Power Optimization and Management in Embedded Systems[1]

Massoud Pedram
University of Southern California
Dept. of EE-Systems
Los Angeles CA 90089

Abstract

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. This paper reviews techniques and tools for power-efficient embedded system design, considering the hardware platform, the application software, and the system software. Design examples from an Intel StrongARM based system are provided to illustrate the concepts and the techniques. This paper is not intended as a comprehensive review, rather as a starting point for understanding power-aware design methodologies and techniques targeted toward embedded systems.

1 Introduction

A system consists of a set of components that provide a useful behavior or service. System design is the process of implementing the desired behavior or service while optimizing some objectives and satisfying some constraints. For example, a high-end desktop computer system is optimized for maximum clock rate while meeting a price target. On the other hand, a battery-operated portable electronic system for information processing and wireless communication may be optimized for minimum power dissipation subject to a required performance.

There exists a dichotomy in the design of modern electronic systems: the simultaneous need to be low power and high performance. This arises largely from their use in battery-operated portable (wearable) platforms. Accordingly, the goal of low-power design for battery-powered electronics is to extend the battery service life while meeting performance requirements. In fixed, power-rich platforms, the packaging and reliability costs associated with high power and high performance systems also force designers to look for ways to reduce power consumption. Thus, reducing power dissipation is a design goal even for non-portable devices since excessive power dissipation results in increased packaging and cooling costs as well as potential reliability problems. However, the focus of this survey paper is on portable electronic systems.

A typical embedded system consists of a base hardware platform, which executes system and application software and is connected to a number of peripheral devices. The hardware platform refers to the computational units, the communication channels, and the memory units. Examples of peripheral devices include displays, wireless local area network, keyboard, and camcorder. Power efficiency is obtained by reducing power dissipation in all parts of the design. Reducing the power consumption of all of the constituents of the base hardware can minimize its power dissipation. The peripheral devices tend to consume a significant fraction of the total power budget. Although power-efficient design of peripherals is beyond the scope of this paper, we will describe techniques to reduce the data rate and switching activity on the communication links between the hardware platform and these peripherals as well as review dynamic power management policies that aim at turning off the peripherals during their idle periods. Choices for the software implementation (i.e. operating system kernel, application-level code) affect the power efficiency of the base hardware and peripheral devices. For example, the power dissipation overhead of the operating system calls, the power-efficiency of the compiled code, and the memory access patterns play important roles in determining the overall power dissipation of the embedded system.

A key part of embedded system synthesis is hardware-software co-design. Hardware-software co-design is the process of concurrently defining the hardware and software portions of an embedded system while considering dependencies between the two. Designers rely on their experience with past systems when estimating the resource requirements of a new system. Since ad-hoc design exploration is time-consuming, an engineer typically selects a conservative architecture after little experimentation, resulting in an unnecessarily expensive system. Most research in the area of hardware-software co-design has focused on easing the process of design exploration. Automating this process falls within the realm of co-synthesis. Finding optimal hardware-software architecture entails selection of processors, integrated circuits, memories, and communication links such that the cost of the architecture is at a minimum and all real-time and peak power consumption constraints are met. Average and peak power consumptions of the architecture have an impact on the battery life, packaging costs and reliability. Thus, it is important to consider power consumption during hardware-software co-synthesis. Considering these problems early in the design cycle usually reduces their required overhead.

The paper is organized as follows. In Section 2 we provide a review of the power reduction techniques for embedded systems. Section 2 contains the description of the target embedded system and a number of examples of power-saving techniques. We conclude the paper in Section 4.

[1] This work is sponsored in part by DARPA PAC/C program under contract number DAAB07-00-C-L516.

2 Literature Review

2.1 Power-aware software compilation

Power analysis techniques have been proposed for embedded software based on instruction-level characterization [1] and simulation of the underlying hardware [2]. Techniques to improve the efficiency of software power analysis through statistical profiling have been proposed in [3]. The SOC design paradigm has fueled research in co-estimation of hardware and software power consumption [4]-[7]. Optimizations of embedded software power consumption through compiler optimizations [8], source-level transformations [9], and memory management schemes [10] [11] have been reported.

