System Level Power Management for Embedded RTOS: An Object Oriented Approach

Ankur Agarwal ankur@cse.fau.edu
Department of Computer Science and Engineering
Florida Atlantic University
Boca Raton, FL 33431, USA

Eduardo Fernandez ed@cse.fau.edu
Department of Computer Science and Engineering
Florida Atlantic University
Boca Raton, FL 33431, USA

Abstract

Power management systems for embedded devices can be developed in real-time operating system (RTOS) or in applications. If power management policies are applied in operating system (OS), then designers and developers will not have to worry about complex power management algorithms and techniques. They can rather concentrate on application development. The OS contains specific and accurate information about the various tasks being executed. An RTOS further has a comprehensive set of power management application programming interfaces (APIs) for both device drivers and applications within a power management component. Therefore, it is logical to place policies and algorithms in the OS that can place components not being used into lower power states. This can significantly reduce the system energy consumption. We present here an abstract model of a system power manager (PM), device power managers, and application power managers. We present relationship and interactions of these managers with each other using Unified Modeling Language (UML) class diagrams, sequence diagrams and state charts. We recommend that the PM must be implemented at the OS level in any embedded device. We also recommend the interfaces for interactions between PM and the devices power manager, as well as PM and application power manager. Device driver and application developers can easily use this object oriented approach to make the embedded system more power efficient, easy to maintain, and faster to develop.

Keywords: *Embedded Device, Object Oriented Design, Policy Manager, Real-Time Operating Systems, System Level Power management*

1. INTRODUCTION

Due to the nature of the use and blend of computationally extensive applications, power consumption is one of the major concerns in developing real-time OS devices [1, 2, 3]. There is always a need for longer battery life in order to avoid catastrophic data loss [4, 5]. The environmental impact of power consumption from electronic systems has raised serious concern [6]. Furthermore, excessive heat dissipation is a major obstacle in improving performance. Power management is widely employed to contain the energy consumption in power-constrained devices. The advancement in processor and display technology has far outpaced similar advancements in battery technology [4]. On the other hand, the battery capacity has

improved very slowly (a factor of two to four over the last 30 years), while the complexity of applications, the computational demands, and therefore the power needs have drastically increased over the same time frame. Power management techniques date back to 1989, when Intel shipped processors with the technology to allow the CPU to slow down, suspend, or shut down part or all of the system platform, or even the CPU itself, to preserve and extend the time between battery charges [7]. Since then, several power consumption strategies have been developed [8, 9, 10, 11, 12, 13, 14].

Figure 1 shows the embedded processor performance and power consumption evolution for the data in Table 1. It should be noted that higher MHz/mW implies higher efficiency. Thus, power management techniques have evolved along time. These techniques have been able to reduce (or keep constant) power consumption in portable devices.

Table 1: Embedded Processor performance and Power Consumption Data [15]

Processor	Intel StrongARM SA-1110	Intel Xscale PXA-250	Intel Xscale PXA-250	Intel Xscale PXA-250
Clock Speed (MHz)	206	200	300	400
Power Draw (mW)	800	256	411	563
MHz/mW	0.26	0.78	0.73	0.71

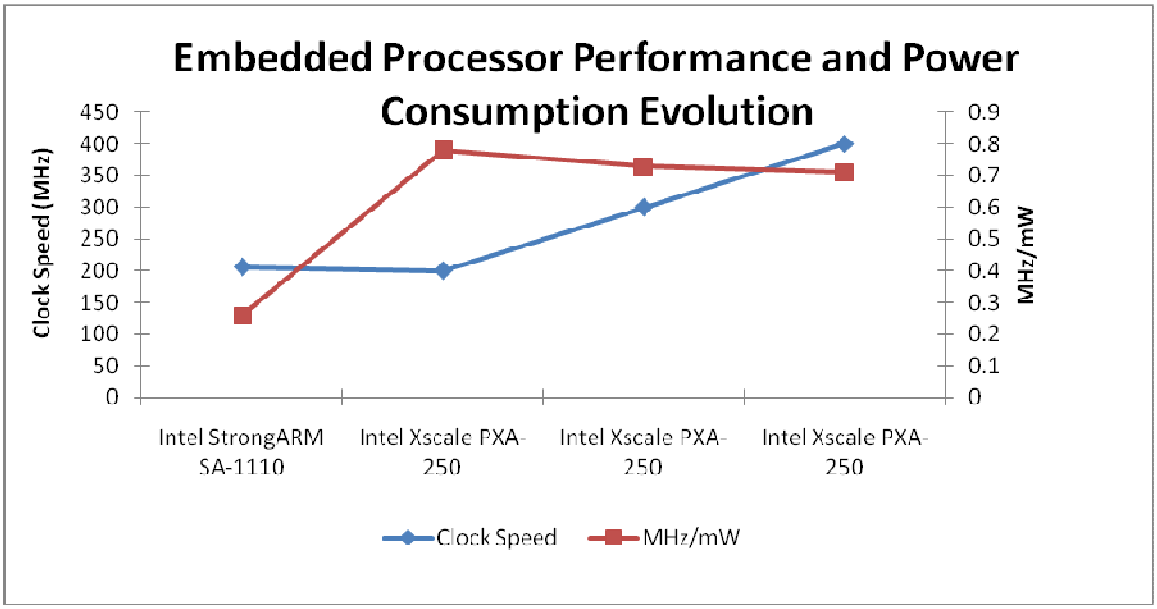

Figure 1: Embedded Processor performance and Power Consumption Evolution [15]

Power management has been a center of focus since the early 90's. Power management policies have been described at different levels of abstraction starting from the lowest level of abstraction: Transistor Level. Figure 2 shows the use of power management techniques at various levels of abstractions.

