Low-Power Design for Embedded Processors

BILL MOYER, MEMBER, IEEE

Invited Paper

Minimization of power consumption in portable and battery-powered embedded systems has become an important aspect of processor and system design. Opportunities for power optimization and tradeoffs emphasizing low power are available across the entire design hierarchy. A review of low-power techniques applied at many levels of the design hierarchy is presented, and an example of low-power processor architecture is described along with some of the design decisions made in implementation of the architecture.

Keywords**—***Circuit design, clock distribution, clock gating, CMOS circuits, CPU microarchitecture, instruction set design, low-power architecture, low-power design, low-power synthesis, low-power systems, power dissipation, power minimization, power optimization, RISC, state assignment, system design.*

I. INTRODUCTION

The increasing prominence of portable electronics and consumer-oriented devices has become a fundamental driving factor in the design of new computational elements in CMOS very large-scale integration (VLSI) systems on a chip. As the focus shifts away from tethered desktop computing to the mobile appliance, a rethinking of design optimizations traditionally targeting ever-increasing performance goals and high clock rates at almost any cost are required in order to optimize battery life and extend the utility of these devices. The trend in the desktop world of continuous growth in complexity and size of the underlying CPU in terms of instruction issue strategies and the supporting microarchitecture needs to be re-examined for these devices, as the tradeoffs in energy consumption versus the improved performance obtained may dictate a different set of design choices. Power consumption arises as a third axis in the optimization space in addition to the traditional speed (performance) and area (cost) dimensions.

Improvements in circuit density and the corresponding increase in heat generation must be addressed even for high-end desktop systems. Current trends in technology scaling of CMOS circuits cannot be reliably sustained without addressing power consumption issues. Environmental concerns relating to energy consumption by computers and other electrical equipment are another reason for interest in low-power designs and design techniques.

Low-power design can be an important element in lowering system cost as well. Smaller packages, batteries, and reduced thermal management overhead result in less costly products, with higher reliability as an added benefit. Size, available power budget, and weight of a device are important metrics, and to a large extent, the power source is the primary determinant of these metrics. Energy efficient designs maximize the useful lifetime of this source, while attempting to meet throughput and peak performance requirements of the overall application. Power efficient design implies that the system minimizes the peak demands on this source, thus improving its operating efficiency. The rate of energy use can have a dramatic effect on the amount of energy available from a battery source as well as its cost [1], [2], thus, there is value in not only minimizing average power consumption, but also peak power consumption as well. Portable product utility is constrained by the physical size and weight of the power source. Current battery technologies, such as Nickel–Metal Hydride systems, are available in "AA" sizes with a capacity of 1600 mAh at a nominal voltage of 1.2 V. For a portable device containing a pair of these cells, run-time between charges of approximately 4 h is possible when the system is dissipating 1 W of average power. For a device to remain usable for a month between charges, the average power dissipation must drop below 5 mW. For systems with an active duty cycle of 10%, the power consumed by the entire system when active must be less than 50 mW, several orders of magnitude below today's notebook computing devices.

Opportunities for design tradeoffs emphasizing low power are available across the entire spectrum of the overall design process for a portable system, and are effectively applied at many levels of the design hierarchy. From algorithm selection to silicon process technology details, opportunities abound. Generally speaking, the higher the level of abstraction, the greater the opportunity for power savings. Much research as well as practical development has occurred in the

Manuscript received December 29, 2000; revised June 10, 2001.
The author is with Motorola Inc., Austin, TX 78729 USA.
Publisher Item Identifier S 0018-9219(01)09683-9.

past 30 or so years regarding low-power design. In the last decade, popularity of the subject has produced a wealth of technical information [3]–[7], as well as annual international symposia and workshops dedicated to latest research and developments [8]–[10].

While the bulk of commercial activity addressing low-power processor systems has focused on well-known clocked CMOS design styles, important research and commercial work in the area of asynchronous logic design techniques continues as an alternative approach to lowering power dissipation in systems. These techniques may also provide a solution to the increasing problem of clock management and distribution as device frequencies approach and even exceed 1 GHz. While not the focus of this paper, the interested reader is referred to the overview presented by Hauck [11] as a starting point for asynchronous design styles.

II. Power Dissipation in CMOS Circuits

Power dissipated in CMOS circuits consists of several components as indicated in (1)

$$P_{\text{total}} = P_{\text{switching}} + P_{\text{shortcircuit}} + P_{\text{static}} + P_{\text{leakage}}. \tag{1}$$

The individual components represent the power required to charge or switch a capacitive load ($P_{\text{switching}}$), short circuit power consumed during output transitions of a CMOS gate as the input switches ($P_{\text{shortcircuit}}$), static power consumed by the device (P_{static}), and leakage power consumed by the device (P_{leakage}). Components $P_{\text{switching}}$ and $P_{\text{shortcircuit}}$ are present when a device is actively changing state, while the components P_{static} and P_{leakage} are present regardless of state changes. The largest active component, $P_{\text{switching}}$, is defined as

$$P_{\text{switching}} = C * V_{dd} * V_{\text{swing}} * \alpha * f \tag{2}$$

where C represents the capacitance being switched, V_{dd} is the supply voltage, V_{swing} corresponds to the change in voltage level of the switched capacitance, α represents a switching activity factor based on the probability of an output transition, and f represents the frequency of operation. The product $\alpha * C$ is also referred to as the effective switched capacitance, or C_{eff}. In most circuits, V_{swing} is equal to V_{dd}, so (2) is commonly written as

$$P_{\text{switching}} = C_{\text{eff}} * V_{dd}^2 * f. \tag{3}$$

The term $P_{\text{shortcircuit}}$ occurs due to the overlapped conductance of both the PMOS and NMOS transistors forming a CMOS logic gate as the input signal transitions. This term has a complicated derivation, but in simplified form can be written as [12]

$$P_{\text{shortcircuit}} = I_{\text{mean}} * V_{dd} \tag{4}$$

where I_{mean} represents the average current drawn during the input transition. I_{mean} is minimized for a single gate with short input rise and fall times, and with long output transition times, thus presenting a tradeoff in device sizing. When a set of gates is considered, it is generally optimal to target equal input and output transition times. For large devices such as input–output (I/O) buffers or clock drivers, special design considerations are often used to minimize the overlap current [13]. For properly sized and ratioed gates, the contribution to overall dynamic power due to $P_{\text{shortcircuit}}$ is on the order of 10%–20%, although this factor may increase with increased device scaling [14].

P_{static} is not usually a factor in pure CMOS designs, since static current is not drawn by a CMOS gate, but certain circuit structures such as sense amplifiers, voltage references, and constant current sources do exist in CMOS systems and contribute to overall power.

P_{leakage} is due to leakage currents from reversed biased PN junctions associated with the source and drain of MOS transistors, as well as subthreshold conduction currents. The leakage component is proportional to device area and temperature. The subthreshold leakage component is strongly dependent on device threshold voltages, and becomes an important factor as power supply voltage scaling is used to lower power. For systems with a high ratio of standby operation to active operation, P_{leakage} may be the dominant factor in determining overall battery life.

