A Predictive Dynamic Power Management Technique for Embedded Mobile Devices

Young-Si Hwang, Sung-Kwan Ku and Ki-Seok Chung, *Member*, IEEE

Abstract — *It is becoming crucial to manage devices intelligently to reduce the power consumption of mobile embedded systems. Power-aware system management relies on techniques of collecting and analyzing information on the status of Input/Output (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 degraded. Therefore, designing a Power Management Unit (PMU) that collects information and analyzes the information with some hardware support is very important. In this paper, we propose a novel PMU design that collects and analyzes access patterns to I/O devices while an application is being executed. We also propose a predictive power-aware management scheme which is carried out based on the collected information. Experiments with various applications have been conducted to show the effectiveness of our approach*[1].

Index Terms — Low-Power Design, Dynamic Power Management, Embedded OS, Power Management Unit.

I. INTRODUCTION

The number of functions that an average embedded mobile device executes is increasing rapidly, and the number of I/O devices that an embedded system should control increases accordingly. With rapid advances in HW and SW technologies, building a 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. From system designers' perspective, minimizing power consumption in mobile devices has become one of the most important issues. Thus, they sometimes sacrifice delay or area to reduce power consumption. With increasing demands for low power techniques, research topics on how to reduce power consumption broadly cover from the circuit/logic level to architecture, software, and system level techniques. Among them, system-level power management techniques have been actively studied because, to reduce power consumption, management techniques are often more important than low power design techniques themselves [8]. Specifically, two of the most commonly applied techniques are Dynamic Power Management (DPM) and Dynamic Voltage & Frequency Scaling (DVFS) [9][10][11][12][13][14][15][16].

To reduce power consumption of embedded processors, hardware-based DVFS techniques are widely accepted. To reduce power consumption of I/O devices, a Power Management Unit (PMU) with DPM capability is often employed. However, managing DVFS and DPM relies on system software such as operating systems. Software-based power-aware management has the merit of flexible control, but it has a potential problem of suffering from significant runtime overhead to decide how to manage effectively. Therefore, comprehensive power management techniques should take into account the overall hardware and software management overhead to achieve a true power reduction. In this paper, we propose an advanced PMU design that minimizes the overall runtime overhead of power-aware system management by implementing a block for collecting and analyzing access patterns in hardware.

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

[1] This research was supported by the MKE(The Ministry of Knowledge Economy), Korea, under the ITRC(Information Technology Research Center) support program supervised by the NIPA(National IT Industry Promotion Agency)" (NIPA-2010-C1090-1031-0009)

Young-Si Hwang is with the Department of Electronics, Computer & Communication Engineering, Hanyang University, Seoul, Korea, (e-mail: ysturtle@hanyang.ac.kr).

Sung-Kwan Ku is with the Samsung Techwin Co., LTD, Seoul, Korea, (e-mail: skwan.ku@samsung.com)

Ki-Seok Chung is with the Department of Electronics, Computer & Communication Engineering, Hanyang University, Seoul, Korea, Ki-Seok Chung is the corresponding author (e-mail: kchung@hanyang.ac.kr).

Contributed Paper
Original manuscript received November 18, 2009
Revised manuscript received April 13, 2010
Current version published 06 29 2010;
Electronic version published 07 06 2010.

0098 3063/10/$20.00 © 2010 IEEE

does not cause any significant runtime overhead unlike other software-based management schemes.

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 works in the last section.

II. Background and Motivation

Power-aware design methodology has emerged as one of the most crucial system design issues [18][19]. Active studies are being conducted 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 in the near future. The difficulty in doing this lies in the fact that nobody can predict the future perfectly. On average it takes tens of mili-seconds to switch a hard disk from standby mode to an active state. Due to its long delay, devices may have to stay in an idle state quite a long time before they are put into sleep mode. To reduce power consumption, modern hard disks have at least four different operation modes (busy, idle, standby and sleep) as shown in Table 1. A hard disk consumes a fairly large amount of power when it stays in an idle state. Thus, 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 two intermediate idle and standby states are added due to the design trade-offs between power savings and wake-up delay. However, the two states are not very effective for power reduction. We claim that by having a predictive power management capability, we can shut down the device as soon as the device becomes idle. By predicting the next re-activation time in advance, we do not have to suffer from the performance penalty due to a 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 realized 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. Learning Tree (LT) [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 would have to be used to obtain the clues for predictions. PCAP [1] is a Program Counter (PC) 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 Operating System (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

