# Power Optimization of Variable-Voltage Core-Based Systems

Inki Hong, Darko Kirovski, Gang Qu, Miodrag Potkonjak, and Mani B. Srivastava

***Abstract*— The growing class of portable systems, such as personal computing and communication devices, has resulted in a new set of system design requirements, mainly characterized by dominant importance of power minimization and design reuse. The energy efficiency of systems-on-a-chip (SOC) could be much improved if one were to vary the supply voltage dynamically at run time. We develop the design methodology for the low-power core-based real-time SOC based on dynamically variable voltage hardware. The key challenge is to develop effective scheduling techniques that treat voltage as a variable to be determined, in addition to the conventional task scheduling and allocation. Our synthesis technique also addresses the selection of the processor core and the determination of the instruction and data cache size and configuration so as to fully exploit dynamically variable voltage hardware, which results in significantly lower power consumption for a set of target applications than existing techniques. The highlight of the proposed approach is the nonpreemptive scheduling heuristic, which results in solutions very close to optimal ones for many test cases. The effectiveness of the approach is demonstrated on a variety of modern industrial-strength multimedia and communication applications.**

***Index Terms*— High-level synthesis, scheduling, synthesis for low power, system-on-a-chip (SOC).**

## I. Introduction

### A. Motivation

THE growing class of portable systems, such as personal computing and communication devices, demands data- and computation-intensive functionalities with low power consumption. For such systems, power consumption is the primary design goal since the battery life is a primary constraint on power, due to the fact that the battery technology has not followed the progress pace of the semiconductor industry. Recent advances in power-supply technology along with custom and commercial CMOS chips that are capable of operating reliably over a range of supply voltages make it possible to create processor cores with supply voltage that can be varied at run time according to application timing constraints. The variable voltage processor core can be made to operate at different optimal points along its power-versus-speed curve in order to achieve much higher energy efficiency than existing techniques for a wider class of applications. Such systems also require design flexibility, which results in the need for implementation on programmable processor platform. In fact, embedded software running on RISC and digital signal processing (DSP) processor cores has emerged as a leading implementation methodology for such applications as speech coding, modem functionality, video compression, and communication protocol processing [3], [46]. Current semiconductor technology allows the integration of programmable processors and memory structures on a single die, which enables the implementation of a system on a single chip. Similarly, the exponential growth of both applications and implementation technology has outpaced the design productivity of the traditional synthesis process. The shrinked time-to-market window has exacerbated the situation. There is a wide consensus that only through reuse of highly optimized cores can the demands of the pending applications and the potential ultra-large-scale integration be matched. Therefore, a low-power core-based system-on-a-chip (SOC), consisting of a variable voltage programmable processor core and a memory hierarchy, attracted much attention from virtually all silicon vendors.

Our synthesis technique targets typical a modern application-specific SOC, consisting of a variable voltage processor core, an instruction and data cache, and a number of optional hardware accelerators and control blocks. The distribution of power dissipation by the components of an application-specific SOC depends on the actual applications running on the system. However, extensive studies indicate that the power consumption of the processor and cache cores accounts for significant portion of the overall power consumption of the described SOC [2], [16]. Therefore, in this paper, we focus on the power optimization of the processor and cache cores.

Average power consumption in CMOS technology is given by $P = \alpha C_L V_{\rm dd}^2 f$ [6], where $f$ is the system clock frequency, $V_{\rm dd}$ is the supply voltage, $C_L$ is the load capacitance, and $\alpha$ is the switching activity. The most effective way to reduce power consumption of a processor core is to lower the supply-voltage level, which exploits the quadratic dependence of power on voltage [6]. Reducing the supply voltage, however, increases circuit delay and decreases clock speed. The resulting processor core consumes lower average power while meeting the deadlines. Unfortunately, this technique is ineffective when tight deadlines are present in systems. Another power optimization technique for processor cores is the system shutdown [46]. The system shutdown technique, though usable even in the presence of tight deadlines, is inferior to the supply-voltage reduction technique for the cases when both techniques can

Manuscript received May 22, 1998; revised May 14, 1999. This paper was recommended by Associate Editor E. Macii.

I. Hong is with Synopsys Inc., Mountain View, CA 94043 USA.

D. Kirovski, G. Qu, and M. Potkonjak are with the Computer Science Department, University of California, Los Angeles, CA 90095-1596 USA.

M. B. Srivastava is with the Department of Electrical Engineering, University of California, Los Angeles, CA 90095-1596 USA.

Publisher Item Identifier S 0278-0070(99)09915-7.

TABLE I
THE CHARACTERISTICS OF THE TWO TASKS USED TO ILLUSTRATE THE MOTIVATION FOR DYNAMICALLY VARIABLE-VOLTAGE APPROACH

| task | arrival | deadline | execution time at 3.3 V |
|---|---|---|---|
| A | 0 | 6 | 5 |
| B | 3 | 20 | 5 |

be applied. The limitations of the techniques arise due to the fact that systems are designed with a fixed supply voltage. The supply-voltage reduction technique attempts to find a single optimal voltage level for the entire processor operation, while the system-shutdown technique makes a binary run-time decision on whether to turn on or off the power supply.

The goal of the research presented in this paper is to develop the design methodology for the low-power, core-based, real-time SOC based on dynamically variable voltage hardware. We consider real-time systems since almost all embedded systems have to meet real-time constraints dictated by their environment [3]. The key challenge is to develop effective scheduling techniques that treat voltage as a variable to be determined, in addition to the conventional task scheduling and allocation. Our synthesis technique also addresses the selection of the processor core and the determination of the instruction and data cache size and configuration so as to fully exploit dynamically variable voltage hardware, which will result in significantly lower power consumption for a set of target applications than existing techniques. This paper is an extended version of [20].

### *B. Motivational Example*

To illustrate the key point of the proposed dynamically variable voltage approach, we consider a set of tasks as a motivational example, shown in Table I. Two independent computations $Task_A$ and $Task_B$ need to be executed on an embedded processor core in the time interval [0, 20]. Each task can be executed immediately after its arrival and is required to be finished by its deadline time. Preemption is not allowed due to the high context-switching cost.

Let the maximum supply-voltage level be $(V_{\rm dd})_{\rm ref} = 3.3V$. Power is normalized to its value at the reference point, i.e., $P(3.3V) = 1W$. Reducing supply voltage results in increased circuit delay; and, to a good accuracy, the circuit delay is given by $k \times (V_{\rm dd}/(V_{\rm dd} - V_t)^2)$, where $V_t$ is the threshold voltage and $k$ is a constant that depends on capacitance, mobility, and transistor dimensions [6]. Assuming a typical value of 0.8 $V$ for the threshold voltage, the execution time $T(V_{\rm dd})$ is the following function of supply voltage $V_{\rm dd}$:

$$\frac{T(V_{\rm dd})}{T(3.3V)} = \frac{V_{\rm dd}/(V_{\rm dd} - V_t)^2}{3.3V/(3.3V - V_t)^2} = 1.89V_{\rm dd}/(V_{\rm dd} - 0.8V)^2$$

The power consumption, as mentioned previously, is given by $P = \alpha C_L V_{\rm dd}^2 f$. Since $f$ is inversely proportional to delay, it follows that $P(V_{\rm dd})$ is the following function of supply voltage $V_{\rm dd}$:

$$\frac{P(V_{\rm dd})}{P(3.3V)} = \frac{V_{\rm dd}(V_{\rm dd} - V_t)^2}{3.3V(3.3V - V_t)^2} = \frac{V_{\rm dd}(V_{\rm dd} - 0.8V)^2}{20.62V}.$$

The system consumes 1 W when no power reduction techniques are applied. We now consider the application of shutdown, supply-voltage reduction, both shutdown and supply-voltage reduction, and dynamically variable voltage approach, respectively.

With the shutdown technique, the system will operate at $V_{\rm dd} = 3.3V$. Task$_A$ is executed in the interval [0, 5]. Task$_B$ is executed in the interval [5, 10]. The processor can be shut down for the interval [10, 20] and then be resumed for the next task. The duty cycle of the processor is 50%, so the average power consumption is 0.5 W.

With the supply-voltage reduction technique, the system will operate at a lower but fixed supply-voltage level. Unfortunately, the tight deadline on Task$_A$ means that the supply voltage cannot be lowered less than $V_{\rm dd} = 2.97$ V since otherwise Task$_A$ will miss its deadline. Thus, the system operates at $V_{\rm dd} = 2.97$ V with $P(2.97{\rm V}) = 0.67W$. Task$_A$ is executed in the interval [0, 6]. Task$_B$ is executed in the interval [6, 12]. The average power consumption is 0.67 W. Since the system can be shut down during the interval [12, 20], the average power consumption can be lowered to $0.67 \times (12/20) = 0.40$ W, which results in 20% power reduction, compared to the shutdown technique.