Prior work on power-aware compilation techniques is limited. In [12], the authors studied the switching activity on address, instruction and data buses and estimated the power consumption of a microprocessor. They proposed a technique called “Cold Scheduling.” In [13], the authors proposed an instruction-level power model and a number of simple compiler techniques (such as operand swapping) for reducing power consumption. The techniques were applied to a Fujitsu DSP.

2.2 Dynamic power management

Dynamic power management – which refers to selective shut-off or slow-down of system components that are idle or underutilized – has proven to be a particularly effective technique for reducing power dissipation in such systems. Incorporating a dynamic power management scheme in the design of an already-complex system is a difficult process that may require many design iterations and careful debugging and validation. The goal of a dynamic power management policy is to reduce the power consumption of an electronic system by putting system components into different states, each representing a certain performance and power consumption level. The policy determines the type and timing of these transitions based on the system history, workload, and performance constraints.

The activity of many components in a computing system is event-driven. An intuitive way of reducing the average power dissipated by the whole system consists of shutting down the resources during their periods of inactivity. In other words, one can adopt a system-level power management policy that dictates how and when the various components should be shut down. A simple and widely used technique is the "time-out" policy, which turns on the component when it is to be used and turns off the component when it has not been used for some pre-specified length of time. The effectiveness of the "time-out" policy largely depends on the time-out settings, as well as the access pattern characteristics of the component. Srivastava et al. [14] proposed a predictive power management strategy, which uses a regression equation based on the previous “On” and “Off” times of the component to estimate the next “turn-on” time. In [15], Hwang and Wu introduced a more complex predictive shutdown strategy that yields better performance. These heuristic power management policies have major limitations. For example, they cannot handle components with more than two power modes; nor can they handle complex system behaviors; and they cannot guarantee optimality.

Choosing a policy that minimizes power dissipation under a performance constraint (or maximizes performance under a power constraint) is a new kind of constrained optimization problem that is highly relevant to low-power electronic systems. This problem is often referred to as the policy optimization (PO) problem. In [16], Benini et al. proposed a stochastic model for a rigorous mathematical formulation of the problem and presented a procedure for its exact solution. The solution is computed in polynomial time by solving a linear optimization problem. Their approach is based on a stochastic model of power-managed devices and workloads and leverages stochastic optimization techniques based on the theory of *discrete-time Markov decision processes*. The system model consists of four components: a power manager (PM), a service provider (SP), a service requestor (SR), and a service request queue (SQ) (cf. Figure 1). The objective function is the expected performance (which is related to the expected waiting time and the number of jobs in the queue) and the constraint is the expected power consumption (which is related to the power cost of staying in some server state and the energy consumption for the transfer from one server state to the next). Non-stationary user request rates can be treated using an adaptive policy interpolation presented in [17]. A limitation of both of these policies is that decision-making is done periodically, even when the system is idle, thus wasting power.

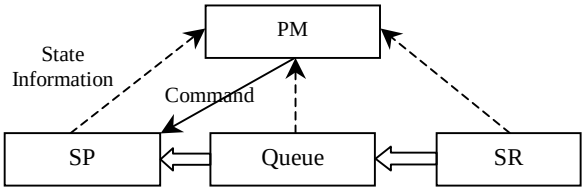

Figure 1 A model of a power-managed system.

In [18] [19], the authors model the power-managed system using a *continuous-time Markov decision process*. In this approach, the PM issues commands asynchronously upon event occurrences instead of at fixed time intervals. System transitions are assumed to follow exponential distribution, even though in some practical cases, the transition times may be non-exponential. This assumption can be relaxed by using a stage method [20], which approximates the general distributions using the series or parallel combinations of exponential stages. The authors of [21] [22] extend the modeling technique using the *semi-Markov decision process* and the *time-indexed semi-Markov decision process*. Both techniques are in a continuous-time domain. The work in [23] uses a modeling technique based on *generalized stochastic Petri nets* (GSPN) for large system with complex behaviors.

2.3 Low power bus encoding

The major building blocks of a computer system include the processor, the memory chips, and the communication channels dedicated to providing the means for data transfer between the processor and the memory. These channels tend to support heavy traffic and often constitute the performance bottleneck in many systems. At the same time, the energy dissipation per memory bus access is quite high, which in turn limits the power efficiency of the overall system.

