International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2014): 5.611

A Review of Embedded Base Power Management Unit

Utsava Khare[1], Megha Gupta[2]

[1]M. Tech Scholar, RIT, Ujjain, India
[2]Assistant Professor, ECE Department, RIT, Indore, India

Abstract: ***This paper Power-efficient design requires reducing power dissipation in all parts of the design and during all stages of the design process subject to constraints on the system performance and quality of service (QoS). Power-aware high-level language compilers, dynamic power management policies, memory management schemes, bus encoding techniques, and hardware design tools are needed to meet these often-conflicting design requirements. Power consumption by embedded devices is a critical issue. There is always a need to extend battery life and/or reduce the environmental impact of a system. Historically, this was purely a hardware issue, but those days are past. In modern embedded systems software takes an increasing responsibility for power management. This article reviews how power management is achieved while a device is operating and looks at the techniques employed to minimize power consumption when a device is inactive.***

Keywords: Embedded system, DSP, DPM

1. Introduction

An embedded system is a computer system with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints. It is *embedded* as part of a complete device often including hardware and mechanical parts. Embedded systems control many devices in common use today[1].

Examples of properties typical of embedded computers when compared with general-purpose ones are low power consumption, small size, rugged operating ranges, and low per-unit cost. This comes at the price of limited processing resources, which make them significantly more difficult to program and to interface with. However, by building intelligence mechanisms on the top of the hardware, taking advantage of possible existing sensors and the existence of a network of embedded units, one can both optimally manage available resources at the unit and network levels as well as provide augmented functionalities, well beyond those available [4]. For example, intelligent techniques can be designed to manage power consumption of embedded systems [5].

Modern embedded systems are often based on microcontrollers (i.e. CPUs with integrated memory or peripheral interfaces) but ordinary microprocessors (using external chips for memory and peripheral interface circuits) are also still common, especially in more complex systems. In either case, the processor(s) used may be types ranging from general purpose to those specialized in certain class of computations or even custom designed for the application at hand. A common standard class of dedicated processors is the digital signal processor (DSP)[19].

Since the embedded system is dedicated to specific tasks, design engineers can optimize it to reduce the size and cost of the product and increase the reliability and performance. Some embedded systems are mass-produced, benefiting from economies of scale.

Power management for computer systems has traditionally focused on regulating the power consumption in static modes such as *sleep* and *suspend*. These are de-activating states, often requiring a user action to re-activate the system. There are usually significant latencies and overheads for entering and exiting these states, and in desktop and server systems a firmware layer typically supports these modes [20].

Dynamic power management refers to power management schemes implemented while programs are running [**1**]. Much architecture provides the equivalent of a *halt* instruction that reduces CPU power during idle periods. The operating system and device drivers may also manage power of peripheral devices, for example spinning down disks during periods of inactivity. Highly integrated processors with on-board peripherals often include software-controlled clock management capabilities to reduce power consumed by inactive peripherals and peripheral controllers. The memory subsystem also provides a profitable area for dynamic power management, either through the memory controller implementation or through software-based schemes [15].

Recent advances in processor design techniques have led to the development of systems that support very dynamic power management strategies based on dynamic voltage and frequency scaling. Since CPU power consumption typically decreases with the cube of voltage while frequencies scale linearly with voltage, significant opportunities exist for tuning the power-performance tradeoff to the needs of the application. Processors such as the Transmit Crusoe, Intel Strong and Scale processors, and the recently announced IBM PowerP 405LP allow dynamic voltage and frequency scaling of the processor core in support of these dynamic power management strategies. Aside from the Transmit system, all of the processors named above are highly integrated system-on-a-chip (SOC) processors designed for embedded applications. The applications of these processors typically do not include traditional BIOS; therefore control of the dynamic power state of the system must be implemented in the operating system [12].

2. The Need to Save Power

Energy consumption is becoming more of a concern as this slice is getting an increasingly larger percentage of the overall operating costs. Imagine superstores with lines and lines of check-out lanes, each with a cash register, a credit-card reader, a scanner and a weight measuring station. It is really a waste if these equipments are not designed to be energy efficient with abilities to power down between customers or during non-operating hours. When multiplied by the number of stores, the number of cities and the operating life of the product, the total accumulated portion of the energy bill that could be saved is in the millions of dollars.

