YOUR SYSTEM'S POWER
EFFICIENCY DEPENDS
ON HOW WELL YOU
MATCH YOUR HARD-
WARE- AND SOFTWARE-
DESIGN DECISIONS WITH
YOUR APPLICATION'S
OPERATING BEHAVIOR.

SQUeEZE PLAY

WRING THE POWER OUT OF YOUR DESIGN

OWER SYSTEM-POWER DISSIPATION is not a design constraint just for battery-powered applications; it is also a primary concern for many high-performance wired systems. The processor or processors you use in an embedded design may represent a relatively small amount of the system's overall power budget, but your decisions regarding the system and software architecture can significantly impact the overall processing performance, power consumption, and EMI (electromagnetic-interference) performance. For battery-powered systems, lower overall power consumption can mean your design benefits from a longer battery life, or it can enable you to select smaller batteries to minimize your system's size, weight, and cost.

For wired systems, lower power dissipation can result in reducing your system's requirement for cooling fans and air-conditioning, because your system is generating less heat. Reducing your cool-

Illustration by Don Arday

ing requirements in this way may also enable your system to operate more quietly, because you can use smaller power supplies and fewer or quieter fans to remove the heat from a rack or a case environment. Lower peak power dissipation in wired systems can also enable you to increase component density that was constrained by hot-spot limits to increase your system's channel-processing capacity. When you lower your design's power consumption, you may be able to reduce your system's overall size and cost.

To best optimize your system's power consumption, you need to match both your hardware power features and your software-architecture decisions with your application's expected operating behavior. Whether your application's operating behavior is primarily continuous or event-response-based drives how you can maximize your system's power efficiency. Continuous operation is characterized by high processing activity and low idle time; event-response operation is characterized by burst activity separated by long idle times.

The total power dissipation of a CMOS circuit consists of the contribution from both static and dynamic power dissipation. Static power dissipation, which includes transistor leakage currents, occurs even when the circuit is inactive, inde-

pendently of any switching activity. Leakage currents occur in all CMOS transistors and are due to reverse-bias-source or drain-diode currents, drain-to-source weak-inversion currents, and tunneling currents. The process technology and cell libraries predominantly drive the magnitude of these currents. The sum of these leakage components captures the non-ideal current flow from the power supply to ground through the transistor, even when the transistor is logically in an off state (see **sidebar** "Addressing leakage"). Static power dissipation can represent most of the total power for applications that rely mostly on event-response operation separated by long idle periods.

Dynamic or active power dissipation occurs when the circuit and signals are transitioning between logic states, and the magnitude of the power dissipation

correlates with the system voltage, clock frequency, and switching activity. Dynamic power dissipation usually dominates the system-power efficiency for continuously operating applications. You can approximate your system's dynamic power consumption by the equation: $P = CFV^2$, where C is the dynamic capacitance, F is the switching frequency, and V is the supply voltage. Your system's dynamic capacitance is primarily fixed, based on the process technology and cell libraries it uses. The quadratic relationship between power and supply voltage make scaling the supply voltage the largest proportional influence on power consumption. However, the minimum usable supply voltage correlates with a circuit's maximum switching frequency; a higher clock frequency requires a higher relative supply voltage within the same process technology.

ESTIMATING POWER

Choosing components that have been designed for power efficiency is a good start to lowering your power consumption. A benchmark that is often available for comparing processor-power performance is milliwatts per megahertz. A problem with using this benchmark is

AT A GLANCE

▷ Total power consumption is a composite of static and dynamic power-dissipation sources.

▷ The power-management features included in the hardware architecture determine how software can drive down power consumption.

▷ Your software-design decisions can significantly impact your overall power consumption, as well as the product size and cost.

that it does not tell you how much work a processor performs per clock cycle. A lower milliwatts-per-megahertz score would not necessarily reveal that one processor might require twice the frequency and could consume more power to complete the same work as another processor.