With the variable voltage hardware, one can schedule the two tasks such that Task$_A$ is executed in the interval [0, 6] at 2.97 V with $P(2.97{\rm V}) = 0.67$ W, and Task$_B$ is executed in the interval [6, 20] at 1.95 V with $P(1.95{\rm V}) = 0.11$ W. The average power consumption is $(0.67 \times 6 + 0.11 \times 14/20) = 0.28$ W, which is 44% lower than the shutdown technique and 30% lower than the supply-voltage reduction in combination with the shutdown technique.

The preceding example illustrates that it is always better to reduce voltage than to shut down when both techniques are applicable, and that dynamically variable supply voltage helps unleash this potential.

### *C. What Is New?*

In this paper, we develop the first approach for power minimization of variable-voltage core-based application-specific systems. The power minimization is addressed at several synthesis tasks including task scheduling, instruction and data cache size determination, and processor core selection. We explore static scheduling algorithms that treat voltage as an optimization degree of freedom for the applications with real-time constraints. By selecting the most efficient voltage profile in the presence of multiple timing constraints, our algorithms result in much larger savings in energy than other techniques such as the system shutdown and the supply-voltage reduction.

### *D. Paper Organization*

The rest of this paper is organized in the following way. Section II presents the related work in the field of low-power system- and behavioral-level synthesis and variable-voltage systems. Section III explains the necessary background material on variable-voltage systems. The design flow of the novel synthesis approach is presented in Section IV. In Section V, the optimization problems are recognized and their

computational complexities are established. In the same section, the synthesis algorithms are proposed and explained in detail. Section VI presents experimental data and discussion to evaluate the effectiveness of the approach. This paper is concluded in Section VIII.

## II. Related Work

Low-power design has emerged as an important area of research with the recent growth of wireless communications applications. We review the research results relevant to low-power systems based on dynamically variable-voltage hardware.

The prior research activity on technology, circuits, and techniques for variable-voltage systems is of direct relevance to our research. At the technology level, efficient dc–dc converters that allow the output voltage to be rapidly changed under external control have recently been developed. For example, Namgoong *et al.* from Stanford University [38] recently presented a high-efficiency dc–dc buck-converter switching regulator with adjustable output supply voltage. The output voltage ranges from 1.5 to 3.5 V while delivering a power level of 10 to 200 mW, with an efficiency of 83–93%. The time taken to reach steady state at the new voltage is under 6 ms/V. Stratakos *et al.* [47] have described an integrated buck-converter circuit with greater than 90% efficiency. While their design delivered a fixed 1.5 V from a 6-V battery, the design is generalizable to produce variable supply voltages. Suzuki *et al.* [49] reported a RISC processor with variable supply voltage, which can vary from 0.4 to 2.9 V from a 3.3-V external power supply. Such supply-voltage chips are crucial enablers for this research.

At the hardware design level, research work has been performed on chips with dynamically variable supply voltage that can be adjusted based on 1) process and temperature variations and 2) processing load as measured by the number of data samples queued in the input (or output) buffer. The first adjustment allows the minimum possible voltage for current process and temperature conditions to be used for a given clock frequency, whereas the second adjustment allows the computation to be slowed down and the supply voltage correspondingly lowered during periods of low workload. Dynamically adapting voltage to operate at the point of lowest power consumption for given temperature and process parameters was first suggested by Macken *et al.* [36]. Nielsen *et al.* [39] extended the dynamic voltage adaptation idea to take into account data-dependent computation times in self-timed circuits. Recently, researchers at the Massachusetts Institute of Technology [4] have extended the idea of voltage adaptation based on data-dependent computation time from [39] to synchronously clocked circuits. The approach is useful when the computation time is data dependent, which is often the case in signal processing, for example, a forward error corrector chip for a digital cassette recorder where the processing time depends on the number of errors in the input sample. Buffering data and averaging processing rate can yield power reductions of an order of magnitude in some applications, but with increased latency [18], [39]. Because these approaches rely on run-time reactive approaches to dynamic voltage adaptation, they work fine only where *average throughput* is the metric of performance. Therefore, these approaches are inapplicable to hard real-time systems such as embedded control applications with strict timing requirements.

There has been research on scheduling strategies for adjusting CPU speed so as to reduce power consumption. Most of the existing work is in the context of a nonreal-time workstation-like environment. Weiser *et al.* [52] proposed an approach where time is divided into 10–50-ms intervals, and the CPU clock speed (and voltage) is adjusted by the task-level scheduler based on the processor utilization over the preceding interval. The CPU speed is slightly lowered or raised depending on whether the utilization has been low or high, and is raised to the maximum level if the CPU is falling behind. Energy is saved by trying to spread cycles into what would otherwise have been idle time. Govil *et al.* [17] enhanced [52] by proposing and comparing several predictive and nonpredictive approaches for voltage changes and concluded that smoothing helps more than prediction. Yao *et al.* [54] described a minimum energy schedule and an average rate heuristic for job scheduling with *preemption* for independent processes with deadlines.

Chou and Borriello [8] presented a static nonpreemptive software scheduling algorithm for reactive hard real-time systems in the context of hardware/software cosynthesis. Given a constraint graph for tasks with several timing constraints, their algorithm is guaranteed to find a feasible ordering if one exists. The drawback of the algorithm is high complexity, which is exponential in the worst case. Dave *et al.* [10] have also proposed a real-time scheduling algorithm for software scheduling, which allows both preemptive and nonpreemptive scheduling.

Landman and Rabaey [31], [32] have performed the work on macro- and microlevel power estimation. The work at Princeton and Fujitsu on the estimation of power consumption for programmable processors [50] is of particular relevance.

Evans and Franzon [12] and Su and Despain [48] have proposed power estimation models for cache, specifically for SRAM. Many researchers have proposed power-efficient cache designs, e.g., [28]. Compiler-directed techniques for minimizing power consumed by cache have also received significant attention in the research community [30], [40]. Panda *et al.* [42] presented a technique for determining the best data cache size for a given application. Panda *et al.* [41] presented techniques to minimize data cache conflicts for direct-mapped caches by analyzing the array access patterns. Kirovski *et al.* [26] developed techniques to minimize instruction/data cache conflicts for direct-mapped caches. The techniques proposed in [26], [30], and [40]–[42] are complementary to our approach since they can be applied as a preprocessing step. The techniques proposed in [26], [30], and [40] reduce the cache conflicts, while those proposed in [41] and [42] reduce the size of the cache or improve the performance of the cache, mainly by optimizing memory and cache data assignment. While the previous works concentrated on the memory and cache data assignment, our work considers other complementary issues such as processor selection, cache sizing, task scheduling, and voltage scheduling simultaneously.

One well-known technique that results in substantial power reduction is reducing the voltage to reduce the switching power, which is the dominant source of power dissipation in a CMOS circuit and is proportional to $V_{\rm dd}^2$, where $V_{\rm dd}$ is the supply voltage [6]. Indeed, supply-voltage reduction of VLSI chips to save power has long been used in memories and watches, where the on-chip supply voltage is reduced to a fixed value [36]. However, this reduction came with a speed penalty due to increased gate delays. Chandrakasan *et al.* [6], [5] have shown that the strategy of operating at a fixed reduced voltage can be coupled with architectural-level parallelism and pipelining to compensate for lower clock rate due to voltage reduction so that the overall throughput remains the same but the overall power is still lowered, although at the cost of increased latency.

Several researchers have addressed the issue of power in event-driven systems and proposed various techniques for shutting down the system or parts of the system. Examples include the predictive shutdown of software programmable processor in a wireless X terminal [22], [46] and disk shutdown [11]. At an implementation level, circuit techniques for low-power hardware have been proposed [6]. Compilation techniques for low-power software have emerged for both general-purpose computation [50] and DSP computations [21]. Numerous behavioral synthesis research efforts have also addressed power minimization; for example, see the survey of [43].

Four research groups in particular have addressed the use of multiple (in their software implementation, restricted to two or three) different voltages for behavioral synthesis [7], [23], [35], [44]. Interestingly, although they used the term "variable voltage," they actually addressed scheduling when a fixed number of simultaneously available voltages are used. Therefore, there is no similarity between our work and these efforts beyond accidental similarity of terminology.

## III. Preliminaries

In this section, we describe the task model and hardware and power models for processor cores, caches, and variable-voltage hardware.

### A. Computational and Timing Models

A set $J$ of independent tasks is to be executed on a SOC. Each task $j \in J$ is associated with the following parameters:

- $a_j$ is its arrival time;
- $d_j$ is its deadline;
- $p_j$ is its period;
- $r_j$ is its execution time at the nominal (maximum possible) voltage level $V_{\rm ref}$.

We assume that all tasks are defined on semi-infinite or very long streams of data. A task consumes one set of input and produces one set of output at each iteration. The duration of each iteration for a task is a period for the task. The execution time of a task is a worst case execution time since all tasks should always satisfy their timing constraints. Many real-time systems can be modeled using our computational/timing model. An example system is a set of avionics tasks that runs on an avionics application-specific system [1], [45]. The task set consists of 24 independent periodic tasks. Another example is a multimedia server that provides multimedia data to multiple users [37].

