Designing Low-Power Circuits: Practical Recipes

*by Luca Benini **
Giovanni De Micheli
Enrico Macii

1531- 636X/03/$10.00©2001IEEE

Abstract—*The growing market of mobile, battery-powered electronic systems (e.g., cellular phones, personal digital assistants, etc.) demands the design of microelectronic circuits with low power dissipation. More generally, as density, size, and complexity of the chips continue to increase, the difficulty in providing adequate cooling might either add significant cost or limit the functionality of the computing systems which make use of those integrated circuits.*

In the past ten years, several techniques, methodologies and tools for designing low-power circuits have been presented in the scientific literature. However, only a few of them have found their way in current design flows.

The purpose of this paper is to summarize, mainly by way of examples,what in our experience are the most trustful approaches to low-power design. In other words, our contribution should not be intended as an exhaustive survey of the existing literature on low-power design; rather, we would like to provide insights a designer can rely upon when power consumption is a critical constraint.

We will focus solely on digital circuits, and we will restrict our attention to CMOS devices, this technology being the most widely adopted in current VLSI systems.

Introduction

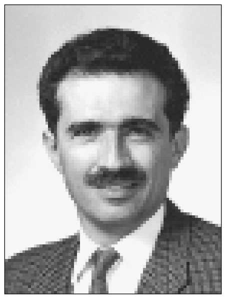
Power dissipation has become a critical design metric for an increasingly large number of VLSI circuits. The exploding market of portable electronic appliances fuels the demand for complex integrated systems that can be powered by lightweight batteries with long times between re-charges (for instance, the plot in Fig. 1 shows the evolution of the world-wide market for mobile phones). Additionally, system cost must be extremely low to achieve high market penetration. Both battery lifetime and system cost are heavily impacted by power dissipation. For these reasons, the last ten years have witnessed a soaring interest in low-power design.

The main purpose of this paper is to provide a few insights into the world of low-power design. We do not intend to review the vast literature on the topic (the interested reader is referred to the many available surveys, e.g., [1–3]). Our objective is to give the readers a few basic concepts to help understanding the "nature of the beast", as well as to provide "silicon-proven recipes" for minimizing power consumption in large-scale digital integrated circuits.

The power consumed by a circuit is defined as $p(t) = i(t)v(t)$, where $i(t)$ is the instantaneous current provided by the power supply, and $v(t)$ is the instantaneous supply voltage. Power minimization targets maximum instantaneous power or average power. The latter impacts battery lifetime and heat

. . . continued on Page 8

Luca Benini


Giovanni De Micheli

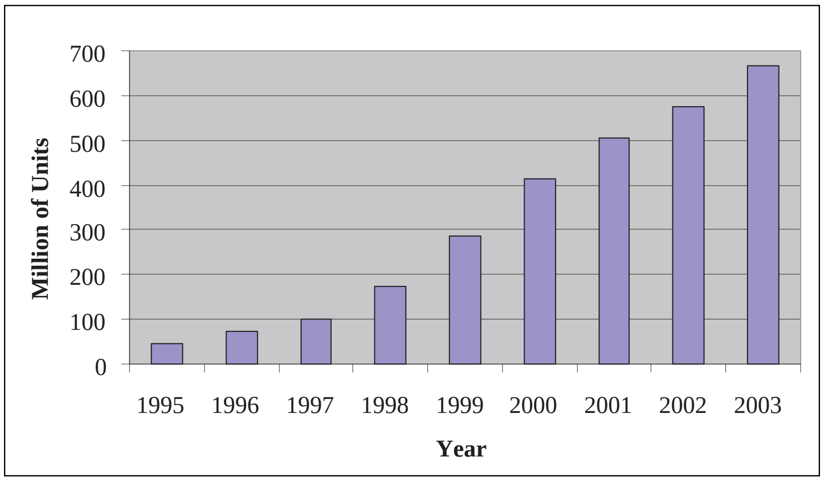
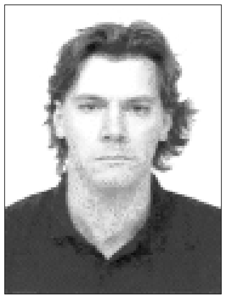


Figure 1. Global market for cellular phones.


Enrico Macii

* Contact Person: Luca Benini, Università di Bologna, DEIS, Viale Risorgimento 2, Bologna, ITALY 40136; Phone: +39-051-209.3782; Fax: +39-051-209.3073; E-mail: lbenini@deis.unibo.it

Low Power Circuits … continued from Page 7

dissipation system cost, the former constrains power grid and power supply circuits design. We will focus on average power in the remainder of the paper, even if maximum power is also a serious concern. A more detailed analysis of the various contributions to overall power consumption in CMOS circuits (the dominant VLSI technology) is provided in the following section.

It is important to stress from the outset that power minimization is never the only objective in real-life designs. Performance is always a critical metric that cannot be neglected. Unfortunately, in most cases, power can be reduced at the price of some performance degradation. For this reason, several metrics for joint power-performance have been proposed in the past.

It is important to stress from the outset that power minimization is never the only objective in real-life designs. Performance is always a critical metric that cannot be neglected. Unfortunately, in most cases, power can be reduced at the price of some performance degradation. For this reason, several metrics for joint power-performance have been proposed in the past. In many designs, the *power-delay product* (i.e., energy) is an acceptable metric. Energy minimization rules out design choices that heavily compromise performance to reduce power consumption. When performance has priority over power consumption, the *energy-delay product* [4] can be adopted to tightly control performance degradation. Alternatively, we can take a *constrained optimization* approach. In this case, performance degradation is acceptable up to a given bound. Thus, power minimization requires optimal exploitation of the slack on performance constraints.

Besides power vs. performance, another key trade-off in VLSI design is *power vs. flexibility*. Several authors have observed that application-specific designs are orders of magnitude more power efficient than general-purpose systems programmed to perform the same computation [1]. On the other hand, flexibility (programmability) is often an indispensable requirement, and designers must strive to achieve maximum power efficiency without compromising flexibility.

The two fundamental trade-offs outlined above motivate our selection of effective low-power design techniques and illustrative cases. We selected successful low power design examples (real-life products) from four classes of circuits, spanning the flexibility vs. power-efficiency trade-off. To maintain uniformity, we chose designs targeting the same end-market, namely multimedia and Internet-enabled portable appliances, where power efficiency is not out-weighted by performance requirements. Our illustrative examples are (in order of decreasing flexibility):

1. Transmeta's Crusoe processor family [5]. This design is representative of the class of general-purpose *x86-compatible* processors for high-end portable appliances, and it features aggressive power optimizations as key market differentiator.

2. Intel's StrongARM family [6]. This design is representative of the class of low-power integrated core-and-peripherals for personal digital assistants and palmtop computers.

3. Texas Instrument TMS320C5x family [7]. This design is a typical very-low power digital signal processor for baseband processing in wireless communication devices (i.e., digital cellular phones).

4. Toshiba's MPEG4 Codec [8]. This is a typical application-specific system-on-chip for multimedia support, targeted for PDAs and digital cameras.

Needless to say, our selection neither implies any endorsement of the commercial products derived from the above mentioned designs, nor implies any form of comparison or benchmarking against competing products.

The remainder of this paper is organized as follows. In *Basic Principles*, we analyze in some detail the sources of power consumption in CMOS technology, and we introduce the basic principles of power optimization. In the section *Technology and Circuit Level Optimizations* we describe a few successful power optimization methods at the technology and circuit level. The section *Logic and Architecture Level Optimizations* covers logic and architecture-level optimizations, while *Software and System Level Optimizations* deals with software and system-level optimizations. Whenever possible, we will describe the optimization techniques with reference to the example designs.

