A Low-Power Real-Time Operating System for ARC (Actual Remote Control) Wearable Device

Moon-Haeng Cho and Cheol-Hoon Lee*

Abstract — *In recent years, consumer electronic devices have been expanding their application domains from traditional consumer electronic devices such as cellular phones, PDAs(Personal Digital Assistants), PMPs(Portable Multimedia Players), Navigation, to Next Generation Personal Computers(NGPCs) such as eating consumer electronic devices, implanted consumer electronic devices, wearable consumer electronic devices. A wearable consumer electronic device is a computer that is subsumed into the personal space of the user, controlled by the user, and is always on and always accessible. Therefore, among the most salient aspect of wearable consumer electronic devices should be small form factor and long battery lifetime. In this paper, we propose a novel low-power real-time operating system (RTOS) specialized for the ARC(Actual Remote Control). An ARC is a wearable wristwatch-type remote controller; it also can serve as a universal remote control for various consumer electronic devices. The proposed RTOS has about 9KB footprints and is equipped with low-power techniques composed of the dynamic power management and the device power management. Experimental results with wearable application show that we can save energy up to 47 % using the proposed low-power techniques*[1].

Index Terms — Next generation personal computers, Wearable consumer electronic devices, Real-time operating systems, Low-power management

I. INTRODUCTION

Computation and communication systems have been rapidly moving toward mobile and portable consumer electronic devices. In recent years, with continued miniaturization and increasing computation power, we see ever growing use of powerful microprocessors running sophisticated, intelligent control software in a vast array of consumer electronic devices including digital convergence consumer electronic devices such as PDAs (Personal Digital Assistants), PMPs (Portable Multimedia Players), UMPCs (Ultra Mobile Personal Computers). Now, digital convergence consumer electronic devices have been expanding their application domains to next generation personal computers (NGPCs) such as eating consumer electronic devices, implanted consumer electronic devices, wearable consumer electronic devices [1]-[7].

A wearable consumer electronic devices is a computer that is subsumed into the personal space of the user, controlled by the user, and has both operational and interactional constancy, i.e. is always on and always accessible. Most notably, it is a consumer device that is always with the user, and into which the user can always enter commands and execute a set of such entered commands, and in which the user can do so while walking around or doing other activities. Therefore, among the most salient aspect of wearable consumer electronic devices should be small form factor and long battery lifetime.

In spite of continuous advances in semiconductor and battery technologies that allow microprocessors to provide much greater computation per unit of energy and longer total battery life, the fundamental tradeoff between performance and battery life remains critically important. Therefore, power management and power-performance trade-offs are more important for wearable consumer electronic devices because of their performance intensive user interfaces and their tighter constraints on size, weight, and battery capacity.

Consequently, to solve the problems of hardware constraints, wearable consumer electronic devices must use small and low-power real-time operating systems [9]-[14].

In this paper, we propose a novel low-power real-time operating system (RTOS) designed specifically for ARC[8] wearable device with battery capacity and memory size limitations. The proposed RTOS has about 9KB footprints and is equipped with the low-power techniques composed of the dynamic power management and the device power management.

In the next section, we survey some related works on wearable computing, embedded operating systems, and low-power management techniques. In Section Ⅲ, the features of the proposed RTOS are presented. We conducted some experiments to evaluate the performance and energy saving capability of the proposed RTOS, and the results are given in Section Ⅳ. The experimental results show that the proposed RTOS is quite efficient in performance and saves energy up to 47% using the low-power management when running wearable consumer electronic device application. Finally, we draw some conclusions and mention possible future works in Section Ⅴ.

[1] This work was supported by the IT R&D program of the Korean MKE and IITA (2008-F-048, Wearable Personal Companion for u-computing collaboration)

M.-H. Cho and C.-H. Lee are both with the Department of Computer Engineering, Chungnam National Univ., Daejeon, Korea. (e-mail: {root4567, clee}@cnu.ac.kr).

*C.-H. Lee is the corresponding author

Contributed Paper
Manuscript received 06/02/10
Current version published 09/23/10
Electronic version published 09/30/10.

II. Related Works

A. Wearable computing

Wearable are generally equated with head-up, wearable displays, one handed keyboards, and custom computers worn in satchels or belt packs. Ideally, wearable computing can be described as the pursuit of a style of interface as opposed to a manifestation in hardware. Wearable computing has inherited many of the system issues with which the general computing community grapples. However, wearable represent a current extreme in computing because researchers have not thoroughly studied the area's design trade-off. To provide wearable computing, the problems of power use, heat dissipation, networking, and privacy should be solved. Each of these problems is closely related to the others, and a design change to correct deficits in one often affects the others. However, the most immediately striking challenges are the performance intensive user interfaces and the tighter constraints on size, weight, and battery life [2]-[14].

