Energy Management for Tiny Real-Time Kernels

M. Bambagini, F. Prosperi, M. Marinoni, G. Buttazzo

2nd International Conference on Energy Aware Computing (ICEAC 2011)
Istanbul, Turkey, November 30 - December 2, 2011

TeCIP Institute, Scuola Superiore Sant'Anna
Area della Ricerca CNR, Via Moruzzi, 1
56127 Pisa, ITALY

Outline

Outline

Problem context

Nowadays, the energy awareness is important to:

- allow money saving (longer lifetime, cheaper cooling system)
- curb environmental pollution

Problem context

Nowadays, the energy awareness is important to:

- allow money saving (longer lifetime, cheaper cooling system)
- curb environmental pollution

In CMOS technology, the gate power consumption is:

$$P_{gate} = C_L V^2 p_s f + V I_{short} + V I_{leak} \quad (1)$$

Most important energy saving techniques:

- Dynamic Voltage and Frequency Scaling (DVFS)
- Dynamic Power Management (DPM)

Contribution

This work presents a module for managing energy consumption in tiny real-time embedded systems with limited resources

Contribution

This work presents a module for managing energy consumption in tiny real-time embedded systems with limited resources

- working with periodic independent tasks
- guaranteeing hard real-time deadlines

Contribution

This work presents a module for managing energy consumption in tiny real-time embedded systems with limited resources

- working with periodic independent tasks
- guaranteeing hard real-time deadlines
- implementing DVFS policies for the single core CPU
- implementing ad hoc solutions for the devices

Contribution

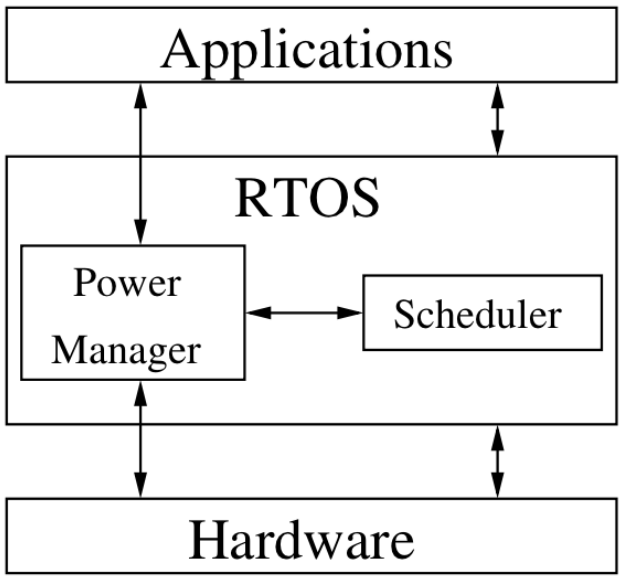
This work presents a module for managing energy consumption in tiny real-time embedded systems with limited resources

- working with periodic independent tasks
- guaranteeing hard real-time deadlines
- implementing DVFS policies for the single core CPU
- implementing ad hoc solutions for the devices
- achieving a high modularity to minimize the footprint size
- providing a uniform interface to access devices and CPU

