POWER MANAGEMENT SYSTEM FOR EMBEDDED RTOS: AN OBJECT ORIENTED APPROACH

Ankur Agarwal,
CSE Dept. FAU,
Boca Raton, FL 33431
email: ankur@cse.fau.edu

Saeed Rajput,
CSE Dept. FAU,
Boca Raton, FL 33431
email : srajput@ieee.org

A. S. Pandya
CSE Dept. FAU,
Boca Raton, FL 33431
email: pandya@fau.edu

Abstract

Power management systems for embedded devices can be developed in operating system (OS) or in applications. If power management policy is applied in OS, then developers can concentrate only on application development. OS contains specific and accurate information about the various tasks being executed. Therefore, it is logical to place algorithms that place components not being used into lower power states. This can significantly reduces the energy consumption by the system. Real-time-operating-system (RTOS) has a comprehensive set of power management application programming interfaces (APIs) for both device drivers and applications within a power management component. In this paper, we provide abstracted concepts of a system power manager (PM), device power managers, and application power managers. We present relationship and interactions of these managers with each using unified modeling language (UML) class diagrams, sequence diagrams and state charts. We recommend that PM must be implemented at operating system 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*: Power management, policy manager, Embedded Device, Policy Manager, Object Oriented Approach*

1. Introduction

Many of the today's computers are small enough to sit in places where we do not see or think about them. For example, devices such as household appliances, cell phones, and car engines all contain embedded computers. Some of these computers are designed and embedded into devices to support real time applications. Due to the nature of usages and blend of computation extensive applications, power consumption is one of the major concerns in developing these devices [1]. There is always a need for longer battery life in order to avoid catastrophic data loss [2, 3]. 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 [5]. Since then, several power consumption strategies have been developed [1, 2, 3, 4, 5, 6].

Power management has been a center of focus since early 90's. Power management policies have been described at different level of abstractions starting from the lowest level of abstraction: Transistor Level. Dynamic voltage scaling (DVS) and dynamic frequency scaling have been exploited to reduce the power consumption in circuits at this level [6]. Extensive research has been done on lower-levels such as transistor level, gate level and application level power management in the past few decades [6, 7]. As per the Moore's law, number of transistors on a chip doubles in every eighteen months [8]. This has resulted in the 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 level of abstraction leading to the era of system level design, system-on-chip and networks-on-chip [9]. 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.

With the mushrooming convergence on the imminence of operating system in embedded systems, there has been a shift of focus lately. 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 the energy consumption [10]. However, the importance of reducing the power consumption in embedded operating systems has not been widely recognized and a large body of work has focused on estimating, managing,

and reducing power consumption in various system components. RTOS serves as an interface between the application software and the hardware. The embedded system design, development 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.

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 have used aggregation and polymorphism to achieve these goals. We present encapsulated behavior of power management features in form of classes and their interaction in form of sequence diagrams. A proper graphical representation of these complex power management processes can give the user the 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.

1.1. Software Architecture of Power Manager in RTOS

An integral part of OS-level power management is PM, which is responsible for the efficient management of the various applications and devices running on the portable system. Figure 1 shows the interaction of devices, applications and battery with the 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.

PM also co-ordinates the different devices, system and processor states to depreciate the 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 driver to inter-communicate about their power states. Software architecture of PM is responsible for providing the services to the device drivers of the system to get notified and respond upon power changes.

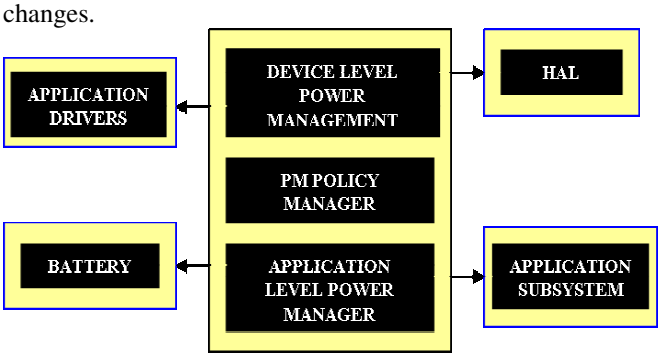

Figure 1. Interaction of power manager with devices and applications

Power management with power manageable hardware comprises of one or more layers of software. Hardware specific power management software and operating system policy manager, which is in-between the hardware independent software interface, is also defined. This creates a layered co-operative 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 productivity by extending the system-battery life.

2. Power States

In any system, under certain circumstances some hardware components are always idle. This also applies to the devices, processor and 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 RTOS, there are different power states associated with applications, devices and the processor. It is quite possible that a processor is in sleeping state, device in soft off and application in full-on state. Figure 2 shows the power state transition for the complete system, among the various states available. Power managed devices receive power state change notifications in the form of I/O control codes (IOCTLs). This partitions device power states from the system power states.

There are seven predefined power states in a 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 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 2. This state diagram can also be considered as the state transition diagram of the active class [11] PolicyManager discussed in the next section.