B. Embedded operating systems for consumer electronic devices

An operating system is the very essence of any consumer electronic device design. Embedded operating systems such as *μC/OS*, *Nucleus*, *pSOS*, *VxWorks*, *Embedded Linux*, *WindowsCE* are designed for consumer electronic devices with relatively limited resources.

1) μC/OS

μC/OS [13], [14] is a fully preemptive real-time kernel written in ANSI C. It can manage up to 64 tasks. It is a scalable kernel, in that the user can select only the kernel features required in the application and leave the rest, thus reducing the amount of memory needed by the system. It provides services such as mailboxes, queues, semaphores, fixed-sized memory partitions, time-related functions, etc. Because of its simplicity and lightweight, this kernel is suitable for small and portable consumer electronic devices that require high-performance. On the negative side, there are only 56 user tasks available and all the tasks should have different priorities. Also, it doesn't provide low power managements.

2) VxWorks

VxWorks is a very popular and powerful RTOS for consumer electronic devices. It provides preemptive multitasking and high-speed interrupt supports. It is a modular and scalable operating system. The kernel provides services such as semaphores, events, mailboxes, queues, memory management, timers, etc.

3) Embedded Linux

Embedded Linux is designed for consumer electronic devices with relatively limited resources, such as smaller sizes of RAM and much more limited secondary storage. It is also usually purpose-made for the required application and target hardware, and thus attempts to be the optimized form of the Linux kernel for that application. Advantages compared to other embedded operating systems include: the source code can be modified and redistributed; relatively small footprint (a typical installation may require less than two megabytes of memory); no royalty or licensing costs; mature and stable; and a large support base.

4) Windows CE

Windows CE is a variation of Microsoft's Windows operating system for minimalistic computers and small and portable consumer electronic devices. It is optimized for consumer electronic devices that have minimal storage (relatively small footprint - Kernel may run in under a megabyte of memory). It conforms to the definition of a real-time operating system with deterministic interrupt latencies. It supports 256 priority levels and uses priority inheritance to deal with priority inversion.

Table 1 shows comparisons about footprint size and the presence of any low-power management techniques among them.

Table1
Comparison of footprint sizes and Low power techniques of operating systems

	μC/OS	VxWorks	embedded Linux	Windows CE
Size (KB)	12	286	1~2MB	1MB
Low power	No	No	Yes	Yes

C. Low power management techniques

In this subsection, we introduce two system-level low-power management techniques: device power management and dynamic voltage scaling. Using these techniques, we can achieve energy reductions either by selectively switching off unused components or by scaling down the performance of individual components in accordance to temporal performance requirements of the consumer electronic device applications [15]-[24].

1) Device power management

System-level device power management algorithms increase the battery lifetime by selectively placing idle components into lower power states. System resources can be modeled using state-based abstraction where each state trades off performance for power. State transitions are controlled by commands issued by a power manager that observes the workload of the system and decides when and how to force power state transitions. The choice of the policy that minimizes power consumption under performance constraints is a constrained policy optimization problem which is very important for all portable systems [18]-[21].

Several heuristic power management policies have been investigated in the past for controlling hard disks and interactive terminals. The most common policy for devices is

a timeout policy implemented in most real-time operating systems. The drawback of this policy is that it wastes power while waiting for the timeout to expire. Another group of policies studied in the past are predictive policies. Predictive policies force the transition to a low power state as soon as a component becomes idle if the predictor estimates that the idle period will last long enough. However, a wrong estimate can cause both performance and energy penalties.

2) Dynamic voltage scaling

The dynamic voltage scaling (DVS) for real-time systems has become a major research area to minimize CPU energy consumption by scaling the voltage and clock frequency while still meeting the real-time deadlines. This is possible because the static CMOS logic has a voltage-dependent maximum operating clock frequency. Thus it can operate at a lower supply voltage. Since the energy dissipated per cycle with CMOS circuitry scales quadratically to the supply voltage ($E \propto V^2$) , DVS can potentially provide a very large net energy savings through frequency and voltage scaling.

Although real-time tasks are specified with worst-case computation requirements, they generally use much less than the worst-case values for most invocations. Ernst and Ye[22] observed that the ratio of the worst-case execution time to the best-case execution time can be as high as 10 in typical applications. To take advantage of this, a DVS mechanism could reduce the operating frequency and voltage when tasks use less than their worst-case time allotment, and, if needed, increase frequency to meet the worst-case needs [23], [24].