Many of today‟s operating systems, like Linux® come with power management support. The features have been available on the mainstream kernel since Linux made headways to lower power portable devices like smart phones, tablets and ebook readers. So even though your design is a plugged-in appliance, you can embrace the “go green initiative” from the ground up by taking advantage of the power management features that are already in place and incorporate them [9].

3. Power Management Requires a System-Level Approach

There are two different components to the power equation from a silicon process standpoint: static (sometimes referred to as *standby*) and active. *Static power* is affected by leakage mainly and increases with temperature and supply voltage. Since leakage is a natural phenomenon that comes with shrinking process technology, the only way to really eliminate it is to shut that component down. Within the SoC, tactics employed so far include power islands, enabling part of the SoC to completely shut down. On the other hand, *active power*, which does increase with supply voltage, but not temperature, depends on chip activity [8]. Strategies here include:

- Dynamic voltage and frequency scaling (DVFS), where the voltage and frequency can be dynamically adjusted to adapt to the performance required
- Clock domain to gate off unused peripheral
- Dynamic power switching (DPS), where software can switch between power modes based on system activity. The “software” is usually part of the operating system
- Adaptive voltage scaling (AVS), a closed-loop, hardware and software cooperative strategy to maintain performance while using the minimum voltage needed based on the silicon process and temperature. From the system standpoint, operations needed for power management include the ability to:

  a) Go to standby (user-application- or system-initiated system service)
  b) Hibernate to memory or storage (user-application- or system-initiated system service)

- Suspend and resume (user-application-initiated system service)
- Transition to different power profiles (user-application condition or state, system initiated and controlled)

Power can also be affected by how the application code is designed. For example, since input/output (I/O) buffers at the pin need to drive current, the traffic through major peripherals like memory controllers, especially double data rate (DDR), can be a good place to inspect. Unnecessarily moving data in and out of the SoC can waste energy. Let‟s take a look at the block diagram of a typical modern embedded system as shown in Figure 1. The processor is highly integrated and includes several types of processors and accelerators for application specific needs as well as all the I/O peripherals to get the data in and out. The system board has external voltage regulators for the different power rails in addition to battery and clock management support integrated circuits (ICs). It also contains external I/O modules and hot swappable devices. To save energy, the application can take advantage of the internal memories by aligning code and data. In this way, algorithms in the pipeline can reuse buffers locally so that the I/O buffers at the pin level do not have to toggle needlessly. Other techniques include matching data types to architecture, correct alignment and use power of two for array sizes to simplify address calculation. These techniques can help reduce power consumption because the lower MIPs required can lower the temperature. Some call this “energy coding,” the third optimization vector besides speed and code size [16].

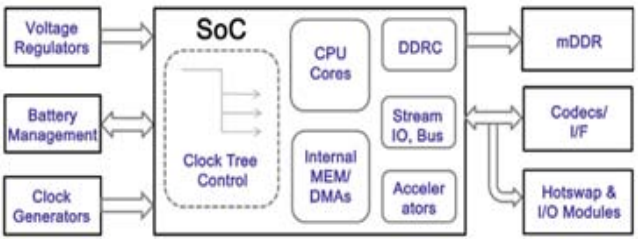

Figure 1: Modern embedded system using complex SoC

Power management is concerted effort. Actions such as going into standby mode can involve a series of hardware and software steps. Therefore, to really “do the job right,” power management needs to be a system-level (i.e., where hardware meets software) design goal, especially if the processor is a complex SoC with multiple internal bus masters. For example, for the “suspend” operation, the software has to take the hardware through the following actions:

- Notify drivers and pending tasks that the system is powering down
- Wait for the safe state to start the shutdown sequence
- Turn off I/Os and accelerators by gating power or clocks
- Save system state to memory (shown as mobile DDR) • Adjust voltage regulators to throttle down
- Set up battery management for suspend
- Transition clocking to a suspend state (usually involving just the real-time clock and mDDR running)

To get into the details of how power management is implemented, we now need to move our discussion on a real device and software

4. Architectural Overview

A high-level overview of our proposed architecture is given in Figure 2. The low-level implementation of the dynamic power management architecture (DPM) is resident in the kernel of the operating system, and power management strategies come from outside of the system. Note that DPM is *not* a self-contained device driver. The low-level implementation of DPM requires enhancements at a few key places in the operating system.