Basic Principles

CMOS is, by far, the most common technology used for manufacturing digital ICs. There are 3 major sources of power dissipation in a CMOS circuit [9]:

$$P = P_{Switching} + P_{Short\text{-}Circuit} + P_{Leakage}$$

$P_{Switching}$, called *switching power*, is due to charging and discharging capacitors driven by the circuit. $P_{Short\text{-}Circuit}$, called *short-circuit power*, is caused by the short circuit currents that arise when pairs of PMOS/NMOS transistors are conducting simultaneously. Finally, $P_{Leakage}$, called *leakage power*, originates from substrate injection and subthreshold effects. For older technologies (0.8 μm and above), $P_{Switching}$ was predominant. For deep-submicron processes, $P_{Leakage}$ becomes more important.

Design for low-power implies the ability to reduce all three components of power consumption in CMOS circuits during the development of a low power electronic product. Optimizations can be achieved by facing the power problem from different perspectives: design and technology. En-

. . . continued on Page 10

Low Power Circuits … continued from Page 9

hanced design capabilities mostly impact switching and short-circuit power; technology improvements, on the other hand, contribute to reductions of all three components.

Switching power for a CMOS gate working in a synchronous environment is modeled by the following equation:

$$P_{Switching} = \frac{1}{2} C_L V_{DD}^2 f_{Clock} E_{SW}$$

where C_L is the output load of the gate, V_{DD} is the supply voltage, f_{Clock} is the clock frequency and E_{SW} is the *switching activity* of the gate, defined as the probability of the gate's output to make a logic transition during one clock cycle. Reductions of $P_{Switching}$ are achievable by combining minimization of the parameters in the equation above.

Historically, supply voltage scaling has been the most adopted approach to power optimization, since it normally yields considerable savings thanks to the quadratic dependence of $P_{Switching}$ on V_{DD}. The major short-coming of this solution, however, is that lowering the supply voltage affects circuit speed. As a consequence, both design and technological solutions must be applied in order to compensate the decrease in circuit performance introduced by reduced voltage. In other words, speed optimization is applied first, followed by supply voltage scaling, which brings the design back to its original timing, but with a lower power requirement.

A similar problem, i.e., performance decrease, is encountered when power optimization is obtained through frequency scaling. Techniques that rely on reductions of the clock frequency to lower power consumption are thus usable under the constraint that some performance slack does exist. Although this may seldom occur for designs considered in their entirety, it happens quite often that some specific units in a larger architecture do not require peak performance for some clock/machine cycles. Selective frequency scaling (as well as voltage scaling) on such units may thus be applied, at no penalty in the overall system speed.

Optimization approaches that have a lower impact on performance, yet allowing significant power savings, are those targeting the minimization of the *switched capacitance* (i.e., the product of the capacitive load with the switching activity). Static solutions (i.e., applicable at design time) handle switched capacitance minimization through area optimization (that corresponds to a decrease in the capacitive load) and switching activity reduction via exploitation of different kinds of signal correlations (temporal, spatial, spatio-temporal). Dynamic techniques, on the other hand, aim at eliminating power wastes that may be originated by the the application of certain system workloads (i.e., the data being processed).

Static and dynamic optimizations can be achieved at different levels of design abstraction. Actually, addressing the power problem from the very

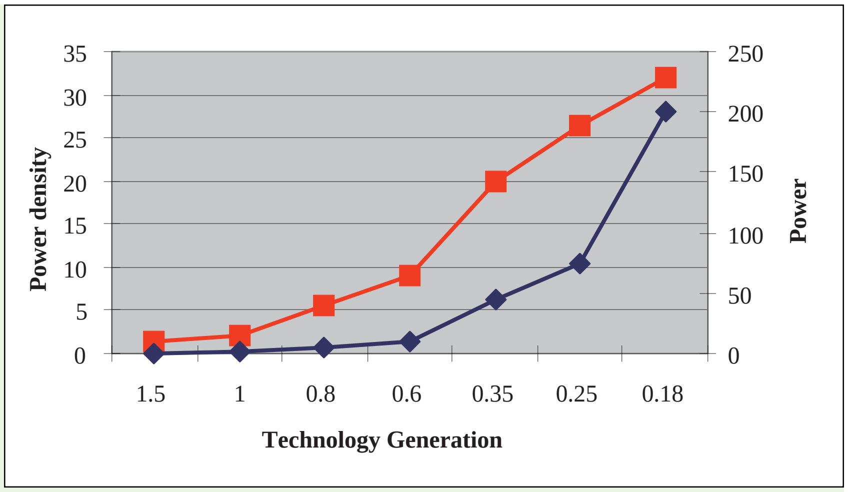


Figure 2. Power density (squares) and power (diamonds) vs. technology generation.

early stages of design development offers enhanced opportunities to obtain significant reductions of the power budget and to avoid costly redesign steps. Power conscious design flows must then be adopted; these require, at each level of the design hierarchy, the exploration of different alternatives, as well as the availability of power estimation tools that could provide accurate feed-back on the quality of each design choice.

In the sequel, we will follow a bottom-up path to present power optimization solutions that have found their way in the development of realistic designs; on the other hand, issues related to power modeling and estimation will not be covered; the interested reader may refer to survey articles available in the literature (see, for example, [10–13]).

Technology and Circuit Level Optimizations

VLSI technology scaling has evolved at an amazingly fast pace for the last thirty years. Minimum device size has kept shrinking by a factor $k = 0.7$ per technology generation. The basic scaling theory, known as *constant field scaling* [14], mandates the synergistic scaling of geometric features and silicon doping levels to maintain constant field across the gate oxide of the MOS transistor. If devices are constant-field scaled, power dissipation scales as k^2 and power density (i.e., power dissipated per unit area) remains constant, while speed increases as k.

In a constant-field scaling regime, silicon technology evolution is probably the most effective way to address power issues in digital VLSI design. Unfortunately, as shown in Fig. 2, power consumption in real-life integrated circuits does not always follow this trend. Figure 2 shows average power consumption and power density trends for Intel Microprocessors, as a function of technology generations [14, 15]. Observe that both average power and power density increase as minimum feature size shrinks.

This phenomenon has two concomitant causes. First, die size has been steadily increasing with technology, causing an increase in total average power. Second, voltage supply has not been reduced according to the directives of constant-field scaling. Historically, supply voltage has scaled much more slowly than device size because: (i) supply voltage levels have been standardized (5 V, then 3.3 V), (ii) faster transistor operation can be obtained by allowing the electric field to raise in the device (i.e., "overdriving" the transistor). This approach is known as *constant voltage scaling*, and it has been adopted in older silicon technologies, up to the 0.8 μm generation.

. . . *continued on Page 12*

Designing Low-Power Circuits: Practical Recipes

Low Power Circuits … continued from Page 11

Probably, the most widely known (and successful) power-reduction technique, known as *power-driven voltage scaling* [3], moves from the observation that constant-voltage scaling is highly inefficient from the power viewpoint, and that we can scale down the voltage supply to trade off performance for power. Depending on the relative weight of performance with respect to power consumption constraints, different voltage levels can be adopted. It is important to observe, however, that transistor speed does not depend on supply voltage V_{DD} alone, but on the *gate overdrive*, namely the difference between voltage supply and device threshold voltage ($V_{DD} - V_T$). Hence, several authors have studied the problem of jointly optimizing V_{DD} and V_T to obtain minimum energy, or minimum energy-delay product [1, 3].

