.CORE0.p_notify 1 ACTIVE

#define state transition triggers: timeout
$pm link_timeout_to_state 2ms CORE0 SLEEP IDLE
```

![Diagram of power model states and transitions](image)

- **sleep** [mW]
- **idle** [mW]
- **active** [mW]

- **notify** idle -> active
- **notify** active -> idle
- **notify** active -> sleep
- **notify** sleep -> active

![Energy/power recording graph](image)
Power Model Instrumentation - Details

Power State Definition

```
set pm [SNPS_PAM create_monitor {Power Analysis} modem_PAM TOP.modem]

$pm add_state modem_fsm Off 0W
$pm add_state modem_fsm Standby 5mW
$pm add_state modem_fsm Transmit 200mW
$pm add_state modem_fsm Receive 150mW
```

Off 0.000W
Standby 0.005W
Transmit 0.200W
Receive 0.150W
Power Model Instrumentation - Details

Dynamic Voltage and Frequency Scaling

• For each power state we define:
  – Reference dynamic power $P_{\text{dyn,ref}}$
    • for a reference frequency $F_{\text{dyn,ref}}$ and reference voltage $V_{\text{dyn,ref}}$
  – Reference leakage power $P_{\text{leak,ref}}$
    • for a reference voltage $V_{\text{leak,ref}}$
  – Trigger frequency and voltage from Virtual Prototype

$\text{pm set_frequency_input signal TOP/MODULE_3/clk_in}$
$\text{pm set_voltage_input signal TOP/PMIC/vdd_out}$
$\text{pm add_state modem_fsm Off 0W 0W 100MHz 1.1V 1.1V}$
$\text{pm add_state modem_fsm Standby 5mW 5mW 100MHz 1.1V 1.1V}$
$\text{pm add_state modem_fsm Transmit 200mW 5mW 100MHz 1.1V 1.1V}$
$\text{pm add_state modem_fsm Receive 150mW 5mW 100MHz 1.1V 1.1V}$
Power Model Instrumentation - Details

*Driving Power States from a (HW) signal*

```
$pm \text{link\_signal\_to\_state} \ TOP/\text{MODULE\_1/pState} \ 0 \ 0xf \ \text{modem\_fsm} \ \text{Off}
$pm \text{link\_signal\_to\_state} \ TOP/\text{MODULE\_1/pState} \ 1 \ 0xf \ \text{modem\_fsm} \ \text{Standby}
$pm \text{link\_signal\_to\_state} \ TOP/\text{MODULE\_1/pState} \ 2 \ 0xf \ \text{modem\_fsm} \ \text{Transmit}
$pm \text{link\_signal\_to\_state} \ TOP/\text{MODULE\_1/pState} \ 3 \ 0xf \ \text{modem\_fsm} \ \text{Receive}
```
# Power Model Instrumentation - Details

*Driving Power States from Software*

<table>
<thead>
<tr>
<th>$pm$ link_instruction_address_to_state</th>
<th>modem_fsm</th>
<th>driver_power_off</th>
<th>Off</th>
</tr>
</thead>
<tbody>
<tr>
<td>$pm$ link_instruction_address_to_state</td>
<td>modem_fsm</td>
<td>driver_power_on</td>
<td>Standby</td>
</tr>
<tr>
<td>$pm$ link_instruction_address_to_state</td>
<td>modem_fsm</td>
<td>send</td>
<td>Transmit</td>
</tr>
<tr>
<td>$pm$ link_instruction_address_to_state</td>
<td>modem_fsm</td>
<td>receive</td>
<td>Receive</td>
</tr>
</tbody>
</table>

**SW function (e.g. Linux device driver)**