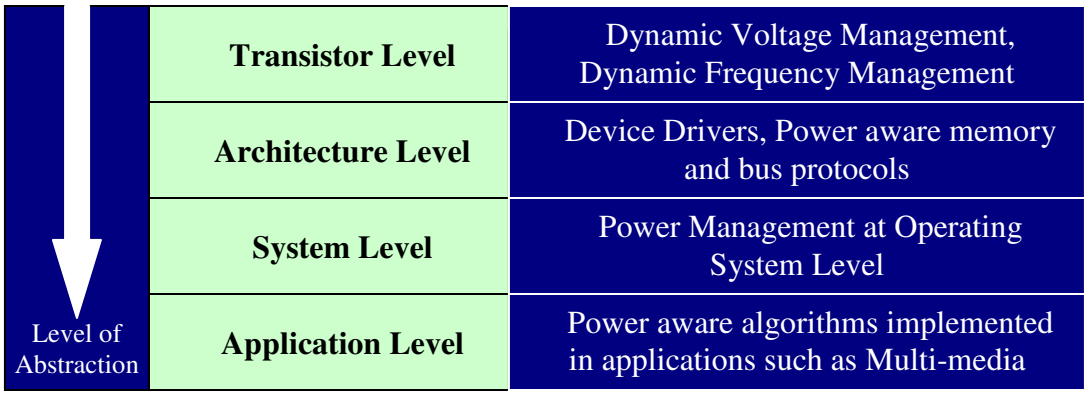

Figure 2: Power Management Schemes at Different Level of Abstractions

At the transistor, level dynamic voltage scaling (DVS) and dynamic frequency scaling have been exploited to reduce the power consumption in circuits at this level [16, 17]. Extensive research has been done on lower-levels such as transistor level, gate level and application level power management in the past few decades [18, 19]. Researchers have developed innovative bus protocols [20, 21], memory optimization algorithms [22, 23], and advancement in cache technology [24, 25] for managing power. Power management techniques in device drivers have resulted in power management techniques at architect's level of abstraction [26, 27, 28]. A system architect performs system level trade-offs by selecting an optimum architecture that satisfies power, area, cost, latency and other quality-of-service (QoS) and performance parameters [29, 30, 31]. Similarly, power management techniques have evolved at the application level [32, 33]. An application is usually given full control to choose the value of CPU power setting and parameters. The OS is then responsible for keeping track of all the power settings of different applications and applying them on different resources such as processors.

As per the Moore's law, the number of transistors on a chip doubles every eighteen months [34]. This has resulted in exponential increase in the complexity of embedded systems. Therefore, power control at lower levels, even though it is more accurate, becomes unfeasibly complex and is compounded by time-to-market pressures. Thus, with time the designs are now being described at higher levels of abstraction leading to the era of system level design, system-on-chip and networks-on-chip [30, 35]. Therefore, there is a strong need for specifying the power issues related to the design at the same level of abstraction, leading to the concept of system level power management. Moreover the application, semiconductor technology, cost, and time-to- market trends are causing a shift towards increased software content in embedded system and systems-on-chip. As a result, designers and users of embedded software must be increasingly aware of power issues. While power dissipation is inherently a property of the underlying system hardware, knowledge of embedded software that runs on the hardware is useful in order to analyze and improve system's power consumption characteristics. Modern OS not only contain precise information about the various tasks being executed but are also well developed with algorithms, that selectively place components into lower power states, thereby drastically reducing energy consumption [1]. However, the importance of reducing the power consumption in embedded OS has not been widely recognized and a large body of work has focused on estimating, managing, and reducing power consumption in various system components. An RTOS serves as an interface between the application software and the hardware. The embedded system design and its issues such as hardware resource management, memory management, process management and development of device drivers can be simplified by providing the designers with a well defined interface. With more features being supported by embedded systems, the applications and their development is becoming complex everyday. RTOS must provide a simple and encapsulated Application Programming Interface (API) so that the software remains portable across product users, product families and companies [36].

The motivation behind this paper is the need for the applications to provide well-defined interfaces between RTOS and device devices that can be used by the power manager to manage the overall power of the system. This paper also aims at providing a simple programming interface for the application developers to inform RTOS about applications' power and device requirements. Conversely, we also

recognize the need for RTOS to inform the application about the current battery status so that the application can keep user informed. Once the RTOS is aware of the power requirements, it should be able to bring the complete system into a lowest possible power state. For a simplified and well-encapsulated design, we provide an object-oriented representation for the power manager components that are embedded in the RTOS, device drivers and applications. We present encapsulated behavior of power management features in the form of classes and their interaction in the form of sequence diagrams. A proper graphical representation of these complex power management processes can give the user a capability to manage the complexity, tight performance and power constraints in the system. These features can then be used by the different application and device vendors to evolve test cases for verifying the compatibility among the various devices and the applications being used with this OS in their prototype.

Section 2 provides typical software architecture of power manager in an RTOS. Section 3 discusses the power states in the RTOS. Section 4 describes the proposed model for power management in RTOS. Section 5 concludes this research.

2. SOFTWARE ARCHITECTURE of POWER MANAGER in RTOS

Figure 3 shows the layered software architecture of an RTOS. The top layer consists of user defined applications and OS supported applications. OS supported application include user interfaces and client-services. Embedded system designers and developers design their applications and link them with some dynamic link library (Core DLL library). Multimedia, networking, and other applications, along with device drivers are supported at the OS layer. Below this layer there is OS kernel that handles the major OS functionality such as scheduling, task management, memory management, I/O management among other. Under this layer, there is a original equipment manufacture (OEM) layer.