Without loss of generality, we assume that all tasks have identical periods. When this is not the case, a simple preprocessing step and application of the least common multiple (LCM) theorem [33] transforms an arbitrary set of periods to this design scenario in polynomial time. As a consequence, when the LCM theorem is applied, there may be a need to run several iterations of a given task within this overall period. Since context switching overhead is usually high for modern systems, we assume no task preemption. Note that this preemption restriction actually does not affect any of the proposed methods, since in all discussed design cases nonpreemptive policies yield superior results in comparison with preemptive policies.

### B. Power Model

It is well known that there are three principal components of power consumption in CMOS integrated circuits: switching power, short-circuit power, and leakage power. The switching power is given by $P = \alpha C_L V_{\rm dd}^2 f$ [6], as indicated earlier. $\alpha C_L$ is defined to be effective switched capacitance. In CMOS technology, switching power dominates power consumption. It is also known that reduced voltage operation comes at the cost of reduced throughput [6]. The gate delay $T$ follows the following formula: $T = k(V_{\rm dd}/(V_{\rm dd} - V_t)^2)$ where $k$ is a constant [6]. The maximum rate at which a circuit is clocked monotonically decreases as the voltage is reduced. From these equations together with the observation that the speed is proportional to $f$ and inversely proportional to the gate delay, the power-versus-speed curve can be derived, in particular the normalized power $P_n$ ($= 1$ at $V_{\rm dd} = V_{\rm ref}$) as a function of the normalized speed $S_n$ ($= 1$ at some reference $V_{\rm dd} = V_{\rm ref}$). For example, assuming $V_{\rm ref} = 3.3$ V and $V_t = 0.8$ V, the power-versus-speed curve follows the following equation:

$$\begin{aligned} P_n =& 0.164 \cdot S_n^3 + \sqrt{0.893 \cdot S_n^2 + 1.512 \cdot S_n} \\ & \cdot (0.173 \cdot S_n^2 + 0.147 \cdot S_n) + 0.277 \\ & \cdot S_n^2 + 0.059 \cdot S_n. \end{aligned}$$

By varying $V_{\rm dd}$, the system can be made to operate at different points along this curve.

From the equation, it is easy to see that the power is a convex function of speed. It immediately follows that if there is a single computation that needs to be executed by some deadline, it is always better to run the computation at a constant speed corresponding to the average speed required than to change the speed (and correspondingly voltage) during the computation. In the proposed scheduling heuristic, we use this observation so that each task is executed at a single voltage. However, the level of this single voltage may change from task to task, and one of our goals is to determine this "single" voltage for each task.

We assume that the power consumption of software running on a processor is proportional to the number of instructions.

This assumption is based on the fact that for some modern processors such as the Fujitsu SPARClite MB86934, a 32-bit RISC microcontroller, power is directly proportional to the number of operations, regardless of the mix of operations being executed [51]. Tiwari and Lee [51] report that all the operations including integer arithmetic logic unit instructions, floating-point instructions, and load/store instructions incur similar power consumption, whereas different instructions draw different amounts of power consumption for other processors such as Intel 486DX2 and Fujitsu DSP processor [34], [50]. In many cases, the same instruction draws different amount of currents depending on the context such as the predecessor and successor instructions [50]. When the power consumption depends on the mix of operations being executed, as in the case of the Intel 486DX2 and Fujitsu DSP processor [34], [50], s more detailed hardware power model may be needed. Note that our approach does not rely on any specific power model and thus can be applicable with any power model.

### C. Architecture and Hardware Model

Several factors combine to influence system performance: instruction and data cache miss rates and penalty, processor performance, and system clock speed. Power dissipation of the system is estimated using processor power dissipation per instruction and the number of executed instructions per task, supply voltage, energy required for cache read, write, and off-chip data access, as well as the profiling information about the number of cache and off-chip accesses. The approach [26], [27] that we use to realistically address the factors used to estimate power consumption leverages on existing cache and processor models. This enables the synthesis approach to be rapidly updated and applied to environments with new technologies.

The system performance is computed using the following formula for cycles per instruction (CPI):

$$\begin{aligned}\text{CPI} = \frac{f}{\text{MIPS}} + (I - \text{Cache Miss Ratio} \\ + \text{D-Cache Miss Ratio}) \\ \cdot \text{Cache Miss Penalty}\end{aligned}$$

where $X-$ Cache Miss Ratio was computed during the trace driven simulation of the cache subsystem and $X = I$ or $D$. $f$ is a system clock frequency, and MIPS stands for million instructions per second. Cache Miss Penalty, $f$, and MIPS are system parameters introduced in Section III. The system energy consumption (SEC) is computed using the following formula:

$$\begin{aligned}\text{SEC} = &(TC - CPUIC) \cdot CPUEPC + CA \cdot EPCA \\ &+ MA \cdot EPMA\end{aligned}$$

where $TC$ is the total number of cycles required to execute the application, $CPUIC$ is the number of cycles while the CPU is idle waiting for data to be fetched from main memory to cache (blocking caches are assumed for low hardware cost), $CPUEPC$ is the average energy consumed per cycle for a particular processor core, $CA$ and $EPCA$ are the number of cache accesses and energy consumed per access, respectively, and $MA$ and $EPMA$ are the number of main memory accesses (reads due to cache misses and write-back updates) and energy consumed per memory access, respectively. Since power consumed by the cache decoder represents approximately 30% of the overall cache power consumption [12], the power savings due to the decrease of transitions in the cache decoder are accounted for by scaling down 30% of the total cache power consumption with the ratio of the number of transitions due to address change in the optimized (relocated) and traditional basic block schedules. The constants in the formulas have been either obtained from the manufacturer's datasheets or computed using our system performance and power evaluation and simulation platform. The above formulas have been adopted from [26] and [27].

Data on microprocessor cores have been extracted from manufacturer's datasheets as well as from the CPU Center Info Web site.[1] We assume that the processor cores follow the power model described in the preceding subsection.

We use CACTi [53] as a cache delay estimation tool with respect to the main cache design choices: size, associativity, and line size. The energy model is adopted from [12] and [48]. The overall cache model is scaled with respect to actual industrial implementations. Caches typically found in current embedded systems range from 128 B to 32 KB. Although larger caches correspond to higher hit rates, their power consumption is proportionally higher, resulting in an interesting design tradeoff [12], [28]. Higher cache associativity results in significantly higher access time [53]. We use a recently developed compiler strategy that efficiently minimizes the number of cache conflicts for direct-mapped caches [26]. We have experimented with two-way set-associative caches, which did not dominate comparable direct-mapped caches in a single case. Cache line size was also variable in our experimentations. Its variations corresponded to the following tradeoff: larger line size results in higher cache miss penalty delay and higher power consumption by the sense amplifiers and column decoders, while smaller line size results in larger cache decoder power consumption. Extreme values result in significantly increased access time. We estimated the cache miss penalty based on the operating frequency of the system and external bus width and clock for each system investigated. This penalty ranged between four and 20 system clock cycles. Write-back was adopted as opposed to write-through, since it is proven to provide superior performance and especially power savings in uniprocessor systems at increased hardware cost [24]. Each of the processors considered is constrained by the Flynn's limit [13] and is able to issue at most a single instruction per clock period. Thus, caches were designed to have a single access port. Cache access delay and a power consumption model were computed for a number of organizations and sizes, assuming the feature size of 0.5 $\mu$m and typical six transistors per CMOS SRAM cell implementation. The nominal energy consumption per single off-chip memory access, 98 nJ, is adopted from [14].

[1] See http://infopad.eecs.berkeley.edu/CIC/.

### D. Variable-Voltage Hardware

The variable voltage is generated by the dc–dc switching regulators in the power supply, which is essentially a feedback control system. The time for the voltage to reach a steady state at the new voltage is a strong function of the feedback loop parameters and the LC output filter in the switching regulator. Reference [38] reported efficient DC–DC switching regulators with fast transition times. DC–DC converters with faster transition times may be obtained with the cost of increased power dissipation within the converter. Dual to the supply-voltage variation is the accompanying variation of the clock frequency. The clock frequency also takes time to stabilize at the new value. The time and power overhead associated with voltage switching is negligible [38] and is not considered in the analysis. In addition, the computation itself can continue even during the voltage and frequency transition.

### E. Summary of Theoretical Results

The *nonpreemptive* scheduling of a set of independent tasks with arbitrary arrival times and deadlines on a *fixed voltage* processor is an NP-complete task [15]. Therefore, the *nonpreemptive* scheduling of a set of independent tasks with arbitrary arrival times and deadlines on a *variable-voltage* processor is also an NP-complete task since the preceding problem is its special case. Yao *et al.* [54] have provided the optimal *preemptive* static scheduling algorithm for a set of independent tasks with arbitrary arrival times and deadlines. The running time of the algorithm is $O(n \log^2 n)$ for $n$ tasks. Since any optimal nonpreemptive scheduling solution is always worse than or equal to the solution of its preemptive version when there is no overhead for preemption, it can serve as a lower bound for *nonpreemptive* scheduling solutions. This lower bound plays a crucial role in speeding up our branch and bound algorithm for the *nonpreemptive* scheduling problem.