```c

driver_power_off();
driver_power_on();
driver_suspend();
receive(int &bytes);
send(int bytes, ...)
```

- **pState==0**
  - Off
- **pState==1**
  - Standby
- **pState==2**
  - Transmit
- **pState==3**
  - Receive
Video: Analyze Power Management
Parameters to Explore

- Architecture and workload parameters
  - Ethernet Bandwidth: 512 MB/s, 1GB/s
  - CPU-cycles per packet: 1k (avg), 2k (high), 4k (stress)
  - Number of CPUs: 1, 2
  - Number of NPUs: 0, 1
  - Number of SRAMs: 0, 1
  - L2 cache Size in KB: 32, 64, 128

- DVFS power management related parameters
  - DVFS-levels: 1, 2, 3, 4, 5
  - Thresholds (up/down): 2/1, 4/1, 8/1, 4/2, 8/2, 8/4
  - Response delays in us: 1, 2, 3, 4, 5

→ Total of 21600 combinations ???
→ Sensitivity Analysis + Divide and Conquer!
Sensitivity Analysis

1. Design Configuration
2. Simulation sweep
3. Metric Extraction
4. Sensitivity Analysis

Spreadsheet In

parameters

scenarios

results

parameters

metrics

Spreadsheet Out

© Accellera Systems Initiative
Video: Explore Architecture
Parameters to Explore

• Best architecture configuration:

<table>
<thead>
<tr>
<th>Workload</th>
<th>Ethernet Bandwidth:</th>
<th>512 MB/s, 1GB/s</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CPU-cycles per packet:</td>
<td>1k (avg), 2k (high), 4k (stress)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Platform</th>
<th>Number of CPUs:</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Number of NPUs:</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Number of SRAMs:</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>L2 cache</th>
<th>Size in KB:</th>
<th>64</th>
</tr>
</thead>
</table>

Results from first sweep

• DVFS power management related parameters

<table>
<thead>
<tr>
<th>DVFS-levels:</th>
<th>1, 2, 3, 4, 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thresholds (up/down):</td>
<td>2/1, 4/1, 8/1, 4/2, 8/2, 8/4</td>
</tr>
<tr>
<td>Response delays in us:</td>
<td>1, 2, 3, 4, 5</td>
</tr>
</tbody>
</table>

3 parameters
150 combinations

© Accellera Systems Initiative
Summary – Power aware Architecture Definition

Reduce risk of wrong design decision due to late power analysis

- Power and Performance duality: Performance impacts power, power (management) impacts performance
- Early power analysis important for taking the right design decisions

Define the right HW/SW architecture to meet power and performance goals

- Power aware HW/SW partitioning, cache and cache coherency analysis, interconnect/memory optimization

Define the right power management strategy

- Grouping of components into power domains
- Run-fast-then-stop vs. DVFS
- Power management thresholds, time-outs and delays
Agenda

• Part #1: Power-aware Architecture Definition

Part #2: Power-aware Software Development

• Questions
Power-aware Software Development

Top three goals for today

- Reduce risk of late power management software development
- Improve robustness of power management software
- Reveal software bugs that can drain the battery
How can software affect power?

Software controls the activity of the power consumers

**Power management – application & use-case level**

- Selection of best effort service
- Example: Turn off WiFi and use 3G when user idle
- Based on user’s performance/power needs

**Power management – operating system level (OSPM)**

- Runtime control of (sub-) system power modes
- Example: Drive WiFi subsystem into standby
- Steered by application level performance/power needs

**Power control – firmware level**

- Control clocks and voltages
- Example: set voltage regulator to 1.1V, set clock to 1GHz
- Initiated by operating system power management
Power Management complexity

Example: Mobile Application Processor

200 pages of clock programming

Specification for each register:

```c
410 static struct clksrc_clk exynos5_clk_dout_armclk = {
411     .clk = {
412         .name = "dout_armclk",
413         .parent = &exynos5_clk_mout_cpu.clk,
414     },
415     .reg_div = { .reg = EXYNOS5_CLKDIV_CPU0, .shift = 0, .size = 3 }
416 };```

Power Management complexity

Example: Mobile Application Processor

Programmer’s Manual – The devil is in the detail:

<table>
<thead>
<tr>
<th>Caution:</th>
<th>It should be guaranteed that S/W does not access IPs whose clock is gated. It may cause system failure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caution:</td>
<td>It should be guaranteed that the ratio between freq (MCLK_CDREX) and freq (ACLK_CDREX) is kept as &quot;2 to 1&quot; all the time.</td>
</tr>
</tbody>
</table>

Typical software problems that can cause days, or often weeks of debugging!