Minimization of these components of power dissipation is important in designing low-power systems, and there are complex interactions that require tradeoffs to be made involving each. Active power minimization involves reducing the magnitude of each of the components in (3). With its quadratic contribution in the power equation, reduction of supply voltage is an obvious candidate technique for power reduction, and can be applied to an entire design. Reducing supply voltage by a factor of two ideally results in a factor of four reduction in $P_{\text{switching}}$. There are limitations to simple supply voltage scaling, however, since the performance of a gate is reduced as V_{dd} is lowered, due to the reduced saturation current available to charge and discharge load capacitance. Gate delay dependence on V_{dd} is approximated [15] by

$$T_{\text{delay}} \propto \frac{C_{\text{load}} * V_{dd}}{(V_{dd} - V_{\text{threshold}})^{1.5}}. \tag{5}$$

The energy-delay product is minimized when V_{dd} is equal to $2 * V_{\text{threshold}}$. Reducing V_{dd} from $3 * V_{\text{threshold}}$ (a typical value for 0.18μm technology) to $2 * V_{\text{threshold}}$ results in an approximate 50% decrease in performance while using only 44% of the power. This is a useful point of leverage if performance goals can still be met. It would seem that reducing threshold voltage of the devices and, thus, a corresponding reduction in V_{dd} offers a path to arbitrarily low-power consumption. Unfortunately, there are practical limits to the degree that $V_{\text{threshold}}$ can be lowered, due to reduced noise margins and since exponentially increased leakage current becomes a limiting factor in contribution to P_{leakage} [16]. Controllability of variations in $V_{\text{threshold}}$ is also an issue in manufacturing, and provides a lower bound on supply voltage scaling [17]. A methodology for selecting supply and threshold voltage targets is further described in [18].

III. Design Techniques for Power Reduction

Power reduction techniques may be applied at all levels of the system design hierarchy. As noted in [19], these levels include Algorithmic, Architectural, Logic and Circuit, and Device technology. A brief description of each is given followed by some specific examples. This section is not intended to be exhaustive.

A. Algorithmic

Algorithmic-level power reduction techniques focus on minimizing the number of operations weighted by the cost of those operations. Selection of an algorithm is generally based on details of an underlying implementation such as the energy cost of an addition versus a logical operation, the cost of a memory access, and whether locality of reference, both spatially and temporally can be maximized. The presence and structure of cache memory, for example, may cause a different set of operations to be selected, since the cost of a memory access relative to an arithmetic operation changes. In general, reducing the number of operations to be performed is a first-order goal, although in some situations, recomputation of an intermediate result may be cheaper than spilling to and reloading from memory. Techniques used by optimizing compilers, such as strength reduction, common subexpression elimination, and optimizations to minimize memory traffic are also useful in most circumstances in reducing power. Loop unrolling may also be of benefit, as it results in minimized loop overhead as well as the potential for intermediate result reuse.

Number representations offer another area for algorithmic power tradeoffs. For example, the choice of using a fixed point or a floating-point representation for data types can have a significant difference in power consumption during arithmetic operations. Selection of sign-magnitude versus two's complement representation for certain signal processing applications can result in significant power reduction if the input samples are uncorrelated and dynamic range is minimized [20]. Operator precision, or bit length, is another tradeoff that can be selected to minimize power at the expense of accuracy. For some floating point algorithms, full precision can be avoided, and mantissa and exponent width reduced below the standard 23 and 8 bits, respectively, for single precision IEEE floating point. In [21], the authors show that for an interesting set of applications involving speech recognition, pattern classification, and image processing, mantissa bit width may be reduced by more than 50% to 11 bits with no corresponding loss of accuracy. In addition to improved circuit delays, energy consumption of the floating point multiplier was reduced 20%–70% for mantissa reductions to 16 and 8 bits, respectively. Truncation of low-order bits of partial sum terms when performing a 16-bit fixed-point multiplication has been shown to result in power savings of 30% due mainly to reduction in area [22]. Adaptive bit truncation techniques for performing motion estimation in a portable video encoder are shown to save 70% of the power over a full bit width implementation [23].

B. Architectural

At the architectural and microarchitectural level, instruction set design and exploitation of parallelism and pipelining are important in minimizing power consumption. Architecture-driven voltage scaling as a method for power reduction is presented in [19]. The approach is based on lowering voltage to reduce power consumption, and then to apply parallelism and/or pipelining to maintain throughput as the speed of a function unit is decreased. This type of approach is useful if enough parallelism exists at the application level to keep the pipeline full, but trades off increased latency and additional area overhead in the form of duplicated structures (parallelism) or pipeline register overhead (pipelined). For general purpose CPU development, exploiting pipelining and parallelism is important for improved performance. Increases in latency due to deeper pipelining affect the metric of instructions per clock due to data dependencies and control flow dependencies. In the search for maximum overall performance, complicated value prediction schemes and speculative fetch and execution of unresolved branch target instruction streams are often employed for deeply pipelined processors designed for highest performance in order to reduce dependency-related stalls. The overhead for these schemes results in extra energy consumption, and additionally, incorrect speculation results in discarding of operations, an additional waste of energy. Low-power designs tend to avoid these deeply pipelined approaches unless the amount of speculation is limited, the overhead for speculation is low, and the accuracy of speculation is high. Meeting required performance for an application without overdesigning a solution is a fundamental optimization. Additional circuitry designed to dynamically extract more parallelism can actually be detrimental, since the power consumption overhead of this logic is not generally controllable, and will be present even when the additional parallelism is absent from the application.

C. Logic and Circuit Level

Many techniques for power reduction are available at the logic and circuit levels. Most focus on reducing the effective switched capacitance, C_{eff} in (3). Others focus on reduced signal swing, thus avoiding the quadratic dependence on supply voltage.

Static and dynamic (clocked) logic families are both utilized in CMOS designs. Depending on signal probabilities, one or the other may offer reduced effective switched capacitance. For a two-input NAND gate, assuming uniform distribution of input values, the probability of the output being 0 (p_0) is 0.25 (both inputs are 1) and being a 1 (p_1) is 0.75. For a static gate, the probability of a power consuming transition from $0 \rightarrow 1$ ($\mathrm{p}_{0\rightarrow1}$) is then 0.1875 ($\mathrm{p}_0{}^*\mathrm{p}_1$). For the dynamic gate with the output precharged to logic 1, power is consumed whenever the output was previously a 0. Relative to a static gate, the probability of a power consuming transition is higher (0.25), and power is consumed even when the logical value of the output remains 0, which is not the case for the static version. The dynamic version typically has

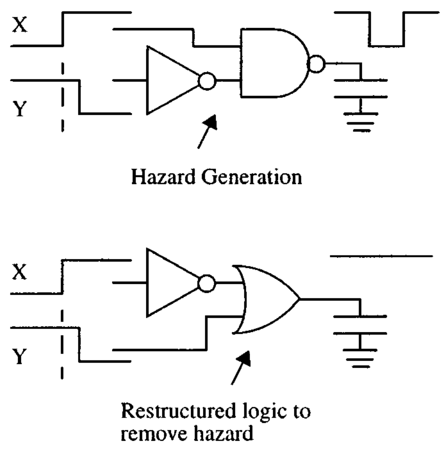

Fig. 1. Glitching in static logic and restructuring for elimination.