Accurate modeling of MOS transistor currents is paramount for achieving acceptable scaled V_{DD} and V_T values. The simplistic quadratic model of CMOS ON-current $I_{DS} = S' \cdot (V_{GS} - V_T)^2$ leads to overly optimistic switching speed estimates for submicrometric transistors. In short-channel transistors, the *velocity saturation* of the electrons traveling between drain and source dictates a different current equation: $I_{DS} = S' \cdot (V_{GS} - V_T)^m$, with $1 \leq m < 2$ (e.g., $m = 1.3$). Furthermore, another important characteristic of CMOS transistors is sub-threshold conduction. When $V_{GS} < V_T$, the current is not zero, but it follows an exponential law: $I_{DS} = R' \cdot e^{V_T/V_o}$, V_o being the technology-dependent *sub-threshold slope*.

Velocity saturation favors supply down-scaling, because it implies diminishing returns in performance when supply voltage is too high. On the other hand, sub-threshold conduction limits down-scaling, because of increased static current leaking through nominally OFF transistors. Intuitively, optimization of threshold and supply voltage requires balancing ON-current and OFF-current, while at the same time maintaining acceptable performance. Optimizing V_{DD} and V_T for minimum energy delay product leads to surprisingly low values of both. V_{DD} should be only slightly larger than $2V_T$, with V_T a few hundred millivolts. This approach is known as ULP, or *ultra-low-power CMOS* [16].

ULP CMOS is not widely used in practice for two main reasons. First, threshold control is not perfect in real-life technology. Many transistors may have sub-threshold currents that are orders of magnitude larger than expected if their threshold is slightly smaller than nominal (remember that sub-threshold current is exponential in V_{GS}). Second, sub-threshold current is exponentially dependent on temperature, thereby imposing tight thermal control of ULP CMOS, which is not cost-effective. Nevertheless, aggressive voltage scaling is commonplace in low-power VLSI circuits: voltage supply and thresholds are not scaled to their ultimate limit, but they are significantly re-

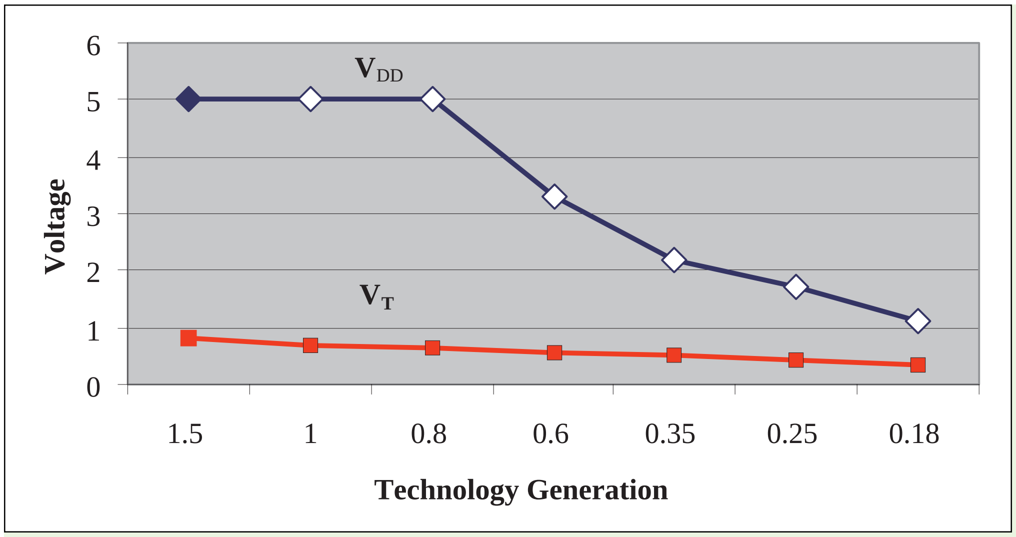


Figure 3. Supply voltage (diamonds) and threshold voltage (squares) vs. technology generation.

duced with respect to constant-voltage scaling. See Example 1.

Power-driven voltage scaling was a “low-hanging fruit” in older technologies. Unfortunately, starting from the 0.8 μm generation, the situation has changed. As shown in Fig. 3, supply voltage has started to decrease with shrinking device size even for high-performance transistors [15]. There are several technological reasons for this trend, which are outside the scope of this paper. From the point of view of power reduction, technology-driven voltage scaling has two important consequences. Aggressively scaled transistors with minimum channel length are becoming increasingly leaky in the OFF state (getting closer to ULP), and there is not much room left for further voltage scaling.

Leakage power is already a major concern in current technologies, because it impacts battery lifetime even if the circuit is completely idle. Quiescent power specifications tend to be very tight. In fact, CMOS technology has traditionally been extremely power-efficient when transistors are not switching, and system designers expect low leakage from CMOS chips. To meet leakage power constraints, *multiple-threshold* and *variable threshold* circuits have been proposed [3]. In multiple-threshold CMOS, the process provides two different thresholds. Low-threshold transistors are fast and leaky, and they are employed on speed-critical sub-circuits. High-threshold transistors are slower but exhibit low sub-threshold leakage, and they are employed in non-critical units/paths of the chip.

Unfortunately, multiple-threshold techniques tend to lose effectiveness as more transistors become timing-criti-

. . . continued on Page 14

Example 1. The StrongARM processor was first designed in a three-metal, 0.35 μm CMOS process, which had been originally developed for high-performance processors (the DEC Alpha family). The first design decision was to reduce supply voltage from 3.45 V to 1.5 V, with threshold voltage $VTN = |V_{TP}| = 0.35$ V, thus obtaining a 5.3x power reduction. The performance loss caused by voltage scaling was acceptable, because StrongARM has much more relaxed performance specifications than Alpha.

As a second example, the research prototype of the TMS320C5x DSP adopted a supply voltage $V_{DD} = 1$ V in a 0.35 μm technology. The aggressive V_{DD} value was chosen by optimizing the energy-delay product.

Example 2. The TMS320C5x DSP prototype adopted a dual-threshold CMOS process, in order to provide acceptable performance at the aggressively down-scaled supply voltage of $V_{DD} = 1$ V. The nominal threshold voltages were 0.4 V and 0.2 V for slow and fast transistors, respectively. Leakage current for the high-V_T transistors is below 1 nA/μm. For the low-V_T transistors, leakage current is below 1 μA/μm (a three-orders-of-magnitude difference!). The drive current of low-V_T transistors is typically twice that of the high-V_T devices.

The MPEG4 Video Codec prototype adopted the variable-threshold voltage scheme to reduce power dissipation. Substrate biasing is exploited to dynamically adjust the threshold: V_T is controlled to 0.2 V in active mode and to 0.55 V when the chip is in stand-by mode.

Low Power Circuits … continued from Page 13

cal. Variable-threshold circuits overcome this shortcoming by dynamically controlling the threshold voltage of transistors through substrate biasing. When a variable-threshold circuit becomes quiescent, the substrate of NMOS transistors is negatively biased, and their threshold increases because of the well known *body-bias effect*. A similar approach can be taken for PMOS transistors (which require positive body bias). Variable-threshold circuits can, in principle, solve the quiescent leakage problem, but they require stand-by control circuits that modulate substrate voltage. Needless to say, accurate and fast body-bias control is quite challenging, and requires carefully designed closed-loop control [3]. See Example 2.

In analogy with threshold voltage, supply voltage can also be controlled to reduce power, albeit in the limited ranges allowed in highly scaled technologies. *Multiple-voltage* and *variable voltage* techniques have been developed to this purpose [3]. In multiple-voltage circuits two or more power supply voltages are distributed on chip. Similarly to the multiple-threshold scheme, timing-critical transistors can be powered at a high voltage, while most transistors are connected to the low voltage supply. Multiple voltages are also frequently used to provide standard voltage levels (e.g., 3.3 V) to input-output circuits, while powering on-chip internal logic at a much lower voltage to save power. The main challenges in the multiple-voltage approach are in the design of multiple power distribution grids and of power-efficient level-shifters to interface low-voltages with high-voltage sections.

