Predictive Power Aware Management for Embedded Mobile Devices

Young-Si Hwang, Sung-Kwan Ku, Chan-Min Jung, Ki-Seok Chung

Dept. of Electronics and Computer & Communications Engineering, Hanyang University
Email : {ysturtle, hopebird99, vanish51, kchung}@hanyang.ac.kr

Abstract

Intelligent power management of mobile devices is getting more important as ubiquitous computing is coming true in daily life. Power aware system management relies on techniques of collecting and analyzing information on the status of I/O devices or processors while the system is running applications. However, the overhead of collecting information using software while the system is running is so huge that performance of the system may be severely deteriorated. Therefore, it is very crucial to design a PMU (power management unit) which collects information in hardware so that the performance of the system is not degraded. In this paper, we propose a novel PMU design which collects access patterns to I/O devices while an application is being executed. And a predictive power aware management is carried out based on the collected information. Experiments with various applications have been conducted to show the effectiveness of our approach.

I Introduction

One of new trends in mobile industry is the convergence of electronics, computing and communication, and it is commonly called as ubiquitous computing environment. Mobile devices are often required to have multiple functionalities. With rapid improvement of HW and SW technologies, building a very complicated mobile device is feasible. However, one of the main challenges lies in how to manage power consumption, because mobile devices should operate with limited battery charge. In perspective of system designers, minimizing power consumption in mobile devices has become the most important issue, and sometimes they sacrifice delay or area to reduce power consumption. With increasing requirement for low power techniques, research topics on how to reduce power consumption broadly cover from circuit and logic level to architecture, software, and system level techniques. Among them, system-level power management techniques have been studied hard because, to reduce power consumption, management technique is sometimes more important than low power design technique itself. Specifically, applying DPM (Dynamic Power Management) and DVFS (Dynamic Voltage & Frequency Scaling) during runtime of a system has been actively studied.

To reduce power consumption of embedded processors, hardware-based DVFS (dynamic voltage frequency scaling) techniques are widely accepted. To reduce power consumption of I/O devices, a PMU with DPM (dynamic power management) capability is often embedded into embedded processors. However, management of both DVFS and DPM heavily rely on system software like operating system. Software-based power aware management has the merit of flexible control, but it has a potential problem of suffering from significant runtime overhead to learn how to manage effectively. Therefore, comprehensive power management techniques should take into account the overall HW and SW management overhead to achieve true power reduction. In this paper, we propose an advanced PMU design which minimizes the overall runtime overhead of power aware system management by implementing logic for collecting and analyzing access patterns in hardware.

DPM is a well-known technique which tries to shut down an unused device to save power. One of the major problems of the DPM is that waking up a sleeping device may take long time. For instance, it may take several seconds to wake up a hard disk in a sleep mode. If a system cannot tolerate such a long wake-up delay, the system may not be able to shut the device down. Maintaining a device at an idle state may result in significantly more power consumption than shutting the device down. A key contribution of our PMU design is that we propose a novel hardware-based method of collecting and analyzing information on the access patterns of I/O devices when the system is running. In our proposed method, we monitor a trace of function calls until the processor initiates an access to a certain I/O device. By analyzing the pattern of PC (program counter) values, we can get good clues to make predictions on when the next I/O device accesses will happen. When we recognize a certain access pattern to an I/O device, we wake up the device in advance before a real I/O access request takes place. By this predictive management, we can reduce the performance penalty due to long wake-up time. Since we rely on HW modules for pattern monitoring and detection, our method does not cause any significant runtime overhead unlike other software-based management schemes.

"This research was supported by the MIC(Ministry of Information and Communication), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA(Institute of Information Technology Advancement)" (IITA-2007-(C1090-0701-0045))

978-1-4244-1922-7/08/$25.00 ©2008 IEEE

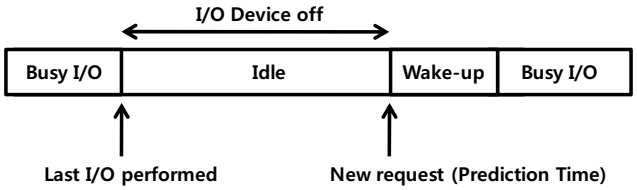

Figure 1 An example of I/O device access pattern

TABLE I
Power consumption of a harddisk for each state[1]

State	Power
Busy power	2.6W
Idle power	1.3W
Standby power	0.25W
Sleep power	0.10W

The rest of the paper is organized as follows. In Section 2, we will address background discussion on I/O device controls, and we will introduce existing I/O device power management techniques. In Section 3, we will explain how to apply a predictive DPM using our PMU on an embedded mobile system. In Section 4, we will explain how we implement our PMU. In Section 5, we will present experimental results. We conclude this paper and present our future work in the last section.

