Power Consumption Optimization for Real-Time Applications

Results on a MAC application implementation

Michel Bourdellès, Julien Marechal
THALES Communications & Security, Gennevilliers, France
{michel.bourdelles} {julien.marechal}@thalesgroup.com

Abstract — **This article proposes a CPU resources control module to be included in a real-time embedded system, and using platform resources tuning capabilities such as Dynamic Voltage and Frequency Scaling (DVFS). This control module has been designed to be generic enough to be adapted to any system, but precise enough to take into account real-time constraints of such systems. The solution proposed does not need any knowledge and pre-execution of the business code of the active components of the system to be monitored. The solution is demonstrated on the Medium Access Control (MAC) of a radio protocol implemented on a Freescale IMX6 board with power gains of more than 40% with frequency scale variations from 800MHz to 400MHz.**

Key words: Power consumption monitoring, Dynamic Voltage and Frequeny Scaling, Real-Time and Embedded Systems, Radio protocol.

I. INTRODUCTION

As shown in several recent reports, the number of embedded electronic devices are increasing in such proportion that it will represent 51% of the ICT footprint in 2020 with a related CO2 footprint of 179 million tons [1], and are expected to increase in the future due to the evolution to the Internet of Things and generalization of mobile internet access. Power consumption optimization increases the autonomy of embedded electronic devices, which is critical for mobile users such as firefighters in remote areas. Power consumption optimization also corresponds to one of the top ten innovations in the next generation of wireless devices in the software defined radio related forum [2].

The new generation of CPUs integrated in current devices offers the capability to tune, from system calls, the platform resources in terms of power consumption. For example Dynamic Voltage and Frequency Scaling (DVFS) allows an user application to tune the tuple (voltage, frequency) level in predefined ranges of the CPU, to offer the capability to adapt this level according to the current application workload demands. This frequency scaling may drastically reduce the amount of power consumed. Indeed, as reminded in [3], the CPU frequency *f* and voltage *Vdd* value can each be linearly reduced on the fly to yield a cubic reduction in the dynamic ($CVdd^2 f$) power. Frequency scaling is implemented by a governor in operating systems (e.g. Linux), which monitors application workloads and scales the frequency level according to an estimate of the CPU needs at a given time [6].

Unfortunately, this kind of scaling is difficult to be applied on real-time applications, due to timing constraints not adapted for DVFS. For example (frequency, voltage) scaling latencies may result in missed critical deadlines. Generally real-time applications dealing with power consumption tuning integrate these constraints in the specification, design and implementation of these systems. To benefit from power consumption optimization through DVFS, in real-time applications, implies to mix both functional and real-time requirements with the full exploitation of the CPU power consumption tuning. This makes more complex the design of such embedded systems. This exploitation is all the more difficult as the application is often designed to be deployed on different execution platforms with different kinds of hardware CPUs.

The challenge faced is thus related to the ease-of-use of power consumption monitoring capabilities with respect of the constraints on the design, implementation and execution of real-time embedded systems.

In this paper we present an innovative solution in order to face these requirements, implemented and validated on a Medium Access Control (MAC) application of a software radio protocol used in mobile ad-hoc wireless network. The application is deployed on an ARM cortex A9 based Freescale IMX6 board [4].

Software radios tend to evolve to cognitive radios, for which the paradigm is the ability to exploit any opportunity of spectrum bandwidth. Therefore, the adaptation of the CPU load to the unpredictable request of the radio protocol processing is getting much more critical as for specific protocol standard, with static spectrum bandwidth and a better knowledge of radio resources exploitation. The new proposed process matches these specific requirements.

The rest of this article is organized as follows: in section II, we remind the requirements to be met for a full use of power consumption monitoring capabilities on real-time embedded systems. We also present the novelty of our solution with respect to the state of the art. In section III, we describe the integration and behavior of the proposed

978-1-4799-3088-3/14/$31.00 ©2014 IEEE

monitoring module with the application to be monitored. In section IV, we describe the real-time constraints related to a protocol stack part implemented in a radio equipment CPU, and focus on the MAC part of this protocol stack used as use case. In section V, we present results of the application of the monitoring module on the MAC layer part implementation. Finally, we conclude and present future work in section VI.