III. THE LOW-POWER REAL-TIME OPERATING SYSTEM

The proposed RTOS have been ported on the ARC wearable device. An ARC is a wearable wristwatch-type remote controller; it also can serve as a universal remote control for various consumer electronic devices. ARC can be worn on the wrist and is based on a 3-axis accelerometer sensor in order to recognize forearm gestures. This makes it suitable for mobile environments where hands-free and portability is necessary. Also, ARC enables natural gesture interface which shows many advantages over the conventional universal remote control. However, ARC has hardware constraints of small-sized memory (less than 1MByte SDRAM), limited battery lifetime, and embedded microprocessors with low-processing power (less than 400MHz).

The proposed RTOS is a small-sized real-time operating system designed specifically for wearable consumer electronic devices such as ARC with limited memory size and battery life time, and low-processing power. As shown in Fig. 1, the proposed RTOS incorporates major features of a real-time operating system, such as task management and multitask scheduling (1.5KB), memory management (1.5KB), inter-task communication and synchronization (1.2KB), interrupt handling (0.2KB), a timer (1.2KB), and power managers (3.2KB). The low-power management is composed of the dynamic power management and the device power management. The total footprint size of the RTOS is about 9 KB.

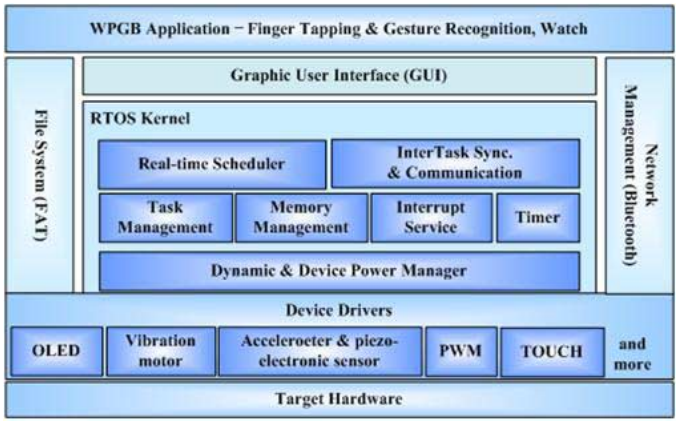


Fig. 1. Overview of the RTOS

A. Task management

Generally, an application of consumer electronic devices is composed of several independent programs. Each of these independent programs is called a task. Each task runs with its own execution functions, stack region, priority, and time slice. The proposed RTOS is a task-based operating system, where the system resources and the address space are shared among tasks, and it provides scheduling methods to solve time constraints of a real-time operating system.

Multitasking means that several tasks are simultaneously run on a single CPU. The proposed RTOS supports such multitasking environments. Each task has its own context and the context is saved in the task control block (TCB). The context of a task includes program counter of the task, CPU registers, stack and function calls, delay and time slice timers, and the task priority.

Each task has one of five states as shown in Fig. 2: SUSPENDED, READY, RUNNING, DELAYED, and PENDING states. Once a particular event occurs, the state of a task is changed accordingly. When a task is waiting to get a particular resource, it is possible to set the maximum pending time. In this case, the task is in the state of PENDING+DELAYED state. Fig. 2 shows the state transition of tasks by events. Once a task is created, it goes into SUSPENDED state. When a created task is activated, it goes into RAEDY state. Also, a task can be deleted from any of the five states.

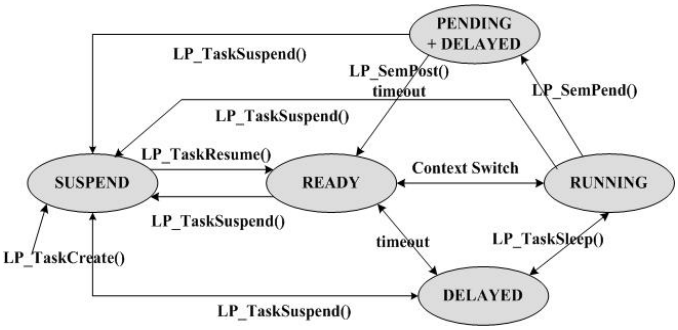


Fig.2. Task state transition diagram

The real-time scheduler and task management service of the proposed RTOS guarantee real-time deadlines. Also, real-time kernel services are deterministic by specifying how long each service call will take to execute. Having this information allows the application designers of consumer electronic devices to better plan their application software [25], [26].