Hey Kukjin, Andrzej,
I recently started playing around with functionfs, and have noticed some strange behavior with my origen board.

If I enable the FunctionFS gadget driver, I see the board hang at boot here:

```
```

LD03 and LD08 are used for powering both device and host phy controllers. These regulators are not handled in USB host driver. Hence we get unexpected behaviour when the regulators are disabled elsewhere.

It would be best to keep these regulators always on.

Signed-off-by: Tushar Behera <tushar.behera@linaro.org>

So your patch worked great for me! Thanks for the analysis and the patch!
Fighting bugs with power impact

*Typical scenario – Here Linux Kernel Mailing List*

> It would be best to keep these regulators always on. No, it's just workaround patch.

It should be handled at USB drivers. We usually used this scheme enable USB power always. **but it consumes lots of power.**

There's no need to enable usb power when there's no usb connection.

So I suggest to enable power when usb is connected only.

In our case, micro IC detects the usb connection and enable usb power at that time.

Thank you,

Kyungmin Park

Shift-Left with Virtual Prototyping

Start earlier – Late Power Management software can have catastrophic consequences on project schedules

News:
“CHIPMAKER reportedly has been delayed in shipping power management software for its XYZ chips. OEM hasn’t yet approved any tablets featuring the CHIPMAKER processor … because the chipmaker hasn’t produced necessary power-management software....”

© Accellera Systems Initiative 2014
Virtual Prototyping approach

Goals
- Enable most critical software development tasks
- As early as possible
- Aligned with software project schedule

Needs
- Develop and deploy virtual prototypes (VP) incrementally
- Enable different software teams to develop in parallel
- Multiple VPs are created with different focus

How
- Model subsystems to support most critical software tasks
- Leverage existing or generic models for simulation of subsystems outside the software development focus

*It's here not a goal of the virtual prototype to represent all the hardware to develop all the software (availability would be too late to make any impact)*
Case Study: SoC Power Management

USB subsystem with our specific PMIC and Clock Controller
Case Study: SoC Power Management

Combine with available VDK for ARM Versatile Express and software

Focus: USB subsystem

<table>
<thead>
<tr>
<th>CPU</th>
<th>Motherboard</th>
<th>Focus: USB subsystem</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM CPUs</td>
<td>UARTs</td>
<td>Clock Controller</td>
</tr>
<tr>
<td>CCI</td>
<td>I2C</td>
<td>Voltage Controller (PMIC)</td>
</tr>
<tr>
<td>Timers</td>
<td>CLCD</td>
<td>USB CORE</td>
</tr>
<tr>
<td>GIC</td>
<td>KMI</td>
<td>USB PHY</td>
</tr>
<tr>
<td>RAMs</td>
<td>KMI</td>
<td></td>
</tr>
</tbody>
</table>

Readily available pre-assembled VP building blocks. Part of VDK for the ARM Versatile Express prototyping system. Allows running stock Linaro Linux kernel and filesystem images.

© Accellera Systems Initiative
Case Study: SoC Power Management

Add power domain information for the USB subsystem

Power domains configuration file

<table>
<thead>
<tr>
<th>MHz</th>
<th>Volt</th>
</tr>
</thead>
<tbody>
<tr>
<td>125</td>
<td>1.1</td>
</tr>
<tr>
<td>250</td>
<td>1.4</td>
</tr>
<tr>
<td>500</td>
<td>1.8</td>
</tr>
</tbody>
</table>

Vdd & Clk valid?

Output = TLM_ERROR

Output = Input

Raise assertion

Power Domain: Core

Power Domain: PHY

IN

Clk

VDD

OUT

© Accellera Systems Initiative
What we just learned

• The scope of a Virtual Prototype need to match the requirements of the software team
  – What is needed to enable the key critical software tasks?
  – A mix of generic and specific HW components
  – Assembly & modeling tools assist the specific HW model creation
  – Enables earliest availability

• A VP can accurately model power management hardware
  – Clock & voltage trees
  – Power Management IC (PMIC)
  – Clock controller
  – Power domain isolation
Power-aware Software Development

Top three goals for today

- Reduce risk of late power management software development
- Improve robustness of power management software
- Reveal software bugs that can drain the battery
Case Study: Normal OS & driver operation

Booting and using USB for a file storage gadget
Case Study: Driver faults from power bug

Bootimg and using USB for a file storage gadget

- Unpowered core
- Unpowered PHY

Abort exception! No response!