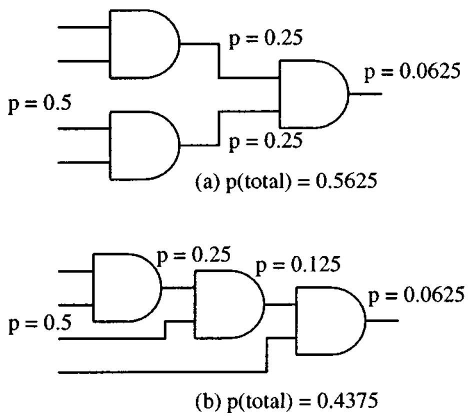

Fig. 2. Equivalent logic mappings with different power costs.

lower input capacitance by a factor of 2 to 3 however since PMOS devices are not driven by logic inputs, thus C_{eff} for the dynamic gate may be much lower, even though it has a higher activity factor. For a wider input static gate, such as a four-input NAND, $p_0 = 0.0625$, and $p_{0\to1}$ is 0.0586. For the dynamic version, $p_0 = p_{0\to1} = 0.0625$. Increasing the number of inputs leads to a lower probability of an output transition. On the other hand, input capacitive loading increases if delay time is held constant, since larger transistors must be used. Intrinsic capacitance of the gate also increases. The power consumed in distributing the precharging signal to the dynamic gate must also considered. A number of different logic families (both static and dynamic) have been proposed in the literature including variants of pass transistor logic (CPL), and cascode voltage switched logic (DCVSL) offering area, speed, and power tradeoffs. An extensive review of the many types of clocked and static logic families may be found in [24].

Static logic may suffer from hazards (or glitches) that result in unnecessary power consumption due to differences in gate input arrival times. These differences in arrival times may cause multiple output transitions, resulting in a value for α that is >1. As an example, the output of a simple two-input circuit in Fig. 1 has unnecessary signal transition from high $\rightarrow$ low $\rightarrow$ high due to the difference in arrival times of inputs X and Y. This hazard may be propagated through additional logic levels and result in multiple gate output transitions before the circuit resolves to a final state, even if the final state is unchanged from the previous state. As the number of logic levels increases in a combinational circuit, the probability of unequal path delays from input to output increases, thus increasing the potential for glitching. Logic restructuring and path delay balancing may be used to reduce glitch power, which can be responsible for $>20\%$ of overall dynamic power consumption in combinational circuits [25]. Fig. 1 shows a restructured circuit realizing the same logic function with reduced glitching. Path delay balancing may be performed by either resizing of individual logic gates to equalize path delay, or by insertion of additional logic elements in faster paths. Since both methods can result in additional switched capacitance, they must be used judiciously. Dynamic logic does not suffer from glitch power since all inputs must be valid before the gate evaluates.

Technology mapping of logic functions to gates may choose to optimize power at the expense of area. A robust standard cell library for low power will include gates with a variety of logic functions as well as multiple drive strengths for each function. Complex gates (AND–OR–INVERT, OR –AND–INVERT, etc.), NAND and NOR gates with inverted inputs, and a rich set of storage elements provide synthesis tools with the flexibility to optimize power consumption. Transition probabilities of the logic being mapped are used in conjunction with loading models of the library elements to select a mapping of the desired Boolean function onto a set of gates in the library which minimizes power, subject to meeting a set of delay constraints. Fig. 2 shows an example of differences in a four input AND function mapping. In the example, mapping (a) consumes more power than mapping (b) due to differences in the total transition probabilities of the three two-input gates. Improvements averaging 10% on a set of benchmarks were obtained in [26] by using power instead of area as a minimization criteria. Their algorithm resulted in an area increase of 12%, showing that minimized area does not necessarily result in minimum power. A similar result is reported by [27], where average power dissipation is reduced by 21% with a corresponding 13% increase in area. Hiding high-probability switching nodes inside of complex gates is used to minimize total switched capacitance.

Synthesis techniques using a hybrid library composed of static CMOS gates in conjunction with pass logic cells have also been shown to be effective in improving power dissipation [28]. Reordering of equivalent inputs of gates and reordering of transistors in complex gates are also techniques available to reduce power. Fig. 3 shows transistor diagrams of a complex gate realizing the logic function $(A + B)^*C$ with an example of input reordering and transistor reordering. Input and transistor ordering affect the amount of switched internal capacitance of the gate, and also affect the speed of the gate and its static power dissipation. In general, inputs signals with high probability of being off are placed nearest the output node of the gate, subject to timing constraints being met, and signals with high probability of being on are placed nearest the supply node.

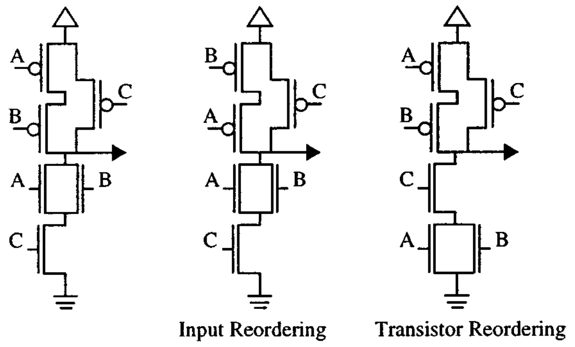

Fig. 3. Input and transistor reordering.

Signals with a high probability of switching (high transition density) are placed nearest the output. A set of rules for ordering simple and complex gates and experimental results are found in [29], where an average 10% savings in power was found between the worst and best orderings.

Sequential circuits are also a focal point for power reduction. Clocks typically consume a large fraction of overall power in synchronous systems; depending on the design target, 30%–40% of total system power is consumed by clock generation and distribution. Low-power optimizations are targeted at minimizing unnecessary transitions on clock signals as well as in combinational logic used for state machine control. Storage element design is also important, and speed/power tradeoffs are available here as well [30]. State assignment for low power has also been explored. In general, the state assignment problem has targeted minimizing area, and this approach tends to reduce power as well. As with combinational logic minimization, area may be traded for reduced power. Low-power state assignment techniques augment the state transition graph (STG) of the state machine with state probabilities and transition probabilities between states, and use these probabilities to guide the state assignment. Adjacent binary encodings are assigned to states connected with high probability edges of the graph. This minimizes the number of state signal transitions, thus attempting to minimize transitions in the next state and output signal combinational logic. One approach attempts to minimize area in conjunction with switching activity by generating multiple sets of state encodings with similar switching energy costs from which a final assignment is chosen on the basis of area [31].

Clock power reduction is important in synchronous systems, since as was noted earlier, it can contribute to a large portion of the overall power budget. Minimization of clock power falls in to several categories including clock distribution optimizations, clock gating, and low-swing clocking techniques.

Gated clocking is a commonly applied technique used to reduce power by gating off of clock signals to registers, latches, and clock regenerators. Gating may be done when there is no required activity to be performed by logic whose inputs are driven from a set of storage elements. Since new output values from the logic will be ignored, the storage elements feeding the logic can be blocked from updating

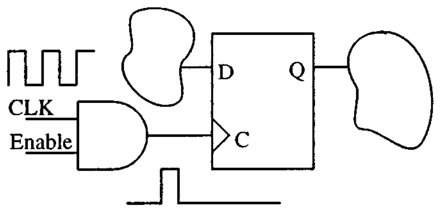

Fig. 4. Clock gating.

to prevent irrelevant switching activity in the logic. Fig. 4 shows an example of clock gating. Clock gating may be applied at the function unit level for controlling switching activity by inhibiting input updates to function units such as adders, multipliers, and shifters whose outputs are not required for a given operation. Entire subsystems may be gated off by applying clock gating in the distribution network. This provides further savings in addition to logic switching activity reduction since the clock signal loading within the subsystem does not toggle. Overhead associated with generation of the enable signal must be considered to ensure that power saving actually occurs, and this generally limits the granularity at which clock gating is applied. It may not be feasible to apply clock gating to single storage elements due to the overhead in generating the enable signal, although self-gating storage elements have been proposed that compare current and next state values to enable local clocking [32]. If the switching rate of input values is low relative to the clock, a net power saving may be obtained.