Low power bus codes can be classified as algebraic, permutation, and probabilistic. Algebraic codes refer to those codes that are produced by encoders that take two or more operands (e.g. the current source word, the previous source word, the previous code word, etc.) to produce the current code word using arithmetic or bit-level logic operations. Permutation codes refer to a permutation of a set of source words. Probabilistic codes are generated by encoders that

examine the probability distribution of source words or pairs of source words, and use this distribution to assign codes to the source words (Level Signaling) or pairs of source words (Transition Signaling). In all cases, the objective is to minimize the number of transitions for transmitting the source words.

Many bus-encoding schemes have been proposed. The encoding function can be optimized for specific access patterns such as sequential access (Gray [24][25], T0 [26], Pyramid [27]) or random data (Bus Invert [28]), for special data types such as floating point numbers in DSP applications [29]. The encoding may be fully customized to the target data (Working Zone [30], Beach [31]). The encoding function can be fixed (Gray, Pyramid), synthesized (Working Zone, Beach), or non-deterministic (Bus Invert, Transition Signaling or both [32]). The encoded bits may be redundant (Bus Invert, T0, Limited-weight code [33]) or irredundant (Gray, Pyramid). The key idea behind all of these techniques is to reduce the Hamming distance between consecutive address and data transfers.

2.4 Dynamic voltage scaling

Dynamic voltage scaling (DVS) [34]-[36] refers to runtime change in the supply voltage levels supplied to various components in a system so as to reduce the overall system power dissipation while maintaining a total computation time and/or throughput requirement. This problem can be formulated as a variable (or multiple) voltage scheduling problem and solved using mathematical (or dynamic) programming. This formulation, however, does not capture the adaptive online characteristic of the DVS problem. Figure 2 shows an example DVS architecture.

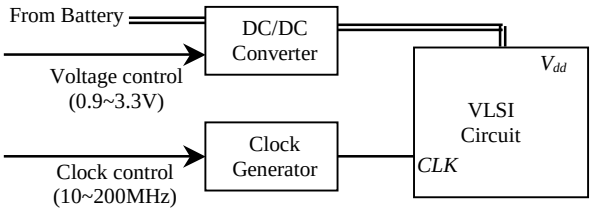

Figure 2 A dynamic voltage-scaling example.

Since static voltage scaling cannot deal with variable workload situations in real-time systems, one must be able to change the supply voltages dynamically for different workloads. Meanwhile, one must carefully design the clock generation circuitry because clock rate must decrease (or increase) with the decrease (or increase) of the supply voltage to make sure that the circuit can work properly.

2.5 Low power system synthesis

Researchers initially focused their interest on hardware-software co-synthesis of one-CPU-one-ASIC architectures, where attempts were made to move operations from hardware to software or vice versa to minimize cost and meet deadlines. Some of these techniques have been extended to handle multiple processing elements and communication links. A constructive co-synthesis system, called COSYN, was presented in [37], which tackles the power consumption problem. For task graphs from the literature for which mixed integer linear programming-based optimal results are known, COSYN also obtains the same optimal results in many orders of magnitude smaller CPU time. This work has been extended to hierarchical task graphs and hierarchical embedded system architectures in a co-synthesis system called COHRA [38].

Much of the recent work that targets low power has been at the register-transfer level (RTL). In the first of such work [39] [40], parallelism is proposed to enable voltage reduction, and a variety of circuit transformations are used to reduce circuit power. In [41], circuit metrics based on switching activity are used to perform power optimizations in a class of digital filters. A multiple clocking scheme for RTL designs is presented in [42], where idle areas of the circuit are shut off to save power. A controller re-specification technique is provided in [43] to reduce switching activity in the data path.

Optimizing for power in memory-intensive applications has also been recognized as an important issue and data-dominated applications have been targeted in [44] [45]. Most of the other high-level synthesis techniques for low power also target data-dominated circuits and include scheduling and binding methods [46]-[51]. The only other work for reducing power in CFI applications is given in [52], where a profile-driven high-level synthesis technique for low power is presented. This method, however, is limited to simple conditional constructs and cannot handle multiple nested loops and branches. A method called common-case computation (CCC) was recently introduced in [53], which works at either the behavior or register-transfer level. It identifies commonly occurring sub-behaviors in a behavioral description of an ASIC. It separately synthesizes these sub-behaviors and puts the original circuit to sleep when it detects these sub-behaviors during normal operation.