## IV. Design Methodology for Variable-Voltage Systems: Global Design Flow

In this section, we describe the global flow of the proposed synthesis system and explain the function of each subtask and how these subtasks are combined into a synthesis system. The goal is to choose the configuration of processor, I-cache, and D-cache and the variable-voltage task schedule with minimum power consumption that satisfies the requirements of multiple nonpreemptive tasks. To accurately predict the system performance and power consumption for target applications, we employ the approach that integrates the optimization, simulation, modeling, and profiling tools. The synthesis technique considers each nondominated microprocessor core and competitive cache configuration and selects the hardware setup that requires minimal power consumption and satisfies the individual performance requirements of all target applications. The application-driven search for a low-power core and cache system requires usage of trace-driven cache simulation for each promising point considered in the design space. We attack this problem by carefully scanning the design space using search algorithms with sharp bounds and by providing powerful algorithmic performance and power estimation techniques.

We use the system performance and power evaluation and simulation platform based on SHADE [9] and DINEROIII [19]. SHADE is a tracing tool that allows users to define custom trace analyzers and thus collect rich information on run-time events. SHADE currently profiles only SPARC binaries. The executable binary program is dynamically translated into host machine code. The tool also provides a stream of data to the translated code, which is directly executed to simulate and trace the original application code. A custom analyzer composed of approximately 2500 lines of C code is linked to SHADE to control and analyze the generated trace information. The analyzer sources relevant trace information from SHADE and builds a control flow graph (CFG) corresponding to the dynamically executed code. The analysis consists of two passes. The first pass determines the boundaries of basic blocks, while the second pass constructs a CFG by adding control flow information between basic blocks. We also collect the frequencies of control transfers through each basic block and branch temporal correlation [26]. Once the CFG is obtained, an algorithm is employed to reposition application basic blocks in such a way that instruction cache misses and cache decoder switching activity are minimized [27]. Our experimentation uses a basic block relocation lookup table to simulate the relocation of basic blocks in main memory. An entry in the basic block relocation table consists of two elements: the original and optimized starting address of the basic block. To simulate cache performance of a given application and data stream, we use a trace-driven cache simulator DINEROIII. Cache is described using a number of qualitative and quantitative parameters such as instruction and data cache size, replacement policy, associativity, etc.

The system optimization process is composed of a sequence of activations of these tools. The SHADE analyzer traces program and data memory references as well as the CFG. The CFG is used to drive the code reposition module, which produces a new application mapping table. Streams of references are sent to a program that uses the basic block relocation lookup table to map from the original address space into the optimized address space. The remapped trace of addresses, along with all unmodified data memory references, is sent to DINEROIII for cache simulation. We use our custom program, which computes performance and power consumption based on the formulas given in Section III-C.

The synthesis search loop generates iteratively hardware configurations. For each nonevaluated cache configuration, the synthesis platform estimates its performance when a particular application is executed. Then, the performance data are used during the scheduling process for estimation of task run times. If the scheduler finds a feasible schedule, it reports the power consumption of this hardware configuration to the resource allocation algorithm.

## V. Synthesis Algorithms

The optimization problems encountered in the synthesis of a low-power variable-voltage SOC and competitive optimization algorithms are described in this section.

```
Sort processor cores in a list L in an increasing order of EnergyPerInstruction / SystemClockFrequency at the nominal voltage
Delete the dominated processor cores from L.
For each processor core in L in the order of appearance
    For I-cache = 512B..32KB and CacheLineSize = 8B..512B
      For D-cache = 512B..32KB and CacheLineSize = 8B..512B
        Check bounds; if exceeded break.
        If (current I- and D-cache configuration has never been evaluated)
            Evaluate performance and power consumption of the cache structure
        Using the cache system analysis, evaluate the system power consumption by variable voltage task scheduling.
        Memorize Configuration C if power consumption is minimal.
```

Fig. 1. Pseudocode for the resource allocation procedure.

### A. Resource Allocation

In this phase of the synthesis approach, a search is conducted to find an energy-efficient system configuration. The search algorithm is described using the pseudocode shown in Fig. 1. Since performance and power evaluation of a single processor, I- and D-cache configuration requires a trace-driven simulation, the goal of our search technique is to reduce the number of evaluated cache systems using sharp bounds for cache system performance and power estimations. In addition, a particular cache system is evaluated using trace-driven simulation only once since the data retrieved from such simulation can be used for overall system power consumption estimation for different embedded processor cores with minor additional computational expenses.

The algorithm excludes from further consideration processor cores dominated by other processor cores. One processor type dominates another if it consumes less power at same or higher frequency and results in higher MIPS performance at the same nominal power supply. The competitive processors are then sorted in ascending order with respect to their power consumption per instruction and clock frequency ratio. Microprocessors, which seem to be more power efficient, are therefore given priority in the search process.

The search for the most efficient cache configuration is bounded with sharp bounds. A bound is determined by measuring the number of conflict misses and comparing the energy required to fetch the data from off-chip memory due to measured conflict misses and the power that would have been consumed by twice larger cache for the same number of cache accesses assuming zero cache conflicts. Note that we assume that cache power is a monotone function of size. We terminate further increase of the cache structure when the power consumption due to larger cache would be larger than the energy consumed by the current best solution. Similarly, another bound is defined at the point when the energy required to fetch the data from off-chip memory, due to conflict cache misses for twice smaller cache with the assumption of zero-energy consumption per cache access, is larger than the energy required for both fetching data from cache and off-chip memory in the case of the current cache structure. We abort further decrease of the cache structure if the amount of energy required to bring the data due to additional cache misses from off-chip memory is larger than the energy consumed by the current best solution.

When evaluating competitive hardware configurations, the target applications are scheduled with the variable-voltage task scheduler using the predicted system performance and power on the configuration considered. Before the nonpreemptive task scheduling is performed, a more efficient optimal preemptive scheduler [54] is applied to get a lower bound. If the result is worse than the current best solution, the configuration is worse than the current best one.

### B. Task Scheduling

As described in Section III, the nonpreemptive scheduling of a set of independent tasks with arbitrary arrival times and deadlines on a variable-voltage processor is an NP-complete task. We have developed an efficient and effective heuristic for the general variable-voltage task-scheduling problem, which leverages least constraining, most-constrained heuristic paradigm in order to obtain competitive solutions. The algorithm is described using the pseudocode shown in Fig. 2.

We first define the terms used to describe the algorithm. The *lifetime* of a task is defined as the time between the task's arrival time and its deadline. The *time partition* and *time interval* represent any contiguous time period between two time points. In particular, the time partition is used to refer to one of the two split time intervals of a task. The *start (finish) time* of a task is the time when the task starts (finishes) its execution.

The algorithm consists of two phases. In the first, all tasks are scheduled at the nominal voltage. If a feasible schedule is found in the first phase, in the second phase the supply voltages are adjusted for low power. These two phases are repeated $LOOPS$ times, and the best solution obtained from all the iterations is chosen as the final solution. Each iteration of the two phases generates different solutions since the objective functions used in the first phase to guide the search strategy are randomized by a random offset. The complexity of the algorithm is $O(n^3)$ for $n$ tasks.

In the first phase, the task scheduling at the nominal voltage is done in the following way. Initially, the time period in which the tasks appear is divided into time regions such that each time region has its start or end time equal to a start or end time of some task and there does not exist a task that has either an arrival or deadline time within the time region. Then, for each time region $tr$, an objective function $OBJTR(tr)$ is computed as

$$OBJTR(tr) = maxVoltage(tr) \cdot aveVoltage(tr)$$

where the functions $maxVoltage(tr)$ and $aveVoltage(tr)$

**Given:** a set of tasks $T$, where each task $t \in T$ is characterized by its arrival time $a_t$, its deadline $d_t$, and its execution time $r_t$ at the nominal voltage

**Repeat** $LOOPS$ times

**Scheduling tasks at the nominal voltage**

**Repeat**

Create a set $TR$ of time regions $tr_i[start_i, end_i] \in TR$ where $start_i$ and $end_i$ correspond to an arrival or deadline time of two not necessarily different tasks and there does not exist another task $t_o \in T$ with $start_i < a_{t_o}, d_{t_o} < end_i$.

Select the task $t \in T$ such that $OBJ(t) > OBJ(x), x \in T, x \neq t$ and $\neg\exists s \in T$ such that $a_s > a_t$ and $d_s < d_t$.

Schedule $t$ to the time interval with the smallest $S$.

Delete $t$ from the list of tasks $T$ and update the set of time regions $TR$ correspondingly.