Reduced swing clock drivers have been explored as another method to reduce clock power. Reducing clock driver supply voltage by 50% and providing specially designed flip-flops that receive the half-swing clock results in a theoretical power saving of 75%, and a reported savings of 63% in [33]. The drawback to this approach is an increase in the flip-flop delay of $> 2\times$. Another approach in [34] reduces the swing of a pair of complementary clocks by 50% and overcomes the issue with increased flip-flop delay by providing full V_{dd} to the clocked nodes of the flip-flop circuit. In this approach the theoretical power savings is 50%, and an actual savings of 43% is achieved.

Differential clock signaling is an alternative that allows the clock swing to be reduced well below 50% of V_{dd}. Differential signaling typically consumes static power, thus the power savings due to a differential clock network are dependent on the operating frequency of the clock and the load being driven. With a signaling technique using a pair of differential lines that swing at $0.2^{*}V_{dd}$, the theoretical saving in the clock distribution network is 60%. Static power consumption in the driver and receiver reduces this saving. Duty cycle and receiver skew effects must also be managed.

Using both edges of the clock to update registers is an option that allows equivalent throughput at half the original clock rate, thus cutting clock power in half. Dual-edge-triggered flip-flops (DETFF) have been developed that update state on both edges of the clock. Although larger than standard single-edge flip-flops, and increased loading on the clock, the 50% reduction in clock distribution power can result in significant power reductions. One drawback of the

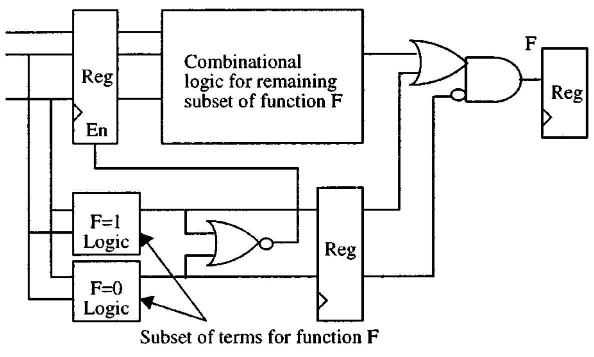
Reg
En
Combinational logic for remaining subset of function F
F
Reg
F=1 Logic
F=0 Logic
Reg
Subset of terms for function F

Fig. 5. Precomputation structure.

DETFF relative to the single edge version is the duty cycle of the clock is now a factor in determining cycle time. A comprehensive comparison of various DETFF implementations is provided in [35].

Retiming of sequential circuits and pipelined datapath logic is a technique traditionally used to increase operating speed of a circuit by balancing the delay of each stage of logic in the circuit. Registers are moved either forward or back along combinational logic paths until the total delay between registers is equalized. As the registers are moved, the number of required registers may increase or decrease based on the number of signals crossing the register boundary. Also, combinational logic optimization opportunities may occur as new logic groups are exposed, thus further improving the circuit speed. The balanced circuit may then be operated at a lower frequency or voltage, thus reducing power consumption further.

One observation made in [36] is that propagation of unnecessary switching activity due to glitches can be halted by insertion of a register in a combinational logic path. The register output will transition once per clock cycle at most, even if the input makes multiple transitions. By placing registers at high fanout nodes, switched capacitance can be minimized, assuming that the additional capacitive load created in adding the register is low enough relative the original load, and the original node had multiple transitions per cycle. Retiming for low power is an approach that attempts to minimize glitch power in a pipeline by moving the registers forming the pipeline to positions that optimally minimize switching activity in the logic network. Since delay of the pipeline stages must be considered, only a subset of nodes in the circuit are candidates for register placement, i.e., those nodes which would not violate delay constraints. Additionally, there is a desire to minimize the number of registers due to area costs as well as the additional clock power consumed.

Precomputation is an optimization technique for sequential circuits which minimizes switching activity by selectively precomputing the output values of a logic circuit before they are required, and then using the computed values to minimize switching activity by disabling inputs to the logic circuit. The precomputed values are then substituted for the original logic circuit output values. Precomputation logic uses a small subset of the original input signals to generate simple logic functions that indicate that the original logic function is either True or False, respectively. By keeping these functions simple, overhead associated with precomputation is minimized. In addition, the original logic function may be simplified since a portion of it is being handled by the precomputation logic itself, and the terms for this portion may be assigned as don't-cares for the original function. Fig. 5 shows one variant of a precomputation circuit implementing a logic function F. In Fig. 5, the logic function F is implemented by precomputing a simple subset of the input combinations for which F is True ($F = 1$ block) and for which F is False ($F = 0$ block). When either of these blocks is active, the inputs to the larger combinational block computing the remaining terms of F are blocked, and the larger block remains quiescent. The precomputation logic then forces the output of function F to "1" or "0," respectively.

As has been seen with other power saving techniques, increased area is traded for reduced power. In [37], the authors report power savings of 11%–66% using precomputation on a number of combinational logic circuits. Methods for generating the precomputation functions are also described.

Guarded evaluation is a similar technique that relies on input blocking for transition reduction [38]. Transparent latches are added to inputs of existing logic and are appropriately disabled when the logic output can be determined without new input values being driven from the disabled latches. This technique is common in the design of datapath functions in low-power processors as will be described later.

For synthesized portions of a design using gates from a predetermined library, gate sizing should be performed when possible to ensure that no noncritical circuit path is overly fast. Gate size selection is typically based on output loading, and fanout ranges of 3–8 are typical. As fanout increases, delay increases but dynamic power is reduced. Care must be taken not to increase fanout to the degree that signal rise and fall times become an issue in increased short circuit power. Custom portions of a design have an additional degree of freedom in that individual transistors may be sized to minimize power. Algorithms have been developed to size individual transistors in a design to minimize delay, power, or the power-delay product within an area constraint. Edge rate constraints are also considered [39].

D. Device Technology

At the device level, threshold voltage selection plays an important role in the tradeoff between performance and leakage power. Supply and threshold voltage selection was discussed earlier [16]–[18]. Alternative process technologies to bulk CMOS such as silicon on insulator (SOI) may be attractive due to lowered parasitic capacitance and reduced body effect. Dual device threshold technologies are also an approach to lowering power consumption. High-threshold devices may be used in noncritical delay paths, while reserving low-threshold devices for speed-critical paths, thus minimizing standby power consumption. A methodology for selection of individual device sizes and thresholds to optimize speed and standby power goals is described in

[40]. Alternate approaches for standby power reduction are to raise the threshold of all devices while in standby mode by providing a transistor well biasing circuit.

IV. Embedded Processor Example

Low-power embedded processors fall into several categories. At the extreme low power range, these are typically 8-bit CPUs with power dissipation measured in microwatts, which power devices such as digital watches, calculators, and other long-life devices. In the midrange, 16- and 32-bit processors power handheld devices with dissipation measured in milliwatts. Higher performance 32-bit processors dissipating watts of power cover high-end applications, such as notebook computers.

In the midrange of performance, one example of a 32-bit processor architecture designed specifically for portable and low-power applications is the Motorola M*CORE family. This architecture and its implementations were specifically designed from the ground up to address low-power embedded applications with a range of power and performance constraints, but targeted initially at the midrange applications requiring tens to hundreds of MIPS of performance, while dissipating tens to hundreds of milliwatts of power. Cost is an important factor that cannot be ignored in the design of a commercial, high-volume application, and cost considerations were balanced with power optimizations in both the architecture definition and implementation aspects. Some details of the architecture and implementations are described in the following subsections.