3 The Target Embedded System

3.1 Platform Description

Because of its high performance, aggressive power efficiency, and low cost, many designers use the StrongARM® embedded microprocessors, e.g., the SA-110, the first product in the StrongARM family, which features a highly integrated single-chip design [54]. The latest StrongARM microprocessor from Intel®, the SA-1110, is a device optimized for meeting portable and embedded application requirements [55]. The SA-1110 incorporates a 32-bit StrongARM core (SA-10) capable of running at up to 206 MHz and delivering up to 235 Dhrystone 2.1 MIPS. The 206 MHz SA-1110 core runs at a supply voltage level of 1.75 V and consumes < 400 mW in the *Normal* mode, < 100 mW in the *Idle* mode, and < 50 uA in the *Sleep* mode. The SA-1110 has split instruction and data caches, memory-management unit (MMU), and read/write buffers. It is the focus of this effort to explore these features in addition to the low voltage feature of SA-1110. The SA-1110 contains the CPU core, memory controller, LCD controller and other peripheral controllers. On the processor, the memory only functions as cache. One or more Dynamic RAM chips are externally connected to function as the processor's main memory.

Our target embedded system is similar to the Intel StrongARM SA-1110 evaluation board (Assabet) plus the companion SA-1111 board (Neponset). The peripherals include an active LCD, a wireless LAN, a keyboard, a mouse, external FLASH memory, a camcorder, and a GPS set.

3.2 Power Saving Techniques

Exemple 1:

Consider a scenario where the application software for the system has already been developed. In this case the option of rewriting the application software to improve its power efficiency may not exist. However, a power-aware high-level

language compiler can significantly reduce power dissipation of the application software running on the target system.

Traditional cache organization is vertical in the sense that the "Inclusion Property" is preserved, i.e., L1 is included in L2, L2 is included in memory, and so on. StrongARM processors feature a horizontal data cache organization. The MainCache and MiniCache of StrongARM are exclusive – any cacheable data goes into either the MainCache or the MiniCache, but not both. The compiler can thus use the MiniCache as a temporary storage without polluting the regular access patterns occurring in the MainCache as shown in the matrix multiplication in Example 1. Obviously, this results in performance improvement and power savings.

```
/* Matrix Multiplication. *
 * Cache size: MainDcache=64;
 * MiniDcache=16;ReadBuffer=4;
 */
int a[8][8];  /* Main Dcache */
int b[8];     /* Mini Dcache */
int c[8];     /* Mini Dcache */
int ready;    /* ReadBuffer */
while (! ready) ;
ready = 0;
for (i=0; i<8; i++) {
  for (j=0; j<8; j++) {
    c[i] = a[i][j] * b[j]; }}
ready = 1;
```

Example 2:

Instruction rescheduling is a common compiler technique for improving the code performance. Without affecting the correctness, reordering instructions can eliminate pipeline stalls caused by load delay, branch delay, delay slot, etc. This technique is also very effective for saving power (cf. Figure 3). The idea is to reorder instructions so as to minimize the Hamming distance of consecutive instructions while preserving functional correctness and performance.

Sequential Schedule				**Gray Schedule**			
instruction	Rt	Rs1	Rs2	instruction	Rt	Rs1	Rs2
add r0, r1, r2	000	001	010	add r0, r6, r7	000	110	111
sub r3, r4, r5	011	100	101	sub r1, r2, r5	001	010	101
mul r6, r0, r3	110	000	011	mul r3, r0, r1	011	000	001
Bit Activity	4	3	5	Bit Activity	2	2	2

Figure 3 Effect of instruction scheduling.

Example 3:

We present a simple example to illustrate the basic idea behind dynamic power management and the resulting trade-off space. Consider an I/O device (service provider) that is controlled by the operating system-directed power management software and that can be put into one of two modes: "On" and "Off." The device consumes 10 Watts of power when it is in the "On" state and 0 Watt when it is "Off." It takes 2 seconds and 40 Joules of energy to turn this device from "Off" to "ON" whereas it takes 1 second and 10 Joules of energy to turn it from "On" to "Off." The service request pattern in a time period of 25 seconds is shown in Figure 4(a).