**For** each task $s \in T$

**If** the schedule of $t$ splits the life-time of task $s$ into two partitions both larger than $r_s$

Assign the task $s$ to the partition that includes the time interval with the smallest $S$.

Update the task arrival and deadline times accordingly.

**Until** all tasks are scheduled.

**Adjusting voltages of scheduled tasks for low-power**

**For** each task $t$

Compute the earliest start $st_t$ and latest finish $fi_t$ times according to the existing scheduling order.

Compute the minimal voltage $v_t$ at which the task can run if expanded to the time interval from $st_t$ to $fi_t$.

**Repeat**

**For** each task $t$

Adjust the start/finish times of the task $t$ and its adjacent tasks according to Figure .

**Until** no further energy consumption improvements.

Compute system energy $E$.

**If** $E < BestE$

$BestE = E$.

**Return (BestE)**

Fig. 2. Pseudocode for the variable-voltage task-scheduling algorithm.

assume that all tasks are scheduled from their arrival until their deadline regardless of time overlaps and return the maximum and average voltage for all tasks alive at time region $tr$, respectively. Larger $OBJTR(tr)$ implies that the time region $tr$ is more constrained. The objective functions (or constraints) of time regions are used to compute the constraints of tasks. For each task $t$, an objective function $OBJ(t)$ is computed as

$$OBJ(t) = \sum_{tr \in [a_t, d_t]} OBJTR(tr) \cdot \frac{r_t^2}{(d_t - a_t)^2}.$$

Larger $OBJ(t)$ implies that the task $t$ is more constrained.

The most constrained task $t$, which does not completely include the lifetime of another tasks in its lifetime, is then scheduled in the time interval equal to its run time at the nominal voltage, which spans over a subset of time regions with the lowest sum of objective functions among any feasible subset of time regions. When comparing the subset of time regions, we only consider the cases that the start time of the task is only at the start time of each time region in the task's lifetime or the end time of the task is only at the end time of each time region. For each time interval [start, end] of all eligible time intervals, we compute $S = \Sigma_{tr \in [start, end]}\ OBJTR(tr)$. We schedule the task $t$ in the time interval with the smallest $S$. Upon scheduling the task $t$, other tasks' deadlines and arrival times are updated in the following way.

- If the scheduled task's lifetime is included in the lifetime of some other task $s$, and the task $s$ can be scheduled at both partitions of its lifetime upon deleting the time regions that contain the task $t$, then both partitions of the task's lifetime are evaluated for scheduling. Objective functions are computed for all the time intervals with their length equal to the task's nominal voltage run time such that the start time of the time interval is only at the start time of each time region, or the end time of the time interval is only at the end time of each time region, shown in the lower part of Fig. 3. For each time interval, we compute $S$ using the same formula in the above. The partition that includes the time interval with the minimal $S$ is selected since it is likely to least constrain other tasks in the later steps.
- Otherwise, each remaining task's arrival time and deadline is updated if its lifetime intersects with the lifetime of the task $t$.

This process is repeated until all tasks are scheduled at the system nominal voltage.

We explain the process using the example shown in Fig. 3. Six tasks T[0], T[1], T[2], T[3], T[4] and T[5] are shown with their arrival, deadline, and nominal voltage run times. The set of tasks creates a set of nine time regions TR[0]–TR[8]. Based on objective functions computed, task T[4] is selected as the most constrained. Once this task is scheduled, the arrival times and deadlines of tasks T[2] and T[3] are updated as shown (see arrows in Fig. 3). However, task T[1] can be scheduled at both TR[1, 2, 3] and TR[5, 6, 7]. There are five possible schedules with respect to the existing time regions at TR[1, 2, 3] and three (actually five, but only three distinct) at TR[5, 6, 7], as shown in the bottom part of Fig. 3. The least constraining schedule (marked in Fig. 3) is in the TR[5, 6, 7] partition since during these time regions, T[1] can be scheduled in the time interval, where there are no other alive tasks. Therefore, the lifetime of task T[1] is reduced to TR[5, 6, 7]. Next, the task T[3] is most constrained. The task T[3] is scheduled at the start of TR[5], and the lifetime of the task t[1] is adjusted.

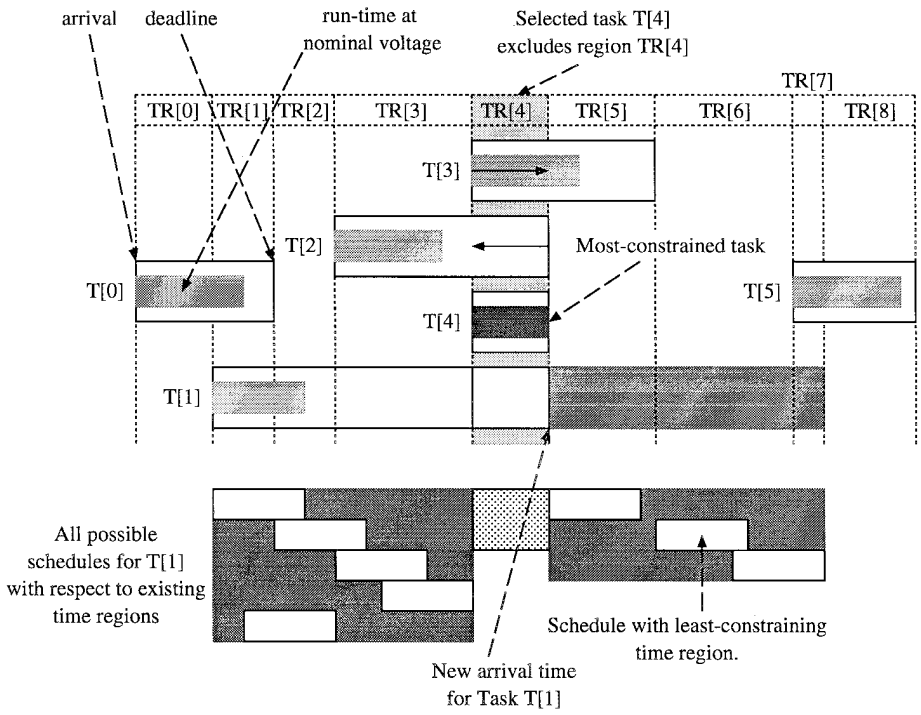


Fig. 3. Scheduling tasks at nominal voltage.

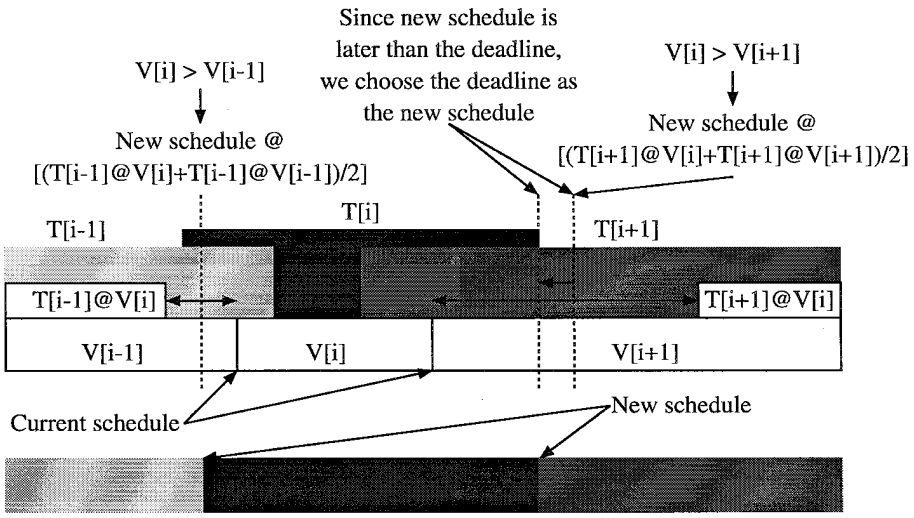


Fig. 4. Adjusting the voltage of scheduled tasks for low power.

The task T[5] is now most constrained and is scheduled to be finished at the end time of TR[8]. The lifetime of the task T[1] is again adjusted. At this point, the lifetimes of the tasks T[0], T[1], and T[2] do not have any intersections with other tasks' lifetimes, so they are scheduled at their arrival times. Note that the tasks can be also scheduled to be finished at their deadlines since both schedules have the same $OBJ$ value. When the two schedules have the same $OBJ$ value, we choose the one with earlier start time in our implementation.

Once tasks are scheduled, the procedure that tunes the supply voltage of each task is invoked. It iteratively traverses the set of tasks and tries to lower the voltage of each task until no energy dissipation improvement occurs. Whenever a task $t$ has a neighboring task $s$ with a lifetime that intersects the lifetime of $t$ and supply voltage $v_s$ lower than $v_t$, the border between the tasks is moved according to Fig. 4. For example, in Fig. 4, task T[i] has been assigned a supply voltage higher than task T[i-1]. The new arrival time of T[i] and deadline time of T[i-1] is moved to the arithmetic mean of the current arrival/deadline time, and the arrival/deadline time as if T[i-1] is supplied with T[i]'s voltage V[i]. When the arithmetic mean is later (earlier) than the deadline (arrival time) of T[i], we choose the deadline (arrival time) as the new schedule time. For a fixed task-scheduling problem, this voltage adjustment algorithm results in solutions arbitrarily close to the optimal solution, given the task schedule at the nominal voltage.