A. Instruction Set Design, Programmer's Model

At the architectural level, the specification of an instruction set can have a large effect on system power dissipation as well as performance. As is to be expected, there are tradeoffs to be made. RISC, CISC, and VLIW architectures are examples of approaches to instruction set design, each with their own merits. For low-cost systems, instruction code density is an important factor, since the cost of instruction memory is directly related to the size of the binary images of the programs embedded into the system. CISC designs typically provide good code density due to the complexity of individual instructions and due to their use of variable length instruction formats. Traditional RISC and VLIW instruction sets trade code density for simplified decoding and straightforward instruction fetch units. While code density remains high with CISC approaches, the complications in control circuitry for fetching, decoding, and sequencing tend to cause increased overhead in power, and either cost or performance tend to suffer.

For a low-power focus, the desire is to have as large a percentage of power consumption utilized for the fundamental computational operations required by the algorithm being executed. Fetch, decode, and sequencing of instructions represents overhead associated with managing the computational task, and an approach that reduces the power in these areas is important. Traditional RISC architectures define a fixed-length instruction that is not highly encoded, thus reducing the sequencing overhead significantly. Typically a load–store (or register–register) model is chosen in which operations are performed using a set of general-purpose registers, and the only operations on memory are loads and stores. Ease of decoding and the ability to pipeline operations with low control overhead are advantages. The increased instruction fetch bandwidth required represents a drawback, as the typical RISC instruction is encoded as a 32-bit word. Average instruction lengths for CISC architectures with variable length instructions are on the order of 22–24 bits, and these instructions have more semantic content than a RISC instruction. They typically support operations on memory directly, via a set of complex addressing modes.

An instruction set design based on a fixed-length 16-bit instruction format was selected for the M*CORE architecture, as well as a RISC load-store model with a 16-entry general purpose register file, where the only operations performed on memory are loads and stores. The ISA departs from a pure RISC approach in several areas to achieve improved code density, such as support for instructions that save and restore a group of general-purpose registers to and from memory for increased code density. Relative to a 32-bit ISA, the limitations of 16-bit instructions cause longer execution pathlengths due to limitations on the size of immediate fields, effective address offsets, and a 2-operand instruction format in which one of the source registers also serves as the destination. Using compiler-driven instruction definition during development minimized these limitations. Trace analysis was used to minimize instruction bandwidth requirements and instructions were selected to minimize the overhead for common code sequences.

The instruction set supports byte, halfword and word (32-bit) data types, and a complete set of logical, shift, bit manipulation, and arithmetic operations that operate on a register and either another register or a 5-bit immediate field. Load and store instructions provide a single base $+$ 4-bit scaled displacement addressing mode. A single condition code bit is defined, and conditional branch instructions test the value of this bit for either true or false. Branch instructions support an 11-bit displacement field, sufficient to satisfy $>98\%$ of all displacements. Providing multiple compare instructions allows any Boolean relationship of variables to be generated, and requires less precious opcode space than providing conditional branch instructions that test for multiple conditions, due to the size of branch displacements. Sizes of immediate fields are limited, so special instructions are provided for generation of commonly occurring constants. Constants from 0–128, all powers of two, and all powers of two -1 are available directly in the ISA. Larger arbitrary values are either synthesized with a pair of instructions, or are loaded from memory as 32-bit constants with a PC-relative load word instruction (LRW). A single storage location for these large constants may be referenced by multiple LRWs, thus amortizing the storage cost. Conditional move, increment, decrement, and clear instructions are provided to eliminate some branches. A complete description of the MCORE processor architecture and ISA can be found in [41] and [42].

By careful selection of instruction semantics and immediate/displacement widths, we find that object code compiled for this ISA is less than 70% the size of code for a typical 32-bit RISC, which results in a significant cost advantage. The penalty in terms of pathlength increase (number of instructions executed) across a variety of embedded applications is on the order of 15%–20% relative to a 32-bit RISC instruction encoding. Similar conclusions were reached in [43]. From a power perspective, this means memory traffic (in bytes) is reduced dramatically since instructions are 16 bits in length. In spite of the greater number of instructions executed, the overall power consumption is reduced, since on-chip instruction memory power consumption is typically 1.5–2$\times$ greater than the CPU in our designs, and instruction memory traffic has been reduced by 40%.

Other advantages related to power and performance are realized. For designs utilizing cache memory, the instruction cache capacity is effectively doubled, since approximately twice as many instructions can be stored. Cache miss rates of typically sized embedded cache designs (4–32 kB) may be reduced 30%–50% with this effect. Given that accessing the next level of the memory hierarchy can result in factors of 20$\times$ greater power consumption or more due to traversing chip boundaries, this reduction in miss rate is significant. In cacheless designs where memory is embedded on-chip, the power consumption of memory is reduced due to the reduced capacity requirement. For on-chip memories or caches, a 32-bit data path is typically provided, which results in double the effective fetch bandwidth relative to a 32-bit instruction word, allowing instruction memory to be accessed every other cycle on average, even with a target of single cycle instruction execution. For low-cost designs where instruction memory is off chip, the ability to fetch a pair of instructions at a time across a 32-bit interface reduces effective memory latency. Even a narrow 16-bit interface path results in greatly reduced performance degradation relative to a wider instruction word.

After selecting the set of operations to minimize code size and execution pathlength and defining the instruction formats, the task of encoding of opcodes remained. We performed an initial encoding assignment and then iterated it to reduce the number of terms and literals in a two-level programmable logic array targeted for controlling a processor data path which implemented the data operations defined by the instruction set, as well as control of an instruction prefetch and program counter unit. By viewing this task as a state assignment problem for sequential logic minimization, each instruction opcode is assigned to a state. A Moore-machine model was used in which control outputs are a function of present state only. Inputs to the state machine are the next instruction opcode, and all states are completely interconnected via an exhaustive set of edges. Next state equations are ignored, since they are a function of only the inputs, not current state. By casting the opcode assignment problem in this fashion, state assignment tools were used to automate the process. This process was iterated as the control signal requirements were altered to further minimize area. Often, multiple equivalent control sets can be used to obtain the desired function. As an example, to implement the logical NOT instruction, we can either exclusive–or the source value with -1 in a logical unit, or we may perform a subtract from 0 with inverted carry-in in the add unit. Since the energy used by the logical unit is lower than the adder, it is the obvious first choice. In some circumstances, however, utilizing the adder results in lower overall energy usage, since it may allow additional reduction in control circuitry transitions by collapsing control terms in the output equations of the control decoder. This is particularly true when the instruction or function in question has a low dynamic frequency of execution. Compiler-directed feedback was used to determine the best tradeoffs between control decoder power and execution unit power in a number of instances.

In addition to area minimization, minimizing control unit power consumption is desired. This was done by instrumenting an instruction set simulator to capture the frequency of execution of all instructions, as well as instruction pairs. Opcodes were ordered by frequency and by frequency of execution pairs, and an initial state assignment was performed on the most frequently occurring instructions, with the objective of assigning adjacent states to frequently occurring instruction pairs. The remainder of the state assignments were made with automated state assignment tools. We achieved control section power savings of approximately 15% with this approach to opcode assignment for our baseline machine, with no increase in area.

Beyond just CPU power reduction, system-level power savings are supported by the ISA with three low-power operating mode instructions. The WAIT, DOZE, and STOP instructions are provided to enable a system to be placed in increasingly lower power modes as appropriate for operating conditions. When the CPU encounters one of these instructions, it completes all previous instructions in the pipeline, finishes all outstanding prefetch operations, and then enters a state where internal clocks are gated off. A pair of control outputs that encode the present operating mode are driven to the rest of the system to allow specific low-power operating conditions to be defined by the system designer. The CPU will exit these modes and resume normal operation once a pending wakeup request is recognized. As an example of system use, the WAIT mode might be used to disable only the CPU, while keeping system PLLs and peripherals active. If there is not expected to be a need for processing for a longer period of time, the DOZE mode might be defined to disable PLLs and certain peripherals that are unnecessary in that mode. Wakeup from this state would entail a longer period of time. The STOP mode can be used to enter a deep power-down state in which all clocks are stopped at the system level, and power supply voltage either reduced or totally switched off to major subsystems.