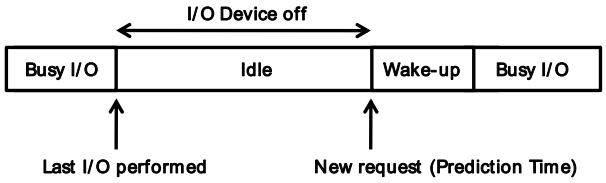

Fig. 1. An example of I/O device access pattern.

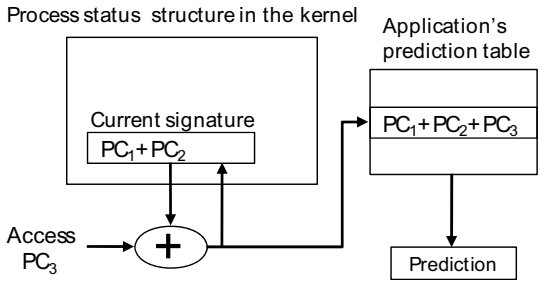

Fig. 2. Structure of PCAP[1].

TABLE I
Power consumption of a hard_disk (Fujitsu 1.8-inch HDD) for each state

State	Power
Spin-up	4.5W
Busy (RD/WR)	1.6W
Idle	0.5W
Standby	0.2W
Sleep	0.1W

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 the branching history and how the stored information is used to predict the future branches in PCAP. When a certain access to an I/O device is initiated, a signature, which is the sum of accessed PC values to the device, is computed. The computed signature is compared with the signatures stored in the application's prediction table. When there is a match, a predictive shutdown action takes place. When there is no match, the current signature is stored in the prediction table without any predictive power management. 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 the runtime overhead as much as possible. The main focus of existing studies such as timer-based prediction or Learning Tree is how to execute a set of special-purpose instructions to minimize power consumption in the OS kernel. However, this may result in a significant runtime overhead. In our approach, monitoring I/O access patterns and recognizing a certain pattern are implemented in hardware so that the 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 the I/O access control signal and the PC values. Therefore, we do not need any large history buffer.

III. PREDICTIVE DPM ALGORITHM

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

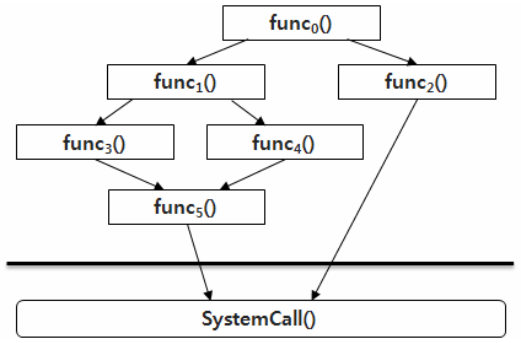

Fig. 3 A function call trace graph

A. A key observation in building PMU Design

One of the key steps to achieve the proposed PMU design is to detect the start of an I/O device access in advance. To detect accurately when an I/O device access is about to start, a set of conditions to be satisfied is defined. Our detection algorithm is based on the observation that when the OS branches to a device driver routine, a system call mechanism is invoked. Therefore, by monitoring the invocation of such a system call, we can tell whether an I/O device access is about to start. As you see in Figure 3, 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 of execution traces to be saved, only the PC values that correspond to function calls are saved in the trace buffer. It may 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 I/O patterns, the OS informs the PMU of the execution address of an application program, which contains an I/O access and the address of the system call, so that the PMU recognizes 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 that should be saved in the trace buffer. The saved data is stored in the trace buffer as shown in Figure 5. The trace buffer is a First-In First-Out (FIFO), which is mainly used to detect the start and the end patterns of an I/O access. Also, it stores time tags with the PC values. A time tag will tell when a specific function call happens. A PC value of selected function calls collected in Figure 5 corresponds to the execution trace of a task. For example, if the processor repeatedly executes "Task1" in Figure 4, the same execution trace is found in the trace buffer. More specifically, Figure 5 shows the execution pattern when