## VI. Experimental Results

In this section, we report the results of the conducted experiments. First, we describe the benchmarks and its input data used for the evaluation of our approach. Next, we analyze the experimental results.

We used nine applications, shown in Table II, to demonstrate the effectiveness of the approach. All nine applications are in the public domain and can be obtained via the Internet.[2] JPEG software from the Independent JPEG Group implements JPEG baseline, extended-sequential, and progressive compression processes. We used integer discrete cosine transform (DCT) for decoding a JPEG file to a PPM file. Integer DCT and progressive compression process options were used for compression. GSM software used in the experimentation is an implementation of the European GSM 06.10 provisional standard for full-rate speech transcoding, prI-ETS 300 036, which uses residual pulse excitation/long term prediction coding at 13 kbit/s. A 16-bit linear pulse-code-modulation data file is used to measure execution parameters of the GSM encoder. The measurement of the GSM decoder execution characteristics was done using the output of the GSM encoder. EPIC (Efficient Pyramid Image Coder) implements lossy image compression and decompression utilities. The compression algorithm is based on a critically sampled (imperfect-reconstruction) dyadic wavelet decomposition and a combined run-length/Huffman entropy coder. The remaining benchmarks (Mipmap, Osdemo, and Texgen) use a three-dimensional (3-D) graphic library called Mesa. The author of the library claims that Mesa is a valid alternative to OpenGL since it uses the OpenGL application programming interface. Mipmap is a simple mipmap texture mapping example. Osdemo is a demo program that draws a set of simple polygons using the Mesa 3-D rendering pipe. Last, Texgen renders a texture mapped version of the Utah teapot.

In order to evaluate the efficiency of our approach for variable-voltage system synthesis, we conducted a set of experiments. The processor cores are assumed operational at the dynamically variable [0.8, 3.3]-V supply voltage. Performance and power consumption data have been scaled according to the performance-versus-voltage and power-versus-voltage relationships, respectively. We experimented with our variable-voltage scheduling algorithm by scheduling six different sets of applications and timing constraints on a number of hardware configurations. Each application set was defined using a set of $K$ tasks ($K = 10 \cdots 50$). Each task encompasses one execution of an application from Table II. For each application, we established four different quality regimes, which resulted in four different power-timing performances. The regime of the application execution was not changed at run time. The tasks were scheduled within the least common multiple of their periods. The timing constraints (start and end time) were determined pseudorandomly with a goal of achieving tight schedules. The input traces of the task mix have been adopted from state-of-the-art video stream modeling reports [29]. We compared the obtained results (column *new*) with a lower bound obtained using Yao's preemptive scheduling algorithm with assumed zero preemption cost (column LB). We also compared them with the results of a fixed voltage system, where the fixed voltage used was the minimum voltage that can satisfy all constraints (column FV). Last, to demonstrate the effectiveness of variable voltage systems over multiple voltage systems, we have performed an additional comparison with a system supplied with three different optimized voltage supplies (column 3VS). The results are presented in Table III. The values in Table III are SEC values defined in Section III-C. The low-power processor-cache application-mix champion configurations are described in the first five columns of Table III. The next six columns present the power consumption lower bound, the result obtained using our algorithm, and the result obtained by a fixed voltage system for six different sets of applications extracted from Table II.

[2] See http://www.cs.ucla.edu/ leec/mediabench/applications.html

TABLE II
A Brief Description of the Applications Used in the Experimentation: IC—Number of Dynamic Instructions in Millions

| Application | IC | Source | Description |
|---|---|---|---|
| JPEG encoder | 14.1 | Independent | JPEG image |
| JPEG decoder | 3.8 | JPEG Group | encoding/decoding |
| GSM encoder | 184.2 | Technische | European wireless voice |
| GSM decoder | 73 | Universität | coding standard |
| EPIC encoder | 50.4 | University of | Wavelet image |
| EPIC decoder | 7.3 | Pennsylvania | encoding/decoding |
| Mipmap | 48.5 | University of | 3-D rendering examples |
| Osdemo | 8.9 | Wisconsin | using Mesa graphics |
| Texgen | 84.7 | | library |

Note that our nonpreemptive scheduling algorithm resulted in solutions that were only 1.47% higher on average than the lower bound determined using Yao's preemptive scheduling algorithm over a series of more than $10^4$ scheduling solutions. In addition to the results presented in Table III, in order to emphasize the advantage of relying on variable voltage while scheduling, we report an average standard voltage deviation per schedule to be 0.61 V over the same set of scheduling solutions. The average maximal voltage per schedule was 3.12 V, while the average minimal voltage was 1.08 V with respect to the nominal 3.3 V and threshold voltage $V_t = 0.8$ V. The importance of effective allocation for more complex systems is emphasized in Table III, where for each application mix and set of champion configurations, the average power consumption among the champion configurations is presented in rows 8, 16, and 24. As shown, the importance of having allocation analysis is increasingly important as the system complexity increases. While for the ten-task mix the best configuration is only 3% better than the average, our allocation technique finds a configuration for the case of a 50-task mix such that 25% power savings are achieved compared to the average configuration.

We have also applied the variable voltage synthesis strategy to a problem of scheduling 24 avionics tasks [1] and scheduling 104 tasks of an elbow manipulator [25]. Since the code for both applications was not available, we did not use the code relocation technique in the optimization process. However, the power consumption was obtained by employing the scheduling technique for variable voltage and a single instruction per cycle execution model. The energy dissipation equaled:

TABLE III
EVALUATION OF THE SCHEDULING ALGORITHM ON A SET OF ARCHITECTURES AND APPLICATION MIXES: LB—LOWER BOUND, FV—FIXED VOLTAGE APPROACH, 3VS—THREE FIXED VOLTAGE SUPPLIES

| Processor Core | I-cache | | D-cache | | 10 Tasks - Energy ($J$) | | | | 15 Tasks - Energy ($J$) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Cache | Block | Cache | Block | LB | New | 3VS | FV | LB | New | 3VS | FV |
| Motorola 68000 | 1KB | 128B | 2KB | 128B | 2.46 | 2.50 | 4.11 | 5.53 | 2.61 | 2.66 | 4.87 | 6.49 |
| ARM7 LPower | 1KB | 128B | 2KB | 128B | 2.42 | 2.47 | 4.03 | 5.47 | 2.50 | **2.54** | 4.72 | 6.21 |
| LSI TR4101 | 512B | 64B | 4KB | 32B | 2.43 | 2.50 | 4.17 | 5.54 | 2.52 | 2.58 | **4.68** | 6.39 |
| LSI CW4001 | 512B | 64B | 1KB | 128B | **2.39** | **2.44** | **3.91** | **5.40** | **2.49** | **2.54** | 4.70 | **6.20** |
| PowerPC403 | 1KB | 32B | 2KB | 32B | 2.48 | 2.55 | 4.24 | 5.64 | 2.75 | 2.78 | 4.91 | 6.79 |
| LSI CW4011 | 1KB | 64B | 2KB | 64B | 2.50 | 2.59 | 4.33 | 5.73 | 2.72 | 2.77 | 4.87 | 6.76 |
| Average | | | | | *2.44* | *2.51* | *4.13* | *5.52* | *2.58* | *2.65* | *4.77* | *6.47* |

| Processor Core | I-cache | | D-cache | | 10 Tasks - Energy ($J$) | | | | 15 Tasks - Energy ($J$) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Cache | Block | Cache | Block | LB | New | 3VS | FV | LB | New | 3VS | FV |
| Motorola 68000 | 1KB | 128B | 2KB | 128B | 11.1 | 11.9 | 43.9 | 56.0 | **27.3** | **27.5** | 101.7 | **130.0** |
| ARM7 LPower | 1KB | 128B | 2KB | 128B | 12.3 | 13.1 | 44.1 | 61.6 | 29.9 | 31.8 | 112.0 | 150.3 |
| LSI TR4101 | 512B | 64B | 4KB | 32B | **10.6** | **11.0** | **41.7** | **51.7** | **27.3** | 27.9 | **96.3** | 131.5 |
| LSI CW4001 | 512B | 64B | 1KB | 128B | 13.8 | 15.0 | 46.3 | 70.5 | 35.0 | 37.4 | 131.4 | 176.9 |
| PowerPC403 | 1KB | 32B | 2KB | 32B | 14.1 | 14.4 | 44.0 | 67.8 | 37.2 | 38.7 | 135.7 | 183.1 |
| LSI CW4011 | 1KB | 64B | 2KB | 64B | 14.8 | 15.4 | 46.4 | 72.4 | 39.6 | 40.7 | 147.3 | 192.5 |
| Average | | | | | *12.8* | *13.4* | *44.4* | *63.3* | *32.7* | *34.0* | *120.7* | *160.7* |