© Accellera Systems Initiative
Case Study: Root cause analysis

Using a VDK, there is more to see!

Standard Linux Kernel Messaging System

Debug message: Exception fault!
Case Study: Root cause analysis

At that time:

- Access to USB CORE
- USB CORE in Power Down State! Why?
- USB PHY is powered on Ok
- PMIC drives voltages: V4 and V6
- PMIC controlled by SW: OK
- Debug message: Exception fault!

Need to correct USB driver to driver V3 regulator!

VDD connectivity: USB CORE connected to V3 and not V4!
What we just learned

• Virtual prototypes do accurately simulate power management fault scenarios
  – Software developer can reproduce and observe same defects like on hardware
  – Deterministic repetition for debug and testing

• Virtual prototypes help accelerating root cause analysis
  – Visibility and traceability of any HW or SW property
  – Cross correlation of HW and SW power management
Power-aware Software Development

Top three goals for today

- Reduce risk of late power management software development
- Improve robustness of power management software
- Reveal software bugs that can drain the battery
## Software bugs impacting power consumption – the reality today

<table>
<thead>
<tr>
<th>Small software bugs can have big impact on power</th>
<th>Impact is use-case dependent</th>
<th>Software developers are often unaware</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Invisible to the developer</td>
<td>• Talk time might not be impacted at all</td>
<td>• No means to test and use-case analyze during development</td>
</tr>
<tr>
<td>• Only revealed in long term scenario measurements</td>
<td>• Standby time might be reduced by multiple hours</td>
<td>• Power bugs continuously slip into production firmware</td>
</tr>
</tbody>
</table>

© Accellera Systems Initiative
How to analyze power bugs?

Soldering skills required...

The hardware way...

- Use a 7-way 0.1” header for each probe (3 channels)
- Super胶ue the back to a spare region of the board
- Add 0V connection
- Add twisted pairs back to the shunt
- Remove old inductor, solder the shunt in place
- Add the other end of the wires to the shunt

The Virtualizer way...

Dynamic power analysis
Instrumentation overlay

Source: How to measure SoC power, Andy Green, TI Landing Team lead, Linaro
Dynamic power analysis

Using instrumentation scripts in Virtualizer VDK
USB Power Model

Abstract power model
• Approximate power per component at a reference frequency & voltage
• Good enough to find bug mentioned in introduction!

Detailed power model
• Approximate power for each power mode in a component
• Needed for run-time power management SW

From K Park:
> It would be best to keep these regulators always on.
No, it's just workaround patch. It should be handled at USB drivers. We usually used this scheme enable USB power always. but it consumes lots of power. There's no need to enable usb power when there's no usb connection.
Dynamic power analysis

big.LITTLE processing – Task migration with DVFS

- Frequencies
- Voltages
- Leakage Energy
- Dynamic Energy
- Power State
- State Utilization

A15 1.1V/1.359Ghz
A15 0.8V/755MHz
A7 0.8V/140Mhz

© Accellera Systems Initiative
What we just learned

• Power awareness has higher priority than power accuracy for software developers
  – Even a simple on/off power model can reveal severe defects
  – Power models can be realized at multiple levels of abstraction

• Power analysis can be added as an overly to a virtual prototype
  – Less intrusive than cutting rails and soldering shunts
  – Accuracy in the same ballpark as HW based measurement
Power-aware Software Development

Top three goals for today

Reduce risk of later power management software

• Complete software development earlier with Virtual prototypes
• Simulate power domain control (PMIC and clock controller)

Improve robustness of power management software

• Simulate fault scenarios such as unpowered hardware
• Efficient root cause analysis from HW though SW stack

Reveal software bugs that can drain the battery

• Simulate approximate power consumption
• Expose power consumption defects to the SW developer

© Accellera Systems Initiative 2014
Questions?
Thank You