II. Background and Motivation

Power aware design methodology has emerged as one of the most crucial design problems. Active studies are being done to reduce power consumption of I/O devices by power aware system management. The goal of such studies is to shut down an unused device as soon as possible if the device is not going to be used any time soon. The difficulty in doing this lies in the fact that nobody can predict the future perfectly. On average it takes several seconds to switch a hard disk in a sleep mode to an active state. Due to its unacceptable long delay, in general, devices may have to stay in an idle state quite long before it goes into a sleep mode. However, as you can see from Table 1, a hard disk consumes fairly large amount of power when it stays in an idle state. To reduce power consumption, most modern hard disks have at least four different operation modes (busy, idle, standby and sleep) shown in Table 1. However, to reduce power consumption more aggressively, what is desirable is to make the hard disk stay only in either the busy or the sleep state. The intermediate two idle and standby states are added as a result of design trade-offs between power saving and wake-up delay. In this paper, we claim that by having a

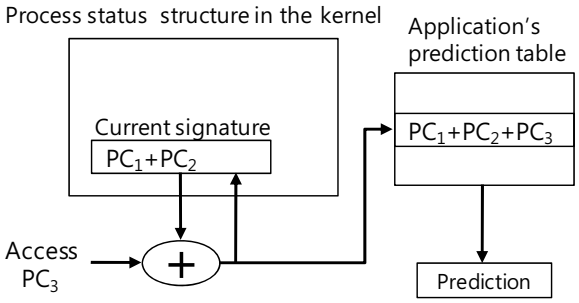

Figure 2 Structure of PCAP[1]

predictive power management capability, we can shut down the device as soon as the device becomes idle. And by predicting the next re-activation time in advance, we do not have to suffer from long wake up delay. Furthermore, instead of switching from an idle state to a standby-state or switching from a standby state to a sleep state by a simple timer based algorithm, a more intelligent mechanism for switching states is achieved by our effective predictive algorithm.

To manage I/O accesses intelligently, accurate prediction capability is really crucial. There have been several different approaches on predictive power management. A prediction method based on timeout [2] is straightforward. LT (learning tree) [3] tries to classify the I/O accesses into roughly two categories: long active states followed by short idle states, and short active states followed by long idle states. These mechanisms are not dynamic in the sense that off-line profiling will be used to obtain data for predictions. PCAP [1] is a PC (program counter) based dynamic prediction scheme. PCAP relies on a branch prediction mechanism which is commonly employed in processor designs. PCAP is based on the observation that the OS kernel executes a branch instruction to a device driver for a specific device when the OS starts accessing the device. Such a branch is stored in a history buffer, and the stored branching history will be used to predict the upcoming I/O access for the device.

Figure 2 shows an example of how a prediction table included in the OS kernel is used to store branching history and how the stored information is used to predict the future branches in PCAP. Our approach is similar to PCAP in the sense that our approach is also a PC-based prediction scheme. However, our goal is to minimize runtime overhead as much as possible. The main focus of existing studies like timer-based prediction or learning tree is how to execute a set of special-purpose codes to minimize power consumption in the OS kernel. However, this will result in a significant runtime overhead. In our approach, I/O access pattern monitoring and recognition are implemented in HW so that runtime overhead is minimized. Also unlike PCAP where we have to maintain a large history table for branch prediction, we detect only the start of a certain I/O device access based on a combination of I/O access control signal and the PC values. Therefore, we do not need any large history buffer.