Example 3. Supply voltage is closed-loop controlled in both the MPEG4 Codec core and the Crusoe Processor. Supply voltage control in the MPEG4 helps in compensating process fluctuations and operating environment changes (e.g., temperature changes). Variable-voltage operation in Crusoe is more emphasized: it automatically lowers the supply voltage when the processor is under-loaded. From the performance viewpoint, the processor gradually slows down when its full performance is not needed, but its supply voltage and clock speed are rapidly raised when the workload increases.

In both MPEG4-core and Crusoe, frequency and power supply are synergistically adjusted thanks to a clever feed-back control loop: a replica of the critical path is employed to estimate circuit slow-down in response to voltage down-scaling, and clock frequency is reduced using a PLL that locks on the new frequency.

Variable-voltage optimizations allow modulating the power supply dynamically during system operation. In principle, this is a very powerful technique, because it gives the possibility to trade off power for speed at run time, and to finely tune performance and power to non-stationary workloads. In practice, the implementation of this solution requires considerable design ingenuity. First, voltage changes require non-negligible time, because of the large time constants of power supply circuits. Second, the clock speed must be varied consistently with the varying speed of the core logic, when supply voltage is changed. Closed-loop feedback control schemes have been implemented to support variable supply voltage operation. See Example 3.

The techniques based on voltage scaling described above require significant process and system support, which imply additional costs that can be justified only for large-volume applications. Less aggressive circuit-level approaches, that do not require ad-hoc processes can also be successful. Among the proposals available in the literature, library cell design and sizing for low power have gained wide-spread acceptance. From the power viewpoint, probably the most critical cells in a digital library are the sequential primitives, namely, latches and flip-flops. First, they are extremely numerous in today's deeply pipelined circuits; and second, they are connected to the most active network in the chip, i.e., the clock. Clock drivers are almost invariantly the largest contributors to the power budget of a chip, primarily because of the huge capacitive load of the clock distribution network. Flip-flop (and latch) design for low power focuses on minimizing clock load and reducing internal power when the clock signal is toggled. Significant power reductions have been achieved by carefully designing and sizing the flip-flops [1].

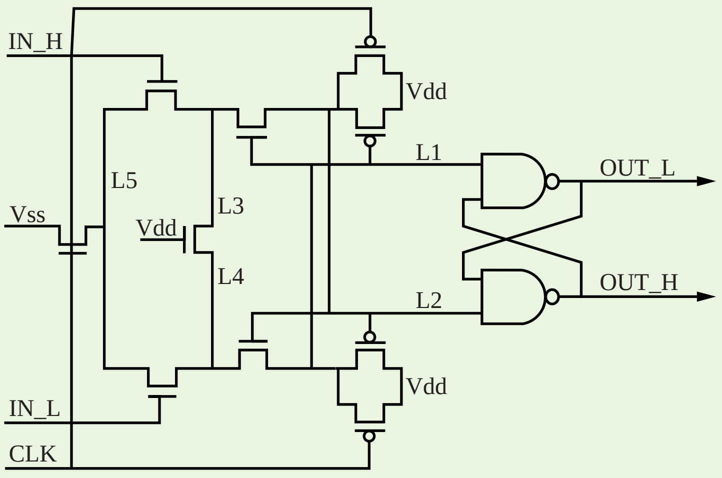


Figure 4. Low-power FF used in the StrongARM design.

Clock power optimization is effectively pursued at the circuit level also by optimizing the power distribution tree, the clock driver circuits and the clock generation circuitry (on-chip oscillators and phase-locked loops) [3]. Higher abstraction-level approaches to clock power reduction are surveyed in the next section. See Example 4.

Transistor sizing is also exploited to minimize power consumption in combinational logic cells. Rich libraries with many available transistor sizes are very useful in low-power design, because they help synthesis tools in achieving optimum sizing for a wide range of gate loads. Power savings can be obtained by adopting non-standard logic implementation styles such as pass-transistor logic, which can reduce the number of transistors (and, consequently the capacitive load), for implementing logic functions which are commonly found in arithmetic units (e.g., exclusive-or, multiplexers).

. . . continued on Page 16

Example 4. The StrongARM processor adopted the edge-triggered flip-flop of Figure 4 to reduce clock load with respect to the flow-through latches used in the Alpha processor. The flip-flop is based on a differential structure, similar to a sense amplifier. Notice that the internal structure of the FF is quite complex in comparison with the simple flow-through latches in Alpha. However,the FF has reduced clock load (only three transistors). This key advantage gave a 1.3 x overall power reduction over the latch-based design.

Example 5. Static power minimization was a serious concern for all our example designs. In the MPEG4 Codec, which adopted a dual supply voltage scheme, significant static power can be dissipated when high-supply gates are driven by a low-supply gate, because the driver cannot completely turn off the pull-up transistors in the fanout gates. For this reason, special level shifter cells were designed that exploited a regenerative cross-coupled structure similar to a sense amplifier to restore full swing in the high-supply section, and minimize static power.

Leakage power was the main concern in the StrongARM design, that could not benefit from multiple threshold or variable threshold control. To reduce leakage power in gates outside the critical path, non-minimum length transistors were employed in low-leakage, low-performance cells.

Low Power Circuits … continued from Page 15

While most traditional power optimization techniques for logic cells focus on minimizing switching power, circuit design for leakage power reduction is also gaining importance [17]. See Example 5.

Logic and Architecture Level Optimizations

Logic-level power optimization has been extensively researched in the last few years [11, 18]. Given the complexity of modern digital devices, hand-crafted logic-level optimization is extremely expensive in terms of design time and effort. Hence, it is cost-effective only for structured logic in large-volume components, like microprocessors (e.g., functional units in the data-path). Fortunately, several optimizations for low power have been automated and are now available in commercial logic synthesis tools [19], enabling logic-level power optimization even for unstructured logic and for low-volume VLSI circuits.

During logic optimization, technology parameters such as supply voltage are fixed, and the degrees of freedom are in selecting the functionality and sizing the gates implementing a given logic specification. As for technology and circuit-level techniques, power is never the only cost metric of interest. In most cases, performance is tightly constrained as well. A common setting is *constrained power optimization*, where a logic network can be transformed to minimize power only if critical path length is not increased. Under this hypothesis, an effective technique is based on *path equalization*.

Path equalization ensures that signal propagation from inputs to outputs of a logic network follows paths of similar length. When paths are equalized, most gates have aligned transitions at their inputs, thereby minimizing spurious switching activity (which is created by misaligned input transitions). This technique is very helpful in arithmetic circuits, such as adders of multipliers. See Example 6.

Glue logic and controllers have much more irregular structure than arithmetic units, and their gate-level implementations are characterized by a wide distribution of path delays. These circuits can be optimized for power by *resizing*. Resizing focuses on fast combinational paths. Gates on fast paths are down-sized, thereby decreasing their input capacitances, while at the same time slowing down signal propagation. By slowing down fast paths, propagation delays are equalized, and power is reduced by joint spurious switching and capacitance reduction. Resizing does not

Example 6. The multiply-accumulate (MAC) unit of the StrongARM processor is based on a Wallace-tree multiplier coupled with a carry-lookahead adder. The Wallace-tree architecture was chosen because it is very fast, but also because it has low dynamic power consumption in the carry-save adder tree. The improvement comes from a sizable reduction in spurious switching, thanks to path delay balancing in the Wallace-tree. A 23% power reduction (as well as a 25% speed-up) is achieved by the Wallace-tree architecture with respect to the array multiplier.