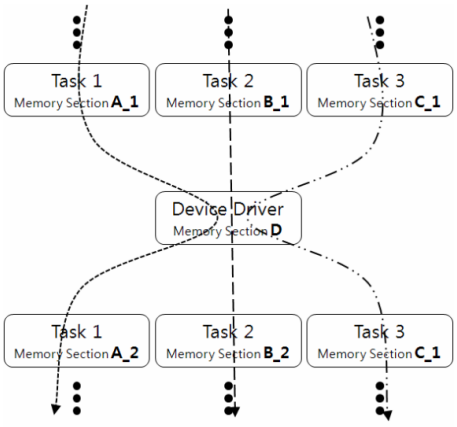

Fig. 4 An application's execution trace with an I/O access.

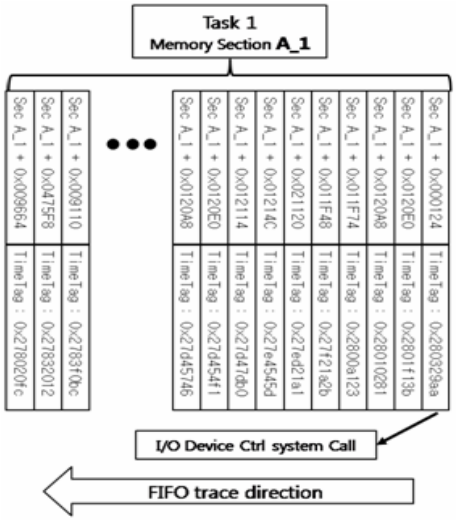

Fig. 5 An example of collecting the start pattern of an I/O.

the processor executes "Memory section A_1.", and this corresponds to the time when an I/O access starts.

To search for a 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 system call for an I/O device control is found. When the specific call is found as in Figure 5, the trace information stored in the FIFO is defined to be the start pattern of the corresponding device. We save the detected start pattern into a table called "DPM Tab.", and this will be used to compare with future I/O access patterns. The 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 patterns that are related to I/O device accesses. To find a pattern, we implement the Longest Common Subsequence

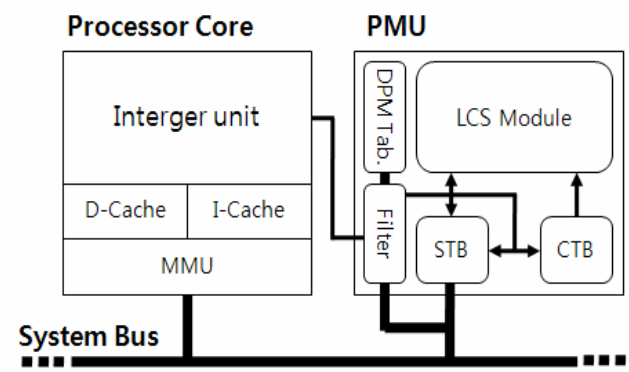

Fig. 6 Structure of processor core and PMU.

```
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]
```

Fig. 7 Predictive power management algorithm.

(LCS) algorithm [4] inside the PMU module. Since LCS is a polynomial time algorithm that 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 Current Trace Buffer (CTB) 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 implement in the PMU.

- 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.

- Save Trace Buffer (STB)

STB has the same structure as CTB, but it 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.

- Filter

The filter module forces only the PC values that correspond to a function call to be stored in CTB and STB. This filter also checks the address range so that only the addresses that fall into the application's memory area will be delivered to CTB and STB.

- LCS Module

This is a hardware implementation of the LCS algorithm. This searches for the longest common subsequence between the patterns in STB and those in CTB. The newly acquired common pattern will be stored in STB for future comparisons.

- DPM Tab.

This table stores the information that is necessary for a predictive I/O access control. Each field of the table is a quintuple {matched address0, matched address1, matched address 2, matched address 3, prediction}. The four matched addresses are extracted patterns from the LCS module, and will be used for prediction. (The number of matched addresses can be any number. We choose four addresses in the current implementation.)