B. CPU Microarchitecture

While many processor implementation techniques in extremely high-end designs are focused on extracting all possible instruction-level parallelism, these techniques tend to have a correspondingly high level of power inefficiency.

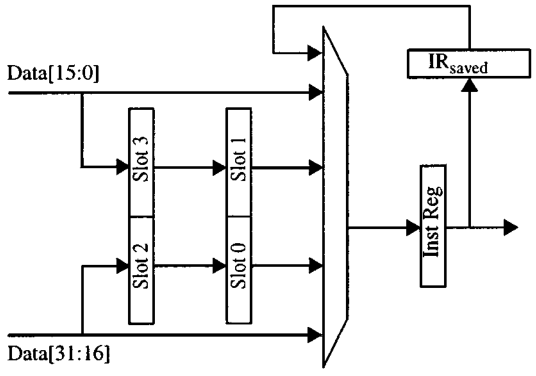

Fig. 6. Instruction buffer supporting the unified bus architecture.

Many embedded control algorithms do not display a high degree of opportunity to exploit parallelism, except in the areas of signal processing and multimedia. Power efficient solutions for both of these domains tend to rely on specialized hardware acceleration, not general purpose computing solutions. For midrange controller applications, a simple pipelined microarchitecture offers a reasonable balance between performance, cost, and power efficiency.

We selected a five-stage instruction pipeline (Fetch, Decode, Execute, Memory, and Writeback) and optimized for power consumption in initial M*CORE implementations. A unified memory system was chosen with a 32-bit-wide interface, as opposed to dual instruction and data memory ports. This was due to the 16-bit instruction word size. Since the goal of the initial CPU microarchitecture was to achieve an ideal execution rate of one instruction per clock and instruction fetch bandwidth of two instructions per clock is available, the additional overhead and inefficiency of memory utilization for dual (Harvard-style) memories was avoided. As long as the relative frequency of data memory operations is less than 50%, the memory port remains underutilized. In our typical benchmark suite, load and store instructions comprise about 23% of the overall dynamic instruction mix. For situations requiring more data bandwidth, load and store instructions are available that move 128 bits of data. Priority is given to data accesses across the unified interface since an instruction buffer is provided in the CPU. Fig. 6 shows a diagram of the instruction buffer structure. The buffer captures a pair of instructions per transfer into an even and an odd slot. Idle cycles on the unified bus are used to fill empty slot pairs, providing an increase in effective instruction bandwidth. More aggressive microarchitectures that attempt to issue multiple instructions per clock would likely require either a wider port or separate instruction and data ports to memory.

Custom logic design was used in the datapath of the processor for the register file, function units, operand multiplexers, and writeback logic. Synthesized logic was used in the control section. Evaluation of synthesized logic for datapath elements showed an average area increase of 2× and power dissipation increase of 2.5× over custom designed units. The functionality of the datapath logic was established early in the design phase, thus, the degree of change was limited. Control logic, on the other hand, typically remains in

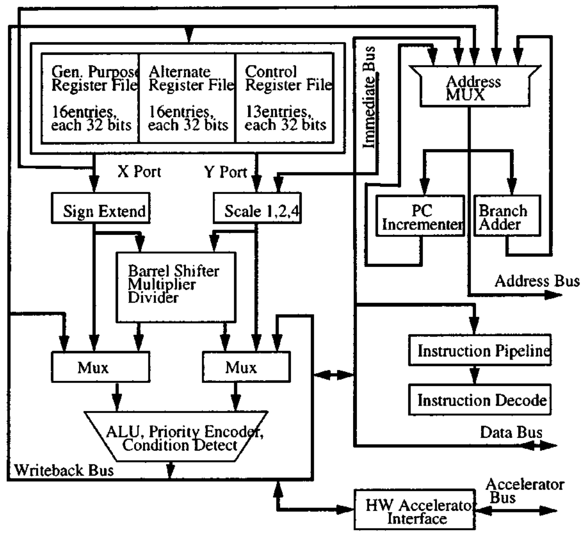

Fig. 7. Processor datapath.

a state of flux until very late in the design cycle, thus, the flexibility of logic synthesis is an overriding consideration. A high-level diagram of the datapath appears in Fig. 7. Initial sizing of datapath circuits was performed manually, followed by an automated sizing tool Focus [39], which provides a set of solutions with various speed and area tradeoffs. Focus begins with a minimally sized circuit netlist, and then iteratively sizes transistors along critical paths based on a sizing merit formula until timing constraints are met. In comparison with the manual device sizing, Focus was able to achieve area savings of 17% on the logic unit with no performance penalty.

Gated clocks and delayed clocks are used to control all datapath control points; there are no free-running clocks in the datapath. This is critical to reduced power. Clock gating elements eliminate unnecessary transitions on the clock distribution circuits as well as preventing unnecessary logic transitions in computational elements that are not being used in a particular cycle. Storage elements are also simplified, since a feedback path from output to input is no longer required to maintain present state. Using an approach similar to the concept of guarded evaluation, the adder, barrel shifter, find-first-one unit, logic unit, multiplier, and branch adder are all preceded by latches that conditionally open based on the currently executing instruction. Gated clocks control these latches, and in contrast to the approach in [38], the latches actually form part of the instruction pipeline, thus introducing no additional overhead. Fig. 8 illustrates an example of the input and output gating for the address adder. Delayed clocks are used to allow inputs or outputs of a unit to settle before being propagated to downstream logic. For example, when calculation of a load or store address is being performed, the calculation begins following the rising edge of the clock. Since the adder is allocated about 60% of the clock cycle to compute the result, driving of the output value onto the highly loaded address bus is delayed until partway into the low portion of the clock cycle to allow the adder to complete its evaluation. The delay is set such that the adder has completed the result calculation for a large

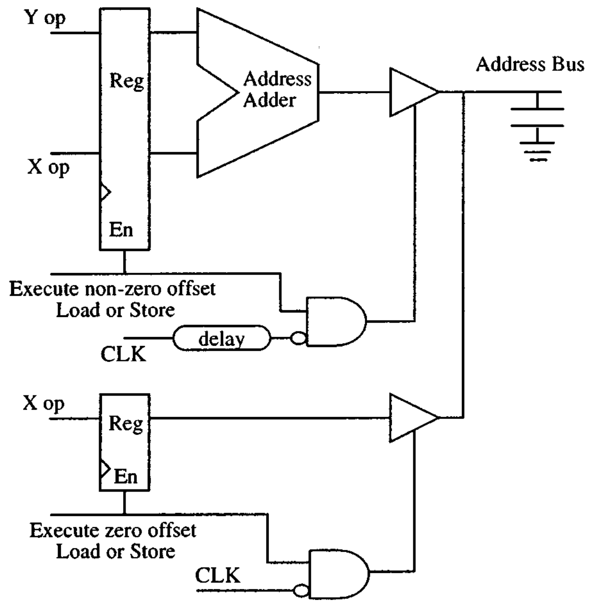
Y op
Reg
Address Adder
Address Bus
X op
En
Execute non-zero offset Load or Store
delay
CLK
X op
Reg
En
Execute zero offset Load or Store
CLK

Fig. 8. Address adder input and output gating.