Outline

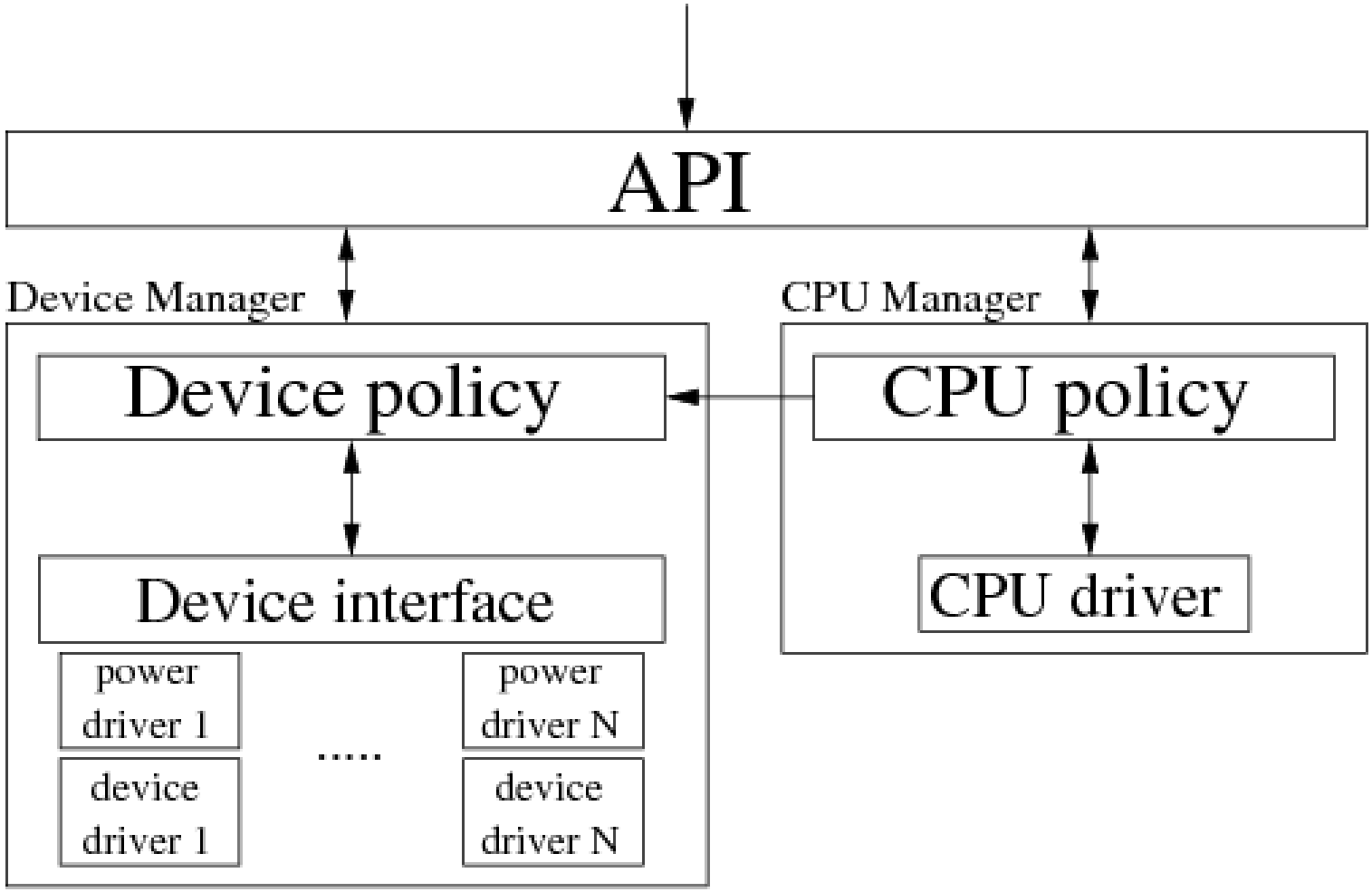
System Architecture

The *Power Manager* is a module of the Real-Time OS



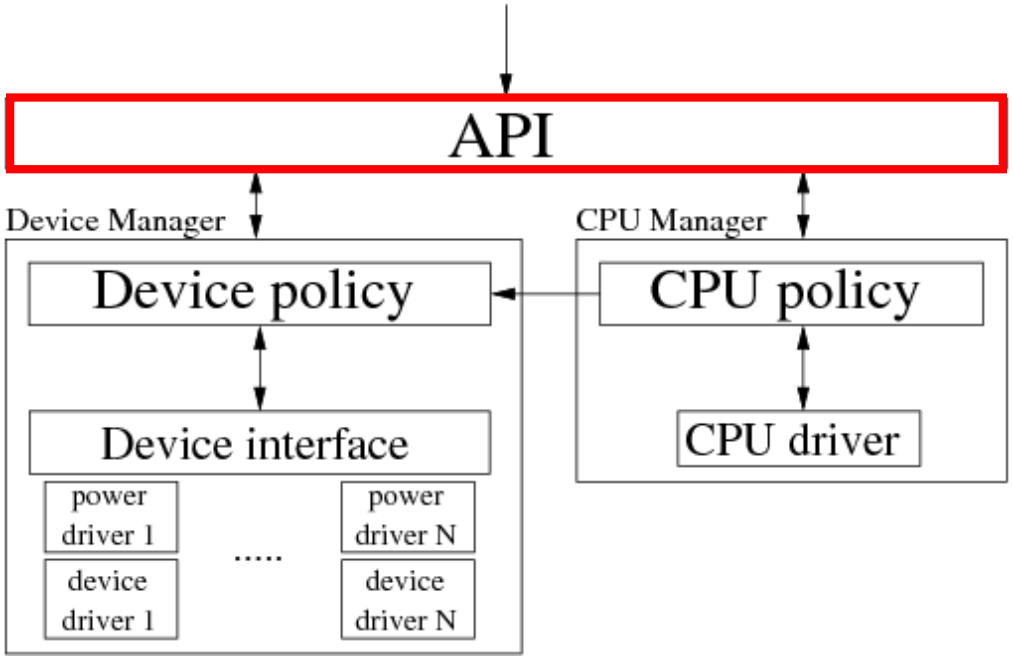
- The *Scheduler* selects independently the tasks to execute (FP or EDF)
- The *Power Manager* chooses an appropriate running configuration (i.e. speed and voltage)
- Although *Applications* and the *Scheduler* can communicate with *Power Manager*, they are independent

Module Architecture



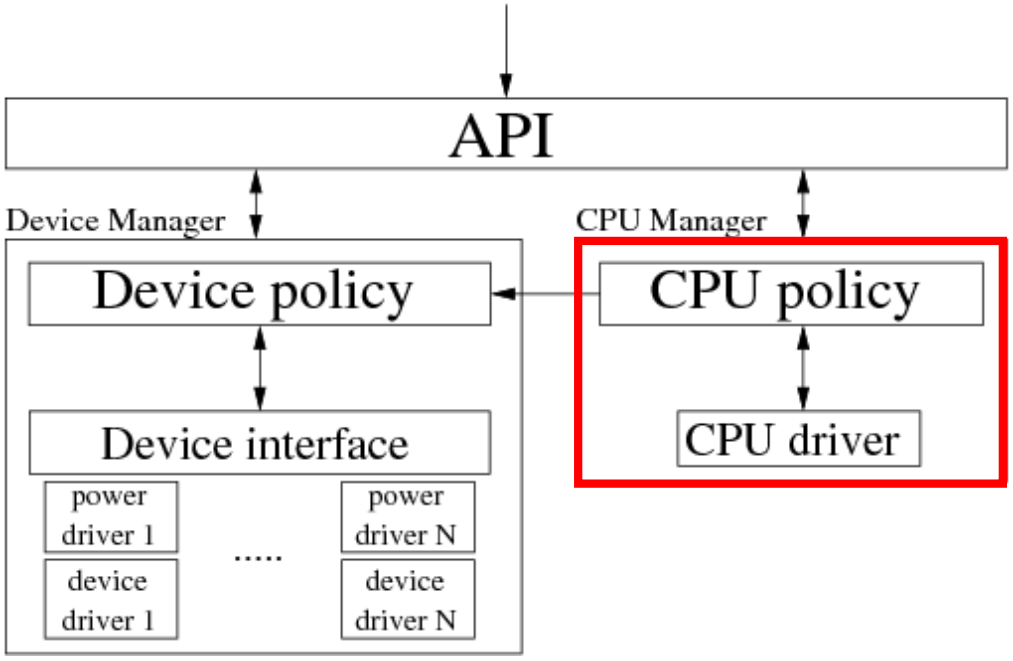
The *Power Manager* is divided in three independent modules.

Module Architecture - API



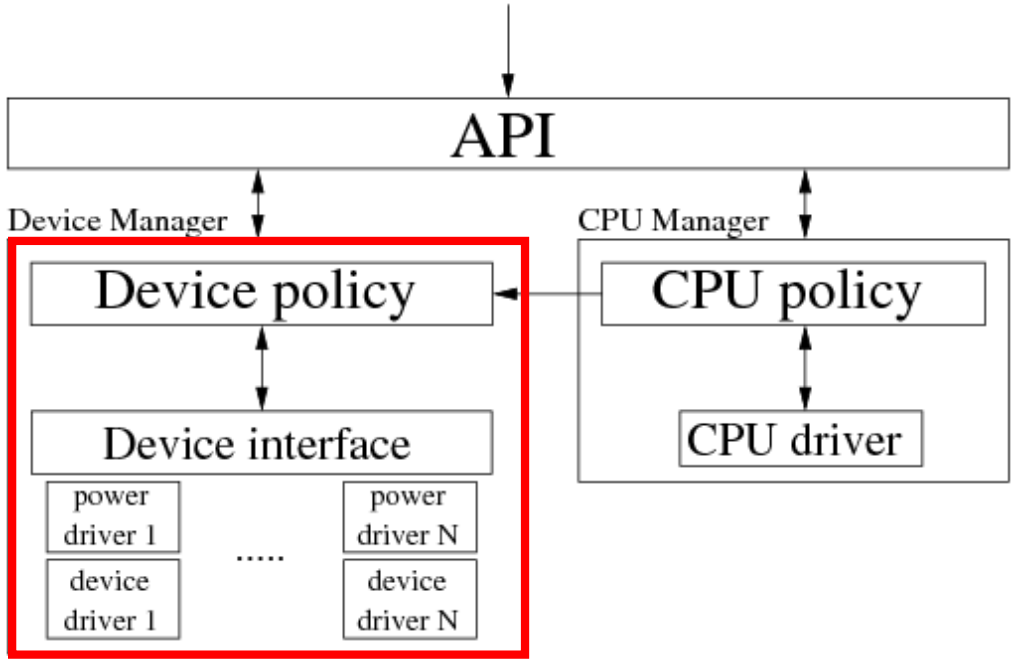
API: interface with the kernel and Applications.

Module Architecture - CPU Manager



CPU Manager: management of the CPU consumption.
CPU policy implements the energy saving policies.
CPU driver makes new configuration operative.