The predictive power management algorithm in Figure 7 proceeds as follows.

a. Set the PC collection state to '1st'. The PC patterns for function calls are saved in STB until a system call to control an I/O device is invoked.
b. When the system call is invoked, the PC collection state is changed to "2nd". Also the PC patterns for function calls are saved in CTB until a new system call for an I/O device control takes place.
c. By running our LCS algorithm, we find the common patterns between STB and CTB, and the found common patterns will be saved in STB.
d. Steps in b and c are repeated a number of times to obtain a reliable access pattern. (Currrently we repeat 10 times.)
e. Among all the found reliable patterns, we select the three most reliable patterns and store them in DPM Tab. (The number of saved patterns is determined by considering trade-offs between resource amount and prediction accuracy.)

The saved patterns in DPM Tab. are compared with the PC patterns that are being accessed. When there is a match, we activate a predictive power management.

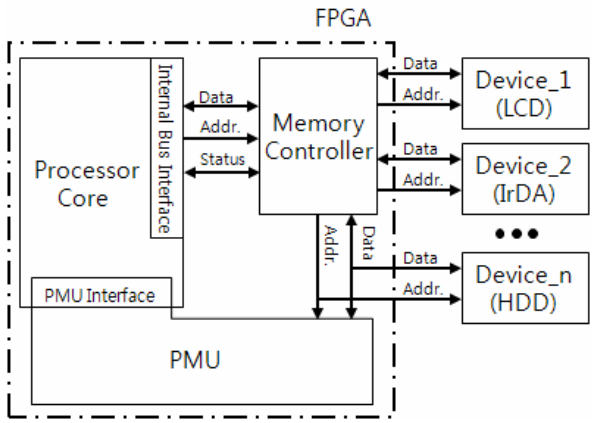

Fig. 8 PMU System Block Diagram.

TABLE II
MEASUREMENT (SYNPLIFY 8.6.2)

		Leon2	PMU
Total Combinational functions	Logic element usage by number of inputs		
	4 input functions	4021	696
	3 input functions	4471	64
	<= 2 input functions	864	75
	Logic elements by mode		
	Normal mode	8908	780
	Arithmetic mode	448	55
	Total	9356	835
Total registers		2579	379
I/O Pins		105	123

TABLE III
AREA AND POWER MEASUREMENT (QUARTUS 6.0)

	Leon2	PMU
Total logic elements	9438 / 33216 (28%)	691 / 33216 (2%)
Total registers	2546	365
Dynamic Power	98.87mW	1.04mW
Static Power	93.25mW	0.21mW

IV. IMPLEMENTATION OF PMU

To show the effectiveness of our approach, we implemented the proposed PMU on a System-on-Chip (SoC) platform and tested with several application programs. Figure 8 shows the system block diagram integrated with the proposed PMU. To implement our PMU on an embedded platform, the following steps have been conducted.

We use an SoC platform equipped with a LEON2 [7] processor core from Gaisler Research and a Field Programmable Gate Array (FPGA) device. Most of the modules to implement the PMU are programmed in the FPGA device (Cyclone-II: EP2C35F484C6N). 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.

First, the proposed PMU designed in Verilog HDL is verified with reasonable I/O access patterns using Mentor Graphics' ModelSim simulator. Second, we program our PMU design into an FPGA device. Next, we 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 upcoming I/O accesses to enable an early wake-up while minimizing power consumption aggressively.

The area of the PMU has been estimated by Synplicity's Synplify 8.6.2 and Altera's Quartus 6.0. To understand the relative size of the PMU, we compare the area with that of the LEON2 processor, which is the main processor to run all the software. The size of the PMU is only 7% of that of the LEON2 processor core as shown in Table 2 and 3. The amount of power dissipated by the PMU itself has been estimated by Altera's PowerPlay. The results show that the amount of power dissipated by the PMU is much smaller than that of the LEON2 processor core.

