A Power Manager for Deeply Embedded Systems

Geovani R. Wiedenhoft, Arliones Hoeller Jr. and Antônio A. Fröhlich
Federal University of Santa Catarina
Laboratory for Software and Hardware Integration
PO Box 476 – 88049-900 – Florianópolis, SC, Brazil
{grw,arliones,guto}@lisha.ufsc.br

Abstract

Deeply embedded systems are designed to perform a certain set of tasks, and present limitations regarding processing and memory capabilities. In many cases, these systems are powered by batteries, requiring efficient power management. In this paper, we present a dynamic power manager with no significant overhead to the application. This manager uses the power management infrastructures present in the EPOS *operating system, and is able to save power in different application scenarios.*

1. Introduction

Deeply embedded systems are platforms designed to perform a certain set of tasks, such as monitoring and controlling different environments. These systems usually present limitations in processing and memory capabilities, and in many cases are powered by batteries. Therefore, battery life is a determining factor for these system's availability.

Given these limitations, power management must be performed in an efficient manner. The hardware platforms for embedded systems allows changing operating modes in order to save power through mechanisms such as DVS (*Dynamic Voltage Scaling*) [7] and hibernation of resources [5]. Changes in operating modes must be performed with care, in order not to interfere with the execution of the application, and to not increase power consumption instead of decreasing, as changing operating modes consumes a considerable amount of power. In order to decrease processing and memory overhead, many times power management is performed by the application itself, and not the operating system. However, if a power management infra-structure exists in the operating system, it should be possible to build an active, application-independent power manager for deeply embedded systems.

This work explores dynamic power management techniques with no significant overhead for deeply embedded applications. Hoeller [4] defined and implemented a power management infra-structure for the EPOS operating system [6]. In this infra-structure every system component answers to an uniform power management API, and the relations between components are formalized through Petri Nets in order to allow automatic generation of power state migration methods, with minimal overhead. We use this infra-structure to build a dynamic power manager for deeply embedded systems. This power manager uses re-pluggable heuristics for power management, allowing configuration and adaptability to specific applications. The power management algorithm used in this work presented promising power saving results in different application scenarios.

2. EPOS Power Management Structure

EPOS (*Embedded Parallel Operating System*) [6] is a hierarchical component framework for the generation of specialized runtime support for embedded applications. Through the analysis of the applications, the EPOS system tools generate a runtime support system that aggregates only the components required by a certain application, without adding unused components. Furthermore, through the separation of system abstractions, hardware mediators and scenario aspects, EPOS allows the development of platform-independent applications.

In EPOS, every system component implements an uniform power management interface [4]. This infra-structure allows power management in different levels through the system's components hierarchy, from a high level abstraction to hardware drivers. Furthermore, it allows power management in different subsystems (e.g. processing, communication) or in the system as a whole, through a global component, who "knows" every other component instantiated in the system. The relationships between components are formalized through Petri Nets to allow automatic generation of consistent power migration methods for every component in the system, with minimal overhead.

The power management interface in EPOS is comprised

1-4244-0826-1/07/$20.00 © 2007 IEEE

by two methods: one to access the current operating mode of the component, and one to change it. This interface allows applications to perform power management with no significant overhead. With this interface, the programmer directly inserts power management statements in the application's source code. For example, if in a certain point of the application the programmer knows that the UART component will no longer be used for a period of time, he manually inserts a statement to change the power mode of the component to OFF or STANDBY . Whenever the component is accessed again, the EPOS power management infra-structure ensures the related components return to their previous active power states (FULL , LIGHT).

EPOS's power management infra-structure is implemented as an aspect, as power management is a non-functional property in the context of operating systems. As such, this infra-structure is isolated from the rest of the system, and may be activated or deactivated as needed by specific applications.

3. The EPOS Power Manager

Dynamic power management is usually performed by an agent external to the application, such as the operating system or the BIOS, which monitors the usage of certain resources. This monitor analyzes the behavior of the system, by gathering usage statistics of different components. This information is used to guide power management strategies, and usually incur in changing the operating mode of certain resources. This frees the application programmer from having to implement power management strategies in the application itself. The outcome of the power management strategy in terms of saved power is directly related to the quality of the power management algorithm used by the monitor. However, the limitations of deeply embedded systems prevent them from using complex power management algorithms.

A simple strategy for dynamic power management makes use of the scheduler in an operating system. Whenever the scheduler has no tasks to schedule, the system's main processor is put in standby mode. However, this technique is very limited, as it does not regard peripheral components, which in deeply embedded systems usually consume more power than the main microcontroller. In order to contemplate these devices, an active power manager is required. This manager checks component usage, and if a certain component is inactive for a given period of time, it changes the components power state to a lower level. One technique for detecting component usage makes use of hardware usage counters for each component. However, embedded systems hardware typically does not feature such counters, and a software-based infra-structure must be used.