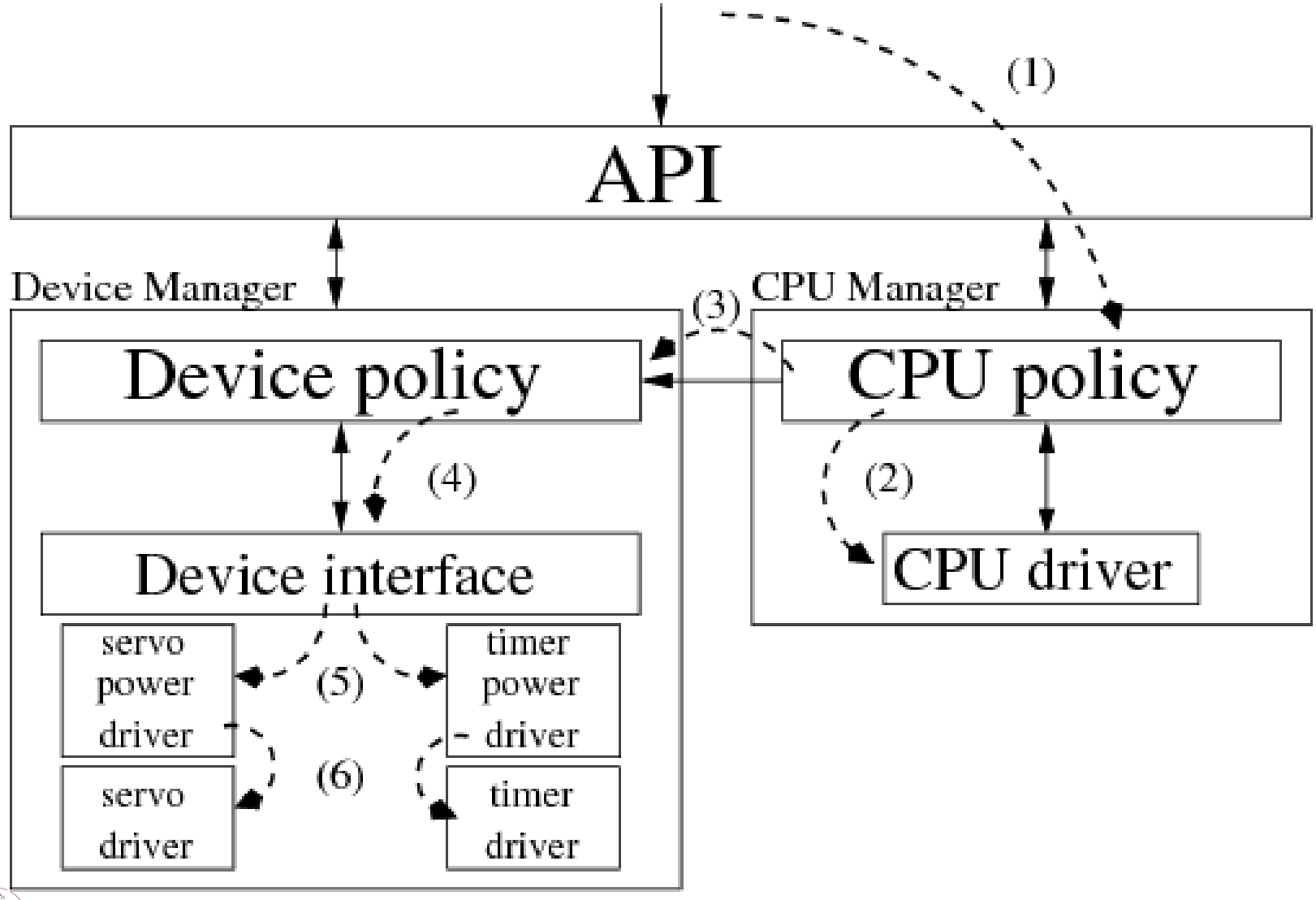
Module Architecture - Device Manager



Device Manager: management of the device consumption.
Device policy implements the device strategies.
Device Interface offers a single access point to the devices.
Power driver and *Device driver* abstract the device behavior.

Example scenario

A new task instance becomes running



CPU policies - Example scenario

Consider the following example:

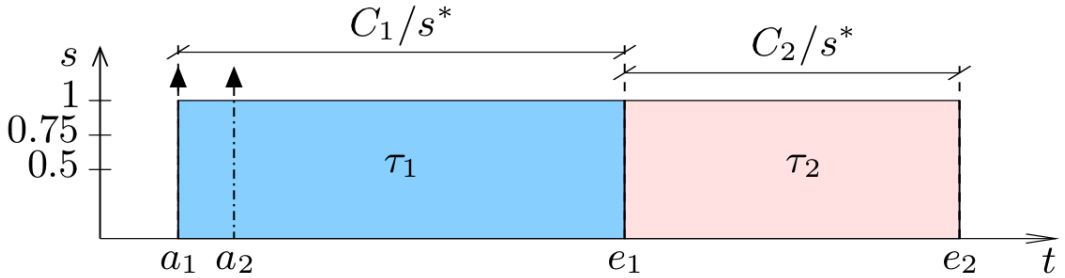
- a CPU with three speeds $S = \{0.5, 0.75, 1\}$
- two tasks: τ_1 and τ_2 (τ_2's priority $<$ τ_1's priority)
- a_i: arrival time, e_i: finishing time in the worst case at $s = 1$
- tasks feasible in the worst case at the highest speed ($s^* = 1$)

CPU policies - Example scenario

Consider the following example:

- a CPU with three speeds $S = \{0.5, 0.75, 1\}$
- two tasks: τ_1 and τ_2 (τ_2's priority $<$ τ_1's priority)
- a_i: arrival time, e_i: finishing time in the worst case at $s = 1$
- tasks feasible in the worst case at the highest speed ($s^* = 1$)

In the worst case the task execution would be:

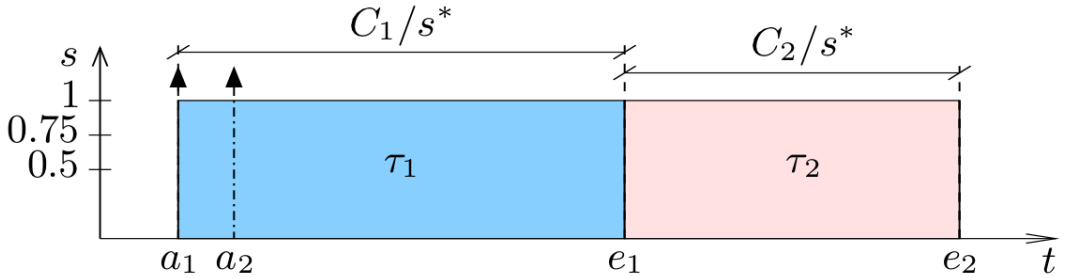


CPU policies - Example scenario

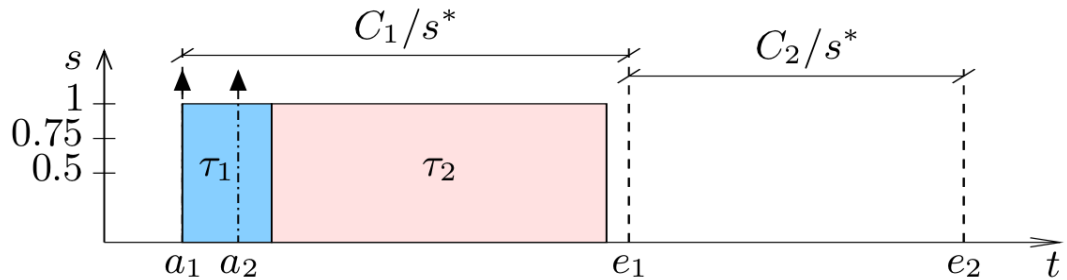
Consider the following example:

- a CPU with three speeds $S = \{0.5, 0.75, 1\}$
- two tasks: τ_1 and τ_2 (τ_2's priority $<$ τ_1's priority)
- a_i: arrival time, e_i: finishing time in the worst case at $s = 1$
- tasks feasible in the worst case at the highest speed ($s^* = 1$)

In the worst case the task execution would be:

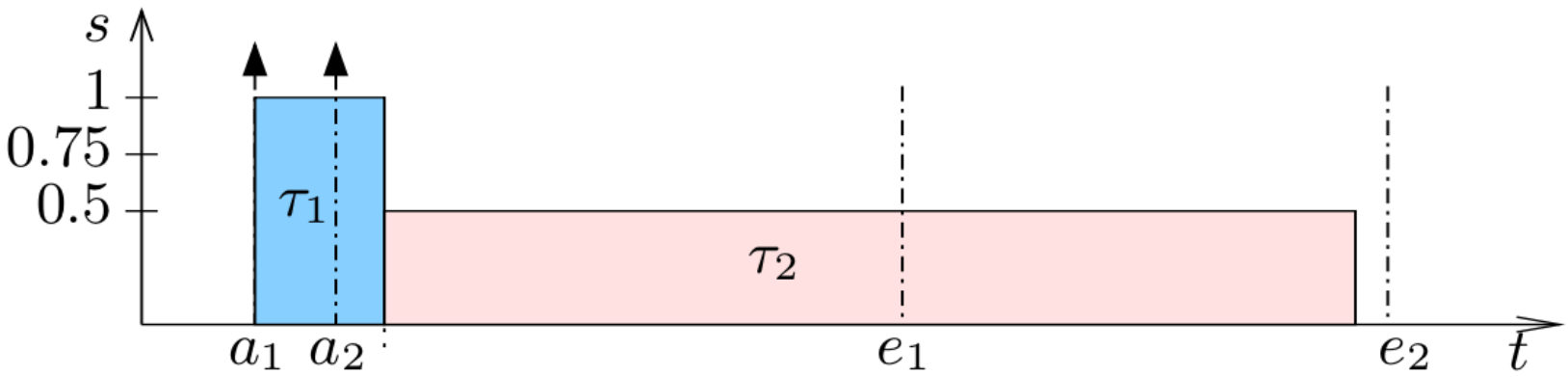


If τ_1 ends earlier than the worst case, the task execution will be:



CPU policies - OLDVS (1/3)

OLDVS [9] selects the lowest available speed to prolong a task execution not later than its worst case finishing time.