The proposed RTOS provides 28 priority levels from 4 to 31 for each task (see Fig. 3) and a ready list is managed for each priority. The scheduler finds the highest priority task in a constant time using special data structure called MK_Priority and then transfers the CPU control to the task. The lowest four bits ("Group Bit") of MK_Priority are bit maps for the four priority groups. If there is at least one task with a priority belonging to a priority group, the corresponding bit of Group Bit is set to 1. Otherwise, it is set to 0. The proposed RTOS provides a priority-based preemptive scheduling policy, and for tasks with the same priority, it also provides round-robin scheduling policies.

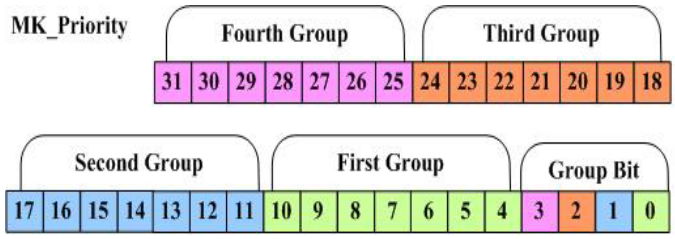


Fig. 3. Data structure for MK_Priority.

B. Memory management

Dynamic memory management algorithms play an important role in the modern software engineering paradigms and techniques. Using dynamic memory management increases the flexibility and functionalities of applications. The goal of these algorithms is to provide dynamically to the applications the amount of memory required at run time [27]-[29]. The proposed RTOS provides a heap storage manager for dynamic memory management (see Fig. 4). The heap storage manager uses the algorithm with address-ordered first-fit memory management for fast response times.

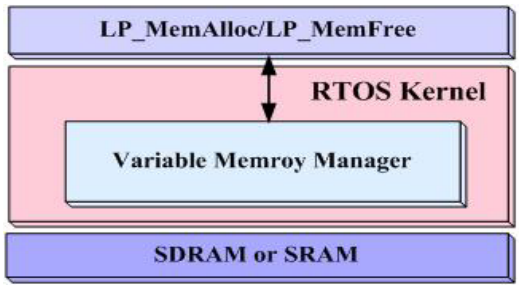


Fig.4. Heap manager

C. Semaphore

Semaphores are important means to satisfy the requirements of mutual exclusion and inter-task synchronization. It manages shared resources through mutual exclusion so it is more effective than interrupt or scheduling locks. There are three kinds of semaphores.

Binary semaphore: a binary semaphore is a semaphore which can take one of only two values (usually values 0 and 1). The binary representation is all that is required to provide mutual exclusion protection for a critical section. Binary semaphores are therefore the most commonly used semaphore forms.

Mutual exclusion semaphore: as a special case of binary semaphore, it serves as a key to access a shared resource.

Counting semaphore: a counting semaphore is a semaphore that can take any value between 0 and an upper integer value. Counting semaphores are normally used for synchronization purposes when counting out resources in use.

D. Inter-task communications

The proposed RTOS provides various inter-task communication methods such as: shared memory which uses global variables; semaphores which provide synchronization and mutual exclusion against access to shared resources; message port and task port which deliver messages among tasks.

When a message is sent to a task or tasks, the message port delivers a message pointer to reduce copy overhead for delivering message (see Fig. 5).

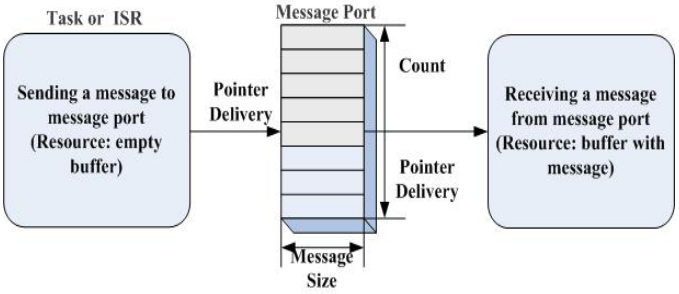


Fig. 5. Message port.

The task port provides one-to-one communication among tasks (see Fig. 6). It is implemented using a field of the task control block (TCB) rather than a separate data structure, and it is used for sending short messages among tasks.