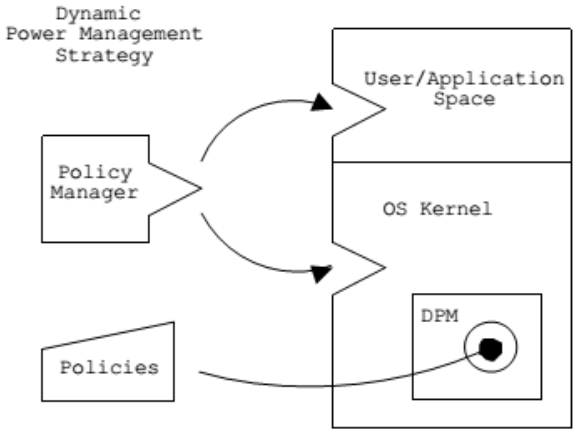

Figure 2: A high-level view of our dynamic power management proposal

Is shown above, we expect a complete dynamic power management strategy to be defined in advance for each application, by a system designer familiar with the characteristics of the embedded system and its special features and requirements. The strategy is communicated to DPM in two ways: as a predefined set of policies and as an application/policy-set specific policy manager that manages them.[15,12,11]

5. History of Embedded

One of the very first recognizably modern embedded systems was the Apollo Guidance Computer, developed by Charles Stark Draper at the MIT Instrumentation Laboratory. At the project's inception, the Apollo guidance computer was considered the riskiest item in the Apollo project as it employed the then newly developed monolithic integrated circuits to reduce the size and weight. An early mass-produced embedded system was the Autonetics D-17 guidance computer for the Minuteman missile, released in 1961. When the Minuteman II went into production in 1966, the D-17 was replaced with a new computer that was the first high-volume use of integrated circuits. Since these early applications in the 1960s, embedded systems have come down in price and there has been a dramatic rise in processing power and functionality. An early microprocessor for example, the Intel 4004, was designed for calculators and other small systems but still required external memory and support chips. In 1978 National Engineering Manufacturers Association released a "standard" for programmable microcontrollers, including almost any computer-based controllers, such as single board computers, numerical, and event-based controllers.

As the cost of microprocessors and microcontrollers fell it became feasible to replace expensive knob-based analog components such as potentiometers and variable capacitors with up/down buttons or knobs read out by a microprocessor even in consumer products. By the early 1980s, memory, input and output system components had been integrated into the same chip as the processor forming a microcontroller. Microcontrollers find applications where a general-purpose computer would be too costly.

A comparatively low-cost microcontroller may be programmed to fulfill the same role as a large number of separate components. Although in this context an embedded system is usually more complex than a traditional solution, most of the complexity is contained within the microcontroller itself. Very few additional components may be needed and most of the design effort is in the software. Software prototype and test can be quicker compared with the design and construction of a new circuit not using an embedded processor

6. Power Management API

In order to introduce a discussion about power management application programming Interfaces (APIs), let us first recall how energy consumption requirements arise during the design of an energy-aware embedded system and how they are usually captured. In such systems, designers look for available energy-efficient components and, eventually, specify new components to be implemented. During this process, they inherently acquire knowledge about the most adequate operating strategy for each component and for the system as a whole. Whenever the identified strategies are associated to modifications in the energy level of a given component, this can be captured in traditional design diagrams, such as sequence, activity, and timing, or by specific tools [3].