II. NOVELTY WITH THE STATE OF THE ART

In a first part of this section, we list the set of requirements to be met by the solutions to be easily integrated in the embedded systems development process flow. In a second stage, we discuss the current proposed solutions with respect to these requirements. We finally set our proposed solution as an introduction of section III.

A. Requirements for power management monitoring

Req1: The first requirement dealing with real-time application is the reliability. That means that power-consumption optimization does not modify the behavior of the system.

Req2: The second requirement is to consider the system to be optimized as much as possible as a black-box. This requirement is due to the will of the minimal intrusion into the business code and requirement on the most limited expertise of this code. This intrusion limitation stands for code analysis along with its execution knowledge. In particular no knowledge of execution timings of the application shall be needed, which is also related to the first requirement, as it is difficult to have an exhaustive knowledge of real size applications, when hopefully the behaviors are predictable, which is not generally the case.

Req3: Life cycle and evolution of an embedded equipment design leads to a lot of bug corrections and patches set on the initial code. Adaptation of an accurate tuning of the power optimization after each modification may go to a nightmare. One other requirement is the capabilities of post processing modification of the business code without any impact on the resource monitoring application integration.

Req4: Due to the independence of the software application design and the platform and middleware design encouraged in particular as one the software defined radio paradigm, this separation has to be taken in account in the power management monitoring proposals.

Req5: As it will be shown in section IV, real-time application deployed in CPU may integrate processes of different level of periodicity as these processes or any other active software element. These processes may be periodic which means the duration between two triggers is equal to a known value, aperiodic the triggering is not predictable or sporadic which assumes a minimal duration between two triggers. These processes may also have different level of criticality, from critical (which means the hardware resources have to be set at the expected values before the triggering factors of the corresponding critical process), to not critical (which means the power hardware resources adaptation may be set only at the reception of the triggering factors). The capability to adapt the power consumption monitoring mechanism is also a requirement.

Req6: Moreover, and as a complement of the second requirement expressed, no assumption has to be stated on the restriction of the full exploitation of the platform resources offered in the design of the business code. This includes in particular the processes creation and deletion, in addition to the activities migration from a hardware entity element to one another. This includes also the capability at application level to dynamically modify the tasks scheduling, for example in changing the order of tasks execution in the task scheduling list.

Req7: Finally, the solution proposed should be adaptable to any hardware platform with no assumption on specificities about hardware and operating system (multicore architectures, OS scheduling policies). The only need is the capability from the monitoring module to exploit timing and application driven power adaptation devices.

Req8: For portability reason on different Hardware platforms, each one has its specific power management device (for example with different latencies on the transition from one (voltage, frequency) tuple to one another. The power management module has to be easily adaptable from one platform to one another.

B. Discussion on the novelty of the approach compared to the state of the art

A wide range of studies have been provided in order to optimize the power consumption within real-time embedded systems.

Power consumption optimization by design: One of these sets of studies consists on naturally taking into account the system specification to integrate the power consumption reduction in the equipment to be designed, by identifying predefined coarse grain “power idle state” period situations. We may cite as example the 802.11s standard [5] which offers the capability to the devices to switch in awake, light sleep and deep sleep modes with respect to temporization and message exchange between peer nodes for synchronization. Although, these optimizations may highly optimize the power consumption of the system, they can’t be applied on other real-time system designs. Moreover, they are based on a coarse grain analysis and can’t identify potential gain on the exploitation of situations dynamically identified for low power application, and these systems execution might be refined to reduce much more power consumption. The solution proposed in this paper may be applied to identify and exploit these situations in addition to specific application optimization.

Generic power monitoring: at the opposite, there exist proposed solutions implemented into the operating system, without taking into account application specificities. We may mention the Linux Governor Implementation [6] which dynamically evaluates the power consumption needed and adapts the hardware resources. The main drawback of these solutions is the incapacity to manage real-time constraints.

Low-power scheduling: in order to face real-time constraints, a low power scheduling approach has been proposed in [7], taking in account real-time constraints and tuning the DVFS (voltage, frequency) CPUs assignments. This scheduling principle is based on the exploitation of "slack time" between a task's real execution time and its Worst Case Execution Time. One drawback of this method is that the knowledge of a task's execution paths has to be known by the low power scheduler, as well as the real and worst-case execution times for a particular sequence. Evaluations give good results for systematic execution applications such as video encoder/decoder.