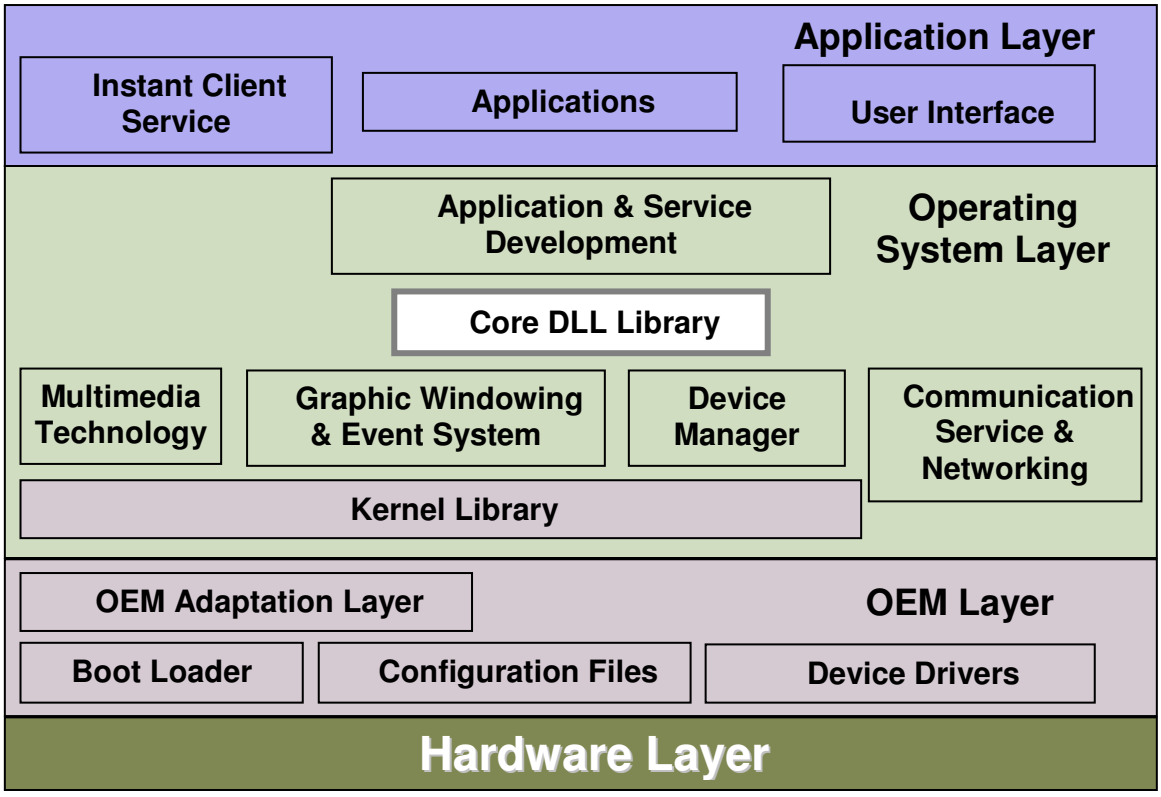

Figure 3: Layered Software Architecture of OS

OEM layer usually supports all the OEM hardware and contain device drivers for various hardware devices originally supported by OS. This layer also contains boot loader and configuration manager which loads OS every time we start this system. This part of the OS is write-protected and application/device/embedded system developers are not allowed to alter the content of this layer.

We have abstracted power management functionality of an OS in Figure 4. An integral part of OS-level power management is the policy manager. Policy Manager is responsible for the efficient management of the various applications and devices running on the portable system. This is one typical scenario of OS and is very similar to WindowsCE.NET. Other RTOS may or may not follow the similar architecture. Figure 4 shows the interaction of devices, applications and battery with the power manager (PM). The PM acts as a mediator between devices, applications, and the processor. Information from the various interfaces about their power status is then collected and managed by PM. On the basis of the collected information, the entire system is put into the lowest possible power state for a particular application.

The PM also co-ordinates the different devices, system and processor states to decrease the overall system power consumed. While maintaining critical resources in the system and monitoring processor utilization to ensure its operation at the lowest possible state, it also provides a means for the drivers to inter-communicate about their power states. The Software architecture of the PM is responsible for providing the services to the device drivers of the system, to get notified, and respond upon power changes.

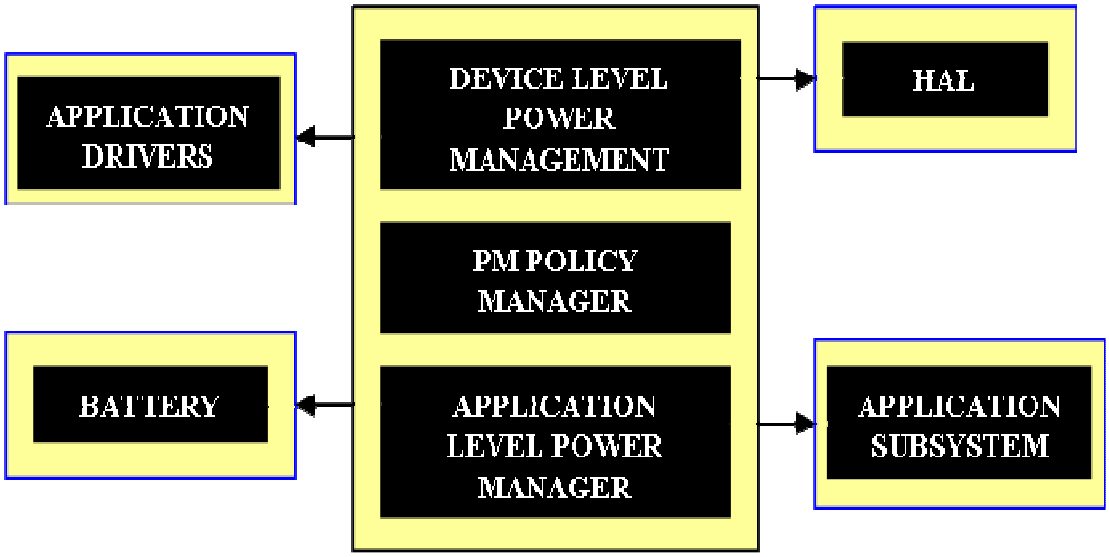

Figure 4: Interaction of power manager with devices and applications

Power management with power manageable hardware comprises one or more layers of software. Hardware specific power management software and operating system policy manager, which are in between the hardware independent software interface, are also defined. This creates a layered cooperative environment through software interfaces, and allows applications, operating systems, device drivers and the PM to work together, thereby reducing the power consumed. The higher-level application software is therefore able to use PM without any inkling of the hardware interface as the details of the hardware are masked by the PM. This leads to increased usage due to extended system battery life.

3. POWER STATES

In any system, under certain circumstances some hardware components are always in an idle state. This applies to the devices, to the processor and to the applications. Under these conditions, when no task is

being executed, the particular device or application can be put into a lower power state. In an RTOS, there are different power states associated with applications, devices and the processor. Thus, it is possible that a processor may be in sleeping state, a device in soft off and an application in full-on state. Figure 5 shows the state transitions for a complete system. Power managed devices receive power state change notifications in the form of I/O control codes (IOCTLs). This diagram separates device power states from the system power states.