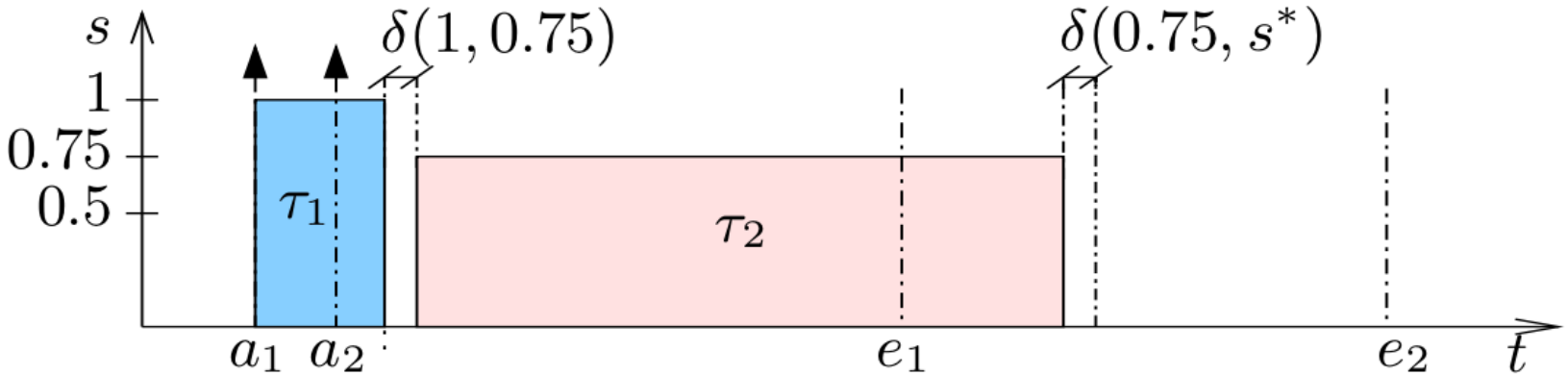


[9] Lee and Shin, "On-line Dynamic Voltage Scaling for Hard Real-Time Systems Using the EDF Algorithm"

CPU policies - BSDVFS (2/3)

BSDVFS, a variant of OLDVS proposed in this work, takes the following switching overheads into account:

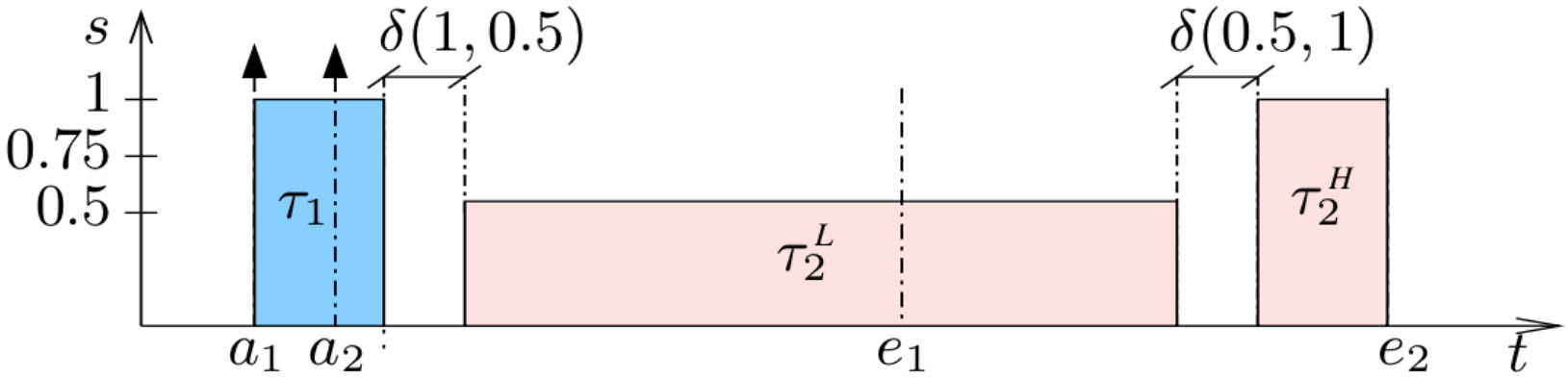
- to scale from the actual speed to the new speed
- to restore the speed s^*.



CPU policies - BSDVFS* (3/3)

BSDVFS* extends OLDVS* [10] to consider switching overheads.

It splits the remaining computation time in two parts, executed at the lower and the higher adjacent speeds computed by BSDVFS.



[10] Gong, Seong, and Lee, "On-line dynamic voltage scaling on processor with discrete frequency and voltage levels"

Device policies - Timers

Despite the negligible energy consumption, timers are taken into account to maintain the consistency of the system time.

No policy is provided and the *Power Driver* offers only two states: ON and OFF.

Device policies - Servomotors

The *Power Driver* offers *m* states, each one identified by a specific PWM period.

The implemented policy selects the best power state which produces the lowest consumption when a specific torque is required (information provided by the Application level).

Outline

CPU Results

The *Power Manager* has been developed as module of the Erika Enterprise kernel, which implements the OSEK API, and tested on the Evidence Flex boards, equipped with Microchip dsPic33 MCUs.

CPU Results

The *Power Manager* has been developed as module of the Erika Enterprise kernel, which implements the OSEK API, and tested on the Evidence Flex boards, equipped with Microchip dsPic33 MCUs.

Frequencies: 40, 35, 30, 20, 16, 10, 8 and 2 MIPS. Overheads:

- $\sim$1ms: from 2MIPS to any other
- between 4μs and 40μs: otherwise

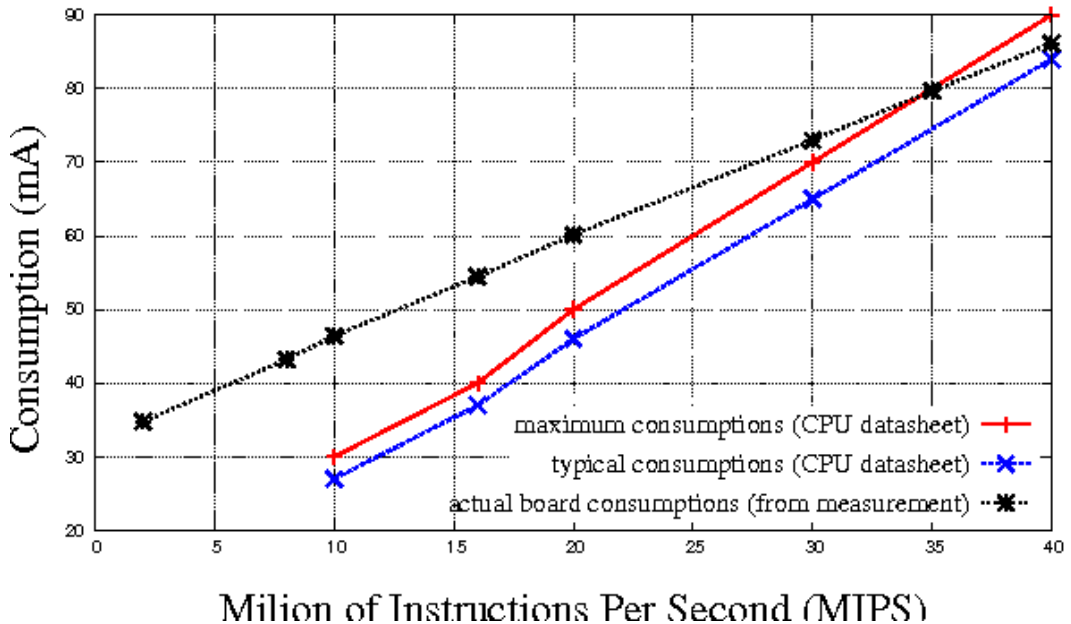
CPU Results

The *Power Manager* has been developed as module of the Erika Enterprise kernel, which implements the OSEK API, and tested on the Evidence Flex boards, equipped with Microchip dsPic33 MCUs.