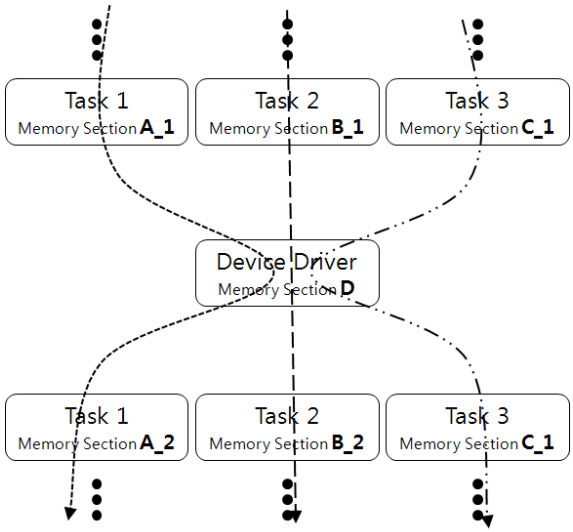

Figure 3 An application's execution trace with an I/O access

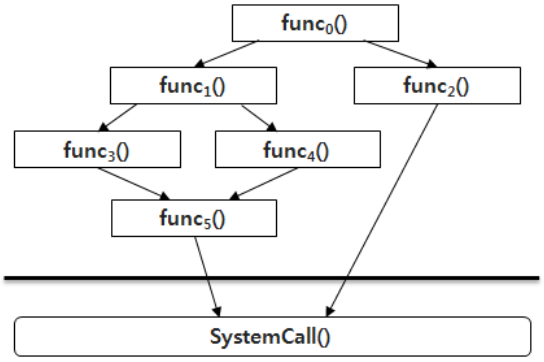

Figure 4 A function call trace graph

III. Predictive DPM Algorithm

The main functionality of our proposed PMU is to monitor and predict the I/O activity with the collaboration of HW and SW. The specific roles of HW and SW in the PMU with a key DPM algorithm are presented in the following.

A. A key observation in building PMU Design

One of the key steps to achieve low overhead PMU design is to detect the start of an I/O device access in advance. To detect when an I/O device access is about to start effectively, a set of conditions to be satisfied is defined. This paper is based on the observation that when the OS branches to a device driver routine, a system call mechanism is invoked. Therefore, by monitoring invocation of such system call, we can tell whether an I/O device access is about to start. As you see in Figure 4, an execution trace of function calls is collected until we detect the start of an I/O access and then it is analyzed in search of any common patterns among any previously saved execution traces. To minimize the amount

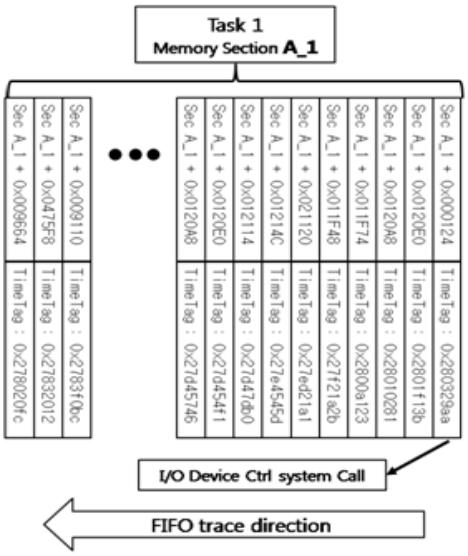

Figure 5 An example of collecting the start pattern of an I/O

of execution traces to be saved, only the PC values that correspond to function calls are saved in the trace buffer. It can be intuitively understood that execution traces based on function calls will be quite accurate since software programs are typically written in terms of functions and function calls.

B. Detection of start and end patterns

Before collecting the I/O patterns, the OS informs the PMU of the execution address of application program which contains an I/O access and the address of the system call so that the PMU receives only the appropriate information. In other words, to take advantage of layered software structures, only the execution address of related applications will be collected while disregarding other information. By doing this, we can reduce the amount of data which should be saved in our trace buffer. The saved data is stored in the trace buffer as shown in Figure 5. The trace buffer is a FIFO (First-In First-Out) mainly used to detect the start and the end patterns of an I/O access. Also it stores the time tag with the PC values. The time tag will tell when a specific function call happens. The PC values of function calls collected in Figure 5 correspond to the execution trace of Task1 in Figure 3. That is, if the main processor repeatedly executes Task1, the same execution trace may be found in the trace buffer. More specifically, Figure 5 shows the execution pattern when the processor executes "Memory section A_1", and this corresponds to the time when an I/O access starts.