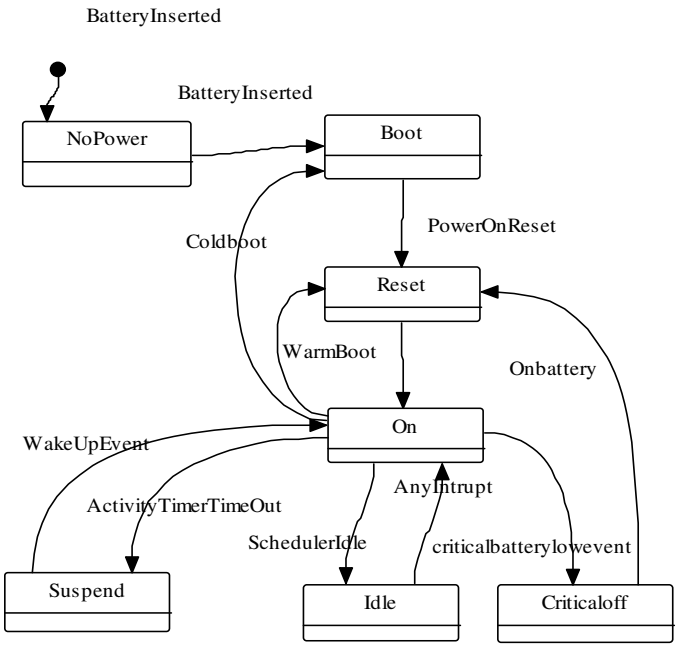

Figure 2. System Power States

A system may have 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 the 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.

3. Power Management Classes

We have abstracted power management features and represented into three different types of classes: PolicyManager, DeviceDriverPolicyManager, and ApplicationPolicyManager. These classes have been well encapsulated. We show the aggregation relation of these classes with policy manager.

3.1. 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 ApplicationPowerManager (APM) and to the devices using the interface specified in DeviceDriverPowerManager (DDPM). On detecting a low battery state, 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.

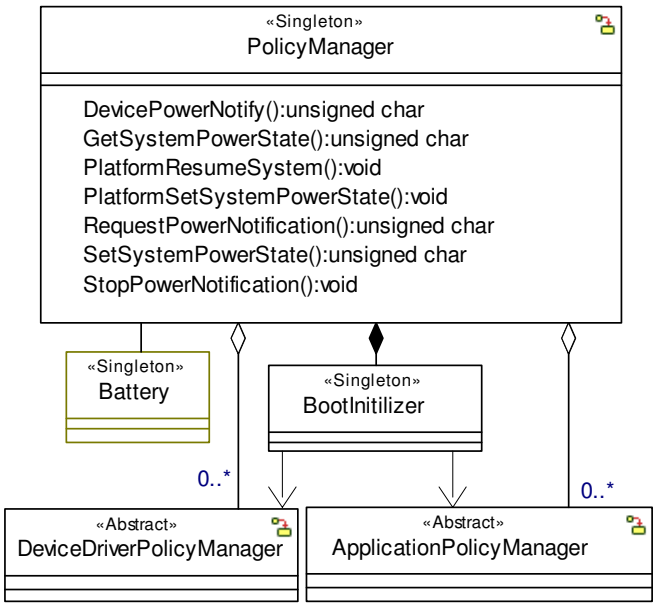

Figure 3. UML Class Diagram of Power Management

Figure 3 depicts the class diagram of power management in RTOS. It can be seen from the figure that Policy Manager and Battery are the singleton class where as the DDPM and APM are the abstract class associated with 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 the important application and the drivers while the system is booted.

3.2. Device Driver Power Manager

Figure 4 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 are presented here just as examples. In a specific embedded system some of these subclasses may not be presented, but they may also have other subclasses. 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. DDPM's interaction with Policy Manager is shown through a sequence diagram in Figure 5 for one specific scenario.

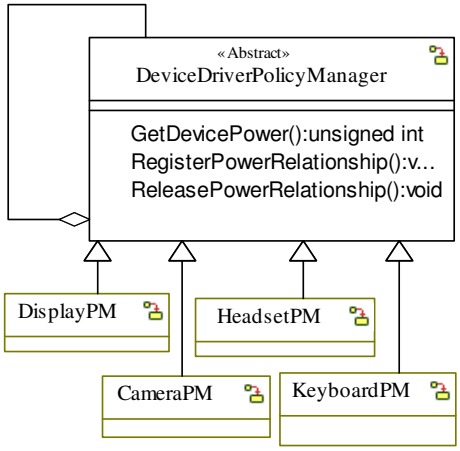

Figure 4: UML Class Diagram of DDPM

Initially, all the device drivers register themselves with Power 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 notify the policy manager to change its state. DevicePowerNotifiy() informs the device about the change in its power state.