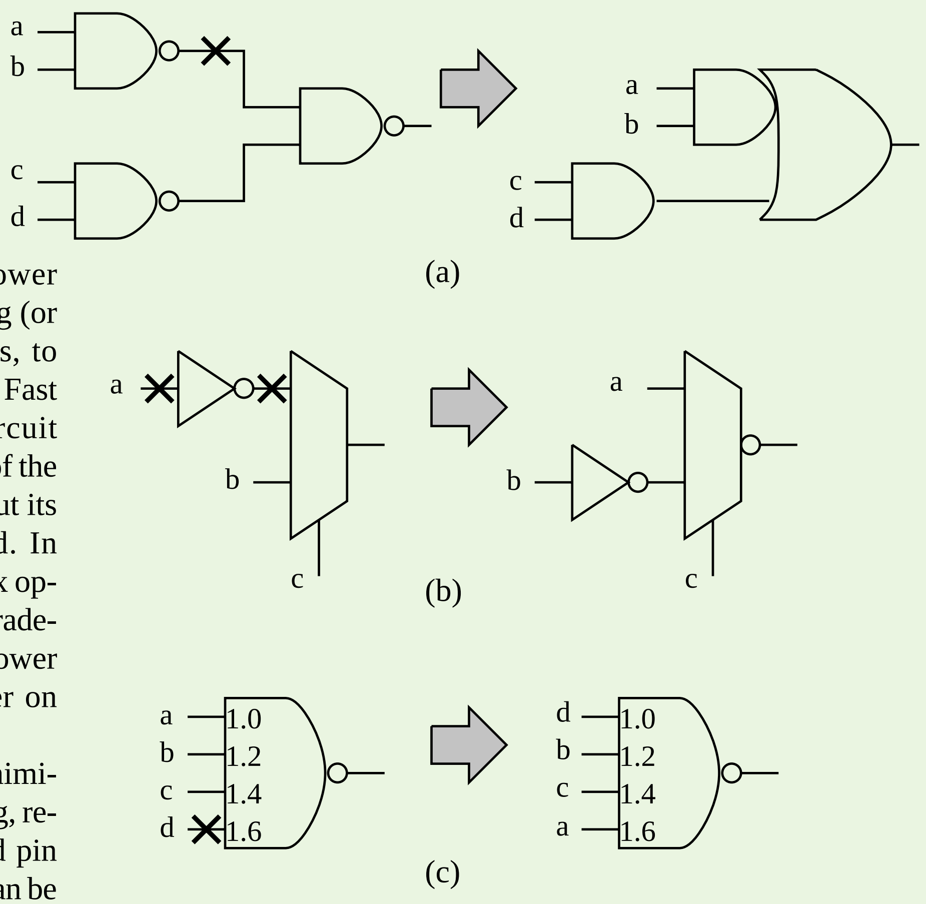


Figure 5. Local transformations: (a) re-mapping, (b) phase assignment, (c) pin swapping.

always imply down-sizing. Power can be reduced also by enlarging (or buffering) heavily loaded gates, to increase their output slew rates. Fast transitions minimize short-circuit power of the gates in the fanout of the gate which has been sized up, but its input capacitance is increased. In most cases, resizing is a complex optimization problem involving a trade-off between output switching power and internal short-circuit power on several gates at the same time.

Other logic-level power minimization techniques are re-factoring, re-mapping, phase assignment and pin swapping. All these techniques can be classified as *local transformations*. They are applied on gate netlists, and focus on nets with large switched capacitance. Most of these techniques replace a gate, or a small group of gates, around the target net, in an effort to reduce capacitance and switching activity. Similarly to resizing, local transformations must carefully balance short circuit and output power consumption. See Example 7.

Logic-level power minimization is relatively well studied and understood. Unfortunately, due to the local nature of most logic-level optimizations, a large number of transformations has to be applied to achieve sizable power savings. This is a time consuming and uncertain process, where uncertainty is caused by the limited accuracy of power estimation. In many cases, the savings produced by a local move are below the “noise floor” of the power estimation engine. As a consequence, logic-level optimization does

. . . continued on Page 18

Example 7. Figure 5 shows three examples of local transformations. In Fig. 5 (a) a re-mapping transformation is shown, where a high-activity node (marked with `x`) is removed thanks to a new mapping onto an `and-or` gate. In Fig. 5 (b), phase assignment is exploited to eliminate one of the two high-activity nets marked with `x`. Finally, pin swapping is applied in Fig. 5 (c) to connect a high-activity net with the input pin of the 4-input `nand` with minimum input capacitance.

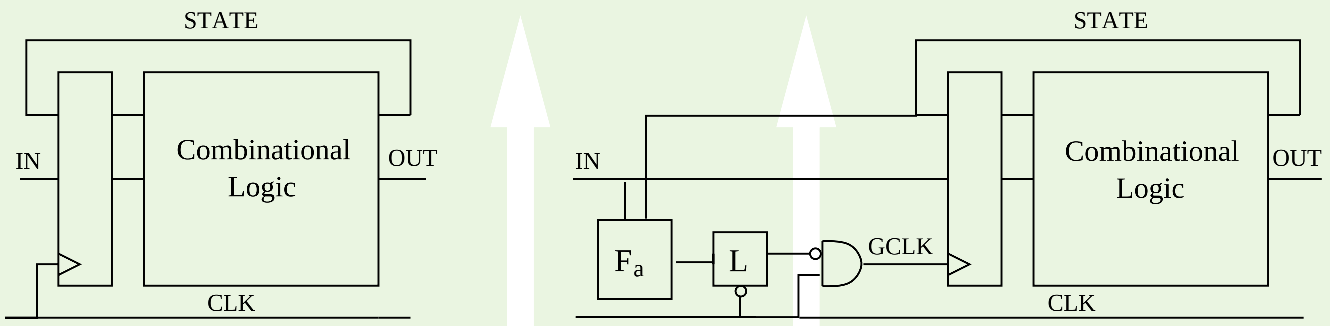


Figure 6. Example of gated clock architecture.

Low Power Circuits … continued from Page 17

not result in massive power reductions. Savings are in the 10 to 20% range, on average. More sizable savings can be obtained by optimizing power at a higher abstraction level, as detailed next.

Complex digital circuits usually contain units (or parts thereof) that are not performing useful computations at every clock cycle. Think, for example, of arithmetic units or register files within a microprocessor or, more simply, to registers of an ordinary sequential circuit. The idea, known for a long time in the community of IC designers, is to disable the logic which is not in use during some particular clock cycles, with the objective of limiting power consumption. In fact, stopping certain units from making useless transitions causes a decrease in the overall switched capacitance of the system, thus reducing the switching component of the power dissipated. Optimization techniques based on the principle above belong to the broad class of *dynamic power management* (DPM) methods. As briefly explained in the section Basic Principles, they achieve power reductions by exploiting specific run-time behaviors of the design to which they are applied.

The natural domain of applicability of DPM is system-level design; therefore, it will be discussed in greater detail in the next section. Nevertheless, this paradigm has also been successfully adopted in the context of architectural optimization.

Clock gating [20] provides a way to selectively stop the clock, and thus force the original circuit to make no transition, whenever the computation to be carried out by a hardware unit at the next clock cycle is useless. In other words, the clock signal is disabled in accordance with the idle conditions of the unit.

As an example of use of the clock-gating strategy, consider the traditional block diagram of a sequential circuit, shown on the left of Fig. 6. It consists of a combinational logic block and an array of state registers which are fed by the next-state logic and which provide some feed-back information to the combinational block itself through the present-state input signals. The corresponding gated-clock architecture is shown on the right of the figure.