To search for the start pattern in accessing a certain I/O device, the proposed PMU stores a trace of function calls in the FIFO buffer until a specific I/O device control system call is found. When the specific call is found as in Figure 5, the history information stored in the FIFO is defined to be the start pattern of the corresponding device. We save the found start pattern into a table called "DPM Tab", and this will be used to compare with future I/O access patterns. The

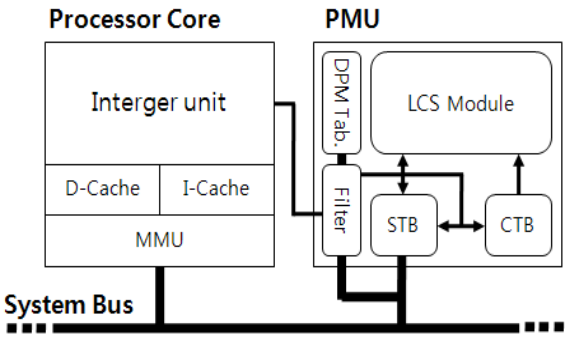

Figure 6 Structure of processor core and PMU

detection of the end pattern is found by checking the last PC value in the FIFO. If the PC value corresponds to a system call, the PC history values from the start of the I/O access till the end of the I/O access are stored. If no new access is made during a predetermined amount of time, the saved pattern is defined to be the end pattern of the I/O access. This end pattern may be used to predict the end of I/O accesses in advance for aggressive power management.

C. Detection of Longest Common Patterns

From the stored PC values in the trace buffer, we need to extract a pattern which is related to I/O device accesses. Therefore, we implement a LCS (longest common subsequence) algorithm [4] inside the PMU module. Since LCS is a polynomial time algorithm which finds the longest common subsequence from the two given patterns, it enables us to maximize the prediction time range. Figure 6 shows the PMU structure where the PC values delivered by the processor and stored in the CTB (current trace buffer) are used by the LCS module to extract the longest common pattern for I/O device accesses.

Here is a brief explanation of each module that we implemented in the PMU.

1. Current Trace Buffer (CTB)
   A FIFO buffer of which individual field is {Address, Time Tag} which is a pair of the address of a function call and the time when the call is invoked.
2. Save Trace Buffer (STB)
   The same structure as CTB, but STB contains the pattern to be matched in the CTB. The result from the LCS module will be updated in the STB for future pattern matching.
3. Filter
   Filter module forces only the PC values which correspond to a function call to be stored in CTB and STB. This filter also checks the address range so that only the addresses which fall into the application's memory area will be delivered to CTB and STB.
4. LCS Module
   A hardware implementation of the LCS algorithm. This searches for the longest common subsequence between the patterns in STB and ones in CTB. The newly acquired common pattern will be stored in STB for future comparison.
5. DPM Tab.
   This table stores the information which is necessary for predictive I/O access control. Each field of the table consists of a quintuple, {matched address0, matched address1, matched address 2, matched address 3, prediction}. The four matched addresses are extracted address patterns from the LCS module, and will be used for prediction. (The number of matched addresses can be any number but we chose four addresses in the current implementation.)

TABLE 2
Predictive power management algorithm

```
state ← 1st
repeat_cnt ← 0

pattern_gathering :
While TRUE
    Do repeat
        Until I/O_Dev_Ctrl
    If state = 1st
        Then STB ← Filter(inst, addr)
    Else CTB ← Filter(inst, addr)

Switch state
    Case 1st
        state ← 2nd
      Goto pattern_gathering
    Case 2nd
        STB ← LCS(CTB, STB)
        If repeat_cnt < 10
            Then repeat_cnt ← repeat_cnt + 1
                Goto pattern_gathering

DPM Tab{Match_Addr0} ← STB[Highest+0]
DPM Tab{Match_Addr1} ← STB[Highest+1]
DPM Tab{Match_Addr2} ← STB[Highest+2]
```

The algorithm to predict the start pattern for an I/O device access is given in Table 2. When the PMU executes the algorithm above, the results will be stored in the DPM Tab. The PMU constantly compares the information in the DPM Tab with the current PC patterns, and if there is a match, the PMU sends a wake-up command to the device in advance.