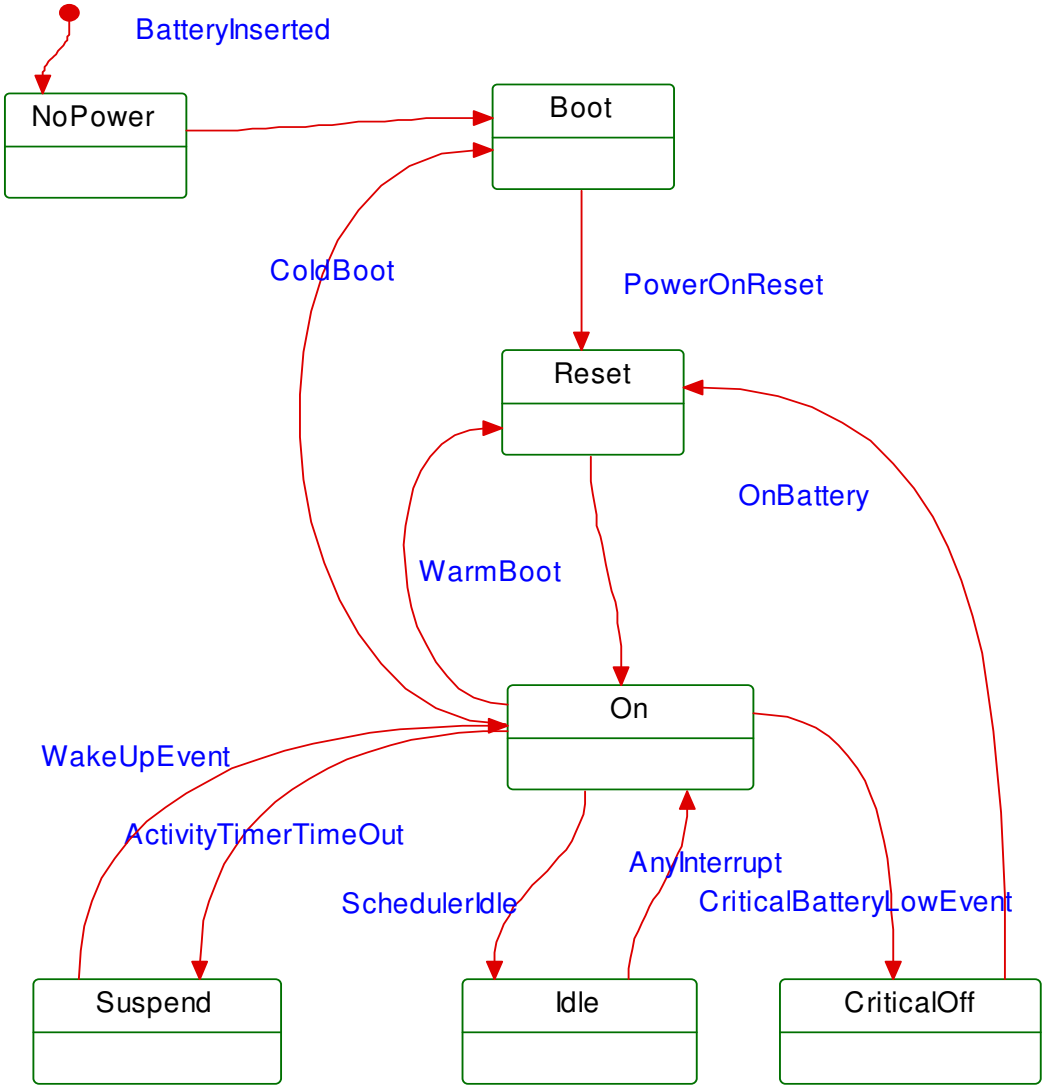

Figure 5: System Power States

There are seven predefined power states in a typical system. 1) The system is said to be in the “NoPower” state (S0), when the system has no power. 2) Upon the insertion of the battery the system moves to “Boot” state (S1). 3) The “On” state (S2) used for the normal operation, is a state in which the system dispatches user mode (application) threads for execution. Dynamic Frequency Management (DFM) and Dynamic Voltage Management (DVM) optimizations are done in this state. 4) The “Idle” state (S3) is a low-wake-latency sleep state. In this state, system context is maintained by the hardware while there is no loss of system context in the CPU or peripheral devices. Upon detection of some events such as user activity, the system moves back to the On state. 5) The “Suspend” state (S4), is a low-wake-latency sleep state where all system contexts are lost except system memory. 6) The “Critical Off” State

(S5) is a non-volatile sleep state. In this state system context is saved and restored when needed. Here the operating system saves the necessary information in the non-volatile memory and tags the corresponding context markers. 7) Finally, the "Reset" state (S6)is also referred to as soft restart state. In this state the system contexts are properly saved before being lost. All the data and other user information are also stored in memory before the system restarts itself. These system states are shown in Figure 5. This state diagram can also be considered as the state transition diagram of the active class [37] Policy Manager discussed in the next section.

A system may be in any of the above-explained seven states depending upon the task, which is being executed on the device. However, transition among states can consume some time depending on the thread executed by the PM.

The PM makes state transition decisions according to same power management policy, which is discussed in the next section. The power states of devices and processor are based on the same structure as that of the system. However, different devices may be in different power states, while the system and processor are in another state. For instance, a device can be in the Off state while the system is in the On state.

4. A MODEL for POWER MANAGEMENT

We have developed a UML model for power management features. The model uses three Manager (controller) classes that coordinate the basic power related functions.

4.1. Policy Manager

The Policy Manager constantly monitors battery state. It also orchestrates all system-level and device-level state changes. These notifications are passed to applications polymorphically using the interface specified in the ApplicationDriverPowerManager (APM) and to the devices using the interface specified in DeviceDriverPowerManager (DDPM). On detecting a low battery state, the Policy Manager decides to force the system into the Idle state. It sends a notification to the APM and DDPM, which in turn notifies all application and drivers registered with it respectively.

Figure 6 depicts the class diagram of power management in an RTOS. It can be seen from the figure that Policy Manager and Battery are singleton class whereas the DDPM and APM are abstract class associated with the Policy Manager. The relationships shown are used to notify drivers and applications about the various system level state changes. For example, when a new application is plugged in, the drivers are notified through the policy manager notification interface regarding the different system level state changes that may occur. The "BootInitilizer" class is for loading important application and drivers while the system is booted.

