Towards power efficient mixed criticals systems

OSPERT Workshop, 9th July 2013, Paris
Florian Broekaert, Laurent San, Agnes Fritsch (Thales Communications & Security)
Sergey Tverdyshev (SYSGO)
1. **Introduction**
   1. Context
   2. System integrators motivations
   3. Power management techniques overview

2. **A Low-Power scheduler implementation**
   1. Low-Power scheduling algorithm example
   2. End-User usage
   3. Implementation - A portable LowPower scheduler framework

3. **PikeOS - scheduling for mixed critical systems**
   1. PikeOS
   2. PikeOS scheduling with time partitionning

4. **Low Power scheduling perspectives for mixed critical systems**
   1. Extensions of PikeOS scheduler
   2. Integration at user VM
   3. PikeOS extensions
Context & Trends:

- Ever increasing functionalities & complexity
  → More performances required!
  → New SoC architectures are emerging:
    Multicores, GP-GPU, Manycores...

- Battery technology evolution is slower
  → Power Management becomes a must!
Power consumption impacts:

- Thermal dissipation
- Components reliability
- Cost

→ Issue for the whole system!
Introduction - TCS motivations

Objectives

- Power consumption reduction of mono & multi processors embedded systems hosting real-time applications
- Contribution to ↑ autonomy / ↓ consumption / ↓ thermal embedded products

Approach

- Safe usage of HW platforms power saving knobs
- Specification of OS services to monitor and control power consumption
- Use a modular runtime environment with minimum impact on the SW application development
- Environment enabling development of power mngt policies
- Portable design on various OS and HW platforms
CMOS circuit power consumption breakdown:

\[ \text{Power} = P = P_{\text{Static}} + P_{\text{Dynamic}} \]

Energy required for executing a Task during a time \( T \)

\[ E = P \times T \propto V^2 \]

Knowing: \( T \propto F^{-1} \)

**Static Power**

\[ P_{\text{Static}} \propto V_{dd} \times I_{\text{leak}} \]

With

\[ I_{\text{leak}} \propto (W/L) e^{(-V_{TH}/T)} \]

Evolve with CMOS technology and temperature (T)

\[ \Rightarrow \text{Gains} \propto V \]

**Dynamic Power**

\[ P_{\text{Dynamic}} \propto \alpha \times C \times V_{dd}^2 \times F \]

With

\[ \alpha : \text{switching factor} \]
\[ C : \text{Capacitance} \]
\[ V_{dd} : \text{Supply voltage} \]
\[ F : \text{Frequency} \]

\[ \Rightarrow \text{Gains} \propto V^2 \cdot F \]
**Introduction – power management techniques**

**DVFS (Dynamic Voltage and Frequency Scaling) knobs:**
- HW mechanism consisting in varying the voltage/frequency couple
- Technique enabling the reduction of the processor power consumption by providing just the necessary energy to execute a job

![Diagram](image)

**Theoretical example:** a task with workload \( W \) which execution must be finished before its deadline \( D \)

\[
P_{\text{Dynamic}} \propto \alpha \times C \times V_{dd}^2 \times F
\]

\( (V_1,F_1) \Rightarrow E_1 \)

\( E_2 < E_1 \)

\( \Rightarrow \) By reducing the \( V,F \) couple, CPU power consumption can be reduced!

**Offline strategy**
- Global frequency (for all tasks) : based on the schedulability test according to the Worst Case Execution Time (WCET) of a task set

**Online strategy**
- Local frequency (for each job) : based on the schedulability test, The Worst Case Execution Time is substituted by the actual execution time → Lot of switch

**Slow down factor:**

\[ S = \frac{f_1}{f_2} \]
DPM (Dynamic Power Management) knobs:
- Input voltage of the component is turned off
- Several low-power mode
  - Idle: data in cache are saved
  - Sleep: data are lost

**Real time system:**
Offline or online policy based on the schedulability test

**Soft system:**
Probability based estimations of inactivity time, period, duration

Data in the cache memory?
**DVFS or DPM?**

**DVFS**

- Frequency/Voltage
- Task 1: Capacity W
- Deadline
- F2 < F1, V2 < V1
- E' = E_{task1,F2,V2} + E_{switch}
- E' < E_{task1,F1,V1} < E

**DPM**

- Processor in sleep mode
- Task 1: Capacity W
- Deadline
- E'' = E_{task1,F1,V1} + E_{switch} + E_{sleep}
- E'' > E_{task1,F1,V1} (>E') < E

**Introduction – power management techniques**

F2 < F1, V2 < V1

\[ E = E_{task1,F1,V1} + E_{idle} \]

\[ E' = E_{task1,F2,V2} + E_{switch} \]

\[ E' < E_{task1,F1,V1} < E \]

\[ E'' = E_{task1,F1,V1} + E_{switch} + E_{sleep} \]

\[ E'' > E_{task1,F1,V1} (>E') < E \]
Application driven solution:
- Intrusive, not flexible, multi-application issues

OS driven:
- General purpose OS (Linux/Windows) solutions:
  - OS heuristics evaluate the processor load for a given past period and take actions if the processor reaches a specific workload threshold
  - No modifications are required for the applications to use this framework

But,
- Not suited for real-time (soft & hard) applications
- Usually disabled by system integrators to avoid system misbehaviour

→ System integrators need power management solutions compatible with real-time and critical systems! No standard solutions exist
1. Introduction
   1. Context
   2. System integrators motivations
   3. Power management techniques overview

2. A Low-Power scheduler implementation
   1. Low-Power scheduling algorithm example
   2. End-User usage
   3. Implementation - A portable LowPower scheduler framework