Without a power management policy, this device is always "ON" as depicted in Figure 4(b). The shaded areas in the figure identify the time periods when the device is processing the requests. Under a "greedy power management policy", the OS turns on the device whenever there is a service request for it and turns off the device whenever it has nothing to do. The working pattern of the device under this policy is shown in Figure 4(c). If, however, the average delay for processing a request can be increased, then the power management policy shown in Figure 4(d) will result in minimum power dissipation (total energy in 25 seconds) for the device under the specified delay constraint. The comparison between the three policies in terms of total energy consumption and average delay is presented in

Table 1.

This simple example illustrates the fact that there is a fundamental trade-off between power dissipation and latency per request (while maintaining the same throughput) and that a "good" power management policy is one that can exploit this trade-off dynamically.

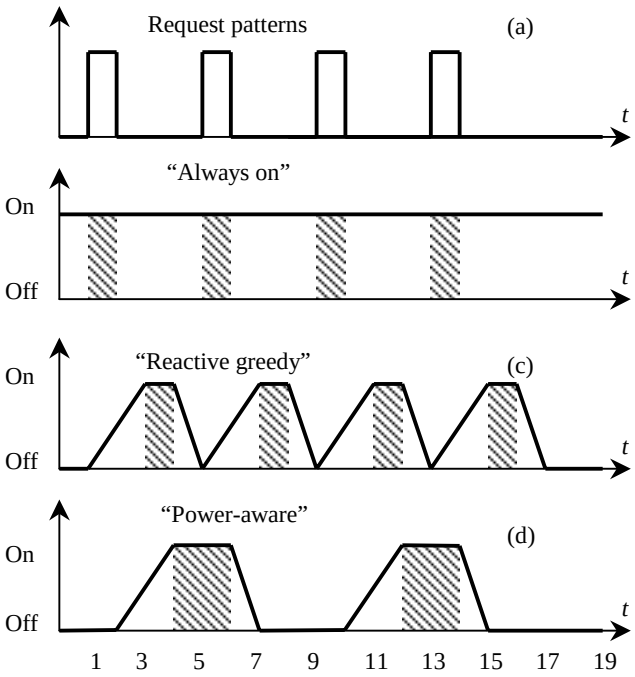

Figure 4 A PM problem with solutions.

Table 1 Results of different PM policies.

Policy type	Energy consumption in 25 sec	Avg. Latency per request
"Always on"	250 J	1 sec
"Reactive Greedy"	240 J	3 sec
"Power-aware"	140 J	2.5 sec

Example 4:

Dynamic RAM (DRAM) is usually laid out in a 2-dimensional array. To identify a memory cell in the array, two addresses are needed: row address and column address. The row address is sent over the bus first and is latched in the DRAM decoder. Subsequently, the column address is sent to complete the address. We will refer to this kind of DRAM as Conventional DRAM. As a result, the DRAM bus is almost always time-multiplexed between the row and column addresses, and the pin count needed for addresses can be reduced by a factor of

two. Because the switching activity on a DRAM bus is totally different from that of a non-multiplexed bus, we need another Gray-code-like encoding scheme to minimize the switching activity for sequential memory access on a DRAM bus.

Table 2 Comparing bus encoding techniques.

Bin	Gray	BI	T0	Conv. DRAM	Page Mode	Burst Mode	Pyra-mid	Burst Pyra-Mid
0000	0000	0000-0	0000-0	00 00	00 00	00 00	00 00	00 00
0001	0001	0001-0	0000-1	00 01	01		00 01	
0010	0011	0010-0	0000-1	00 10	10	00 10	01 01	01 00
0011	0010	0011-0	0000-1	00 11	11		01 00	
0100	0110	1011-1	0000-1	01 00	01 00	01 00	00 10	01 10
0101	0111	1010-1	0000-1	01 01	01		10 01	
0110	0101	0110-0	0000-1	01 10	10	01 10	01 10	11 00
0111	0100	0111-0	0000-1	01 11	11		10 10	
1000	1100	0111-1	0000-1	10 00	10 00	10 00	10 00	00 10
1001	1101	0110-1	0000-1	10 01	01		00 11	
1010	1111	1010-0	0000-1	10 10	10	10 10	11 01	10 10
1011	1110	1011-0	0000-1	10 11	11		01 11	
1100	1010	0011-1	0000-1	11 00	11 00	11 00	11 10	11 10
1101	1011	0010-1	0000-1	11 01	01		10 11	
1110	1001	1110-0	0000-1	11 10	10	11 10	11 11	10 00
1111	1000	1111-0	0000-1	11 11	11		11 00	
=20	**=16**	**=28**	**=2**	**=32**	**=24**	**=16**	**=16**	**=12**