| Processor Core | I-cache | | D-cache | | 10 Tasks - Energy ($J$) | | | | 15 Tasks - Energy ($J$) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Cache | Block | Cache | Block | LB | New | 3VS | FV | LB | New | 3VS | FV |
| Motorola 68000 | 1KB | 128B | 2KB | 128B | 59.9 | 64.1 | 190.7 | 254.8 | 141.7 | 142.0 | 291.8 | 478.5 |
| ARM7 LPower | 1KB | 128B | 2KB | 128B | 64.4 | 64.5 | 201.4 | 256.4 | 139.9 | 140.2 | 283.2 | 472.5 |
| LSI TR4101 | 512B | 64B | 4KB | 32B | 63.7 | 63.9 | 213.5 | 252.5 | 127.3 | 127.9 | 269.0 | 431.0 |
| LSI CW4001 | 512B | 64B | 1KB | 128B | 69.4 | 73.7 | 240.7 | 293.0 | 135.0 | 136.3 | 277.6 | 459.3 |
| PowerPC403 | 1KB | 32B | 2KB | 32B | **53.0** | **53.4** | **175.1** | **212.3** | 109.7 | 110.7 | 224.9 | 373.1 |
| LSI CW4011 | 1KB | 64B | 2KB | 64B | 55.2 | 55.4 | 188.5 | 220.3 | **100.2** | **100.6** | **203.0** | **339.0** |
| Average | | | | | *61.0* | *62.5* | *201.7* | *248.2* | *125.6* | *126.3* | *258.2* | *425.6* |

- LB = 84 J, New = 88 J, 3VS = 121 J, and FV = 208 J for the set of 24 avionics tasks;
- LB = 107J, New = 113 J, 3VS = 172 J, and FV = 283 J for the set of 104 tasks of the elbow manipulator.

The results were obtained respectively for the Yao's preemptive scheduling algorithm with assumed zero preemption cost (column LB), our scheduling heuristics for variable voltage (column New), multiple voltage (three voltage supplies—column 3VS), and fixed voltage systems (column FV).

## VII. DISCUSSION

Variable-voltage cores involve overhead in terms of area, timing, and power due to the variable-voltage supply hardware. According to the results of Suzuki *et al.* [49], this overhead is negligible for a processor core, which our research targets. When this overhead outweighs the gains from power reduction, other approaches such as using multiple voltages may be more effective.

We currently consider only nonpreemptive scheduling policies. Since we target a static scheduling problem, both preemptive and nonpreemptive scheduling policies can be used. There are two advantages of using nonpreemptive scheduling policies over preemptive ones. First, modern processors that use deep pipelining and high clock speed have high preemption overhead. Thus, the gains by using preemptive scheduling policies do not usually justify the high overhead. Second, preemptive scheduling algorithms require higher overhead for run-time scheduling decisions. On the other hand, introduction of preemption to a scheduling problem can greatly simplify the problem. While a scheduling problem can be an optimization problem of polynomial time complexity when preemption is assumed, it might be an NP-complete problem when no preemption is allowed [15]. Considering the high context switching cost for preemption, however, the cost should be taken into account in the preemptive scheduling problem, which might reestablish its computational intractability. We note that this overhead may not be critical for some modern systems. For such systems, it might be better to use preemptive scheduling over nonpreemptive scheduling. However, we should also consider the overhead of changing voltage levels in the preemptive scheduling problem since we may need to change the voltage levels frequently and drastically (e.g., from 3 to 1 V and vice versa). In the nonpreemptive scheduling problem, such overhead can be considered negligible. Yao *et al.* [54] has proposed an optimal algorithm for the preemptive scheduling problem, if we do not consider the voltage change overhead. It would be an interesting research problem to determine a better scheduling policy for the given design scenario with a realistic preemption penalty model. In our design cases, nonpreemptive policies yielded superior results in comparison with preemptive policies when considering the preemption penalty of context switch and voltage changes.

In this research, we assume that the time taken to reach steady state at the new voltage and power consumed by the dc–dc converter is negligible. Namgoong *et al.* [38] reported that the time taken to reach steady state at the new voltage was under 6 ms/V. For some fine-grained applications, this timing overhead may be significant and should be considered.

Suzuki *et al.* [49] reported that the area overhead of the dc–dc converter was under 1% of the chip size. Based on the observation that power consumption is roughly proportional to area, we estimate the power consumed by the converter will be less than 1 mW for the processor. We believe this power overhead is negligible.

Although many real-time application-specific systems can be modeled using our computational, timing, and hardware model, we can generalize our approach to handle more general models. Possible directions are to handle dependency constraints and data-dependent task execution times. We should also verify if our approach can be generalized for other architectures such as superscalar and multiprocessor machines.

## VIII. Conclusion

We developed the design methodology for the low-power core-based real-time SOC-based on dynamically variable-voltage hardware. The key contribution was to develop effective scheduling techniques that treat voltage as a variable to be determined, in addition to the conventional task scheduling and allocation. Our synthesis technique also addressed the selection of the processor core and the determination of the instruction and data cache size and configuration so as to fully exploit dynamically variable voltage hardware, which resulted in significantly lower power consumption for a set of target applications than existing techniques. The highlight of the proposed approach was the nonpreemptive scheduling heuristic, which resulted in solutions very close to optimal ones for many test cases. The effectiveness of the approach was demonstrated on a variety of modern industrial-strength multimedia and communication applications, such as MPEG and JPEG encoders and decoders.

## References


[1] J. A. Bannister and K. S. Trivedi, "Task allocation in fault-tolerant distributed system," *Acta Informatica*, vol. 20, no. 3, pp. 261–281, 1983.

[2] T. D. Burd and R. W. Brodersen, "Processor design for portable systems," *J. VLSI Signal Process.*, vol. 13, nos. 2–3, pp. 203–221, 1996.

[3] R. Camposano and J. Wilberg, "Embedded system design," *Design Automation Embedded Syst.*, vol. 1, nos. 1–2, pp. 5–50, 1996.

[4] A. Chandrakasan, V. Gutnik, and T. Xanthopoulos, "Data driven signal processing: An approach for energy efficient computing," in *Proc. Int. Symp. Low Power Electronics and Design*, pp. 344–352, 1996.

[5] A. P. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R. Brodersen, "Optimizing power using transformations," *IEEE Trans. Computer-Aided Design*, vol. 14, no. 1, pp. 12–31, 1995.

[6] A. P. Chandrakasan, S. Sheng, and R. W. Broderson, "Low-power CMOS digital design," *IEEE J. Solid-State Circuits*, vol. 27, no. 4, pp. 473–484, 1992.

[7] J.-M. Chang and M. Pedram, "Energy minimization using multiple supply voltages," in *Proc. Int. Symp. Low Power Electronics and Design*, 1996, pp. 157–162.

[8] P. Chou and G. Borriello, "Software scheduling in the co-synthesis of reactive real-time systems," in *Proc. Design Automation Conf.*, 1994, pp. 1–4.

[9] B. Cmelik and D. Keppel, "Shade: A fast instruction-set simulator for execution profiling," in *Proc. SIGMETRICS Conf. Measurement and Modeling of Computer Systems*, vol. 22, no. 1, 1994, pp. 128–137.

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

[11] F. Douglis, P. Krishnan, and B. Bershad, "Adaptive disk spin-down policies for mobile computers," in *Proc. USENIX Symp. Mobile and Location-Independent Computing*, 1995, pp. 121–137.

[12] R. J. Evans and P. D. Franzon, "Energy consumption modeling and optimization for SRAM's," *IEEE J. Solid-State Circuits*, vol. 30, no. 5, pp. 571–579, 1995.

[13] M. J. Flynn, *Computer Architecture: Pipelined and Parallel Processor Design*. Boston, MA: Jones and Bartlett, 1996.

[14] R. Fromm, S. Perissakis, N. Cardwell, C. Kozyrakis, B. McGaughy, D. Patterson, T. Anderson, and K. Yelick, "The energy efficiency of IRAM architectures," in *Proc. Int. Symp. Computer Architecture*, 1997, pp. 327–337.

[15] M. R. Garey and D. S. Johnson, *Computer and Intractability: A Guide to the theory of NP-Completeness*. New York: Freeman, 1979.

[16] R. Gonzalez and M. Horowitz, "Energy dissipation in general purpose microprocessors," *IEEE J. Solid-State Circuits*, vol. 31, no. 9, pp. 1277–1284, 1996.

[17] K. Govil, E. Chan, and H. Wasserman, "Comparing algorithms for dynamic speed-setting of a low-power CPU," in *Proc. ACM Int. Conf. Mobile Computing and Networking*, 1995, pp. 13–25.