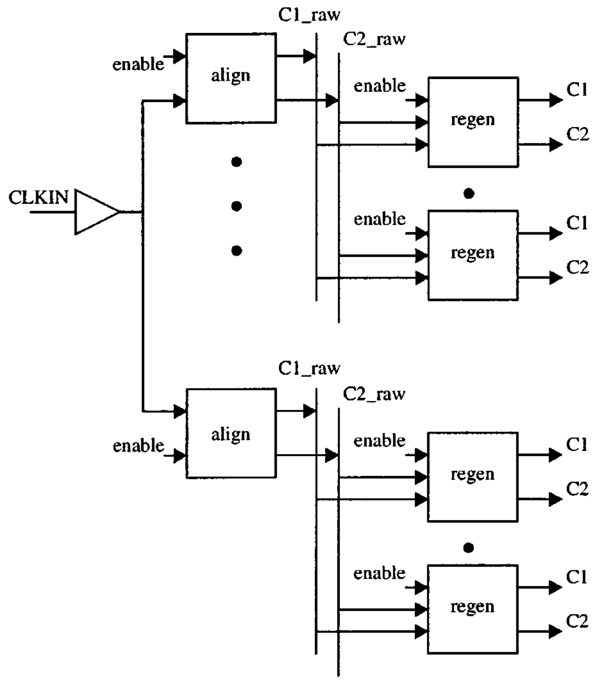

Fig. 9. Two-level clock distribution network.

majority of input value cases, but not for all cases. By doing so, critical path timing is not affected by the delay circuit, but a large majority of calculations are completed prior to the delay interval.

Precomputation is used in the address calculation logic for load and store instructions to detect when the displacement field of the instruction is zero. In this case, no addition of displacement to the base register is required, so the adder is bypassed and remains idle. Fig. 8 shows the bypass logic. From our measurements, almost 50% of load and stores have a zero displacement field.

Floorplanning of the datapath elements was driven by switching activity measurements gathered while executing a set of embedded benchmarks. Relative utilization of each function unit was coupled with capacitive loading of the units, and block placement was performed to minimize the overall bus loading and effective switching frequency of source and destination buses. Source and destination bus segmentation allowed infrequently utilized function units to be placed farther from the centralized register file, and decoupled such that bus segments attached to these units are only driven when the function unit is required for instruction execution.

Branch instruction execution is well recognized as a critical factor in processor performance due to the need to discard operations following a taken branch and then refill the instruction pipeline. Due to the relatively high frequency of change of flow events ($\sim$18% in our systems), branch acceleration techniques are an important element of a microarchitecture. Increased complexity must be balanced with increased power consumption. Our initial implementations of the M*CORE ISA include a dedicated branch adder specifically designed for high-speed branch target calculations. Since the branch displacement is limited to a signed 11-bit value, a specialized design results in faster target calculations than a generalized 32-bit adder. We implement an aggressive branch taken instruction timing of two cycles, with single cycle branch not taken timing. No branch prediction logic is required in this approach. This allows increased performance and less wasted processing to occur, but has some significant circuit timing implications. One implication is that the input values to the branch adder cannot be completely clock gated based on fully decoding a branch instruction and still meet timing requirements. Instead, we perform an incomplete decoding of the branch opcode by combining it with the infrequently executed encodings for Add and Subtract with Carry (ADDC, SUBC). This reduces the gating function delay to a single gate.

Noting that many embedded applications spend a significant portion of execution inside program loops, we have augmented the branch unit with a program loop folding capability that captures information about program loops for use during successive iterations of the loop. This loop folding hardware removes the need to fetch, decode, or execute loop branches for most loop iterations, thus increasing performance and lowering power [44].

C. Clock Distribution

Clock power represents a large portion of overall power consumption in a design optimized for low power. Efficient distribution and gating mechanisms are essential for reducing this component. By adopting a two-level structure for this task, improved power efficiency is realized. The first level of logic is used to align clock edges at various points in the circuitry and to generate two nonoverlapped phases. These nonoverlapped phases are then distributed to a set of clock regenerating cells with clock gating control inputs. Gating of clocks can be performed at both levels depending on the granularity required. Fig. 9 illustrates the two-level approach.

Clock tree generation algorithms have been adopted that allow an unbalanced H-tree structure to be used. As opposed

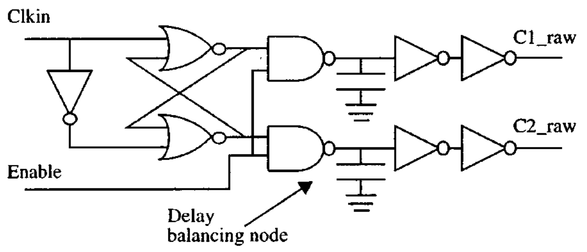

Fig. 10. Clock aligner and low-capacitance balancing nodes.

to traditional clock trees that balance loading at all nodes of the H-tree by adding additional routing capacitance to short nodes, a loading algorithm is adopted that resizes intermediate devices in the clock aligner and regenerator cells to obtained balanced delays. These intermediate device nodes have much lower capacitance than the routing, so small increases in capacitance can be used to obtain the balancing, thus resulting in lower wasted power relative to the traditional tree generation approaches. Delayed clocks are also produced using a similar sizing approach. Fig. 10 illustrates the clock aligner structure and the balancing nodes.

V. Conclusion

Low-power design requires attacking the power dissipation problem at all levels of the design hierarchy. No single target will be sufficient to extract the efficiency required for future handheld products. Voltage scaling has limits that will require additional advanced techniques to be applied at the algorithmic and architectural level for additional power savings. Dynamic voltage scaling based on system loading and processing requirements is an emerging technique with great promise. Clock power optimizations will remain a challenge as higher frequencies and increased pipelining are applied to extract increased performance. Parallelism must be efficiently extracted without sacrificing the goal of low power. Software generation strategies that are based on power cost functions will be increasingly common in these future systems. Even though a broad range of power reducing techniques have been proposed, the challenge still remains to integrate them into a design flow in which power plays as large a role as performance.