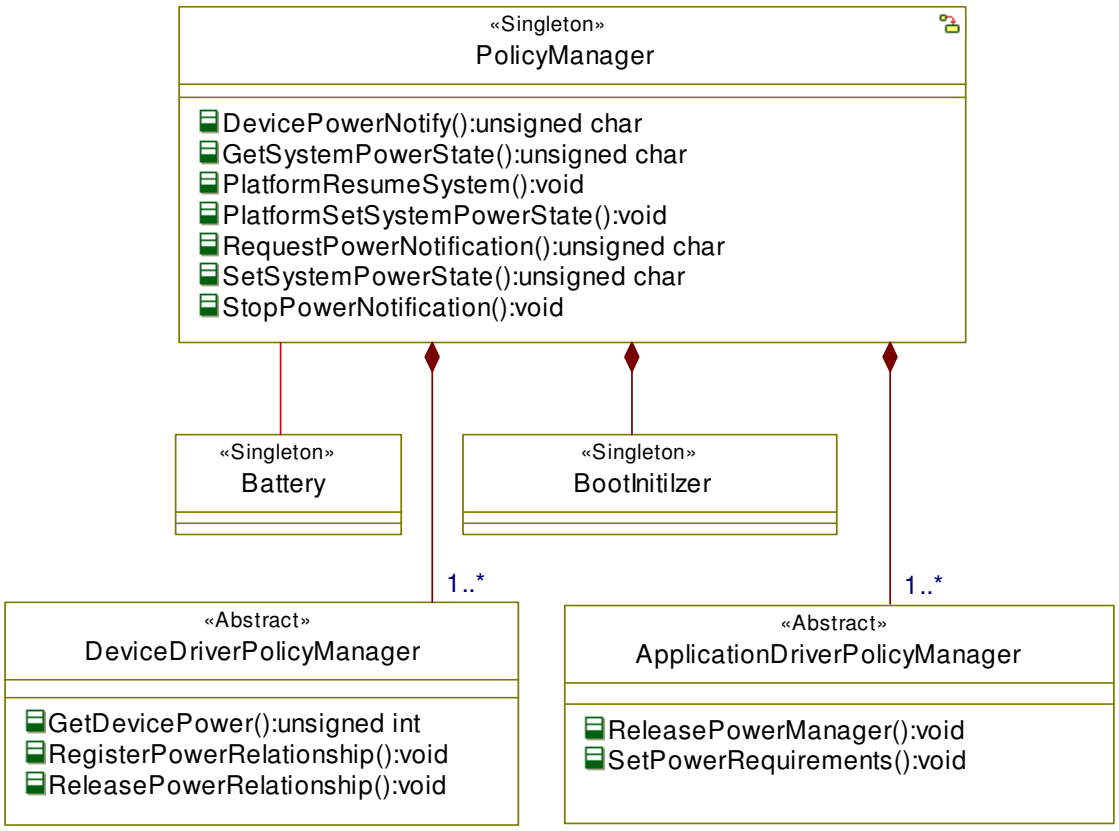

Figure 6: UML Class Diagram of Power Management

4.2. Device Driver Power Manager

Figure 7 depicts several possible concrete subclasses of the DeviceDriverPolicyManager, each associated with a specific device driver such as camera, keyboard, display, headset among others. These devices (referred as subclasses in UML) are presented here just as examples.

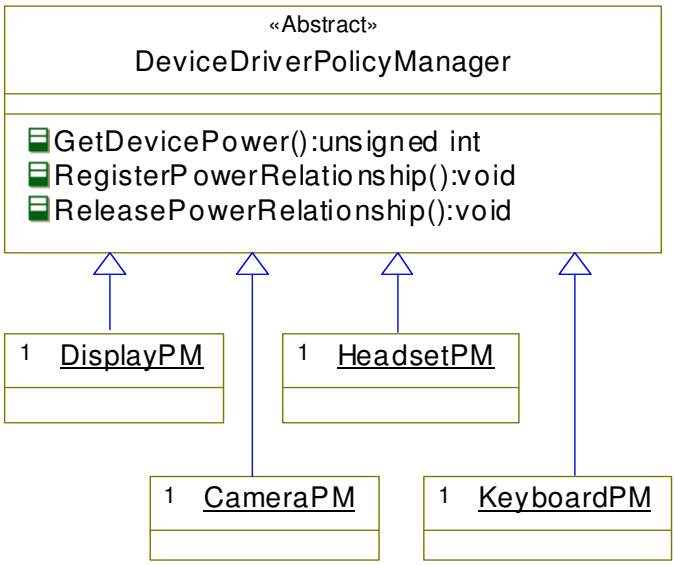

Figure 7: UML Class Diagram of DDPM

In a specific embedded system, some of these subclasses may not be present, but these subclasses may instead have other subclasses contained in them. Various applications, services and device drivers are notified upon the (dis)appearance of device interfaces by the DDPM Interface notification. This feature of RTOS can be regarded as similar to the plug and play of Windows OS. Using the device policy manager's interface, the PM can receive and set specific capabilities of the device driver. However, to be compatible with this power management framework, the device driver must support all the power management states. The DDPM's interaction with the Policy Manager is shown in the sequence diagram of Figure 8.