IV. Implementation of PMU

To show the effectiveness of our approach, we implemented our PMU on an SoC platform and tested with several application programs. Figure 7 shows the system block diagram integrated with our proposed PMU. To implement

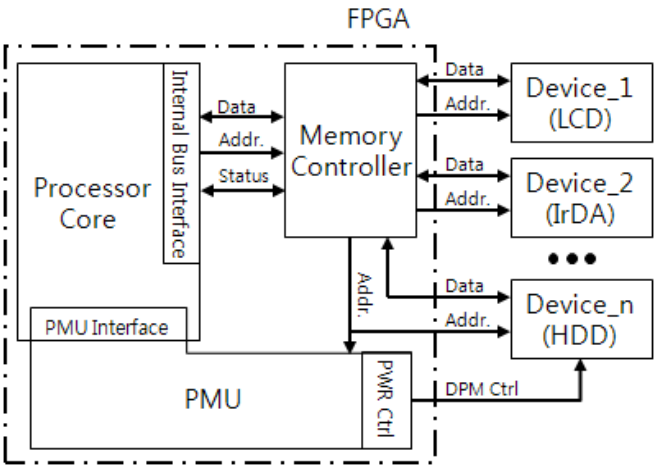

Figure 7 PMU System Block Diagram

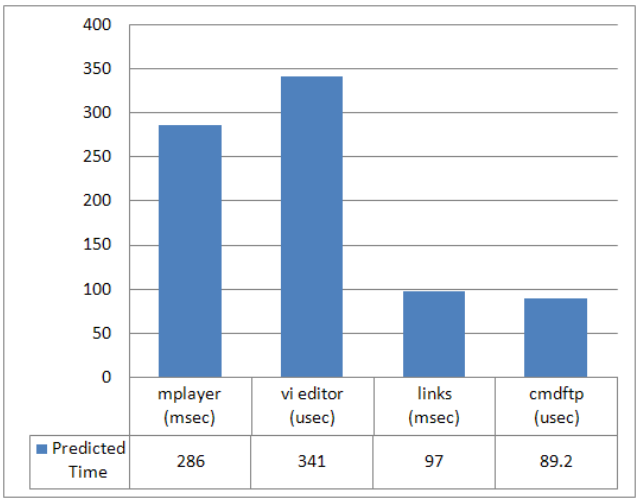

Figure 8 Prediction Time for I/O device control

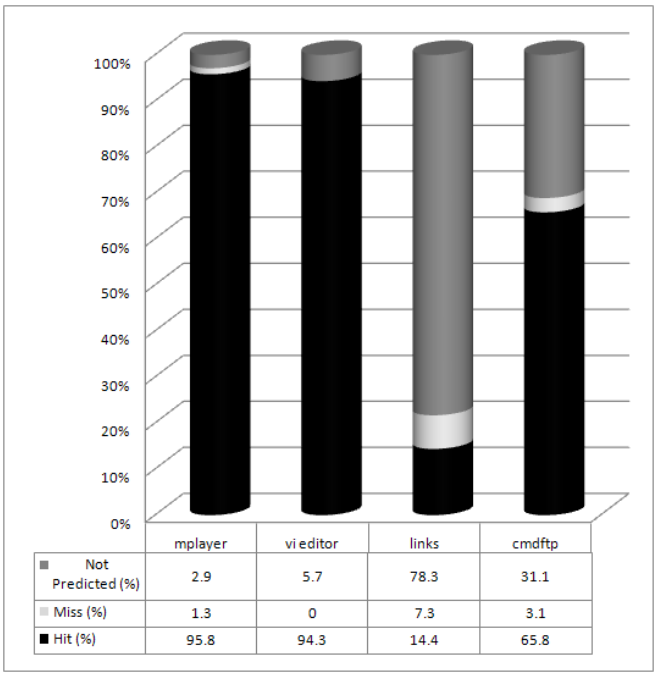

Figure 9 Prediction Accuracy

our PMU on an embedded platform, the following steps have been carried out.

We have used an SoC platform which is equipped with a Leon2 [7] processor core from Gaisler Research and an FPGA device. Most of the modules to implement the PMU are implemented on the FPGA device. The PMU in the FPGA is capable of monitoring the internal information of the LEON2 core including the values of the PC. The LEON2 core operates at 50MHz. To test the capability of PC value monitoring by the PMU, an external hard disk is connected through an IDE interface. The activity is monitored while an application is requesting data on the hard disk. Linux kernel 2.6.11 is ported as the embedded operating system.

The PMU system has been designed by following the next procedure. First, the proposed PMU designed in Verilog HDL is verified with reasonable I/O access patterns using Mentor Graphics' ModelSim simulator. Second, we program the PMU into the FPGA device. Next, we have run four different applications while the PMU records the history of I/O patterns. Based on the saved information, the PMU makes predictive decisions on the future I/O accesses to enable an early wake-up while minimizing power consumption aggressively.