References

[1] R. A. Powers, "Batteries for low power electronics," *Proc. IEEE*, vol. 83, pp. 687–693, Apr. 1995.
[2] J. Costello, "Choosing the right battery to power the portable product," *Electron. Prod.*, Dec. 1992.
[3] A. P. Chandrakasan and R. W. Broderson, *Low Power Digital CMOS Design*. Norwell, MA: Kluwer, 1995.
[4] A. Bellaur and M. I. Elmasry, *Low Power Digital CMOS Design: Circuits and Systems*. Norwell, MA: Kluwer, 1996.
[5] J. M. Rabaey and M. Pedram, Eds., *Low Power Design Methodologies*. Norwell, MA: Kluwer, 1996.
[6] M. S. Elrabaa, I. S. Abu-Khater, and M. I. Elmasry, *Advanced Low-Power Digital Circuit Techniques*. Norwell, MA: Kluwer, 1997.
[7] L. Benini and G. De Micheli, *Dynamic Power Management: Design Techniques and CAD Tools*. Norwell, MA: Kluwer, 1998.
[8] *Power Driven Microarchitecture Workshop*, in conjunction with ISCA '98, Barcelona, Spain, June 28, 1998.
[9] *IEEE Alessandro Volta Memorial Workshop Low-Power Design*, Como, Italy, Mar. 1999.
[10] *Int. Symp. Low Power Electronics and Design*, Rapallo/Portacino Coast, Italy, July 2000.
[11] S. Hauck, "Asynchronous design methodologies: An overview," Univ. Washington, Tech. Rep. UW-CSE-93-05-07, May 1993.
[12] A. Bellaur and M. I. Elmasry, *Low Power Digital CMOS Design: Circuits and Systems*. Norwell, MA: Kluwer, 1996, ch. 4, pp. 135–137.
[13] H. J. M. Veendrick, "Short-circuit dissipation of static CMOS circuitry and its impact on the design of buffer circuits," *IEEE J. Solid-State Circuits*, pp. 468–473, Aug. 1984.
[14] S. R. Vemuru and N. Scheinberg, "Short circuit power dissipation estimation for CMOS logic gates," *IEEE Trans. Circuits Syst.*, pp. 762–765, Nov. 1994.
[15] A. Bellaur and M. I. Elmasry, *Low Power Digital CMOS Design: Circuits and Systems*. Norwell, MA: Kluwer, 1996, ch. 3, p. 90.
[16] D. Liu and C. Svensson, "Trading speed for low power by choice of supply and threshold voltages," *IEEE J. Solid-State Circuits*, pp. 10–17, Jan. 1993.
[17] S. W. Sun and P. Tsui, "Limitation of CMOS supply–voltage scaling by MOSFET threshold variation," in *Proc. CICC*, 1994.
[18] C. Chen, J. Shott, J. Burr, and J. D. Plummer, "CMOS technology scaling for low voltage low power applications," in *Proc. IEEE Symp. Low Power Electronics*, San Diego, CA, Oct. 10–12, 1994, pp. 56–57.
[19] A. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-power CMOS digital design," *IEEE J. Solid State Circuits*, pp. 473–484, Apr. 1992.
[20] A. Chandrakasan and R. W. Brodersen, "Minimizing power consumption in digital CMOS circuits," *Proc. IEEE*, vol. 83, pp. 498–523, Apr. 1995.
[21] Y. F. Tong, R. A. Rutenbar, and D. F. Nagle, "Minimizing floating-point power dissipation via bit-width reduction," in *Power Driven Microarchitecture Workshop* in conjunction with ISCA '98, Barcelona, Spain, June 28, 1998, pp. 114–118.
[22] M. J. Schulte, J. E. Stine, and J. G. Jansen, "Reduced power dissipation through truncated multiplication," in *Proc. IEEE Alessandro Volta Memorial Workshop Low-Power Design*, Como, Italy, Mar. 1999, pp. 61–69.
[23] Z.-L. He, K.-K. Chan, C.-Y Tsui, and M. L. Liou, "Low power motion estimation design using adaptive pixel truncation," in *Int. Symp. Low Power Electronics and Design*, Monterey, CA, Aug. 1997, pp. 167–172.
[24] K. Bernstein *et al., High Speed CMOS Design Styles*. Norwell, MA: Kluwer, 1998, ch. 2–3, pp. 51–131.
[25] A. Raghunathan, S. Dy, and N. K. Jha, "Register transfer level power optimization with emphasis on glitch analysis and reduction," *IEEE Trans. Computer-Aided Design*, pp. 1114–1131, Aug. 1999.
[26] V. Tiwari, P. Ashar, and S. Malik, "Technology mapping for low power," in *30th ACM/IEEE Design Automation Conf.*, 1993, pp. 74–79.
[27] C.-Y. Tsui, M. Pedram, and A. M. Despain, "Technology decomposition and mapping targeting low power dissipation," in *30th ACM/IEEE Design Automation Conf.*, 1993, pp. 68–73.
[28] M. Gallant and D. Al-Khalili, "Synthesis of low power circuits using combined pass logic and CMOS topologies," in *10th Int. Conf. Microelectronics*, Dec. 1998, pp. 59–62.
[29] W.-Z. Shen, J.-Y. Lin, and F.-W. Wang, "Transistor reordering rules for power reduction in CMOS gates," in *Proc. ASP-DAC '95*, 1995, pp. 1–6.
[30] R. P. Llopis and M. Sachdev, "Low power, testable dual edge triggered flip-flops," in *Int. Symp. Low Power Electronics and Design*, Aug. 1996, pp. 341–345.
[31] D. Chen, M. Sarrafzadeh, and G. Yeap, "State encoding of finite state machines for low power design," in *Proc. ISCAS '95*, vol. 3, pp. 2309–2312.
[32] A. G. M. Strollo, E. Napoli, and D. De Caro, "New clock-gating techniques for low-power flip-flops," in *Int. Symp. Low Power Electronics and Design*, July 2000, pp. 114–119.
[33] H. Kojima, S. Tanaka, and K. Sasaki, "Half-swing clocking scheme for 75% power saving in clocking circuitry," *IEEE J. Solid-State Circuits*, pp. 432–435, Apr. 1995.
[34] J.-C. Kim, S.-H. Lee, and H.-J. Park, "A high-speed 50% power-saving half-swing clocking scheme for flip-flop with complementary gate and source drive," in *ICVC '99 6th Int. Conf. VLSI and CAD*, 1999, pp. 574–577.

[35] W. M. Chung and M. Sachdev, "A comparative analysis of dual edge triggered flip-flops," in *2000 Canadian Conf. Electrical and Computer Engineering*, vol. 1, 2000, pp. 564–568.
[36] J. Monteiro, S. Devadas, and A. Ghosh, "Retiming sequential circuits for low power," in *Int. Conf. Computer-Aided Design*, Nov. 1993, pp. 398–402.
[37] J. Monteiro, S. Devadas, and A. Ghosh, "Sequential logic optimization for low power using input-disabling precomputation architectures," *IEEE Trans. Computer-Aided Design*, vol. 17, pp. 279–284, Mar. 1998.
[38] V. Tiwari, S. Malik, and P. Ashar, "Guarded evaluation: Pushing power management to logic synthesis/design," *IEEE Trans. Computer-Aided Design*, vol. 17, pp. 1051–1060, Oct. 1998.
[39] A. Dharchoudhury *et al.*, "Transistor-level sizing and timing verification of domino circuits in the PowerPC microprocessor," in *Proc. IEEE Int. Conf. Computer Design*, Oct. 1997, pp. 143–148.
[40] S. Sirichotiyakul *et al.*, "Standby power minimization through simultaneous threshold voltage selection and circuit sizing," in *Design Automation Conf. '99*, 1999, pp. 436–441.
[41] *M*CORE Reference Manual*: Motorola Inc., 1997.
[42] B. Moyer and J. Arends, "RISC gets small," *Byte Mag.*, Feb. 1998.
[43] J. Bunda *et al.*, "16-bit vs. 32-bit instructions for pipelined microprocessors," in *Proc. 20th Int. Conf. Computer Architecture*, 1993, pp. 237–246.
[44] L. H. Lee, J. Scott, B. Moyer, and J. Arends, "Low-cost branch folding for embedded applications with small tight loops," *MICRO-32*, pp. 103–111, Nov. 1999.

Bill Moyer (Member, IEEE) received the B.S.E.E. degree from Rice University and the M.S.E.E. degree from the University of Texas.

He is a Motorola Fellow and Distinguished Innovator active in low-power architecture and system design. He has been involved with design and development of 32-bit microprocessors in Motorola's 68000, 88000, PowerPC, and M*CORE processor families, and embedded systems employing them for over 20 years, and holds more than 65 patents covering a broad range of microprocessor, memory, and embedded system topics. He has also co-authored a number of low-power oriented publications. He is senior architect for the M*CORE processor family, and is responsible for the definition and design of several family members.