We may state a tradeoff between the knowledge of the application and the easiness of power monitoring taking into account the real-time constraints of the application. We may also state no one of the three families of low power application processes fully matches all of the requirements.

We present in the next section a new process which takes the minimal information from the different active software elements deployed in the platform in order to tune with good accuracy real-time and embedded systems. This process matches all of the requirements expressed in section II.A.

III. Resource Monitoring Application

As an introduction of the description of the power management module, Fig. 1 presents a simplified representation of the multi layers of the deployments from the hardware platform to the final real-time application to be executed. The power management monitoring module will act as a specific active software element deployed on a specific computation unit (supposed to be a CPU), sharing the CPU load with the other application software elements to be tuned, and the operating system.

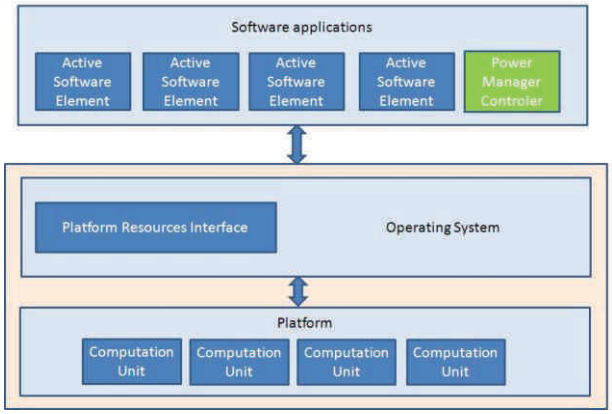

Fig. 1: Controller deployment in the system

In the following, a first section are listed all of the prerequisites of the application, and in the second one the algorithm of the application is presented.

A. Prerequisites for the power monitoring application

1) Initial static prerequisite of information provided from active software elements

Prerequisite1: The information needed to be exchanged from the active software elements and the power manager module are the following: (1) Triggering factors, (2) Criticity and periodicity of these triggering factors and (3) knowledge of the completion of the processing attached to these triggering factors.

As example of triggering factors, we may mention the information from the active software elements at the end of the triggering factors, in writing in a shared memory this information. One other example is the processing of messages received from message queues, from known periodic messages posted for some of them. The Active software elements processing might be considered as completed in the case of empty messages queues status after reaction of initial periodic message post activations.

Prerequisite2: levels of tuple (voltage, frequency) known for the high CPU load mode of the application and for the idle CPU load modes of the application.

As example, for the Freescale IMX6 board tested, the frequency, voltage tuples are (1GHz, 1.25v), (800MHz, 1.15v) and (400MHz, 0.95v).

2) Dynamic prerequisite of information provided from active software elements

Prerequisite3: Provision to the power manager module of the triggering of the factor of the active software elements along with the information of the end of the process related to this triggering.

Prerequisite4: Information on the creation / deletion of active software elements.

3) Platform specific prerequisite information.

Prerequisite5: latencies to transit from one (voltage, frequency) tuple to one another.

Prerequisite6: provision of a time service with a satisfying level of accuracy and capability to arm specific timers.

The prerequisites (information on the triggering, completion and real-time constraints of the tasks, and knowledge of the hardware platform power modules tuning application programming interface) are very general and guarantee the application of the approach on any real-time embedded systems implemented in multi-cores with DVFS monitoring capabilities.

B. Power monitoring application description

With these prerequisites, the power management module is described in the pseudo-code presented in Fig. 2.

The power monitor acts in three main cases:

- Reception of the triggering of a task. In this case the power manager triggers a timer with a value of the period minus the latency to move from a (frequency, voltage) tuple-point to one another. This

latency is platform dependent and has to be provided.

- Reception of the completion of a task. In case of the completion of all tasks the power manager sets the (frequency, voltage) tuple-point to a low value.
- Reception of the completion of one timer of the first case. In this case the power manager sets the (frequency, voltage) tuple-point to a high value.