Besides conventional DRAM, almost every modern DRAM device also supports a Page mode. In the Page mode, after the first data transaction, the row address is latched and several different memory locations in the same row can be read/written by sending only their column addresses. Hyper Page mode or Extended Data Out (EDO) is the same as Page mode, except that the Column Address Strobe (CAS) signal is overloaded with both CAS and Data Out. Synchronous DRAM (SDRAM), named so because it avoids the asynchronous handshaking used in conventional and page mode DRAM, uses the system clock to strobe data. No Data Out signal is needed. To boost the throughput, in Burst mode, several bytes (2, 4, 8 or more) can be read/written continuously without any handshaking signal.

Column 5 in Table 2 shows that the binary code results in a high switching activity. Columns 6 and 7 of Table 2 provide examples of accessing blocks of size two in page mode and burst mode, respectively. Column 8 in Table 2 lists the Pyramid code, which reduces up to 56% of the bit-level switching activity for conventional DRAM.

4 Conclusion

This paper reviewed techniques and tools for power-efficient embedded system design, considering the hardware platform, the application software, and the system software. Design examples from an Intel StrongARM based system were provided to illustrate the concepts and the techniques.

References

[1] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: A first step towards software power minimization," *Proc. Int. Conf. Computer-Aided Design*, Nov. 1994.

[2] T. Sato, Y. Ootaguro, M. Nagamatsu, and H.Tago, "Evaluation of architecture-level power estimation for CMOS RISC processors," *Proc. Int. Symp. Low Power Electronics*, pages 44-45, Oct. 1995.

[3] C.-T. Hsieh, M. Pedram, G. Mehta, and F.Rastgar, "Profile-driven program synthesis for evaluation of system power dissipation," *Proc. Design Automation Conf.*, pages 576-581, June 1997.

[4] Y. Li and J.Henkel, "A framework for estimating and minimizing energy dissipation of embedded HW/SW systems," *Proc. Design Automation Conf.*, pages 188-193, June 1998.

[5] C-T. Hsieh and M. Pedram, "Micro-processor power estimation using profile-driven program synthesis," *IEEE Trans. on Computer Aided Design*, Vol. 17, No. 11, pages 1080-1089, Nov. 1998.

[6] T. Simunic, L. Benini, and G. De Micheli, "Cycle-accurate simulation of energy consumption in embedded systems," *Proc. Design Automation Conf.*, pages 867-872, June 1999.

[7] L. Benini and G. De Micheli, "System-level power optimization: Techniques and tools," *Proc. Int. Symp. Low Power Electronics and Design*, Aug. 1999.

[8] V. Tiwari, S. Malik, and A. Wolfe, "Compilation techniques for low energy: An overview," *Proc. Int. Symp. Low Power Electronics*, pages 38-39, Oct. 1994.

[9] T. Simunic, G. De Micheli, and L. Benini, "Energy-efficient design of battery-powered embedded systems," *Proc. Int. Symp. Low Power Electronics and Design*, Aug. 1999.

[10] F. Catthoor, F. Franssen, S. Wuytack, L. Nachtergaele, and H. De Man, "Global communication and memory optimizing transformations for low power signal processing systems," *Proc. Int. Wkshp. on Low Power Design*, pages 203-208, Apr. 1994.

[11] J. L. da Silva, F. Catthoor, D. Verkest, and H. De Man, "Power exploration for dynamic data types through virtual memory management refinement," *Proc. Int. Symp. Low Power Electronics and Design*, Aug. 1999.

[12] C. L. Su, C-Y. Tsui and A. M. Despain, "Low power architecture design and compilation techniques for high-performance processors," *Proc.IEEE Compcon*, pages 489-498, 1994.