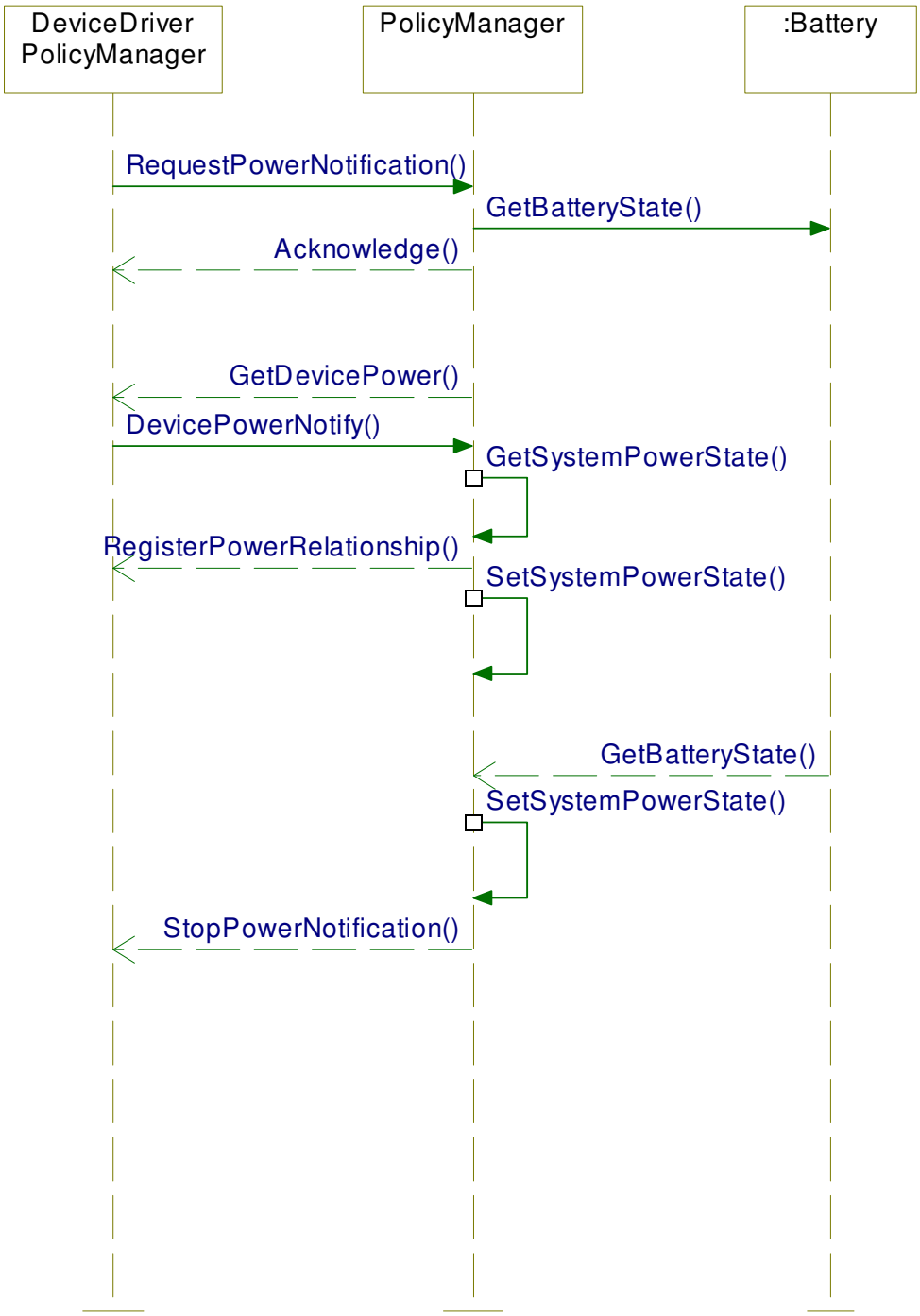

Figure 8: Sequence Diagram of Device Driver Interaction with the Power Manager

Initially, all the device drivers (DeviceDriverPowerManager) register themselves with the Policy Manager through RequestPowerNotification() and receive an acknowledgement. The PM reads a list of device classes from the registry and uses RequestPowerNotifications() to determine when devices of that class are loaded. In order to for a device to get activated in the system, the device finds out its current power state by GetDevicePower() and then notifies the policy manager to change its state. DevicePowerNotifiy() informs the device about the change in its power state. Once the power state of the device has been changed the system changes its previous power state to a new power state in order to accommodate the change in the power state of the device. The Policy Manager constantly monitors battery status. Upon detection of an idle state, the Policy Manager sends a query, to transfer into idle state. Upon the acceptance of the query by the device, the Policy Manager changes the system state. Concurrently, if the Policy Manager detects a low battery state, it notifies the DeviceDriverPolicyManager, to change the device state of all devices to idle. As the device state of the device changes to idle state, an acknowledgement is sent to the Policy Manager, which puts the system state to idle.

4.3. Application Driver Power Manager

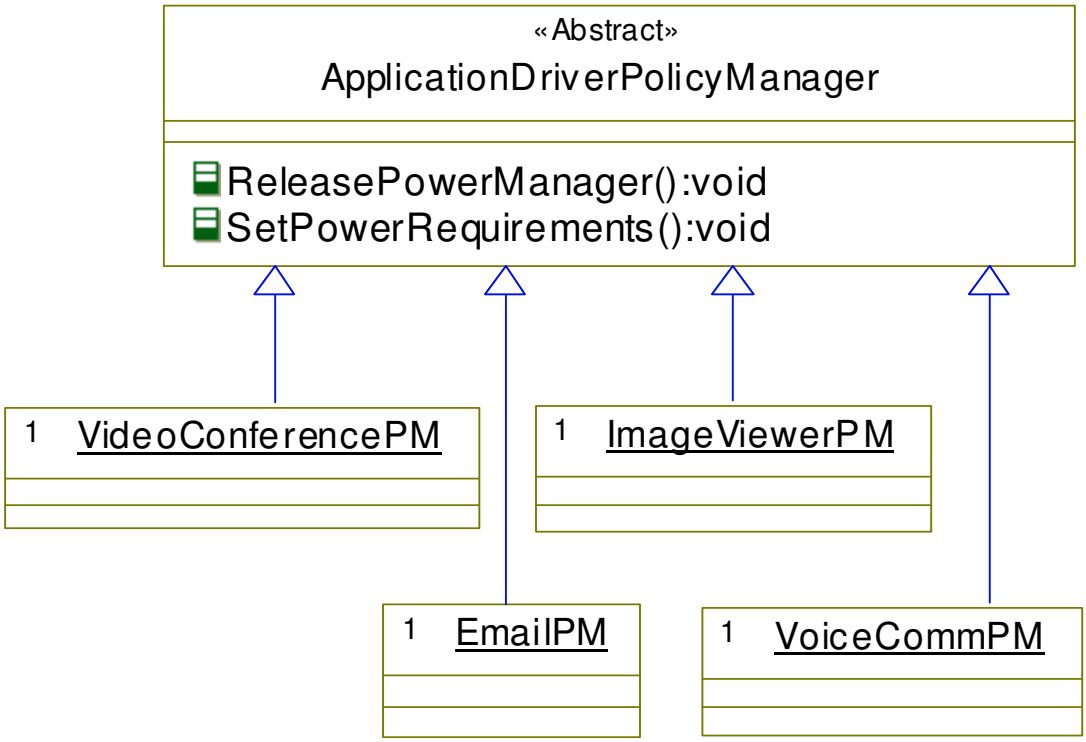

Figure 9: Class Diagram APM

Class diagram for the APM is shown in Figure 9. This Figure depicts some applications such as video conferencing, voice communication, email, audio, and multimedia among others, associated with APM class. Various system level power changes are notified to the applications using the interface specified by APM which in turn is specified by PM. Specific guidelines are set for devices, which applications can request to set system power level by means of DDPM. On specific devices or systems as a whole, a reduced power consuming state can be set if the applications request the power manager to change the device's power state. High-performance "power-smart" applications can also use battery status data provided by PM to provide the best experience for the user by lowering down performance (e.g., lowering frame-rates) in order to preserve battery.

Figure 10 shows the interaction of the APM with the PM and battery as a sequence diagram. Initially the applications register themselves (RequestPowerNotification()) with the PM and receive an acknowledgement. APM notifies the Policy Manager that an application has a specific device power requirement and sets is accordingly (SetPowerRequirement()). The application also requests the power notification for the specific device drivers it needs in order to execute. The system responds to its request by changing the power state of those device drivers (SetDevicePower()). Once the application power requirements are fulfilled the policy manager updates the system power state (GetSystemPowerState(), SetSystemPowerStstate()). The policy manager constantly monitors the battery status. Upon the detection of an idle state event, the APM gets the system power state (GetSystemPowerState()) and device power state (GetDevicePowerState()). Power states of drivers are then forced into sleep state by the Policy Manager. Concurrently, if the Policy Manager detects a low battery state, it notifies the ApplicationDriverPolicyManager, to change the application state of all applications to standby followed by the change in the system state to standby state.