3. PikeOS - scheduling for mixed critical systems
   1. PikeOS
   2. PikeOS scheduling with time partitioning

4. Low Power scheduling perspectives for mixed critical systems
   1. Extensions of PikeOS scheduler
   2. Integration at user VM
   3. PikeOS extensions
Example of a Low-Power scheduling algorithm:

→ EDF (Earliest Deadline First) policy combined with online DVFS usage

- Task WCET is known/estimated for each CPU power modes
- Extend task execution upon a maximum tolerated time (deadline) and find a compliant CPU power mode
- At runtime: Slack time recuperation to take benefit of lower CPU power modes
User interfaces:

- A proposal for POSIX extension to be able to hold extra information about timings for helping resource management policy development
- POSIX limitations:
  - Priority based only
  - doesn’t take into account execution times
- Needs:
  - Concept to associate duration to concurrent flows: The thread segment
Low-Power scheduler implementation - Low Power scheduler Framework (Patent)

Application

User interfaces

User mode implementation (Library)

Standard Linux Interfaces

Linux drivers

Plugable approach

©FR2969776
Results on OMAP3530 with H.264 video codecs:

- ~30% power consumption reduction measured on OMAP3530
- Satisfying results on soft real-time applications
- Multiprocess/Multi application issues

→ Future work: Integration in Kernel space or as a service of RTOS
1. Introduction
   1. Context
   2. System integrators motivations
   3. Power management techniques overview

2. A Low-Power scheduler implementation
   1. Low-Power scheduling algorithm example
   2. End-User usage
   3. Implementation - A portable LowPower scheduler framework

3. PikeOS - scheduling for mixed critical systems
   1. PikeOS
   2. PikeOS scheduling with time partitionning

4. Low Power scheduling perspectives for mixed critical systems
   1. Extensions of PikeOS scheduler
   2. Integration at user VM
   3. PikeOS extensions
The safe and secure virtualization (SSV) RTOS:

- runs concurrently software of different safety and security levels ...
- can provide multiple API, run time environments and guest operating systems ...
- enables a mixture of hard real-time and non real-time applications ...

... on a single embedded device

- Certified technology
- Enables modular certification according to highest industrial standards
- Runs on numerous HW architectures and platforms
- Provides multi-core functionally
PikeOS Architecture

- Guest Operating System
- Guest Runtime Environment
- PikeOS Native

PikeOS System Software

- PikeOS Separation Microkernel
  - Architecture Support Package (ASP)
  - Platform Support Package (PSP)

User Mode
Kernel Mode

Hardware

Up to 62 partitions
**Challenge: Resources sharing**

**Resources**
- CPUs
- Memory, IO memory
- Flies, drivers, devices, buses...

**Safety**
- Integrity, availability
- Isolation, application errors, fail safe

**Security**
- Integrity, availability, confidentiality
- Possible side channels via shared resources
- Resources and API are attack surface

**PikeOS Solution**
- Resource Partitioning

**Challenge: Time sharing**

**Time**
- CPU cycles
- Time effects of accessing shared resources, e.g. buses...

**Safety**
- Availability, deterministic behavior, meeting deadlines
- Right balance between time- and event-triggered tasks

**Security**
- Availability, confidentiality
- Possible timing side channels via shared resources, e.g. caches, busses
- Time is the attack surface

**PikeOS Solution**
- Time Partitioning
- Static allocation of all system resources
- Application has guaranteed access to assigned resources
- Applications cannot access resources of other partitions if not explicitly configured otherwise
- No error propagation throughout other partitions
- Memory protection enforcement using Hardware (MMU)
- All partitions execute in user modes
Time partition principles

- **ARINC 653 compliant**
  - Static configuration of execution order and duration

- **Future ARINC 653 requirements**
  - Support for multiple scheduling schemes
  - Scheduling schemes can be switched during runtime

- **Partition ‘0’ offers additional functionality**
  - Threads with high priority can preempt active partition
  - Threads with low priority can act as global idle-job
1. Introduction
   1. Context
   2. System integrators motivations
   3. Power management techniques overview

2. A Low-Power scheduler implementation
   1. Low-Power scheduling algorithm example
   2. End-User usage
   3. Implementation - A portable LowPower scheduler framework

3. PikeOS - scheduling for mixed critical systems
   1. PikeOS
   2. PikeOS scheduling with time partitionning

4. Low Power scheduling perspectives for mixed critical systems
   1. Extensions of PikeOS scheduler
   2. Integration at user VM
   3. PikeOS extensions
Option 1: Integration as extensions of PikeOS scheduler

- A new parameter must be introduced for the VMs and handled by PikeOS scheduler:
  - Max power budget to be consumed for a VMs during its time partition
  - It only impacts the background time partition execution duration, and/or low priority VMs

- **Traditional ARINC653 scheduling**
- **Scheduling with background time partition**
- **Scheduling with background time partition and limited power budget**
**Option 2: Integration at level of user VM**

- **Apply DVFS/DPM only on Low Priority VMs**
  - Low-priority VMs are allocated to background time partition

- **A new parameter must be handled by PikeOS scheduler: CPU frequency**
  - This is required to ensure the right power mode is used when High priority VMs are scheduled

Traditional ARINC653 scheduling

Scheduling with background time partition

Scheduling with background time partition and DVFS management on this low-priority VMs
Option 3: Integration at the level of PikeOS extensions

- Allocate high performance cores to critical VMs
- Allocate low performance cores to low priority VMs

The system integrator must provide the number of cores and/or frequencies to be used for each partitions to ensure deterministic behavior of the application.
Thank you for your attention!