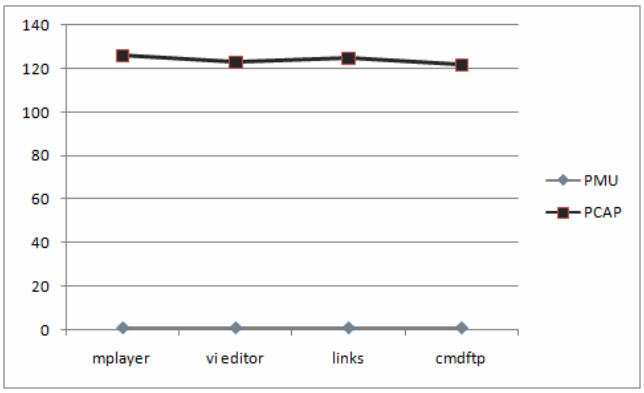

Fig. 9 Runtime overhead of PCAP and the proposed PMU.

V. EXPERIMENTAL RESULT

For performance comparison, we have implemented the PCAP algorithm as a part of the Linux kernel. We have compared the performance by running a set of applications which consists of mplayer (a multimedia player), vi (a text editor), cmdftp (a file transfer program) and links (a text-based web browser). The results are summarized in Figures 9 through 12.

First, we compare the runtime overhead of PCAP and our proposed PMU. The amount of overhead is defined as the ratio between the workload for the power management and the workload of the actual application as shown in Equation (1).

$$\text{overhead} = \frac{\text{power management workload}}{\text{application workload}} \quad (1)$$

Since our proposed PMU collects and analyzes the function call patterns in hardware, the runtime overhead at the kernel is minimal. The control command to manage the states of a hard disk is activated by an interrupt. As shown in Figure 9, the runtime overhead of our proposed method is less than 0.03. On the other hand, in case of PCAP, tracing and collecting PC values and analyzing them are carried out by the kernel code. To measure the overhead of PCAP, Linux "ptrace" system call has been used. The amount of runtime overhead of PCAP estimated by Equation (1) turns out to be more than 120. In other words, the amount of workload to manage I/O devices while an application is running will be roughly 120 times more than the amount of workload to run the application itself. This ratio may be improved significantly by optimizing the kernel code for power management. However, monitoring and analyzing the PC values constantly will inherently cause a significant runtime overhead, and the amount of overhead will often be unacceptably large.

Figure 10 shows the results of the prediction time determined by the PMU. When we evaluate the results solely

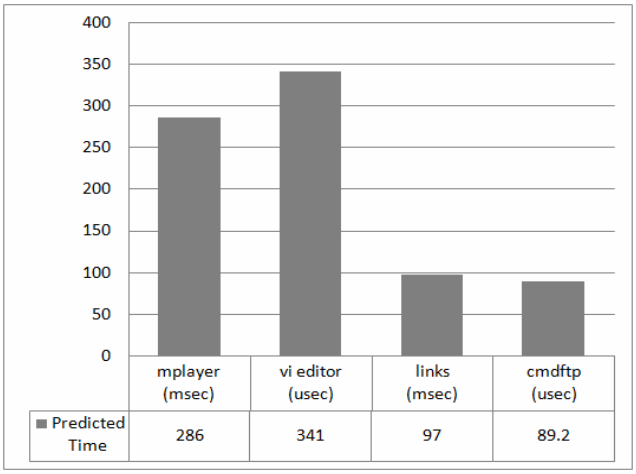

Fig. 10 Prediction Time for I/O device control.

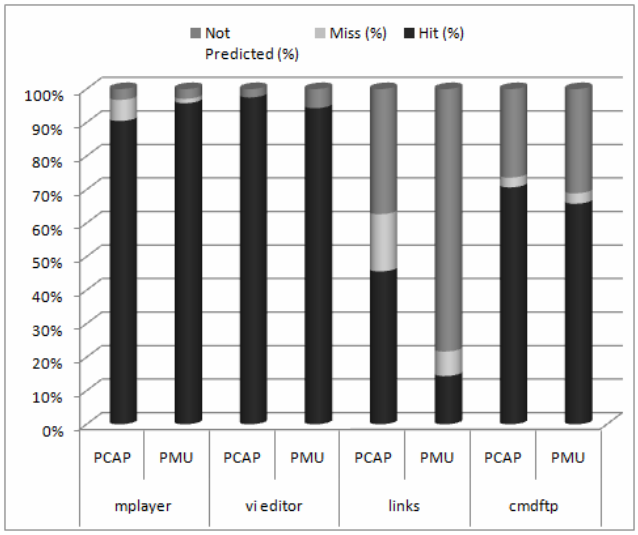

Fig. 11 Prediction Accuracy.

based on the prediction time, the maximum prediction time for four different applications 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 341us and 89.2us, respectively. The main reason that mplayer and links result in longer prediction times is that the relative size of library routines is larger than the application code in these applications. When an application is heavily dependent on library routines, the PMU filter can filter out the addresses that fall into the library routine. Therefore, much less information is stored in CTB and STB. Also, the actual program size may be another decisive factor in determining the prediction time.