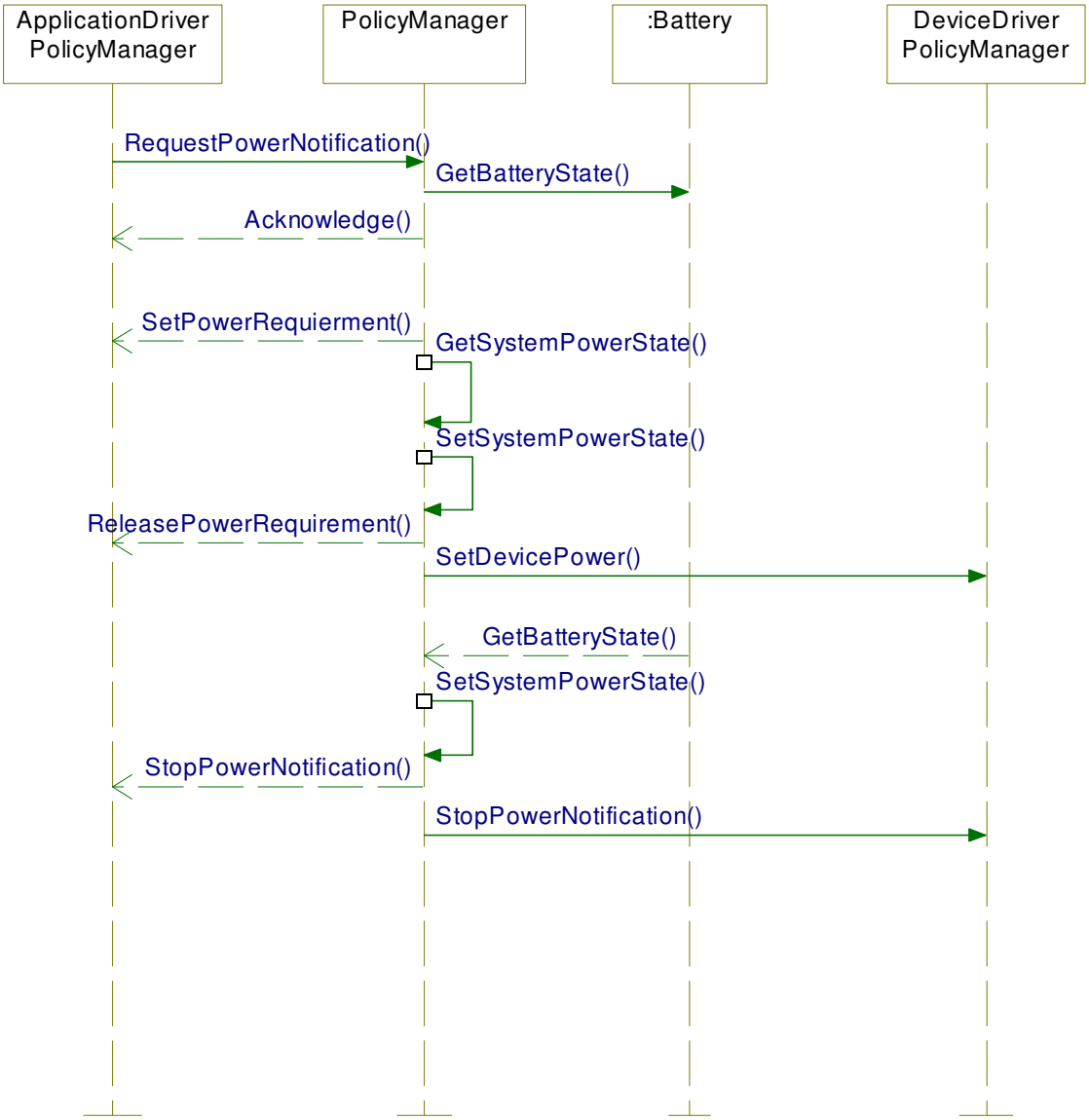

Figure 10: Sequence Diagram of Application Driver Power Manager Interaction with Power Manager

5. CONCLUSION

We have provided an abstract model for implementing power management in an embedded RTOS. System level power management can be designed using an OO approaches We recommend the interfaces for interactions between PM and the devices power manager, as well as PM and application power manager. We present relationships and interactions of these managers with each using UML class diagrams, sequence diagrams, and state charts. This abstract object-oriented representation of power management clarifies the operation of the power manager in conjunction with applications, devices and the processors for the developers of applications and devices. If the operating system and the device drivers are designed using this framework, the task of managing power within applications can be significantly simplified resulting into longer battery life at run time. On the other hand, well defined interfaces also simplify the task of power management in device drivers so that a more granular power management strategy can be used, thus lowering overall power needs of the system. The object-oriented power management interface also simplifies testing for power management scenarios, thus making it possible to test for cases that were not tested earlier. It is our hope that the proposed framework will foster development of embedded systems that are more power efficient, easy to maintain, and faster to develop.

6. REFERENCES

[1] A. Agarwal, S. Rajput, A. S. Pandya, "*Power management System for Embedded RTOS: An Object Oriented Approach*", IEEE Canadian Conference on Electrical and Computer Engineering, May 2006, pp. 2305-2309

[2] Y. H. Lu, L. Benini, G. D. Micheli, "*Power aware operating systems for interactive systems*", IEEE Transactions on Very Large Scale Integration Systems, Volume 10, Issue 2, April 2002, pp. 119-134

[3] S. Vaddagiri, A.K. Santhanam, V. Sukthankar, and M. Iyer, "*Power management in linux-based systems*," Linux Journal, March 2004, http://www.linuxjournal.com/article.php?sid=6699.

[4] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, H. Franke, "*Reducing disk power consumption in servers with drpm*" IEEE Computer, Volume 36, Issue 12, December 2003, pp. 59-66

[5] "*Adaptive Power Management for Mobile Hard Drives*", IBM, April 1999, http://www.almaden.ibm.com/almaden/pbwhitepaper.pdf

[6] H.S. Choi, D. Y. Huh, "*Power Electronics Specialists Conference 2005*", 36th IEEE Power Electronics Specialists Conference, 2005, pp. 2817-2822

[7] L. Benini, R. Hodgson., P. Siegel, "*System-level power estimation and optimization*", International Symposium on Low Power Electronics and Design, IEEE Conference, pp. 173-178, August 1998.