Iterate {
While (
No ((receipt of trigger factor of actives software elements)
or (*completion of active software elements executions*)
or (receipt of trigger of critical deadline – (minus) latency time of tuple (frequency, voltage) change)
or (update of active software element))
) stay asleep.
End While

If (*receipt of trigger factor of actives software elements*)
If (critical active software element) arm for critical deadline -(minus) latency time of tuple (frequency, voltage) change)
End If
If (non critical active software element) and (low level of tuple (frequency, voltage)))
Upper the level of tuple (frequency, voltage)
End If
End If

If (*completion of active software elements executions*)
If (enough time remaining to lower the level of the tuple (frequency, voltage))
lower the level of the tuple (frequency, voltage)
End If
End If

If ((receipt of trigger of critical deadline – (minus) latency time of tuple (frequency, voltage) change) et (((low level of tuple (frequency, voltage)))
Upper the level of tuple (frequency, voltage).
End If
} **End Iterate**

Fig. 2: Power management module description

IV. USE CASE DESCRIPTION

This section has as objectives to give the main figures related to the real-time constraints of the kind of applications we use as a case-study, and more specifically the ones addressed to the Medium Access Controller (MAC) layer of this radio protocol.

A. Introduction to MANET networks

The use case chosen implements a protocol for Mobile Ad-Hoc Network (MANET) communications. MANETs are widely present on Private Mobile Radio (PMRs) and military networks configuration. One characteristic of such networks is their self-organization (creation, self-healing, and dynamic routing). In terms of functionalities the MANET radio equipment has to offer the capability to deal with a variable set of neighbors along with complex routing updates.

B. Real-time constraints on radio-protocol application

As presented in the Fig. 3, the radio protocol processing is based on slot or frame time slice for the MAC (around 5ms and 50ms), and on the authorized topology update variation for Radio Sub Network (RSN) (around 1s, not linked to the Data reception/emission).

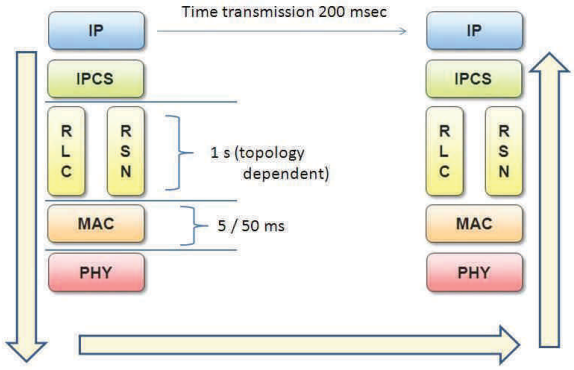

Fig. 3: Real-time constraints on the radio- protocol

Fig. 4 presents the data segmentation and emission/reception over the air, and gives indication of the constraints for a 500 kbits/sec throughput. B/S slots are service/beacon slots and T slots are data transmission/reception slots.

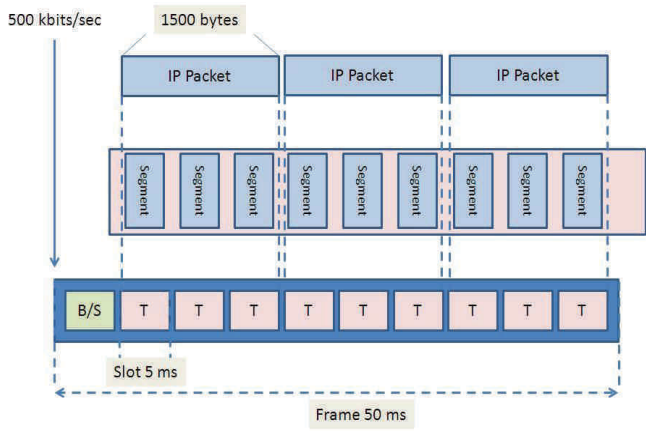

Fig. 4: Timing and throughputs constraints on the MAC

The B/S slot sequence is 9 times in reception and once in emission (in the case of 9 neighbors).

C. *Focus on the MAC layer*

Fig. 5 presents the different processing applied in the MAC layer for B/S or T Packet Data Unit in emission and/or reception.