[18] V. Gutnik and A. Chandrakasan, "An efficient controller for variable supply-voltage low power processing," in *Proc. Symp. VLSI Circuits*, 1996, pp. 158–159.

[19] M. D. Hill and A. J. Smith, "Evaluating associativity in CPU caches," *IEEE Trans. Comput.*, vol. 38, no. 12, pp. 1612–1630, 1989.

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

[21] I. Hong, M. Potkonjak, and R. Karri, "Power optimization using divide-and-conquer techniques for minimization of the number of operations," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1997, pp. 108–113.

[22] C. Hwang and A. C.-H. Wu, "A predictive system shutdown method for energy saving of event-driven computation," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1997, pp. 28–32.

[23] M. C. Johnson and K. Roy, "Datapath scheduling with multiple supply voltages and level converters," *ACM Trans. Design Automation Electron. Syst.*, vol. 2, no. 3, pp. 227–248, 1997.

[24] N. P. Jouppi, "Cache write policies and performance," in *Proc. Int. Symp. Computer Architecture*, 1993, pp. 191–201.

[25] H. Kasahara and S. Narita, "Parallel processing of robot-arm control computation on a multimicroprocessor system," *IEEE J. Robot. Automat.*, vol. RA-1, no. 2, pp. 104–113, 1985.

[26] D. Kirovski, C. Lee, W. Mangione-Smith, and M. Potkonjak, "Application-driven synthesis of core-based systems," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1997, pp. 104–107.

[27] ——, "Synthesis of power efficient systems-on-silicon," in *Proc. Asia and South Pacific Design Automation Conf.*, 1998, pp. 557–562.

[28] U. Ko, P. T. Balsara, and A. K. Nanda, "Energy optimization of multilevel cache architectures for RISC and CISC processors," *IEEE Trans. VLSI Syst.*, vol. 6, no. 2, pp. 299–308, 1998.

[29] M. Krunz and S. K. Tripathi, "On the characterization of VBR MPEG streams," in *Proc. ACM Int. Conf. Measurement and Modeling of Computer Systems (SIGMETRICS 97)*, 1997, pp. 192–202.

[30] C. Kulkarni, F. Catthoor, and H. De Man, "Code transformations for low power caching in embedded multimedia processors," in *Proc. Int. Parallel Processing Symp.*, 1998, pp. 292–297.

[31] P. E. Landman and J. M. Rabaey, "Architectural power analysis: The dual bit type method," *IEEE Trans. VLSI Syst.*, vol. 3, no. 2, pp. 173–187, 1995.

[32] ——, "Activity-sensitive architectural power analysis," *IEEE Trans. Computer-Aided Design*, vol. 15, no. 6, pp. 571–587, 1996.

[33] E. L. Lawler and C. U. Martel, "Scheduling periodically occurring tasks on multiple processors," *Inform. Process. Lett.*, vol. 12, no. 1, pp. 9–12, 1981.

[34] M. T.-C. Lee, V. Tiwari, S. Malik, and M. Fujita, "Power analysis and minimization techniques for embedded DSP software," *IEEE Trans. VLSI Syst.*, vol. 5, no. 1, pp. 123–135, 1997.

[35] Y.-R. Lin, C.-T. Hwang, and A. C.-H. Wu, "Scheduling techniques for variable voltage low power designs," *ACM Trans. Design Automation Electron. Syst.*, vol. 2, no. 2, pp. 81–97, 1997.

[36] P. Macken, M. Degrauwe, M. Van Paemel, and H. Oguey, "A voltage reduction technique for digital systems," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 1990, pp. 238–239.

[37] R. Muntz, J. R. Santos, and S. Berson, "RIO: A real-time multimedia object server," *Performance Eval. Rev.*, vol. 25, no. 2, pp. 29–35, 1997.

[38] W. Namgoong, M. Yu, and T. Meng, "A high-efficiency variable-voltage CMOS dynamic DC-DC switching regulator," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 1997, pp. 380–381.

[39] L. S. Nielsen, C. Niessen, J. Sparso, and K. van Berkel, "Low-power operation using self-timed circuits and adaptive scaling of the supply voltage," *IEEE Trans. VLSI Syst.*, vol. 2, no. 4, pp. 391–397, 1994.

[40] P. Panda and N. Dutt, "Low power mapping of behavioral arrays to multiple memories," in *Proc. IEEE Int. Symp. Low Power Electronics and Design*, 1996, pp. 289–292.
[41] P. Panda, N. Dutt, and A. Nicolau, "Memory data organization for improved cache performance in embedded processor applications," in *Proc. Int. Symp. System-Level Synthesis*, 1996, pp. 90–95.
[42] ——, "Data cache sizing for embedded processor applications," in *Design, Automation Test Eur.*, 1998, pp. 925–926.
[43] M. Pedram, "Power minimization in IC design: Principles and applications," *ACM Trans. Design Automation Electron. Syst.*, vol. 1, no. 1, pp. 3–56, 1996.
[44] S. Raje and M. Sarrafzadeh, "Variable voltage scheduling," in *Proc. Int. Symp. Low Power Design*, 1995, pp. 9–14.
[45] R. S. Ratner, E. B. Shapiro, H. M. Zeidler, S. E. Wahlstrom, C. B. Clark, and J. Goldberg, "Design of a fault tolerant airborne digital computer," in *Computational Requirements and Technology*, vol. 2, SRI Final Rep., NASA Contract NAS1-10920, 1973.
[46] M. Srivastava, A. P. Chandrakasan, and R. W. Brodersen, "Predictive system shutdown and other architectural techniques for energy efficient programmable computation," *IEEE Trans. VLSI Syst.*, vol. 4, no. 1, pp. 42–55, 1996.
[47] A. J. Stratakos, S. R. Sanders, and R. W. Brodersen, "A low-voltage CMOS DC-DC converter for a portable battery-operated system," in *Proc. Power Electronics Specialist Conf.*, 1994, vol. 1, pp. 619–626.
[48] C.-L. Su and A. M. Despain, "Cache design trade-offs for power and performance optimization: A case study," in *Proc. Int. Symp. Low Power Design*, 1995, pp. 63–68.
[49] K. Suzuki, S. Mita, T. Fujita, F. Yamane, F. Sano, A. Chiba, Y. Watanabe, K. Matsuda, T. Maeda, and T. Kuroda, "A 300 MIPS/W RISC core processor with variable supply-voltage scheme in variable threshold-voltage CMOS," in *Proc. IEEE Custom Integrated Circuits Conf.*, 1997, pp. 587–590.
[50] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: A first step toward software power minimization," *IEEE Trans. VLSI Syst.*, vol. 2, no. 4, pp. 437–445, 1994.
[51] V. Tiwari and M. T.-C. Lee, "Power analysis of a 32-bit embedded microcontroller," in *Proc. Asia and South Pacific Design Automation Conf.*, 1995, pp. 141–148.
[52] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," in *Proc. USENIX Symp. Operating Systems Design and Implementation*, 1994, pp. 13–23.
[53] S. J. E. Wilton and N. P. Jouppi, "CACTI: An enhanced cache access and cycle time model," *IEEE J. Solid-State Circuits*, vol. 31, no. 5, pp. 677–688, 1996.
[54] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *IEEE Annu. Foundations Comput. Sci.*, pp. 374–382, 1995.

**Inki Hong** received the M.S. degree in computer science from Stanford University, Stanford, CA, in 1994. He is currently pursuing the Ph.D. degree in the Computer Science Department, University of California, Los Angeles.

He is with Synopsys, Inc., as an R&D Engineer in the Logic Synthesis Department. His research interests include CAD of VLSI circuits, specializing in logic synthesis and high-level and system-level synthesis of VLSI systems. He has published more than 20 refereed papers.

Mr. Hong received the 1996 Chorafas Foundation Award and the 1997 Design Automation Conference Graduate Scholarship Award.

**Darko Kirovski,** for a biography, see p. 1326 of the September 1999 issue of this TRANSACTIONS.

**Gang Qu** received the M.Sc. degree from the Computer Science Department, University of California, Los Angeles, in 1997, where he currently is pursuing the Ph.D. degree.

His research interests include intellectual property protection and core-based system design.

**Miodrag Potkonjak,** for a biography, see p. 1326 of the September 1999 issue of this TRANSACTIONS.

**Mani B. Srivastava** received the M.S. and Ph.D. degrees from the University of California, Berkeley.

He is an Associate Professor of electrical engineering at the University of California, Los Angeles. He worked for several years in Networked Computing Research at Bell Labs. His current research is on wireless multimedia networking, networked embedded systems, reconfigurable systems, low-power systems, and gigabit wireless with funding from NSF, and DARPA's GloMO, SenseIT, and NGI programs. He has received several patents and has published extensively in wireless networking, low-power systems, and system-level tools.

Dr. Srivastava's recent awards include an Okawa Foundation Grant and an NSF CAREER Award.