V. Experimental Result

To evaluate the performance of the PMU, four different applications, mplayer (multimedia player), vi editor (text editor), links (text web browser), and cmdftp (file transfer protocol) are selected and executed. These applications are selected since they are common benchmark programs for the LEON2 processor. The results from our evaluations are summarized in Figure 8.

Figure 8 shows the results on the prediction time determined by the PMU. When we evaluate the results solely based on the prediction time, the maximum prediction time for four different application programs turns out to be 286ms. The prediction time of N seconds means that the PMU notifies the system to wake up the device N seconds earlier than the real I/O access begins. In case of mplayer and links, the prediction times are reasonably long and the actual times are 286ms and 97ms, respectively. In case of vi editor and cmdftp, the prediction times are 341μs and 89.2μs, respectively. The main reason that mplayer and links result in longer prediction times is because the relative size of library routines is larger than application code in these applications. When an application is dependent on library routines heavily, the PMU filter can filter out the addresses which fall into the library. Therefore, much less information is stored in CTB and STB. Also, the actual program size may be also another decisive factor in determining the prediction time.

Figure 9 shows the results on the accuracy of the prediction, which is another important performance metric. In case of mplayer and vi, the ratio of correct predictions (correct prediction/attempted prediction) is 95%. However, in case of cmdftp, the correct prediction ratio is 75%. In case of links, prediction is not attempted in most cases (78.3%), and in 21.7% of prediction attempts, the correct prediction rate is merely 14.4%, and the miss prediction rate is 7.3%. In case of links, the prediction has not been attempted much since there are various ways to enter the device driver. So the pattern matching has not happened much, and the accuracy of the prediction is not great either.

VI. Conclusions

In this paper, we propose a novel PMU architecture which can be used to aggressively control the power states of I/O devices. The distinctive feature of the proposed PMU is that it has the capability of predicting the upcoming I/O accesses by comparing the previously stored access pattern and the currently occurring execution patterns. The PMU is implemented in hardware as a part of the Leon2 processor core. To evaluate the performance of the PMU, four different applications have been executed with the PMU. Experimental results show that depending on the type of applications, different results on the prediction ranges and the prediction accuracies have been obtained. For some applications like mplayer and vi, excellent results are obtained. For I/O accesses which have various ways to enter the device driver, the current PMU has shown poor results in prediction capability. We are in the middle of improving this limitation by implementing more intelligent pattern matching mechanisms.

References

[1] Chirs Gniady, Ali R. Butt, Y. Charlie Hu, and Yung-Hsiang Lu, “Program Counter-Based Prediction Techniques for Dynamic Power Management”, IEEE Transactions on Computers, Vol. 55, No. 6, June 2006

[2] Yung-Hsiang Lu, Eui-Young Chung, Tajana Simunic, Luca Benini, Giovanni De Micheli, “Quantitative Comparison of Power Management Algorithms”, Design Automation and Test in Europe, 20-26, March 2000

[3] Eui-Young Chang, Luca Benini, Giovanni De Micheli, “Dynamic Power Management Using Adaptive Learning Tree”, International Conference on Computer Aided Design, 274-279, Nov 1999

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, “Introduction to Algorithms”, McGraw-Hill Company, pp. 350-356, Second Edition

[5] Luca Benini, Alessandro Bogliolo, Giovanni De Micheli “A Survey of Design Techniques for System-Level Dynamic Power Management” IEEE Transactions on VLSI Systems, Vol. 8, No. 3, June 2000

[6] Fred Douglis, P. Krishnan, and Brian Bershad. “Adaptive disk spin-down policies for mobile computers.” In USENIX Association, editor, Proceedings of the second USENIX Symposium on Mobile and Location-Independent Computing, pages 121-137

[7] URL : http://www.gaisler.com

[8] Eui-Young Chung, Luca Benini, Alessandro Bogliolo, Yung-Hsiang Lu, and Giovanni De Micheli, Fellow “Dynamic Power Management for Nonstationary Service Requests”

[9] IBM and MontaVista Software “Dynamic Power Management for Embedded Systems”

[10] Jason Flinn and M. Satyanarayanan. “Energy-aware adaptation for mobile application.” In Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP’99), Pages 48-63, December