The circuit is assumed to have a single clock, and the registers are assumed to be edge-triggered flip-flops. The combinational block F_a is controlled by the primary inputs, the present-state inputs, and the primary outputs of the circuit, and it implements the activation function of the clock gating mechanism. Its purpose is to selectively stop the local clock of the circuit when no state or output transition takes place. The block named L is a latch, transparent when the global clock signal CLK is inactive. Its presence is essential for a correct operation of the system, since it takes care of filtering glitches that may occur at the output of block F_a. It should be noted that the logic for the activation function is on the critical path of the circuit; therefore, timing violations may occur if the synthesis of F_a is not carried out properly.

The clock management logic is synthesized from the Boolean function representing the idle conditions of the circuit. It may well be the case that considering all such conditions results in additional circuitry that is too large and power consuming. It may then be necessary to synthesize a simplified function, which dissipates the minimum possible power, and stops the clock with maximum efficiency. Because of its effectiveness, clock-gating has been applied extensively in real designs, as shown by the examples below, and it has lately found its way in industry-strength CAD tools (e.g., *Power Compiler* by Synopsys). See Examples 8 and 9.

Power savings obtained by gating the clock distribution network of some hardware resources come at the price of a global decrease in performance. In fact, resuming the operation of an inactive resource introduces a latency penalty that negatively impacts system speed. In other words, with clock gating (or with any similar DPM technique), performance and throughput of an architecture are traded for power. See Example 10.

Clock gating is not the only scheme for implementing logic shutdown; solutions ranging from register disabling and usage of dual-state flip-flops to insertion of guarding logic and gate freezing may be equally effective, although no industry-strength assessment of these design options has been done so far.

. . . continued on Page 20

Example 8. The TMS320C5x DSP processor exploits clock gating to save power during active operation. In particular, a two-fold power management scheme, local and global, is implemented. The clock signal feeding the latches placed on the inputs of functional units is enabled only when useful data are available at the units' inputs, and thus meaningful processing can take place. The gating signals are generated automatically by local control logic using information coming from the instruction decoder. Global clock gating is also available in the processor, and is controlled by the user through dedicated power-down instructions, `IDLE1`, `IDLE2`, and `IDLE3`, which target power management of increasing strength and effectiveness. Instruction `IDLE1` only stops the CPU clock, while it leaves peripherals and system clock active. Instruction `IDLE2` also deactivates all the on-chip peripherals. Finally, instruction `IDLE3` powers down the whole processor.

Example 9. The MPEG4 Codec exploits software-controlled clock gating in some of its internal units, including the DMA controller and the RISC core. Clock gating for a unit is enabled by setting the corresponding bit in a "sleep" register of RISC through a dedicated instruction. The clock signal for the chosen hardware resource is OR-ed with the corresponding output of the sleep register to stop clock activity.

Example 10. The latency for the CPU of the TMS320C5x DSP processor to return to active operation from the `IDLE3` mode takes around 50 μsec, due to the need of the on-chip PLL circuit to lock with the external clock generator.

Example 11. Several schemes [21–23] have been proposed to store compressed instructions in main memory and decompress them on-the-fly before execution (or when they are stored in the instruction cache). All these techniques trade off the efficiency of the compression algorithm with the speed and power consumption of the hardware de-compressor. Probably the best known instruction compression approach is the “Thumb” instruction set of the ARM microprocessor family [24]. ARM cores can be programmed using a reduced set of 16-bit instructions (in alternative to standard 32-bit RISC instructions) that reduce the required instruction memory occupation and required bandwidth by a factor of 2.

Low Power Circuits … continued from Page 19

Software and System Level Optimizations

Electronic systems and subsystems consist of hardware platforms with several software layers. Many system features depend on the hardware/software interaction, e.g., programmability and flexibility, performance and energy consumption.

Software does not consume energy *per se*, but it is the execution and storage of software that requires energy consumption by the underlying hardware. Software execution corresponds to performing operations on hardware, as well as accessing and storing data. Thus, software execution involves power dissipation for computation, storage, and communication. Moreover, storage of computer programs in semiconductor memories requires energy (refresh of DRAMs, static power for SRAMs).

The energy budget for storing programs is typically small (with the choice of appropriate components) and predictable at design time. Thus, we will concentrate on energy consumption of software during its execution. Nevertheless, it is important to remember that reducing the size of program, which is a usual objective in compilation, correlates with reducing their energy storage costs. Additional reduction of code size can be achieved by means of compression techniques. See Example 11.

The energy cost of executing a program depends on its machine code and on the hardware architecture parameters. The machine code is derived from the source code from compilation. Typically, the energy cost of the machine code is affected by the back-end of software compilation, that controls the type, number and order of operations, and by the means of storing data, e.g., locality (registers vs. memory arrays), addressing, order. Nevertheless, some architecture-independent optimizations can be useful in general to reduce energy consumption, e.g., selective loop unrolling and software pipelining.

Software instructions can be characterized by the number of cycles needed to execute them and by the energy required per cycle. The energy consumed by an instruction depends weakly on the state of the processor (i.e., by the previously executed instruction). On the other hand, the energy varies significantly when the instruction requires storage in registers or in memory (caches).

The traditional goal of a compiler is to speed up the execution of the generated code, by reducing the code size (which correlates with the latency of execution time) and minimizing *spills* to memory. Interestingly enough, executing machine code of minimum size would consume the minimum energy, if we neglect the interaction with memory and we assume a uniform energy cost of each instruction. Energy-efficient compilation strives at achieving machine code that requires less energy as compared to a performance-driven traditional compiler, by leveraging the disuniformity in instruction energy cost, and the different energy costs for storage in registers and in main memory due to addressing and address decoding. Nevertheless, results are sometimes contradictory. Whereas for some architectures energy-efficient compilation gives a competitive advantage as compared to traditional compilation, for some others the most compact code is also the most economical in terms of energy, thus obviating the need of specific low-power compilers.

It is interesting to note that the style of the software source program (for any given function) affects the energy cost. Energy-efficient software can be achieved by enforcing specific writing styles, or by allowing source-code transformation to reduce the power consumption of a program. Such transformation can be automated, and can be seen as the front-end of compilation. See Example 12.

Power-aware operating systems (OSs) trade generality for energy efficiency. In the case of embedded electronic systems, OSs are streamlined to support just the required applications. On the other hand, such an approach may not be applicable to OSs for personal computers, where the user wants to retain the ability of executing a wide variety of applications.

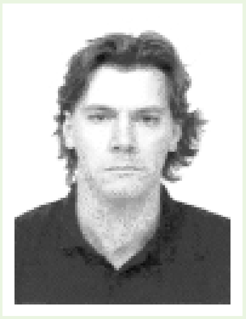
Energy efficiency in an operating system can be achieved by designing an energy aware *task scheduler*. Usually, a scheduler determines the set of start times for each task, with the goal of optimizing a cost function related to the completion time of all tasks, and to satisfy real time constraints, if applicable. Since tasks are associated with resources having specific energy models, the scheduler can exploit this information to reduce run-time power consumption. See Example 13.

Operating systems achieve major energy savings by implementing *dynamic power management* (DPM) of the system resources. DPM dynamically reconfigures an electronic system to provide the requested services and performance levels with a minimum number of active components or a

. . . *continued on Page 22*

Example 12. The IBM XL compilers transform source code (from different languages) into an internal form called W-code. The Toronto Portable Optimizer (TPO) [25] performs W-code to W-code transformations. Some transformations are directed to reducing power consumption. To this goal, switching activity is minimized by exploiting relations among variables that take similar values. Power-aware instruction scheduling can increase the opportunity of clock gating, by grouping instructions with similar resource usage.