Frequencies: 40, 35, 30, 20, 16, 10, 8 and 2 MIPS. Overheads:

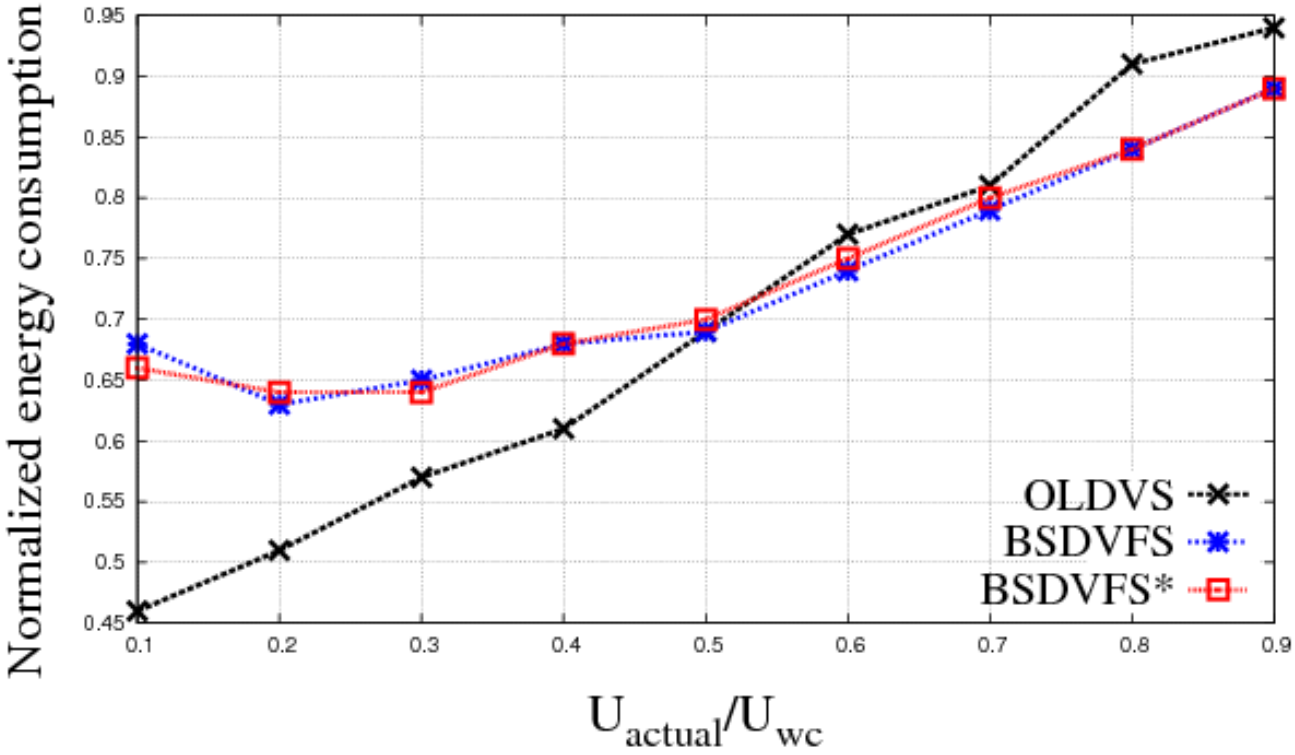
- $\sim$1ms: from 2MIPS to any other
- between 4μs and 40μs: otherwise

Board consumptions varying the frequency



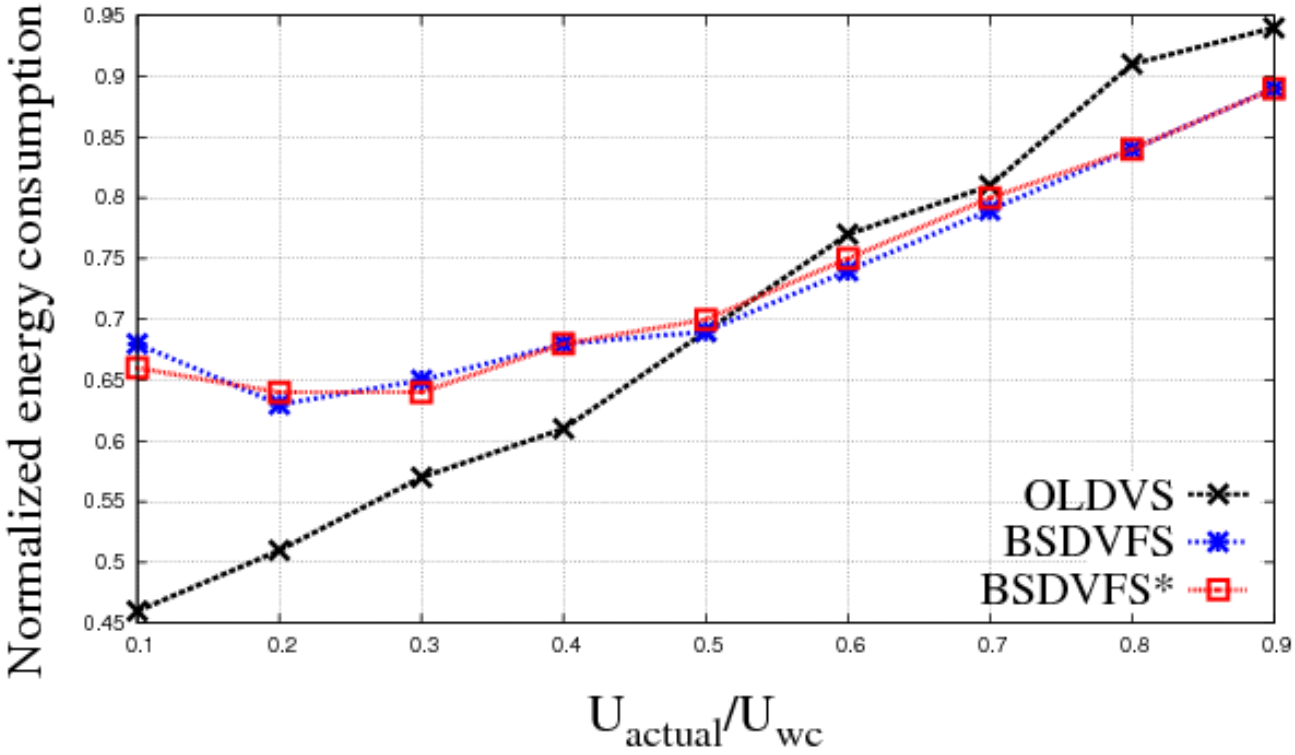
CPU Results

CPU measurements: varying the actual U (10 tasks, $U_{wc} = 0.98$).