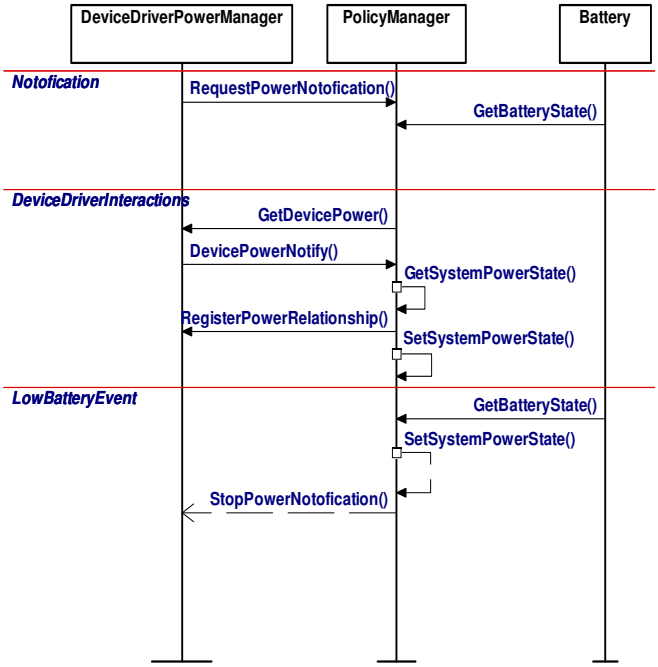

Figure 5: Sequence Diagram of Device Driver Interaction with the Power Manager

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. If it 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 transits to idle state, an acknowledgement is sent to Policy Manager, which puts the system state to idle.

3.3. Application Driver Power Manager

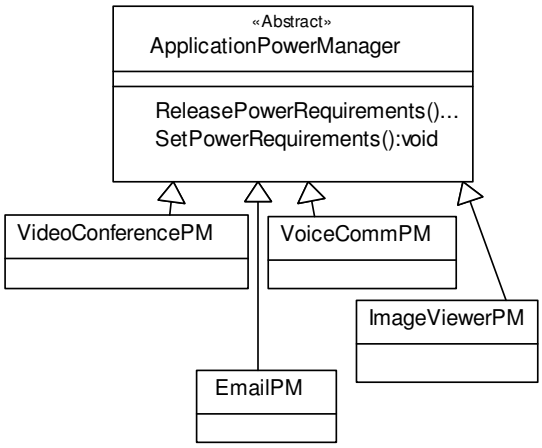

Figure 6: Class Diagram APM

Class diagram for APM is shown in Figure 6. Figure depicts the Power Manager component of some applications such as video conferencing, voice communication, email, as examples of concrete subclasses of the APM. Various system level power changes are notified to the applications using the interface specified by APM by PM polymorphically. 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 transition 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 7 shows the interaction of APM with PM and battery in form of sequence chart. Initially the applications register themselves RequestPowerNotification() with 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 PolicyManager updates the system power state (GetSystemPowerState(), SetSystemPowerStstate()). The policy manager constantly monitors the battery status. Upon

the detection of an idle state event, APM gets the system power state (GetSystemPowerState()) and device power state (GetDevicePowerState()). Power states of drivers are then forced to sleep state by the policy manager. Concurrently, if policy manager detects a low battery state, it notifies the Application Driver Policy Manager, to change the application state of all applications to standby followed by the change in the system state to a standby state.

Figure 7: Sequence Diagram of Application Driver Power Manager Interaction with the Power Manager

4. Conclusion

In this paper we have provided an abstracted concept of implementing Power Management in embedded RTOS. We have shown that system level power management can be implemented in an Object Oriented manner in terms of policy manager, device-driver-power-manager, and application-power-manager. We recommend the interfaces for interactions between PM and the devices power manager, as well as PM and application power manager. We present relationship and interactions of these managers with each using unified modeling language (UML) class diagrams, sequence diagrams and state charts. This abstracted object oriented representation of power management elucidates the operation of the power manager in conjunction with the applications, devices and the processors to the developers of applications and devices from their point of view. If the operating system and the device drivers are design in accordance to 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 interface also simplifies 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 out hope that the proposed framework will foster development of embedded systems that are more power efficient, easy to maintain, and faster to develop.

References

[1] 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.

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

[3] Udani, S., Smith, J., "Power management in Mobile Computing", Department of Computer Information Sciences, Technical Report, University of Pennsylvania, February 1998.

[4] Mario, P., Cathania, I., "Power management System on Silicon For Portable equipment", 8th IEEE International Conference of Electronics, Circuits and Systems, Vol. 1, pp. 13-18, September 2001.

[5] Benini, L., Hodgson, R., Siegel, P., "System-Level Power Estimation and Optimization", International Symposium on Low Power Electronics and Design, IEEE Conference, pp. 173-178, August 1998.

[6] Sinha A., Chandrakasan A. P., "Energy Efficient Real-Time Scheduling [Microprocessors]", *IEEE International Conference of Computer Aided Design*, pp. 458-463, Novemver 2001.

[7] 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.

[8] "Roadmap Architects – The Technology Working Groups" http://public.itrs.net

[9] 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. *In IEEE Computer Society Annual symposium on VLSI*, April 2002, 117-124

[10] 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.

[11] Martin Timmerman, "RTOS Evaluations", 2000, http://www.dedicated-systems.com

[12] Jenson Douglas E., "Real-Time Design Pattern Robust Scalable Architecture for Real Time Systems. Boston, Addition-Wesley, 2002.