Now let us consider the design of a simple application, conceived specifically to illustrate the translation of energy constraints from design to implementation. This application realizes a kind of remote monitoring system, capable of sensing a given property (e.g., temperature), reporting it to a control center, and reacting by activating an actuator (e.g., external cooler) whenever it exceeds a certain limit. Interaction with the control center is done via a communicator (e.g., radio). The system operates on batteries and must run uninterruptedly for one year. A block diagram of the system is shown in Figure 3

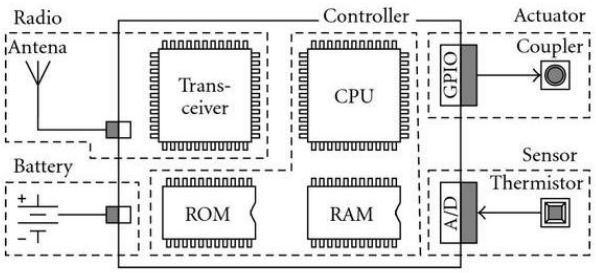

Figure 3: Example monitoring system block diagram

The application is modeled around four tasks whose behavior is depicted in the sequence diagrams of Figures4

through 5: Main allocates common resources and creates threads to execute the other three tasks; Monitor is responsible for periodic temperature monitoring (every second), for reporting the current temperature to the control center (every 10 seconds), and, in case the temperature threshold is exceeded, for triggering the emergency handling thread; Trigger is responsible for triggering the emergency handling thread on command of the control center; Recovery, the emergency handling thread, is in duty of firing a one-shot actuator intended at restoring the temperature to its normal level. Coordination is ensured by properly assigning priorities to threads and by the emergency semaphore.

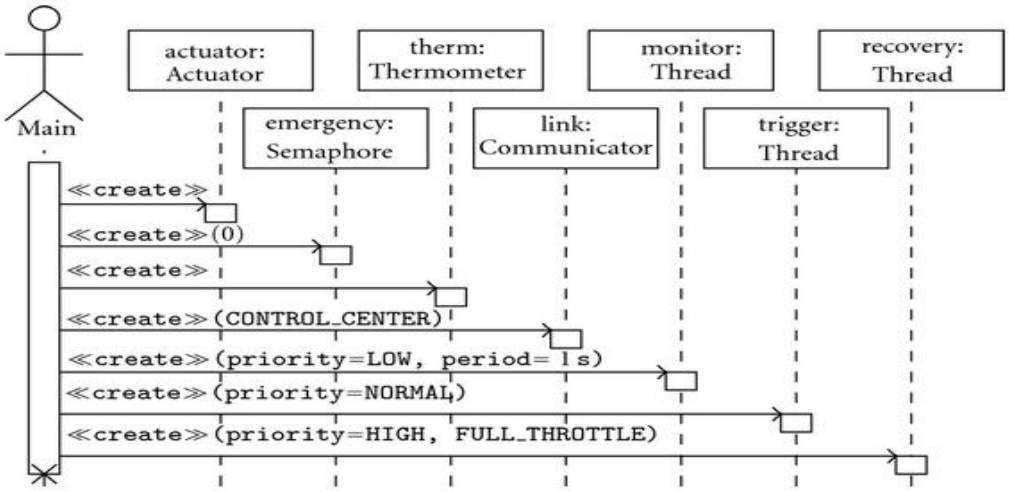

Figure 4: Main thread sequence diagram with power management actions

7. Conclusion

In this paper, power management in embedded systems was addressed from energy-aware design to energy-efficient implementation, aiming at introducing a set of mechanisms specifically conceived for this scenario. A power management API defined at the level of user-visible system components was proposed and compared with traditional APIs. Its implementation was discussed in the context of the necessary infrastructure, including battery monitoring, accounting, autosuspend, and autoresume.

References

[1] J. Monteiro, S. Devadas, P. Ashar, and A. Mauskar, “Scheduling techniques to enable power management,” in Proceedings of the 33rd Design Automation Conference, pp. 349–352, Las Vegas, Nev, USA, 1996.

[2] L. Benini, A. Bogliolo, and G. D. De Micheli, “Dynamic power management of electronic systems,” inProceedings of the IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers (ICCAD '98), pp. 696–702, ACM Press, New York, NY, USA, 1998.