[8] L.S. Brakmo, D.A. Wallach, M.A. Viredaz, "*uSleep: A technique for reducing energy consumption in handheld device*", 2nd IEEE International Conference Proceeding on Mobile Systems Applications, and Services, June 2004.

[9] Y.-H. Lu, G. D. Micheli, "*Comparing system-level power management policies*", IEEE Design & Test of Computers special issue on Dynamic Power Management of Electronic Systems, pages 10–19, March/April 2001.

[10] Sinha A., Chandrakasan A. P., "*Energy efficient real-time scheduling [Microprocessors]*", *IEEE International Conference of Computer Aided Design*, pp. 458-463, November 2001.

[11] Y.-H. Lu, E.-Y. Chung, T. ˇSimuni´c, L. Benini, G. D. Micheli, "*Quantitative comparison of power management algorithms*", IEEE Conference on Design Automation and Test in Europe, March 2000, pp. 20-26.

[12] Q. Qiu, Q. Wu, M. Pedram, "*Dynamic power management in mobile multimedia system with guaranteed quality-of-service*", IEEE Design Automation Conference, Vol. 49, pp 834-849. June 2001.

[13] Udani, S., Smith, J., "*Power management in mobile computing*", Department of Computer Information Sciences, Technical Report, University of Pennsylvania, February 1998.

[14] Yung-Hsiang Lu and Giovanni De Micheli, "*Adaptive hard disk power management on personal computers*", Great Lakes Symposium on VLSI, 1999, pp. 50–53.

[15] "Embedded Processor Performance Parameter Data-Sheet"www.intel.com

[16] Burd T. D., Brodersen R. W., "*Energy efficient CMOS microprocessor design*" In Proceedings of the 28th Annual Hawaii International Conference on System Sciences. Volume 1: Architecture, IEEE Computer Society Press, 1995, pp. 288-297.
[17] Zimmermann, R., and Fichtner, W., "*Low power logic styles: CMOS versus pass-transistor logic*", IEEE Journal of Solid State Circuits, Vol. 32, pp. 1079-1089.
[18] A. Weissel, F. Bellosa, "*Process cruise control: event-driven clock scaling for dynamic power management*", IEEE Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, October 2002.
[19] D. Grunwald, P. Levis, C. Morrey III, M. Neufeld, K. Farkas, "*Policies for dynamic clock scheduling*", Symposium on Operating Systems Design and Implementation, October 2000, pp 78-86
[20] K. Lahiri, A. Raghunathan, "*Power analysis of system-level on-chip communication architectures*", IEEE International Conference on Hardware/Software Codesign and System Synthesis (CODES + ISSS), 2004, pp. 236-241.
[21] Y. J. Lu, A. C. W. Wai, L. L. Wei Fan, B. K. Lok, P. Hyunjeong, K. Joungho, "*Hybrid analytical modeling method for split power bus in multilayer package*", IEEE Transactions on Electromagnetic Compatibility, Volume 48, Issue 1, February 2006, pp. 82-94.
[22] C. Le, P. Nathaniel, L. H. Yung, "*Joint power management of memory and disk under performance constraints*", IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, Volume 25, issue 12, December 2006, pp. 2697-2711.
[23] M. Fukashi, H. Isamu, G. Takayuki, N. Hideyuki, I. Takashi, S. Hiroki, D. Katsumi, A. Kazutami, "*A configurable enhanced TTRAM macro for system-level power management unified memory*", IEEE Journal of Solid-State Circuits, Volume 42, Issue 4, April 2007, pp. 852-861.
[24] Q. Zhu, Y. Zhou, "*Power aware storage cache management*", IEEE Transactions on Computers, Volume 54, issue 5, May 2005, pp. 587-602.
[25] A. Sagahyroon, M. Karunaratne, "*Impact of cache optimization techniques on energy management*", IEEE Canadian Conference on Electrical and Computer Engineering, Volume 4, May 2004, pp. 1831-1833
[26] L. Benini, S. K. Shuklam, R. K. Gupta, "*Tutorial: architectural system level and protocol level techniques for power optimization for networked embedded systems*", 18th International Conference on VLSI Design, January 2005, pp. 18
[27] C. Grelu, N. Baboux, .A. Bianchi, C. Plossu, "*Low switching losses devices architectures for power management applications integrated in a low cost 0.13/spl mu/m CMOS technology*", 35th IEEE Proceedings of Solid-State Device research Conference, September 2005, pp 477-480
[28] J. Byoun, P. L. Chapman, "*Central power management unit as portable power management architecture based on true digital control*", IEEE Workshop on Computers in Power Electronics, August 2004, pp. 69-73
[29] L. Benini, A. Bogliolo, G. D. Micheli, "*A survey of design techniques for system-level dynamic power management*", IEEE Transactions on very large Scale Integration Systems, Volume 8, Issue 3, June 2000, pp. 299-316
[30] T. Simunic, S. P. Boyd, P. Glynn, "*Managing power consumption in network on chips*", IEEE Transactions on Very large Scale Systems, Volume 12, Issue 1, January 2004, pp. 96-107
[31] D. Monticelli, "*System approaches to power management*", 17th Annual IEEE Conference and Exposition on Applied Power Electronics, Volume 1, March 2002, pp. 3-7
[32] Y. H. Lu, T. Simuni´c, G. D. Micheli, "*Software controlled power management*", IEEE International Workshop on Hardware/Software Codesign, 1999, pp. 157–161.
[33] R. M. Passos, C. J. N. Coelho, A. A. F. Loureiro, R. A. F Minim, "*Dynamic power management in wireless sensor networks: an application-driven approach*", 2nd Annual IEEE Conference on Wireless on-demand Network Systems and Services, January 2005, pp. 109-118
[34] "Roadmap Architects – The Technology Working Groups" http://public.itrs.net
[35] S. Kumar, A. Jantsch, J-P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani, "*A network on chip architecture and design methodology*", IEEE Computer Society Annual symposium on VLSI, April 2002, 117-124
[36] Martin Timmerman, "RTOS Evaluations", 2000, http://www.dedicated-systems.com
[37] Jenson Douglas E., "*Real-time design pattern robust scalable architecture for real time systems*", Boston, Addition-Wesley, 2002.