Example 13. Simultaneous multi-threading (SMT) [25] has been proposed as an approach for the exploitation of instruction-level parallelism. In SMT, the processor core shares its execution-time resources among several simultaneously executing threads (programs). Thus, at each cycle, instructions can be fetched from one or more independent threads, and passed to the issue and execution boxes. Since the resources can be filled by instructions from parallel threads, their usage increases and energy waste decreases. Compaq has announced that its future 21x64 designs will embrace the SMT paradigm.



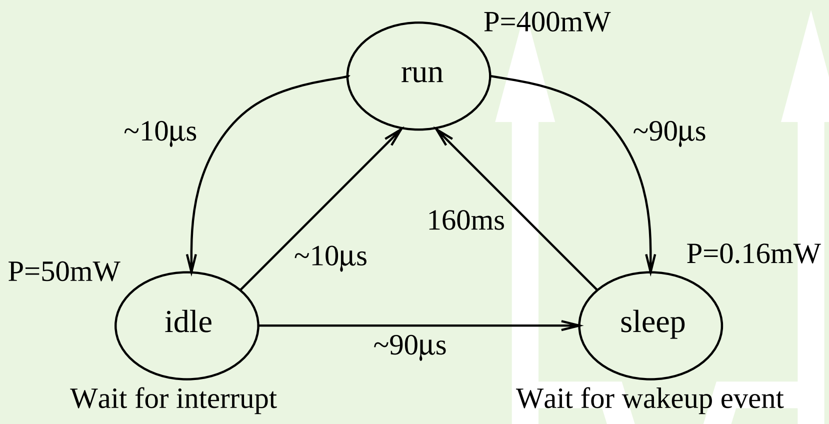


Figure 7. Power state machine for the StrongARM SA-1100 processor.

Low Power Circuits … continued from Page 21

minimum load on such components [26]. Dynamic power management encompasses a set of techniques that achieve energy-efficient computation by selectively shutting down or slowing down system components when they are *idle* (or partially unexploited). DPM can be implemented in different forms including, but not limited to, clock gating, clock throttling, supply-voltage shut-down, and dynamically-varying power supplies, as described earlier in this paper.

We demonstrate here power management using two examples from existing successful processors. The former example describes the shut down modes of the StrongARM processor, while the latter describes the slow down operation within the Transmeta Crusoe chip. See Example 14.

An operating system that controls a power manageable component, such as the SA-1100, needs to issue commands to force the power state transitions, in response to the varying workload. The control algorithm that issues the commands is termed *policy*. An important problem is the computation of the policy for a given workload statistics and component parameter. Several avenues to policy computation are described in another tutorial paper [28]. We consider next another example, where power management is achieved by slowing down a component. See Example 15.

We mentioned before the important problem of computing policies, with guaranteed optimality properties, to control one or more components with possibly different management schemes. A related interesting problem is the design of components that can be effectively power managed. See Example 16.

Several system-level design trade-offs can be explored to reduce energy consumption. Some of these design choices belong to the domain of *hardware/software co-design*, and leverage the migration of hardware functions to software or vice versa. For example, the Advanced Configuration and Power Interface (ACPI) standard, initiated by Intel, Microsoft and Toshiba, provides a portable hw/sw interface that makes it easy to implement DPM policies for personal computers in software. As another example, the decision of implementing specific functions (like MPEG decoding) on specifically-dedicated hardware, rather than on a programmable processor, can significantly affect energy consumption. As a final interesting example, we would like to point out that *code morphing* can be a very powerful tool in reducing energy dissipation. See Example 17.

Conclusions

Electronic design aims at striking a balance between performance and power efficiency. Designing low-power applications is a multi-faceted problem, because of the plurality of embodiments that a system specification may have and the variety of

. . . continued on Page 24

Example 14. The StrongARM SA-1100 processor [27] is an example of a power manageable component. It has three modes of operation: `run`, `idle`, and `sleep`. The `run` mode is the normal operating mode of the SA-1100: every on-chip resource is functional. The chip enters the `run` mode after successful power-up and reset. The `idle` mode allows a software application to stop the CPU when not in use, while continuing to monitor on-chip or off-chip interrupt requests. In the `idle` mode, the CPU can be brought back to the `run` mode quickly when an interrupt occurs. The `sleep` mode offers the greatest power savings and consequently the lowest level of available functionality. In the transition from `run` or `idle`, the SA-1100 performs an orderly shutdown of on-chip activity. In a transition from `sleep` to any other state, the chip steps through a rather complex wake-up sequence before it can resume normal activity.

The operation of the StrongARM SA-1100 can be captured by a power state machine model, as shown in Fig. 7. States are marked with power dissipation and performance values, edges are marked with transition times. The power consumed during transitions is approximately equal to that in the `run` mode. Notice that both `idle` and `sleep` have null performance, but the time for exiting `sleep` is much longer than that for exiting `idle` (10 μs versus 160 ms). On the other hand, the power consumed by the chip in `sleep` mode (0.16 mW) is much smaller than that in `idle` (50 mW).

Example 15. The Transmeta Crusoe TM5400 chip uses the Long Run^R power management scheme, that allows the OS to slow down the processor when the workload can be serviced at a slower pace. Since a slow down in clock frequency permits a reduction in supply voltage, then the energy to compute a given task decreases quadratically. From a practical standpoint, the OS observes the task queue and determines the appropriate frequency and voltage levels, which are transmitted to the phase-locked loop (PLL) and DC-DC converter. As a result, Transmeta claims that the Crusoe chip can play a soft DVD at the same power level used by a conventional x86 architecture in sleep state.

Example 16. The 160-Mhz implementation of the StrongARM described in [6] supports 16 possible operating frequencies generated by a PLL. Moreover, the PLL was designed within a power budget of 2 mW, so that it could be kept running in the `idle` state. Thus, the need for a low-power PLL is dictated by the power budget in the `idle` state. Note that the ability of keeping the PLL running in the `idle` state is key to achieving a fast transition to the `run` state.

Example 17. The Transmeta Crusoe chip executes x86-compatible binaries on a proprietary architecture, which is designed for low-power operation. The operating system performs run-time binary translation to this effect. The code morphing is key in reducing the energy cost associated with any given program.

Designing Low-Power Circuits: Practical Recipes

*by Luca Benini **
Giovanni De Micheli
Enrico Macii

Low Power Circuits … continued from Page 22

degrees of freedom that designers have to cope with power reduction.

In this brief tutorial, we showed different design options and the corresponding advantages and disadvantages. We tried to relate general-purpose low-power design solutions to a few successful chips that use them to various extents. Even though we described only a few samples of design techniques and implementations, we think that our samples are representative of the state of the art of current technologies and can suggest future developments and improvements.

References

[1] J. Rabaey and M. Pedram, *Low Power Design Methodologies*. Kluwer, 1996.

[2] J. Mermet and W. Nebel, *Low Power Design in Deep Submicron Electronics*. Kluwer, 1997.

[3] A. Chandrakasan and R. Brodersen, *Low-Power CMOS Design*. IEEE Press, 1998.

[4] T. Burd and R. Brodersen, “Processor Design for Portable Systems”, *Journal of VLSI Signal Processing Systems*, vol. 13, no. 2–3, pp. 203–221, August 1996.

[5] D. Ditzel, “Transmeta’s Crusoe: Cool Chips for Mobile Computing”, *Hot Chips Symposium*, August 2000.

[6] J. Montanaro, *et al.*, “A 160-MHz, 32-b, 0.5-W CMOS RISC Microprocessor”, *IEEE Journal of Solid-State Circuits*, vol. 31, no. 11, pp. 1703–1714, November 1996.

[7] V. Lee, *et al.*, “A 1-V Programmable DSP for Wireless Communications”, *IEEE Journal of Solid-State Circuits*, vol. 32, no. 11, pp. 1766–1776, November 1997.