[3] P. M. Anderson and A. A. Fouad, "Power system control and stability", A John Wiley & Sons, Inc., p.38.

[4] H. Kobayashi and A. Suzuki, "Stable operation technique for PCS of PV power generation at grid recovery after voltage sag", System Engineering Research Laboratory, Rep.No.R09015, 2010 (in Japanese).

[5] IEEJ technical committee, "Standard models of power system", IEEJ Technical Report, vol. 754, 1999*Hopkins University*, Baltimore, Maryland, USA. 440.

[6] ERSE, Artigo 59º do RRC: Manual de rocedimentos doAcesso e Operação do SEPM, 2004.

[7] CEEL-IST/REN, Determinação da Capacidade de Integração de Energias Renováveisnasilhas da Madeira e do Porto Santo no period 2006-2010 - Impacto do Aumento da Potência EólicaI nstaladana Rede Eléctrica daIlha da Madeira no ano de 2010, 2009.

[8] Ministerio da la Presidencia, Resolución 9613, 2006.

[9] Task Force C2.02.24, Defense plan against extreme contingencies, CIGRE, 2007.

[10] G. Trudel, S. Bernard, “Hydro-Quebec‟s defence plan against extreme contingencies”, IEEE Transaction on power Systems, Vol.14, no.3, pp.958-966, 1999.

[11] S. Diaf, “Estimation de la production éolienne d‟électricité dans la région d‟Adrar”, The first Mediterranean Seminar on Wind Energy, SMEE'2010, pp. 161 – 172, April 11-12 2010, Algiers, Algeria.

[12] Mohamad, H., Mokhlis, H., Bakar, A.H.A., Ping, H.W, “A review on islanding operation and control for distribution network connected with small hydro power plant”, Renewable and Sustainable Energy Reviews, Vol. 15, no. 8, pp. 3952-3962, 2011.

[13] Terzija.V. V, H. J. Koglin, "Adaptive under frequency load shedding integrated with a frequency estimation numerical algorithm", Iee Proceedings-Generation Transmission and Distribution Vol.149, no. 6, pp. 713-718, 2002.

[14] "IEEE standard for interconnecting distributed Resources with electric power systems", IEEE Std 1547-2003, pp. 1- 16, 2003.Ellingwood, Bruce and Redfield, Robert, (1983). “Ground Snow Loads for Structural Design.” *Journal of Structural Engineering*, 109, 950-964.

[15] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. A. Bhattacharya, „„Virtual machine power metering and provisioning,‟‟ in *Proc. st ACM Symp. Cloud Comput.*, 2010, pp. 39–50.

[16] J. Stoess, C. Lang, and F. Bellosa, „„Energy management for hypervisor based virtual machines,‟‟ in *Proc. USENIX Annu. Tech. Conf.*, 2007, pp. 1–14.
[17] Y. Chen, L. Li, L. Liu, H. Wang, and Y. Zhou, „„Metho d and apparatus for estimating virtual machine energy consumption,‟‟ U.S. Patent 13 596 612, Aug. 28, 2012.
[18] Y. Bao *et al.*, „„HMTT: A platform independent full-system memory trace monitoring system,‟‟ *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 36, no. 1, pp. 229–240, 2008.
[19] B. Krishnan, H. Amur, A. Gavrilovska, and K. Schwan, „„VM power metering: Feasibility and challenges,‟‟ *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 38, no. 3, pp. 56–60, 2010.
[20] N. Kim, J. Cho, and E. Seo, „„Energy-based accounting and scheduling of virtual machines in a cloud system,‟‟ in *Proc. IEEE/ACM Int. Conf. Green Comput. Commun. (GreenCom)*, Aug. 2011, pp. 176–181.
[21] J. Levon and P. Elie, *O Profile: A System Profiler for Linux*, 2004. [Online]. Available: ttp://perfmon2.sourceforge.net, accessed May 29, 2014.
[22] *Perfmon2*. [Online]. Available: http://perfmon2.sourceforge.net, accessed May 29, 2014.