CPU Results

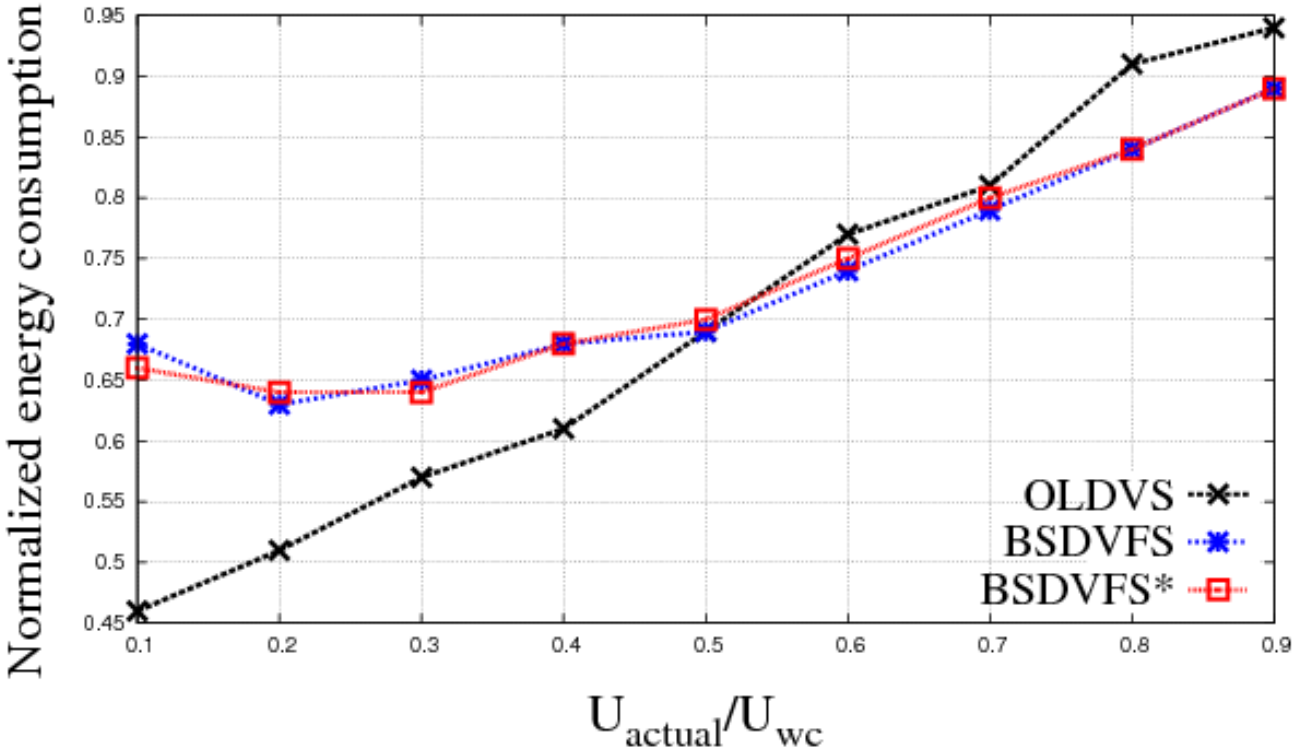
CPU measurements: varying the actual U (10 tasks, $U_{wc} = 0.98$).



BSDVFS: lowest mean speed $\Rightarrow$ BSDVFS consumes less.

CPU Results

CPU measurements: varying the actual U (10 tasks, $U_{wc} = 0.98$).

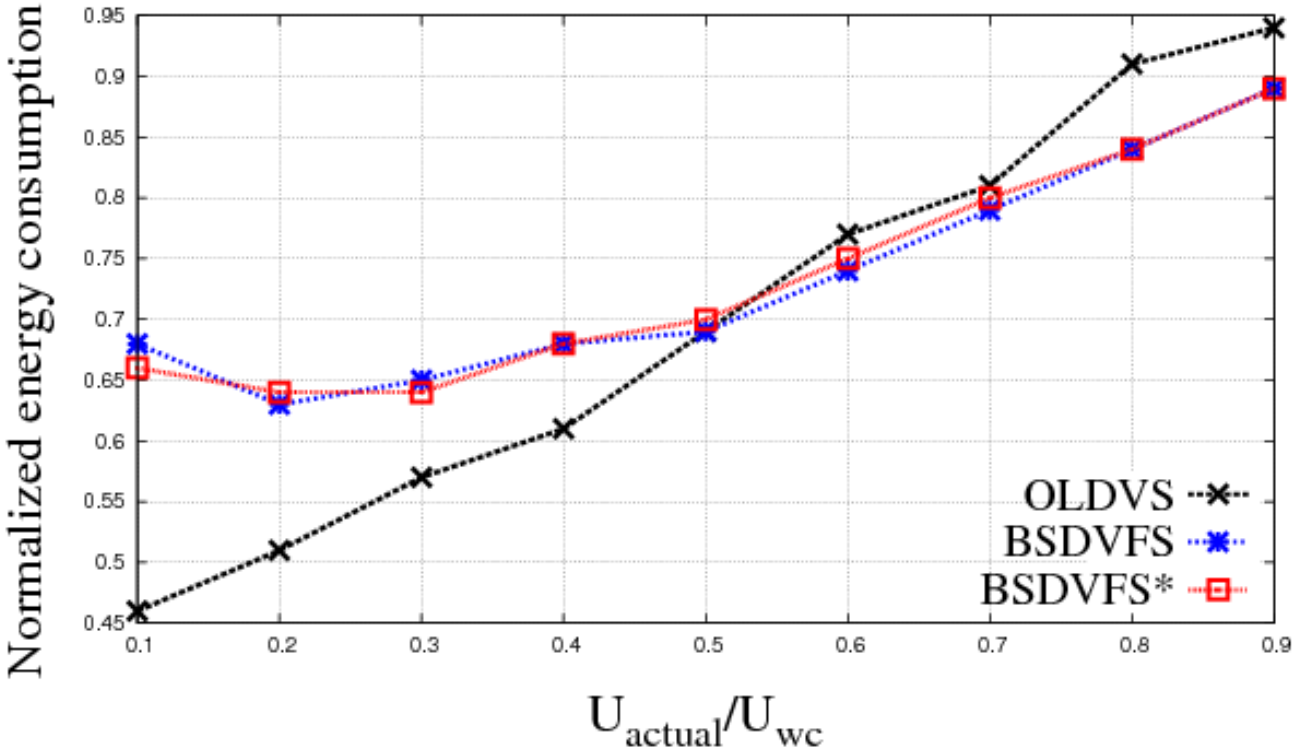


BSDVFS: lowest mean speed $\Rightarrow$ BSDVFS consumes less.

For $\frac{U_{actual}}{U_{wc}} < 0.5$: as OLDVS does not restore s*, it consumes less.

CPU Results

CPU measurements: varying the actual U (10 tasks, $U_{wc} = 0.98$).



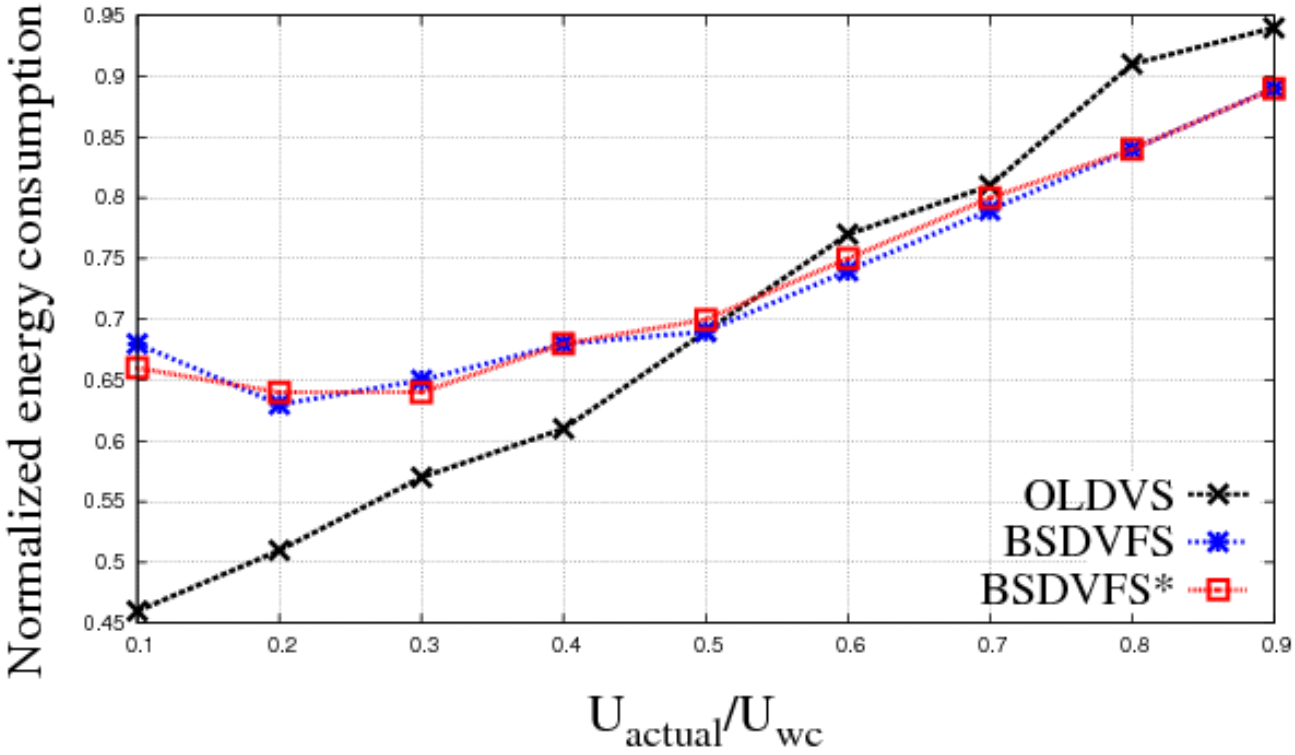
BSDVFS: lowest mean speed $\Rightarrow$ BSDVFS consumes less.