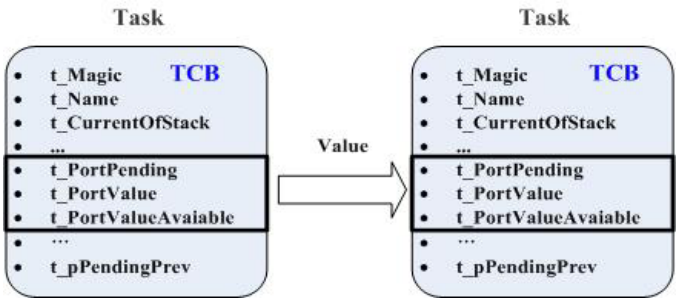


Fig.6. Task port.

E. Low-power management

The low-power management ported on the proposed RTOS is composed of the dynamic power management and device power management (DPM).

As shown in Fig. 7, the proposed RTOS provides a complete low-power management strategy to be defined in advance for each application by a system designer familiar with the characteristics of the embedded systems and its special features and requirements. A DPM policy is a named data structure, installed into the DPM implementation in the operating system, and managed by a policy manager. DPM policy managers are executable programs that activate policies by name. Policy managers may be very active, responding in real time to changes in application power/performance requirements, or may be more passive, for example by changing policies on a longer timescale in response to changes in available battery power.

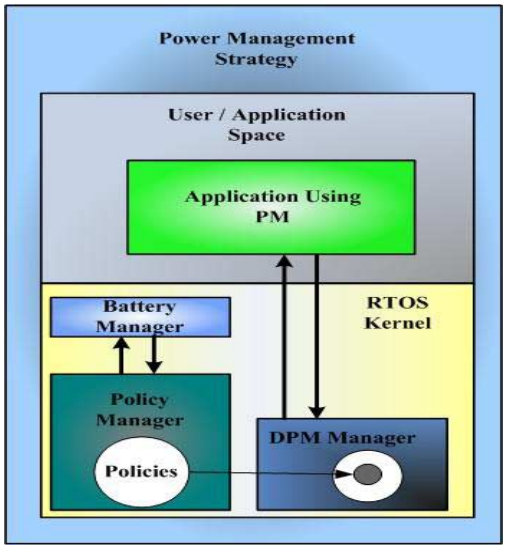


Fig. 7. Power management of the RTOS.

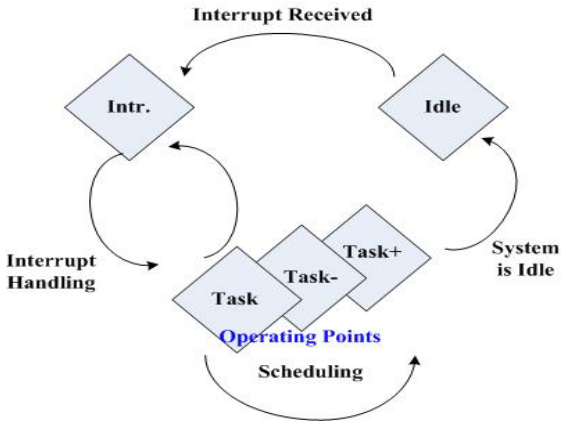


Fig. 8. Operating state transition diagram.

In power management, state transitions (see Fig.8) are controlled by commands issued by the DPM manager that observes the workload and the battery charge state of the system and decides when and how to force power state transitions. In the first, the power manager makes CPU operating frequency and voltage level decisions according to the battery charge state (see Table 2). And then, the power manager makes state transition decisions according to the power management policy.

TABLE 2
CPU OPERATING FREQUENCY AND VOLTAGE LEVEL STRATEGY BASED ON THE WORKLOAD AND THE BATTERY CHARGE STATE OF POWER MANAGEMENT

	Workload High	Workload Mid	Workload Low
Battery High	Highest CPU frequency and voltage	Highest CPU frequency and voltage	Highest CPU frequency and voltage
Battery Low	High CPU frequency and voltage	Middle CPU frequency and voltage	Low CPU frequency and voltage
Battery Critical	Middle CPU frequency and voltage	Low CPU frequency and voltage	Lowest CPU frequency and voltage

IV. EVALUATION AND MEASUREMENTS

To evaluate the performance, we have implemented the proposed RTOS on the ARC wearable device and measured some of the primitive real-time operating system performance characteristics.

One of the significant overheads introduced by a real-time operating system is the scheduling overhead. Fig. 9 shows the scheduling overheads of the proposed RTOS and μC/OS. Here, the scheduling overhead is the sum of the context switch time, the time to insert a task into the ready queue, and the time to select the highest-priority task. As shown in Fig.9, the overhead of the real-time scheduler of the proposed RTOS is lower than that of μC/OS.