Figure 11 shows the results on the accuracy of the prediction. In case of mplayer, the ratios of correct predictions to attempted predictions of PCAP and our PMU are 90.5% and 95%, respectively. The misprediction ratios of the two are 6.3% and 1.3%, respectively.

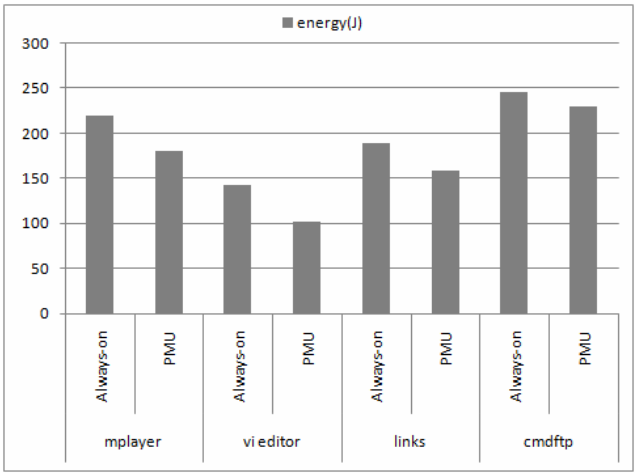

Fig. 12 The amount of energy savings.

Prediction accuracy for mplayer is good since multimedia processing typically has regular access patterns, and our PMU shows slightly better results than PCAP. In case of vi, the correct prediction ratios of the two are 97.5% and 94.3%, respectively. The results for vi are excellent since the text editors like vi have relatively simple read/write disk access patterns. However, in case of cmdftp and links, the attempted prediction percentage and the correct prediction ratio of our PMU are relatively low. This is mainly because our hardware-based prediction mechanism has a limited amount of saved patterns. When the application has various ways to access the hard disk, our PMU cannot handle them well. PCAP is capable of saving and handling various patterns better than our PMU.

Figure 12 shows the amount of energy savings by our PMU. The amount of energy savings is computed by comparing our results with the case where the hard disk is always on. The amount of power consumption is measured every 120 seconds for each benchmark application. The amount of energy saving is 29.49J, 41.27J, 31.30J and 14.67J for mplayer, vi editor, links, and cmdftp, respectively. The main reason that we have the most energy savings in the vi case is that vi has infrequent disk accesses for load and save. The amount of energy savings is said to be dependent on the frequency and the pattern of disk accesses.

VI. CONCLUSIONS

In this paper, we propose a novel PMU architecture that can be used to control the power states of I/O devices. The distinctive feature of the proposed PMU is that it has the capability of predicting 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 that 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 the prediction capability by implementing more intelligent pattern matching mechanisms.

REFERENCES

[1] C. Gniady, Ali R. Butt, Y. Charlie Hu, and Y.-H. Lu, "Program Counter-Based Prediction Techniques for Dynamic Power Management," IEEE Transactions on Computers, Vol. 55, No. 6, June 2006, pp. 24-35

[2] Y.-H. Lu, E.-Y. Chung, T. Simunic, L. Benini, and G. D. Micheli, "Quantitative Comparison of Power Management Algorithms," Design Automation and Test in Europe, March 2000, pp. 20-26

[3] E.-Y. Chang, L. Benini, and G. D. Micheli, "Dynamic Power Management Using Adaptive Learning Tree," International Conference on Computer Aided Design, Nov 1999, pp. 274-279

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

[5] L. Benini, A. Bogliolo, and G. D. Micheli "A Survey of Design Techniques for System-Level Dynamic Power Management," IEEE Transactions on VLSI Systems, Vol. 8, No. 3, June 2000, pp. 231-248