For $\frac{U_{actual}}{U_{wc}} < 0.5$: as OLDVS does not restore s*, it consumes less.

BSDVFS* does not improve the energy saving as expected.

CPU Results

CPU measurements: varying the actual U (10 tasks, $U_{wc} = 0.98$).



BSDVFS: lowest mean speed $\Rightarrow$ BSDVFS consumes less.

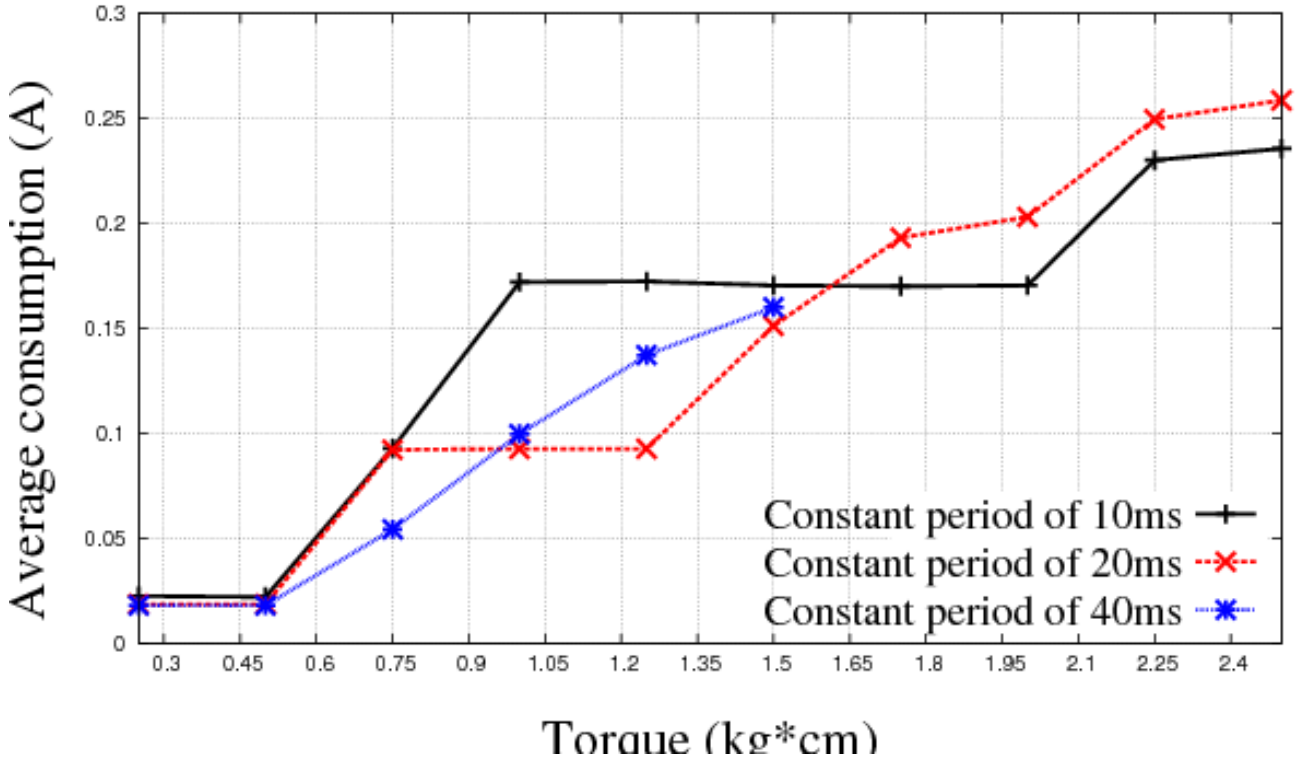
For $\frac{U_{actual}}{U_{wc}} < 0.5$: as OLDVS does not restore s*, it consumes less.

BSDVFS* does not improve the energy saving as expected.

The compiled code requires $\simeq$10K Bytes.

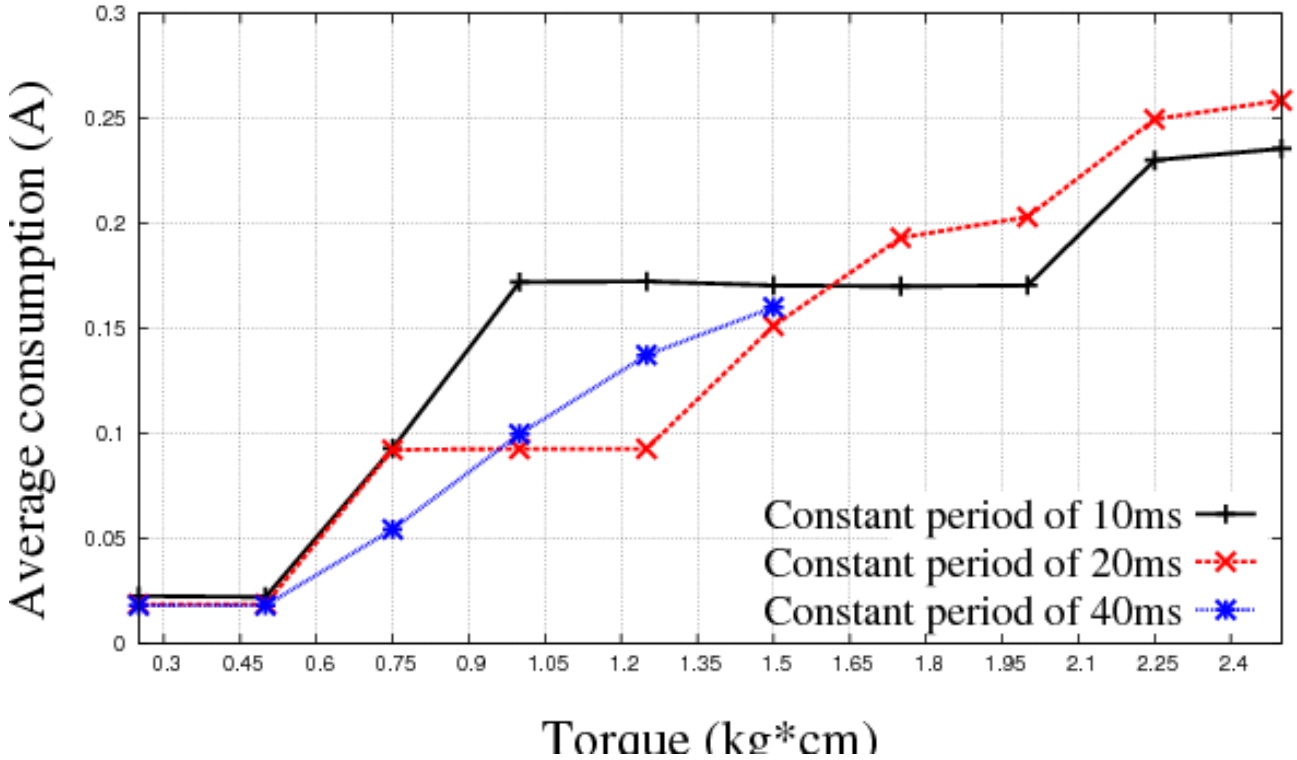
Device Results

Servomotor measurements: mean consumptions varying the applied torque with different PWM periods: 10, 20 and 40ms.