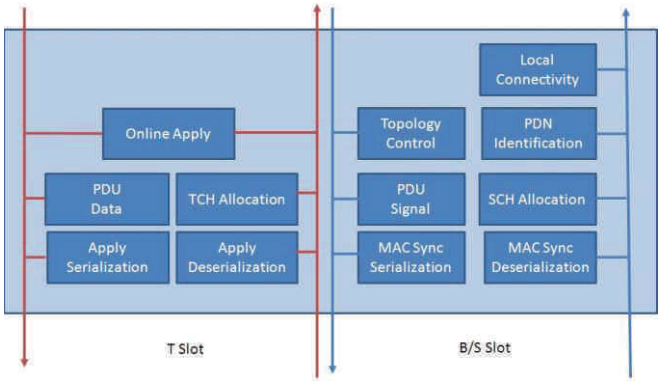

Fig. 5: Processing related to T and B/S slots transmission and reception

TABLE I
ACTIVATIONS AND DEADLINE FOR EACH OF THE PROCESSING ELEMENTS

Process part name	Activation trigger	Slot Type periodicity	deadline
PDN Identification	Reception	B/S	50ms
Topology Control	Emission	B/S	50ms
SCH Allocation	Emission/ Reception	B/S	50ms
TCH Allocation	Reception	B/S	50ms
PDU data	Emission/ Reception	T	5ms
PDU Signal	Emission/ Reception	B/S	50ms
Local connectivity	Reception	B/S	50ms
Apply Serialization	Emission	T	5ms
Apply Deserialization	Reception	T	5ms
MAC Sync Serialization	Emission	B/S	5ms
MAC Sync Deserialization	Reception	B/S	5ms
Online Apply	Emission	B/S	50ms

D. *Part of the CPU load done on the MAC.*

With respect to different protocols (kind of routing and resource allocations) , the processing part dedicated to the MAC layer may vary from 50 to 80% of the whole protocol processing, in terms of CPU load.

V. Experiments and Results

We present in this section results obtained on a Freescale Quad cores IMX6 ARM Cortex A9 based platform with as OS a Linux 3.12.0 kernel patched PREEMPT RT on light busy box file system[8][9][10].

The processing elements of the MAC are distributed on 4 threads, each one able to process the specific activities for the MAC at reception/emission of slots in B/S and T slots in the case of 5 and 50ms deadlines. All these threads are critical and periodic ones.

Fig. 6 presents the information exchanged from/to the power management with this implementation. The power Management is informed of the timers triggering of all the threads. The threads also inform the Power manager of the completion of the processing attached to these actions.

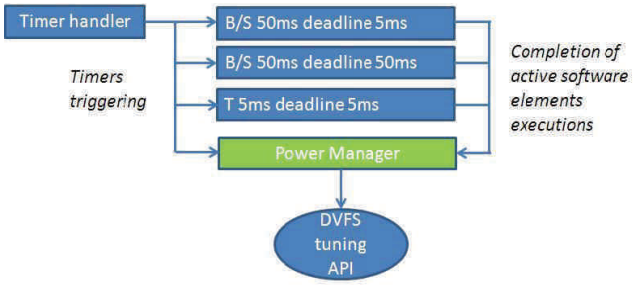

Fig. 6: Implementation details for the evaluation results

As indicated in section IV A), the processing of the MAC is sensitive to the number of the neighbors, on which depends the complexity of the processing.

The evaluation is applied with the frequency scale initialized at 800MHz. We executed the application with a number of neighbors varying from 1 to 40. Each of these computation values is extracted from samples of 200 executions. The minimal time to allow a high to low frequency decision was stated to 1ms. The code evaluated is an instrumented one in order to extract statistics.

Fig. 7 gives results of the percentage of the processing the tuple (voltage, frequency) may be lowered and the related percentage of power gain if the CPU is set to 400MHz in the situations the frequency level may be lowered. The voltage for 800MHz and 400MHz are 1.15V and 0.95V respectively. In abscissa is given the number of neighbors. The power gain is theoretically computed from these figures. The application has been computed on only one core.

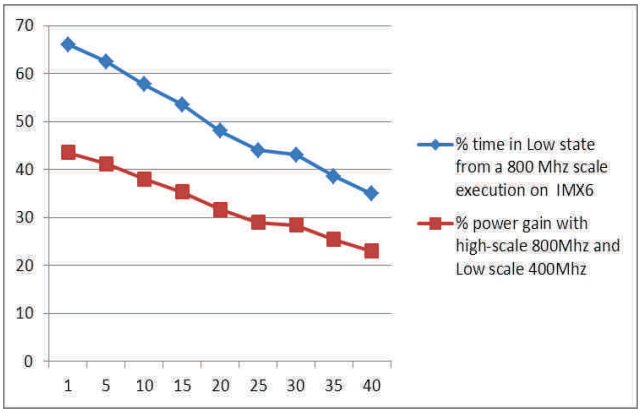

Fig. 7: Overall Low power time percentage with respect to the number of neighbors and related power consumption gain

Fig. 8 presents the visualization of parts of one execution of the application with CPU frequencies variations related to the power manager monitoring.