[6] F. Douglis, P. Krishnan, and B. 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, pp. 121-137

[7] J. Gaisler, "LEON-1 Processor - First Evaluation Results," European Space Components Conference (ESCCON) (2000 : Noordwijk, The Netherlands). Proceedings of the European Space Components Conferences ESCCON 2000, 21-23 March 2000, ESTEC, Noordwijk, The Netherlands Edited by B. Schürmann. Noordwijk, the Netherlands: European Space Agency, 2000. ESA-SP, Vol. 439, p.183

[8] J. Flinn and M. Satyanarayanan. "Energy-aware adaptation for mobile application," in Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP'99), Dec., pp 48-63

[9] L. Benini, A. Bogliolo, and G. D. Micheli, "Policy Optimization for Dynamic Power Management," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 18, No. 6, June 1999, pp. 813-833

[10] C. S. Ellis, "The Case for Higher-Level Power Management," In Workshop on Hot Topics in Operating Systems, Rio Rico, AZ, USA, March 1999, pp. 162-167

[11] R. Nair, "Dynamic path-based branch correlation," in Proceedings of the 28th annual international symposium on Micro-architecture, November 1995, pp. 15-23

[12] Y.–H. Lu, G. D. Micheli, and L. Benini, "Requester-aware power reduction," In Proceedings of the Int'l Symp. on System Synthesis, 2000, pp 18-24

[13] C.-H. Hwang, and A. C. Wu, "A Predictive System Shut-down Method for Energy Saving of Event Driven Computation," ACM trans. On Design Automation of Electronic Systems, Vol. 5, No. 2, April 2000, pp. 226-241

[14] J. E. Smith , "A Study of Branch Prediction Strategies," in Proc. of the 8th Annual Symposium on Computer Architecture, 1981, pp. 135-148

[15] T. Simunic, L. Benini, and G. D. Micheli, "Event Driven Power Management of Portable Systems," Proc. Int'l Symp. System Synthesis, IEEE CS, November 1999, pp. 18-24

[16] T. Simunic et al., "Dynamic Power Management for Portable Systems," Int'l Conf, Mobile Computing and Networking, ACM Press, New York, 2000, pp.11-19

[17] Jaedoo Go, and Minseok Song, "Adaptive disk power management for portable media players", IEEE Transactions on Consumer Electronics, Vol. 54, No. 4, November 2008, pp. 1755-1760

[18] Jaewoo Kim, Ahron Yang, and Minseok Song, "Exploiting flash memory for reducing disk power consumption in portable media players", IEEE Transactions on Consumer Electronics, Vol. 55, No. 4, November 2009, pp. 1997-2004

BIOGRAPHIES

Young-Si Hwang received the B.S. degree in computer engineering from Daejin University, Korea, in 2003 and the M.S. degree from the Department of Electronics Computer Engineering at Hanyang University, Korea in 2008. From 2003-2005, he worked as an engineer for Korea Information & Communications Company of Korea. He now stays in Embedded System-on-Chip Laboratory, College of Information & Communications, Hanyang University to study low power embedded operating system design.

Sung-Kwan Ku received the B.S. degree in Computer Engineering from Texas A&M University, US, in 2005. He received the M.S. degree from the Department of Electronics Computer Engineering at Hanyang University, Korea in 2008. His research interests include low power methodology, low power embedded OS design, and system level power management technique on multi-processors.

Ki-Seok Chung received the B.E. in computer engineering from Seoul National University, Seoul, Korea in 1989, and the Ph.D. degree in computer science from University of Illinois at Urbana-Champaign in 1998. He was a senior R&D Engineer at Synopsys, Inc. in Mountain View, CA from 1998 to 2000, and was a Staff Engineer at Intel Corp. in Santa Clara, CA from 2000 to 2001. He also worked as an assistant professor at Hongik University, Seoul, Korea from 2001 to 2004. Since 2004, he has been an Associate Professor at Hanyang University, Seoul, Korea. His research interests include low power embedded system design, SoC design methodology, reconfigurable processor and DSP design, SoC-platform based verification, and system software for MPSoC.