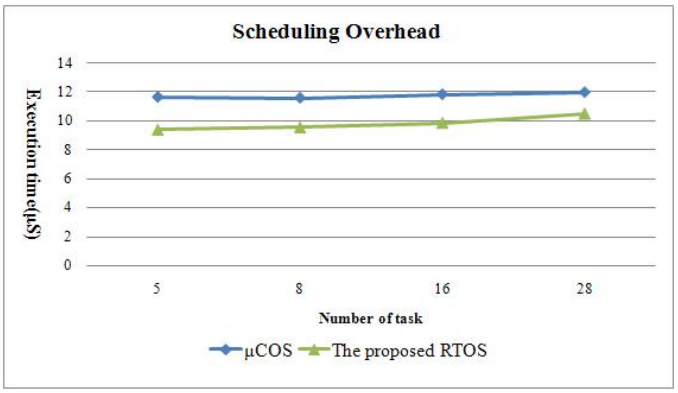


Fig.9.Scheduling overheads.

We also measured the inter-task communication overhead. The proposed RTOS provides two kinds of inter-task communication mechanisms: message ports and task ports. They both transfer just the pointer of each message, thus their overheads do not depend on the message size. The overhead s(i.e., the inter-task communication times) of the message port and the task port are 357.8μs and 48.53μs, respectively. The overhead of the task port is much lower than that of the message port since it makes use of a field in the TCB data structure. Note that, however, the task port can transfer only one message for each communication round.

To evaluate power efficiency of the proposed RTOS, we executed two ARC applications (the gesture recognition with finger tab and the clock application programs), and measured the currebt levels of the ARC using a data acquisition device (sampling frequence : 1500 samples/sec) and a variable voltage supply. The experimentation environment and the DPM policies used for these applications are shown in Fig, 10 and Table 3, respectively.

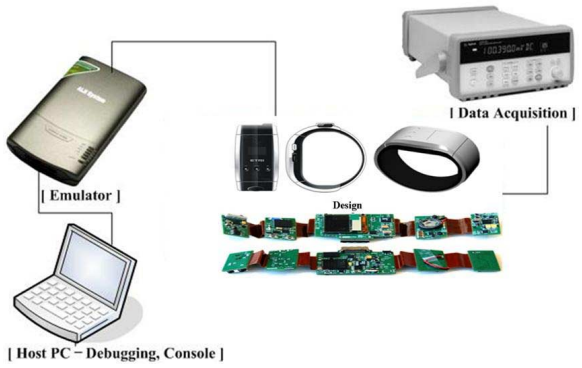


Fig. 10. ARC hardware and experimentation environment

The ARC platform is composed of a ARM926-EJ processor (MAX : 266MHz, MIN : 66MHz)[30], 1-MB SDRAM, an OLED display, 4-MB NAND flash memory, USB and other peripherals. Fig.11 shows the current level profiles for the corresponding policies and applications. The average current levels are summarized in Table 4.

Since the supplying power at each instant is proportional to the current level at that instant, the total energy consumed during a given time interval is proportional to the average current level during the interval. Therefore, Table 4 demonstrates that, using the low-power policy (such as Battery Low and Battery Critical), we could save energy consumption by about 42% and 47% for the ARC application, respectively.

Consequently, we could know that the proposed RTOS with small footprint and low-power management techniques is especially suitable for wearable consumer electronic devices such as ARC considered here.