Device Results

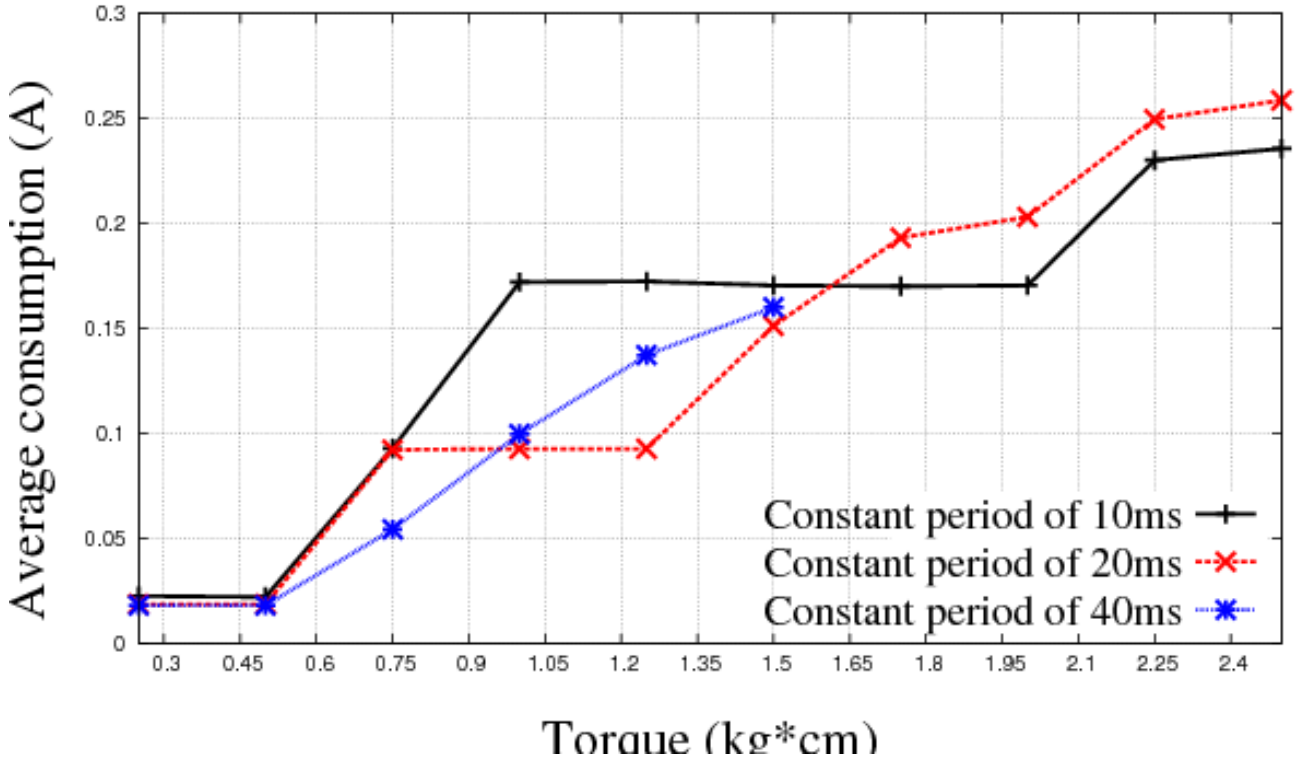
Servomotor measurements: mean consumptions varying the applied torque with different PWM periods: 10, 20 and 40ms.



Light loads $\Rightarrow$ small errors $\Rightarrow$ less frequent update is better

Device Results

Servomotor measurements: mean consumptions varying the applied torque with different PWM periods: 10, 20 and 40ms.

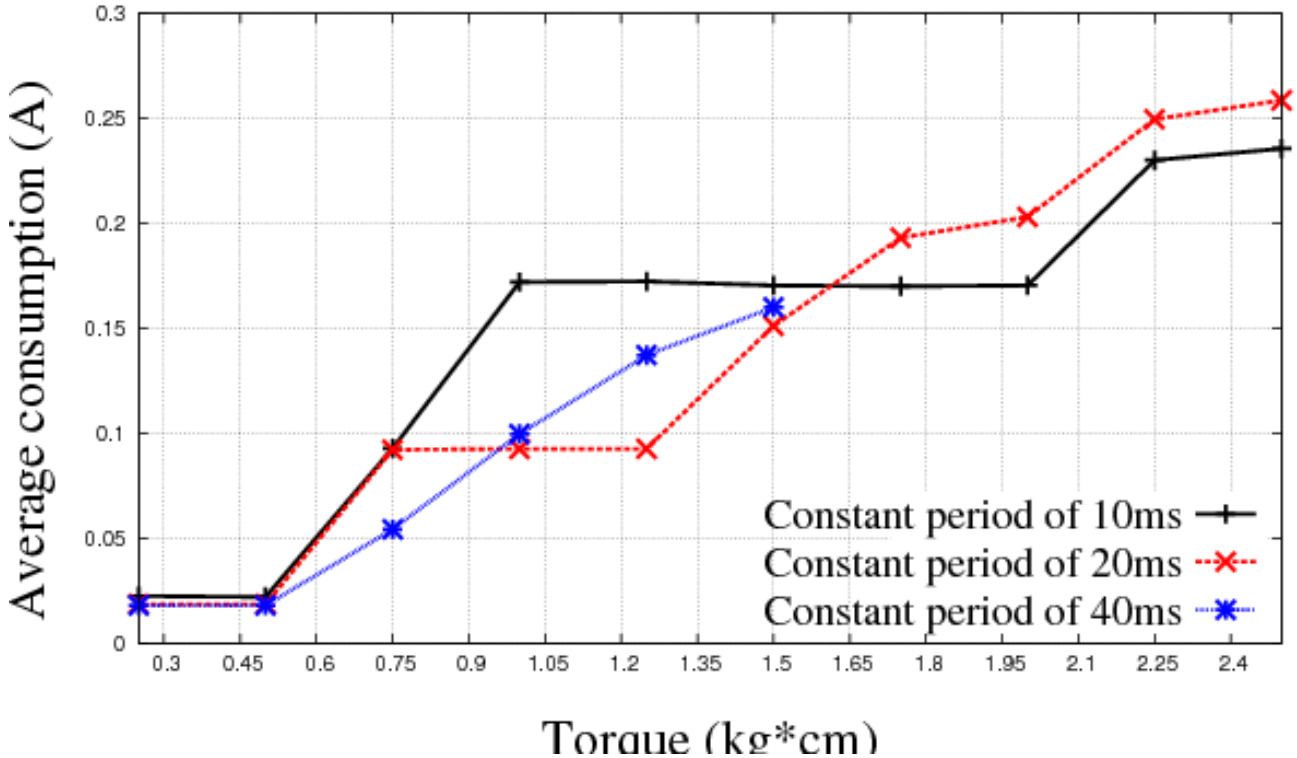


Light loads $\Rightarrow$ small errors $\Rightarrow$ less frequent update is better

Heavy loads $\Rightarrow$ high errors $\Rightarrow$ more frequent update is better

Device Results

Servomotor measurements: mean consumptions varying the applied torque with different PWM periods: 10, 20 and 40ms.



Light loads ⇒ small errors ⇒ less frequent update is better

Heavy loads ⇒ high errors ⇒ more frequent update is better

The compiled code requires <15K Bytes.

Outline

Conclusions

A *Power Manager* for real-time systems has been presented:

- designed to be as modular as possible

Results show the effectiveness of the module for the energy saving of both CPUs and servomotors hosted on an embedded system.

Conclusions

A *Power Manager* for real-time systems has been presented:

- designed to be as modular as possible

Results show the effectiveness of the module for the energy saving of both CPUs and servomotors hosted on an embedded system.

Ongoing work:

- managing radio modules

Conclusions

A *Power Manager* for real-time systems has been presented:

- designed to be as modular as possible

Results show the effectiveness of the module for the energy saving of both CPUs and servomotors hosted on an embedded system.

Ongoing work:

- managing radio modules

Future work:

- porting on different platforms (e.g. Intel Atom and ARM)
- introducing new CPU policies (e.g. DPM algorithms)
- managing other devices (e.g. DC motors and cameras)

Retis
Real-Time Systems Laboratory

thank you

m.bambagini@sssup.it