[13] V. Tiwari, S. Malik and A. Wolfe, "Instruction level power analysis and optimization of software," *Journal of VLSI Signal Processing*, pages 1-18, 1996.

[14] M. Srivastava, A. Chandrakasan, and R. Brodersen, "Predictive system shutdown and other architectural techniques for energy effcient programmable computation," *IEEE Trans. on VLSI Systems*, Vol. 4, No. 1 (1996), pages 42-55.

[15] C.-H. Hwang and A. Wu, "A predictive system shutdown method for energy saving of event-driven computation," *Proc. Int.. Conf. on Computer Aided Design*, pages 28-32, November 1997.

[16] L. Benini, A. Bogliolo, G.A. Paleologo and G. De Micheli, "Policy optimization for dynamic power management," *IEEE Trans. on Computer-Aided Design*, Vol. 18, No. 6 (1999), pages 813-833.

[17] E. Chung, L. Benini and G. De Micheli, "Dynamic power management for non stationary service requests", *Proc. Design and Test in Europe Conference*, pages 77-81, March 1999.

[18] Q. Qiu and M. Pedram, "Dynamic power management based on continuous-time Markov decision processes," *Proc. 36th Design Automation Conf.*, pages 555-561, June 1999.

[19] Q. Qiu, Q. Wu and M. Pedram, "Stochastic modeling of a power-managed system: Construction and optimization," *Proc. Symp. on Low Power Electronics and Design*, pages 194-199, August 1999.

[20] L. Kleinrock, *Queueing Systems*. Volume I: Theory, Wiley-Interscience, New York, 1981.

[21] T. Simunic, L. Benini, and G. De Micheli, "Dynamic Power Management of Laptop Hard Disk", *Proc. Design Automation and Test in Europe*, 2000.

[22] Y. Lu, E. Chung, T. Simunic, L. Benini, and G. De Micheli, "Quantitative Comparison of Power Management Algorithms", *Proc. Design Automation and Test in Europe*, 2000.

[23] Q. Qiu, Qing Wu, and M. Pedram, "Dynamic Power Management of Complex Systems Using Generalized Stochastic Petri Nets", *Proc. the Design Automation Conference*, Jun. 2000.

[24] C. L. Su, C. Y. Tsui, and A. M. Despain, "Saving power in the control path of embedded processors," *IEEE Design and Test of Computers*, Vol. 11, No. 4 (1994), pages 24-30.

[25] H. Mehta, R. M. Owens, and M. J. Irwin, "Some issues in gray code addressing," *Proc. Great Lakes Symposium on VLSI*, Ames, IA, USA, pages 178-181, Mar. 1996.

[26] L. Benini, G. DeMicheli, E. Macii, D. Sciuto, and C. Silvano, "Address bus encoding techniques for system-level power optimization," *Proc. Design Automation and Test in Europe*, pages 861-866, February 1998.

[27] W-C. Chung and M. Pedram, "Power-optimal encoding for DRAM address bus," *Proc. Symposium on Low Power Electronics and Design*, pages 250-252, July 2000.

[28] M. R. Stan, W. P. Burleson, "Bus-Invert Coding for Low-Power I/O," *IEEE Transactions on VLSI Systems*, Vol. 3, No. 1, pages 49-58, Jan. 1995.

[29] K. Kim and P. A. Beerel, "A low-power matrix transposer using MSB-controlled inversion coding," *The First IEEE Asia Pacific Conference on ASIC*,, pages 194–197, 1999.

[30] E. Musoll, T. Lang, and J. Cortadella, "Exploiting the locality of memory references to reduce the address bus energy," *Proc. Symposium on Low Power Electronics and Design*, pages 202-207, August 1997.

[31] L. Benini, G. DeMicheli, E. Macii, M. Poncino, and S. Quer, "System-level power optimization of special purpose applications: The beach solution," *Proc. Symposium on Low Power Electronics and Design*, pages 24-29, August 1997.

[32] Y. Shin and K. Choi, "Narrow bus encoding for lower power systems," *Proc. Asia and South Pacific Design Automation Conf*, pages 217 –220, 2000.