V. Conclusion

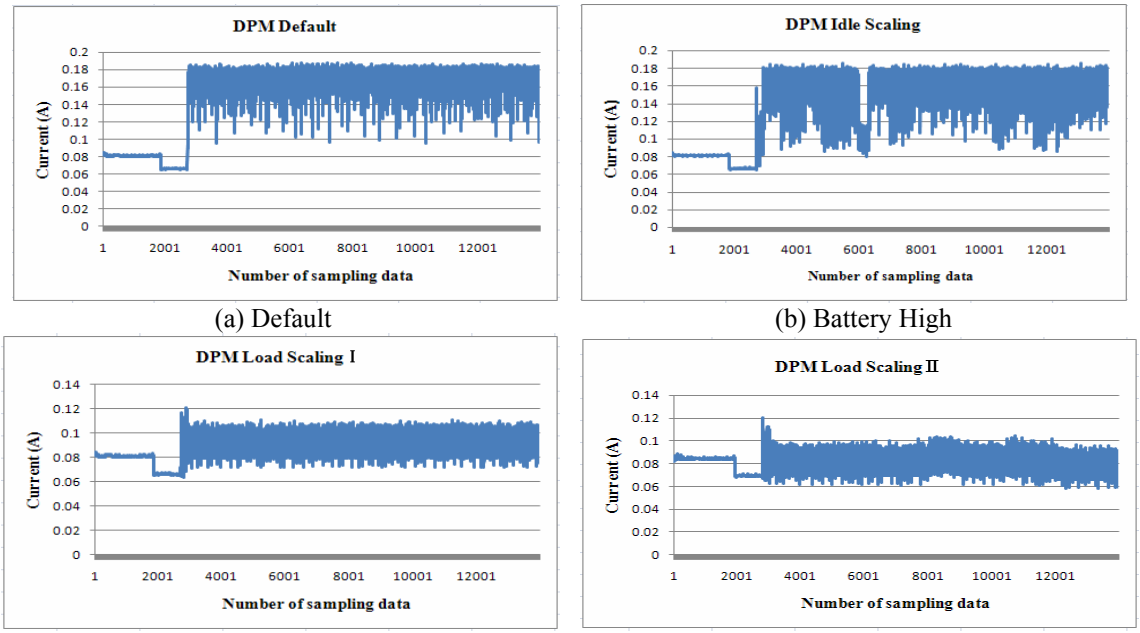
Wearable computers have hardware constraints such as small-sized memory, limited battery life time, and low-processing power. To overcome these constraints, wearable computers must use a small-footprint real-time operating system equipped with power management mechanisms.

In this paper, we proposed a novel RTOS for wearable consumer electronic devices and introduced major features of it. The footprint of the proposed RTOS is about 9KB meaning that it is quite efficient for a wearable consumer electronic device such as ARC considered in this paper, The RTOS is also equipped with two power management mechanisms: the dynamic power management and the device power management mechanisms. Experimental results using wearable applications show that the proposed RTOS could achieve energy savings up to 47%.

In the future, we would like to further adjust the proposed RTOS for other task sets typically found on wearable applications. Also, we would like to explore memory-aware low-power techniques and power-aware OLED management mechanism for wearable consumer electronic devices.

Table 3
CPU operating frequency and voltage level strategy

	Gesture task	Fingertap task	Clock task	Idle task
	Workload High	Workload Mid	Workload Low	Idle
Default	266/133(1.6V)	266/133(1.6V)	266/133(1.6V)	266/133(1.6V)
Battery High (Idle scaling)	266/133(1.6V)	266/133(1.6V)	266/133(1.6V)	133/66(1.5V)
Battery Low (Load Scalingl)	133/88(1.525V)	133/66(1.5V)	133/33(1.45V)	133/16(1.45V)
Battery Critical (Load Scalingll)	133/33(1.45V)	133/16(1.45V)	88/33(1.425V)	66/16(1.425V)



(a) Default (b) Battery High

(c) Battery Low (d) Battery Critical

Fig. 11. Current level profiles

TABLE 4
AVERAGE CURRENT VALUES.

	default	Battery High	Battery Low	Battery Critical
Average current	0.146	0.140	0.085	0.077
Energy saving	0%	4%	42%	47%

REFERENCES

[1] K. G. Shin and P. Ramanathan, "Real-time computing: a new discipline of computer shtcience and engineering," Proc. of the IEEE, vol. 82, no. 1, pp. 6-24, Jan. 1994.

[2] G.Kortuem, Z. Segall, and M. Bauer, "Context-Aware, Adaptive Wearable Computers as Remote Interfaces to Intelligent Environments," Proc. IEEE Intl., Symp., Wearable Computers(ISWC), IEEE CS Press, pp.58-65, 1998.

[3] T. Starner, "Augmented Reality Through Wearable Computing," Presence, vol.6, no. 4, pp. 386-398, 1997.

[4] M. Bilinghurst, "A Wearable Spatial Conferencing Space," Proc. Intl., Symp., Wearable Computers, IEEE CS Press, pp. 76-83, 1998.

[5] S.Fickas, "When Cborgs Meet: Building Communities of Cooperating Wearable Agents," Proc. Intl., Symp., Wearable Computers, IEEE CS Press, pp. 124-132, 1999.

[6] T. Trarner, "The challenges of wearable computing: Part 1", Micro, IEEE, Vol. 21, pp.44-52, 2001.

[7] T. Trarner, "The challenges of wearable computing: Part 2", Micro, IEEE, Vol. 21, pp.54-67, 2001.

[8] D. W. Lee, et al., "Actual Remote Control: A Universal Remote Control using Hand Motions on a Virtual Menu," IEEE Transactions on Consumer Electronics, Vol.55, No. 3, pp.1439-1446 , August 2009

[9] L. M. Thompson, "Using pSOS+ for embedded real-time computing," in COMPCON, pp. 282-288, 1990.