[8] M. Takahashi, *et al.*, “A 60-mW MPEG4 Video Coded Using Clustered Voltage Scaling with Variable Supply-Voltage Scheme”, *IEEE Journal of Solid-State Circuits*, vol. 33, no. 11, pp. 1772–1780, November 1998.

[9] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, “Low-Power CMOS Digital Design”, *IEEE Journal of Solid-State Circuits*, vol. 27, no. 4, pp. 473–484, April 1992.

[10] F. Najm, “A Survey of Power Estimation Techniques in VLSI Circuits”, *IEEE Transactions on VLSI Systems*, vol. 2, no. 4, pp. 446–455, December 1994.

[11] M. Pedram, “Power Estimation and Optimization at the Logic Level”, *International Journal of High-Speed Electronics and Systems*, vol. 5, no. 2, pp. 179–202, 1994.

[12] P. Landman, “High-Level Power Estimation”, *ISLPED-96: ACM/IEEE International Symposium on Low Power Electronics and Design*, pp. 29–35, Monterey, California, August 1996.

[13] E. Macii, M. Pedram, F. Somenzi, “High-Level Power Modeling, Estimation, and Optimization”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 11, pp. 1061–1079, November 1998.

[14] S. Borkar, “Design Challenges of Technology Scaling”, *IEEE Micro*, vol. 19, no. 4, pp. 23–29, July-August 1999.

[15] S. Thompson, P. Packan, and M. Bohr, “MOS Scaling: Transistor Challenges for the 21st Century”, *Intel Technology Journal*, Q3, 1998.

[16] Z. Chen, J. Shott, and J. Plummer, “CMOS Technology Scaling for Low Voltage Low Power Applications”,

ISLPE-98: IEEE International Symposium on Low Power Electronics, pp. 56–57, San Diego, CA, October 1994.

[17] Y. Ye, S. Borkar, and V. De, "A New Technique for Standby Leakage Reduction in High-Performance Circuits", *1998 Symposium on VLSI Circuits*, pp. 40–41, Honolulu, Hawaii, June 1998.

[18] M. Pedram, "Power Minimization in IC Design: Principles and Applications", *ACM Transactions on Design Automation of Electronic Systems*, vol. 1, no. 1, pp. 3–56, January 1996.

[19] B. Chen and I. Nedelchev, "Power Compiler: A Gate Level Power Optimization and Synthesis System", *ICCD'97: IEEE International Conference on Computer Design*, pp. 74–79, Austin, Texas, October 1997.

[20] L. Benini, P. Siegel, and G. De Micheli, "Automatic Synthesis of Gated Clocks for Power Reduction in Sequential Circuits", *IEEE Design and Test of Computers*, vol. 11, no. 4, pp. 32–40, December 1994.

[21] Y. Yoshida, B.-Y. Song, H. Okuhata, T. Onoye, and I. Shirakawa, "An Object Code Compression Approach to Embedded Processors", *ISLPED-98: ACM/IEEE International Symposium on Low Power Electronics and Design*, pp. 265–268, Monterey, California, August 1997.

[22] L. Benini, A. Macii, E. Macii, and M. Poncino, "Selective Instruction Compression for Memory Energy Reduction in Embedded Systems", *ISLPED-99: ACM/IEEE 1999 International Symposium on Low Power Electronics and Design*, pp. 206–211, San Diego, California, August 1999.

[23] H. Lekatsas and W. Wolf, "Code Compression for Low Power Embedded Systems", *DAC-37: ACM/IEEE Design Automation Conference*, pp. 294–299, Los Angeles, California, June 2000.

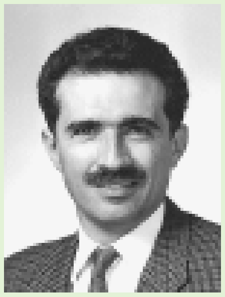
[24] S. Segars, K. Clarke, and L. Goudge, "Embedded Control Problems, Thumb and the ARM7TDMI", *IEEE Micro*, vol. 15, no. 5, pp. 22–30, October 1995.

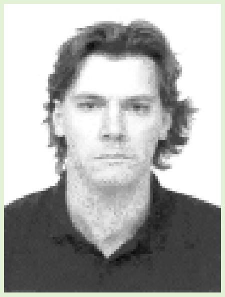
[25] D. Brooks, *et al.*, "Power-Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors", *IEEE Micro*, vol. 20, No. 6, pp. 26–44, November 2000.

[26] L. Benini and G. De Micheli, *Dynamic Power Management: Design Techniques and CAD Tools*. Kluwer, 1997.

[27] Intel, *SA-1100 Microprocessor Technical Reference Manual*. 1998.

[28] L. Benini, A. Bogliolo, and G. De Micheli, "A Survey of Design Techniques for System-Level Dynamic Power Management", *IEEE Transactions on VLSI Systems*, vol. 8, no. 3, pp. 299–316, June 2000.





Luca Benini received the B.S. degree *summa cum laude* in electrical engineering from the University of Bologna, Italy, in 1991, and the M.S. and Ph.D. degrees in electrical engineering from Stanford University in 1994 and 1997, respectively. Since 1998, he has been assistant professor in the Department of Electronics and Computer Science in the University of Bologna, Italy. He also holds visiting professor positions at Stanford University and the Hewlett-Packard Laboratories, Palo Alto, California.

Dr. Benini's research interests are in all aspects of computer-aided design of digital circuits, with special emphasis on low-power applications, and in the design of portable systems. On these topics, he has authored more than 100 papers in international journals and conferences, a book and several book chapters. He is a member of the technical program committee in several international conferences, including the Design and Test in Europe Conference and International Symposium on Low Power Design.

Giovanni De Micheli is professor of electrical engineering, and by courtesy, of computer science at Stanford University. His research interests include several aspects of design technologies for integrated circuits and systems, with particular emphasis on synthesis, system-level design, hardware/software co-design and low-power design. He is author of *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, 1994, co-author and/or co-editor of four other books and of over 200 technical articles.

Dr. De Micheli is a fellow of ACM and IEEE. He received the Golden Jubilee Medal for outstanding contributions to the IEEE CAS Society in 2000. He received the 1987 *IEEE Transactions on CAD/ICAS* Best Paper Award and two best paper awards at the Design Automation Conference, in 1983 and in 1993.

He is editor-in-chief of the *IEEE Transactions on CAD/ICAS*. He was vice president (for publications) of the IEEE CAS Society in 1999–2000. Dr. De Micheli was program chair and general chair of the Design Automation Conference (DAC) in 1996–1997 and 2000 respectively. He was program and general chair of the International Conference on Computer Design (ICCD) in 1988 and 1989 respectively.

Enrico Macii holds the Dr. Eng. degree in electrical engineering from Politecnico di Torino, Italy, the Dr. Sc. degree in computer science from Università di Torino, and the Ph. D. degree in computer engineering from Politecnico di Torino. From 1991 to 1994 he was adjunct faculty at the University of Colorado at Boulder. Currently, he is associate professor of computer engineering at Politecnico di Torino.

His research interests include several aspects of the computer-aided design of integrated circuits and systems, with particular emphasis on methods and tools for low-power digital design. He has authored over 200 journal and conference articles in the areas above, including a paper that received the best paper award at the 1996 IEEE EuroDAC conference.

Enrico Macii is associate editor of the *IEEE Transactions on CAD/ICAS* (since 1997) and associate editor of the *ACM Transactions on Design Automation of Electronic Systems* (since 2000). He was technical program co-chair of the 1999 IEEE Alessandro Volta Memorial Workshop on Low Power Design, and technical program co-chair of the 2000 ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED). Currently, he is general chair of ISLPED'01.