This work has extended the power infra-structure in EPOS to develop a power manager for deeply embedded systems. This manager has configuration capabilities, accessed through re-pluggable heuristics that allow the modification of the power management algorithm. This modification is made available because the management algorithm is one of the main factors in the balance between power saving and processing overhead. This way, reconfigurability allows the application programmer to decide what power management algorithm his system will support, without incurring in significant interference in the application.

This section details the power management algorithm implemented in the dynamic power manager and presents its operating modes.

3.1. Power Management Algorithm

Figure 1 is an UML time diagram that represents our power management algorithm. The states of the algorithm are presented, from the testing for component idle time to the verification of idleness and then changing the component's operating mode, which in this case is the migration from on-mode to off-mode on a UART component. The state and values of the application thread, the power manager thread and the UART component are presented in this figure. In this case, the UART was used as an example of component to be used with our dynamic manager.

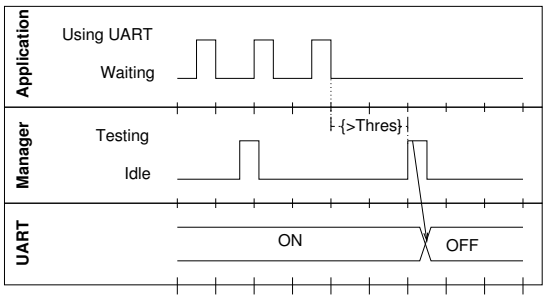

Figure 1. Power management algorithm.

In general terms, the algorithm functions in the following manner: The utilization timestamp is updated every time the component is accessed; The power manager is executed at scheduled times; A test is conducted at each power manager execution, after which the processor is yielded no matter the results; The moment the manager is executed, the current timestamp is obtained. After that, the last timestamp stored on the component is subtracted from it; The component operation mode is modified if the result surpasses a pre-defined threshold that symbolizes idleness.

3.2. Manager Operation Modes

In this work, the power manager was implemented as an aspect, to allow isolation from the system. This has enabled the development the following different power manager operation characteristics: The possibility of the application developer choosing if the power manager is enabled or not; The possibility of configuring which components application wishes the power manager to monitor, to fulfill the requirements of each specific application. This way, power management for components that the application

does not wish to be monitored is not included in the final system, even though it was implemented for every component in the system; There is the possibility of configuring the manager to be active or auxiliary in the system. In active management, the manager is executed actively in determined time intervals. In auxiliary mode, another component must initiate execution of the manager.

4. Evaluation

In this section we evaluate the dynamic power management strategy implemented in this paper, comparing it with the application-driven power management that is available from the EPOS power management structure.

4.1. Test Environment

The tests described in the previous section were performed with an ATMega16 microcontroller, from Atmel, in a STK500 development kit. To do this tests we use two applications: a deterministic one and a non-deterministic one. Figure 2 represents the applications algorithms tested in this work. These applications wait for a established time (deterministic) or random one (non-deterministic), and they write a constant value in the UART.

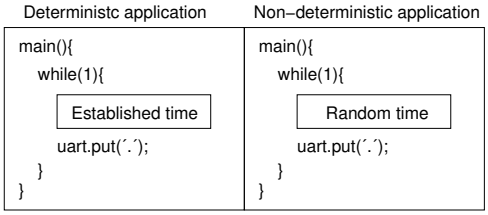

Figure 2. Applications algorithms.

These tests were only performed with the use of the UART component. The UART consumes a considerable energy to perform changes in the operating mode. However, the UART is one of the less energy consuming resources, thus generating low power savings.

In this work we performed measurements using the equipment without power management; with the application-driven power management that is available in the power management infra-structure from EPOS; and with the dynamic power management implemented in this work.

In the measurements with the dynamic power management we intercalated values for *threshold* of 10000, 5000 and 0 microseconds. In all the measurements we executed 20 iterations of 10 seconds each. The results were obtained through the arithmetic average of the remaining values excluding the 20% smallest values and the 20% largest values.

4.2. Results Analysis

In table 1 we have the results obtained in the tests performed with the deterministic application. Through these results we can infer that in a deterministic application the application-driven power management reveal a better result, saveing 4.8% of energy. Besides, we can observe in this table that the results of the dynamic power management tended to a *threshold* equal zero, that is, this technique for a deterministic application tended to an application-driven power management. These results can be explained through the programmer knowledge that knows each detail from deterministic programs.

Table 2 presents the values obtained in the tests with the non-deterministic application. These results show us that in non-deterministics applications the dynamic power management is more satisfactory, and saved 1% of energy. In this way, we can conclude that for non-deterministic applications it is desireable to wait a specific time in order to verify if the component will not be used again before to change its operating mode. This happens because the component consumes more energy if it changes the operating mode than if it stays on and idle for a specific time. Thus, the dynamic power management gets a better advantage in a non-deterministic applications. In these cases, the programmer is unaware of the specific details of what the application will execute and how long this execution will be, because it can be influenced by environment's parameters.