[10] D. Hildebrand, "An architectural overview of QNX," in Proc. Usenix Workshop on Micro-Kernels and Other Kernel Architectures, Apr. 1992.

[11] Nucleus, http://www.atinucleus.com

[12] RTOS Analysis, http:\\www.realtime-info.be

[13] Jean J. Labrosse, µC/OS: The Real-Time Kernel, R&D Publications, Lawrence, 1993.

[14] Jean J. Labrosse, "µ C/OS II: The Real-Time Kernel 2nd Edition", R&D Publications, Lawrence, 2002.

[15] Marcus T. Schmitz, Bashir M. and Al-Hashimi, System-Level Design Techniques for Energy-Efficient Embedded Systems, Kluwer academic publishers, Boston, 2004.

[16] IBM and MontaVista Software, "Dynamic Power Management for embedded systems,", Nov. 2002.

[17] B. Brock and K. Rajamani, "Dynamic Power Management for Embedded Systems", Proc. of IEEE Int' l SoC Conf. (SoCC 2003), pp.416-419, Sep. 2003.

[18] Wang Yue, Zhao Xia, and Chen Xiangqun, "A Task-Specific Approach to Dynamic Device Power Management for Embedded System," in ICESS'05, Vol.00, pp.158-165, 2005.

[19] L. Benini and G. De Micheli, Dynamic Power management: design techniques and CAD tools, Kluwer academic publishers, 1997

[20] Ch.-H Hwang and A. Wu, "A Predictive System Shutdown Method for Energy Saving of Event-Driven Computation", in Proceeding of the Int,l Conference on Computer Aided Design, pp. 28-32, 1997.

[21] M. Srivastava, A. Chandrakasan, R. Brodersen, "Predictive system shutdown and other architectural techniques for energy efficient programmable computation," IEEE Transactions on VLSI Systems, vol. 4, no. 1, pp.42-55, 1996.

[22] R.Emst and W. ye, “Embedded Program timing Analysis Based on Path Clustering and Architecture Classification,” Proc. Int’l Conf. Computer-Aided Design (ICCAD’97), pp. 598-904, 1997.
[23] P. Pillai and K. G. Shin, “Real-time dynamic voltage scaling for low-power embedded operating systems,” in Proc, 18th ACM symp. Operating Systems Principles (SOSP), pp.89-102, 2001.
[24] S. Lee and T. Sakurai, “Run-time voltage hopping for low-power real-time systems,” in Proc. Design Automation Conf., pp.806-809, 2000.
[25] S.-J. Oh, et al., “Deterministic Task Scheduling for Embedded Real-Time Operating Systems”, IEICE Trans. Inf. & Syst., Vol. E87-D, No. 2 , pp. 123-126, Feb. 2004.
[26] M. J. Jung, M. H. Cho, Y. H. Kim, C. H. Lee, “Generalized Deterministic Task Scheduling algorithm for Embedded Real-Time Operating Systems”, in Proc. The 2006 International Conference on Embedded Systems & Applications ESA’06, pp.79-82, June. 2006
[27] T.Ogasawara, "An algorithm with constant execution time for dynamic storage allocation", 2nd International Workshop on Real-Time Computing Systems and Applications, pp.21-27, 1995.
[28] I.Puaut, "Real-Time Performance of Dynamic Memory Allocation Algorithms", 14th Euromicro Conference on Real-Time Systems(ECRTS'02), pp.41-48, 2002.
[29] P.R.Wilson, M.S.Johnstone, M.Neely, "Dynamic Storage Allocation:A Survey and Critical Review", IWMM'95, pp1-116, 1995.
[30] i.MX21 Application Processor Reference Manual, 2, Freescale, 2005.

BIOGRAPHIES

Moon-Haeng Cho received the BS and MS degrees in computer engineering from Chungnam National University, Daejeon, S.Korea in 2004 and 2006. Since 2006, he will receive the Ph.D. Degree in System Software Laboratory in Chungnam National University. His research interests include embedded system, real-time and ubiquitous computing, small and low-power real-time operating systems.

Cheol-Hoon Lee received the B.S. degree in electrical engineering from Seoul National University, Seoul, Korea, in 1983 and the M.S. and Ph. D. degrees in computer engineering from KAIST, Daejeon, Korea, in 1988 and 1992, respectively. Since 1995, he has served Chungnam National University as a professor. His research interests include parallel processing, operating system, real-time kernel, and fault tolerant computing.