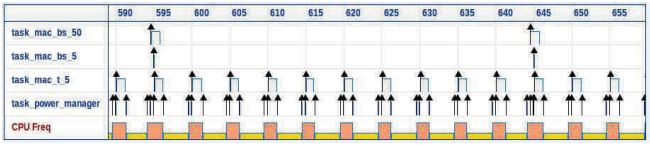

Fig. 8: Snapshot of visualization of trace results

The results show a percentage of power gain of 40% and higher for the MAC layer processing in the configuration of less than 5 radio nodes in the neighborhood. The gain of power frequency falls down to 23% with a configuration of 40 radio nodes in the neighborhood. These values consider background tasks (as other layers processing and operating systems specific tasks like file management) execution when all critical tasks are completed. In the case of idle processing mode, the gain is lower with less interest of the use of the power manager.

The overhead induced by the power manager, in terms of execution time is of 0.4% of the all application execution on an Intel Core-i5 (2.4Ghz) and 1,7% on the IMX6 ARM CortexA9 (1Ghz). The power manager itself is very light (few dozens of c code lines), with a simple application programming interface to connect the business code to the power manager, in particular to receive the information of the triggering and completion of the tasks, implemented as global variable assignments protected by semaphores, and on the criticity, periodicity and deadlines of the tasks.

VI. Conclusion and Future Work

We presented in this paper an original power manager control module that can be adapted to any kind of real-time embedded system. The proposed solution is willing to be in the right tradeoff between power consumption gains and knowledge required for fine power consumption tuning. The proposed solution is non-intrusive and does not require access to the business code of an application, nor measurements of execution times of the application. With this solution, the designer does not have to adapt the application to be compatible with the power manager control module. Furthermore, the designer may post-modify the code of the application, without any impact on the power manager control module. Experiments on a MANET MAC application, of a software radio protocol, show power consumption gains of more than 40%. The application, used in the experiment, has in a common with cognitive radio applications a variable need of CPU load with respect to knowledge of the environment.

As future plan, we expect to continue to implement our solution to other real-time embedded systems to get it as generic as possible. A work on an, as easy as possible, integration into a system design process is also planned.

VII. Acknowledgment

This work is performed in the framework of the ARTEMIS funded project PRESTO (www. presto-embedded.eu) and FP7 funded project PHARAON (pharaon.di.ens.fr). The views expressed in this document do not necessarily represent the views of the complete consortiums. The community is not liable for any use that may be made of the information contained herein.

References

[1] H.Makkar. Green Telecom Layered Framework for Calculating Carbon Footprint of Telecom Network, In IJRET: International Journal of Research in Engineering and Technology, Volume: 02 Issue: 09 | Sep-2013

[2] access groups.winnforum.org/winnforum_top_ten

[3] Tejaswini Kolpey, Antonia Zhaiz, and Sachin S. Sapatnekary. Enabling Improved Power Management in Multicore Processors Through Clustered DVFS. In DATE 2011, Grenoble.

[4] access: www .element14.com/community/community/knode/single-board_computers/sabrelite, access 12 decembre 2013

[5] 802.11s-2011 - IEEE Standard for Information Technology--Telecommunications and information exchange between systems, access:standards.ieee.org/findstds/standard/802.11s-2011.html

[6] Venkatesh Pallipadi, and Alexey Starikovskiy. The ondemand governor: past, present and future. In Proceedings of Linux Symposium, vol. 2, pp. 223-238, 2006.

[7] S. Li and F. Broekaert. Low-Power Scheduling with DVFS for common RTOS on Multicore Platforms. In *Proceedings of the 3rd Workshop on Embedded Operating Systems*, Toulouse, France, 2013.

[8] access: www .kernel.org/

[9] access: www .kernel.org/pub/linux/kernel/projects/rt/3.12/

[10] access: www .busybox.net/