Analyzing both tables we can verify a reduction of memory footprint from table 1 to table 2 in the technique of dynamic power management. The main reason for this reduction is that the non-deterministic application uses *timestamps*, which requires extra components in the system.

5. Related Work

TINYOS [3] is an operating system based in events developed for embedded systems, more specifically, for Wireless Sensor Networks. TINYOS' power management is done by the task scheduler using an interface that provides beginning and end methods for components. When the task queue is empty, the scheduler stops the processor, but the peripherals stay on. The processor is turned back on when interrupt occurs. In TINYOS, more sophisticated mechanisms are left for the application to implement.

MANTIS OS (*Multimodal Networks of In-situ Sensors*) [1] is a *multithread* operating system, that has an API based on POSIX that is adapted for Wireless Sensor Networks needs. MANTIS OS' power management is similar to the UNIX `sleep` function. In order to use it, the application should enable the power management through an specific function, and thus its possible to use another function to specify the time period the application can "sleep". The scheduler is activated through a call to this last function, and if there are no threads to be scheduled, the power management puts the processor in a sleep state. This management controls only the processor and there is no specific control for the peripherals, except those provided by their drivers.

SOS [2] is an operating system for Wireless Sensor

Table 1. Results obtained through tests in a deterministic application.

Architecture	Aggregate size (Bytes)	Energy consumption (J)	Economy%
Without management	-	0.3156	-
Application-driven	548	0.3004	4.80%
Manager Thr.=10000	1488	0.3079	2.50%
Manager Thr.=5000	1488	0.3043	3.60%
Manager Thr.=0	1488	0.3008	4.70%

Table 2. Results obtained through tests in a non-deterministic application.

Architecture	Aggregate size (Bytes)	Energy consumption (J)	Economy%
Without management	-	0.3053	-
Application-driven	552	0.3042	0.40%
Manager Thr.=10000	1040	0.3034	1.00%
Manager Thr.=5000	1040	0.3014	1.30%
Manager Thr.=0	1040	0.3037	0.50%

Networks that aims to handle the dynamic reconfiguration of the message propagation services, dynamic memory allocation and dynamic module loading. SOS' power management is similar to the one found in TINYOS and MANTIS OS. When the scheduler task queue does not have messages to be scheduled, the processor operating mode is changed to a low power state, but the peripherals remain powered on. In SOS the modules can implement individual mechanisms for power management of its used resources.

The main energy treatment realized by these operating systems is changing the processor operating mode in the moment that there are no tasks, threads or messages to be scheduled. Therefore, they do not approach specific techniques for the power management of peripherals.

6. Conclusion

This paper presented the development of a low-overhead power manager for deeply embedded systems. This manager extends the power management infrastructures made available in the EPOS operating system, and is able to save power in different application scenarios. The implementation of this extension as an aspect allowed wide-configuration of the power management strategy including the possibility of choosing if the manager will be on or off, the possibility of configuring only the desired components by the application for the power management, and if the manager will be active or auxiliary in the power management. Furthermore, our power manager features re-pluggable heuristics which allow applications to select what power management strategy is the most adequate for its requirements. Our test results showed that our dynamic, operating system-driven strategy shows advantages to application-driven power management in different application scenarios.

References

[1] H. Abrach, S. Bhatti, J. Carlson, H. Dai, J. Rose, A. Sheth, B. Shucker, J. Deng, and R. Han. Mantis: System support for multimodal networks of in-situ sensors. In *2nd ACM International Workshop on Wireless Sensor Networks and Applications*, pages 50 – 59, San Diego, CA, 2003.

[2] C.-C. Han, R. Kumar, R. Shea, E. Kohler, and M. Srivastava. A dynamic operating system for sensor nodes. In *3rd international conference on Mobile systems, applications, and services*, pages 163–176, New York, NY, USA, 2005. ACM Press.

[3] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *9th international conference on architectural support for programming languages and operating systems*, pages 93–104, Cambridge, Massachusetts, United States, 2000.

[4] A. S. J. Hoeller, L. F. Wanner, and A. A. Frhlich. A Hierarchical Approach For Power Management on Mobile Embedded Systems. In *5th IFIP Working Conference on Distributed and Parallel Embedded Systems*, pages 265–274, Braga, Portugal, Oct. 2006.

[5] R. Kravets and P. Krishnan. Power management techniques for mobile communication. In *4th annual ACM/IEEE international conference on mobile computing and networking*, pages 157–168, New York, NY, USA, 1998. ACM Press.

[6] H. Marcondes, A. S. H. Junior, L. F. Wanner, and A. A. Frhlich. Operating Systems Portability: 8 bits and beyond. In *11th IEEE International Conference on Emerging Technologies and Factory Automation*, pages 124–130, Prague, Czech Republic, Sept. 2006.

[7] J. Pouwelse, K. Langendoen, and H. Sips. Dynamic voltage scaling on a low-power microprocessor. In *7th annual international conference on mobile computing and networking*, pages 251–259, New York, NY, USA, 2001. ACM Press.