[33] M. R. Stan and W. P. Burleson, "Limited-weight codes for low-power I/O," *IEEE Transactions on VLSI*, pages 49-58, Mar. 1995.

[34] T. Pering, T. Burd, and R. Brodersen, "The simulation and evaluation of dynamic voltage scaling algorithms," *Proc. Int. Symp. Low Power Electronics and Design*, Aug. 1998.

[35] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," *Proc. Int. Symp. Low Power Electronics and Design*, Aug. 1998.

[36] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M.B. Srivastava, "Power optimization of variable voltage core-based systems," *Proc. Design Automation Conf.*, pages 176-181, June 1998.

[37] B. P. Dave, G. Lakshminarayana, and N. K. Jha, "COSYN: Hardware-software co-synthesis of distributed embedded systems," *Proc. Design Automation Conf.*, pages 703-708, June 1997.

[38] B. P. Dave and N. K. Jha, "COHRA: Hardware-software co-synthesis of hierarchical distributed embedded system architectures," *Proc. Int. Conf. VLSI Design*, Jan. 1998.

[39] A. P. Chandrakasan, S. Sheng, and R. W. Broderson, "Low-power CMOS digital design," *IEEE J. Solid-State Circuits*, pages 473-484, Apr. 1992.

[40] A. P. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R. W. Broderson, "Optimizing power using transformations," *IEEE Trans. Computer-Aided Design*, Vol. 14, pages 12-51, Jan. 1995.

[41] A. Chatterjee and R. K. Roy, "Synthesis of low power DSP circuits using activity metrics," *Proc. Int. Conf. VLSI Design*, pages 265-270, Jan. 1994.

[42] C. Papachristou, M. Spining, and M. Nourani, "A multiple clocking scheme for low power RTL design," *Proc. Int. Symp. Low Power Design*, pages 27-32, Aug. 1995.

[43] A. Raghunathan, S. Dey, N. K. Jha, and K. Wakabayashi, "Power management techniques for control-flow intensive designs," *Proc. Design Automation Conf.*, pages 429-434, June 1997.

[44] D. Lidsky and J. Rabaey, "Low-power design of memory intensive functions," *Proc. Symp. Low Power Electronics*, pages 16-17, Oct. 1994.

[45] P. R. Panda and N. D. Dutt, "Behavioral array mapping into multiport memories targeting low power," *Proc. Int. Conf. VLSI Design*, pages 268-272, Jan. 1997.

[46] A. Raghunathan and N. K. Jha, "Behavioral synthesis for low power," in *Proc Int. Conf. Computer Design*, pages 318-322, Oct. 1994.

[47] J. Chang and M. Pedram, *Power Optimization and Synthesis at Behavioral and System Levels Using Formal Methods*, Kluwer Academic Publishers, 1999.

[48] R. S. Martin and J. P. Knight, "Power profiler: Optimizing ASIC's power consumption at the behavioral level," *Proc. Design Automation Conf.*, pages 42-47, June 1995.

[49] A. Dasgupta and R. Karri, "Simultaneous scheduling and binding for power minimization during microarchitecture synthesis," *Proc. Int. Symp. Low Power Design*, pages 69-74, Apr. 1995.

[50] L. Goodby, A. Orailoglu, and P. M. Chau, "Microarchitectural synthesis of performance-constrained low-power VLSI designs," *Proc. Int. Conf. Computer Design*, pages 323-326, Oct. 1994.

[51] A. Raghunathan and N. K. Jha, "SCALP: An iterative-improvement-based low-power data path synthesis system," *IEEE Trans. Computer-Aided Design*, Vol. 16, pages 1260-1277, Nov. 1997.

[52] N. Kumar, S. Katkoori, L. Rader, and R. Vemuri, "Profile-driven behavioral synthesis for low-power VLSI systems," *IEEE Design and Test of Computers*, pages 70-84, Sept. 1995.

[53] G. Lakshminarayana, A. Raghunathan, K. S. Khouri, N. K. Jha, and S. Dey, "Common-case computation: A high-level power optimization technique," *Proc. Design Automation Conference*, June 1999.

[54] URL: http://www.arm.com/Pro+Peripherals/MicroP/StrongARM/

[55] URL: http://developer.intel.com/design/strong/