An alternative power benchmark is milliwatts per million instructions per second. You still need to exercise caution when using this benchmark, because you may lack visibility into what instruction sequence forms the basis of deriving the

score. For example, basing the score on executing NOP (no-operation) instructions may yield the lowest power the core can dissipate, but the score is not a useful indication of power for useful work. Basing the benchmark score on executing an algorithm entirely from cache or local memory can represent a better mix of instructions, but operating entirely out of local memory may be neither feasible nor applicable for your application design.

The typical and maximum power-consumption numbers published by processor vendors offer limited value for comparing processor architectures, because power consumption can vary significantly, depending on the instructions executed, the data manipulated, the amount of off-chip activity, and the dynamic power-management features. A more useful benchmark for comparing power consumption is milliwatt per task; however, this measurement is difficult to make, especially in the consistent fashion necessary for comparing two processor architectures.

To assist you in making a more usable power estimate, based on how your application will behave and what power-management features it can exploit, some

ADDRESSING LEAKAGE

Transistor switching speeds continue to increase, and transistor geometries continue to shrink. A byproduct of a smaller and faster transistor is that it becomes harder to turn the transistor completely off. A transistor's leakage current increases exponentially as the transistor's gate-dielectric thickness decreases, because the gate dielectric's insular quality decreases and current leaks through it. Uncontrolled, this conduction causes the transistor to act less like an ideal switch with a purely "on" and "off" state and more like a device with an "on" and "leaky off" behavior.

Reducing the supply voltage allows you to reduce the system's power dissipation, but, to maintain high performance margins when using smaller and

faster transistors, you must also use transistors with a lower threshold voltage. As process geometries continue to shrink beyond 90 nm, leakage current continues to grow as a percentage of the overall power, and more engineering creativity is necessary to manage and reduce the growth in overall leakage for high-performance circuits.

One approach to reducing overall leakage is to use slower, lower leakage transistors for noncritical paths and faster, low-threshold transistors that are more leaky on critical paths. Currently, designers manually select where they can use the slower, lower leakage transistors by performing a static-timing analysis on a circuit using the faster transistors and replacing

the areas that have the highest timing slacks with the slower transistors. This approach reduces the overall leakage current by reducing how much of the circuit is subject to higher leakage, but it is an iterative process that requires a cell library with both high- and low-leakage cells.

Dynamic deactivation is a technique for reducing the leakage of the fast, leaky transistors when they are idle; however, you must be careful that cycling the circuit will yield a power savings, because there is a power cost when you turn the circuit off and on. This technique is not useful if your idle times are too short to amortize the activation and deactivation energy required to turn the circuit off and on. Restarting a

deactivated circuit also entails significant start-up latencies that can limit the usefulness of this technique.

Another technique for managing leakage is body bias, which lets you control the bias of the substrate on which the transistor is built to reduce the leakage. This approach allows you to make the circuit fast when you need it and make it slow and less leaky when it is inactive by controlling the body voltage of the transistors. You can also reduce leakage for non-speed-critical circuits by stacking two transistors in a series, which leaks less than an individual transistor. This technique can make sense if leakage power is a larger limiter than the number of transistors.

processor vendors make available spread-sheets and analysis tools, albeit generally crude ones at this time, that allow you to manually characterize your application behavior, the mix of instructions you will use, the amount and type of off-chip activity, and the impact of using the power-management features built into the device. Power-management features include shutting down power to inactive modules, limiting the propagation of the clock signal to inactive blocks, dynamically scaling the clock frequency, and scaling the supply voltage as a function of the clock frequency.

LOW-POWER FEATURES

Many processor devices include sleep, standby, or low-power modes that remove the power supply to specific modules, such as peripheral devices, the processor core, and oscillators used for clock generation. Selectively shutting down the power to these modules allows you to reduce their power dissipation, including the static power dissipation, to zero for circuit blocks that would otherwise not be performing useful work. To preserve application-state data and avoid the need for restarts, low-power modes often preserve power to the memory structures. When switching a module back on, there is a time delay before the supply voltage and possibly the clock signal stabilizes. For this reason, powering down modules in this fashion is impractical when they will be idle for less than the stabilization time or when they need to more quickly respond to an event than the stabilization time allows. Powering down modules is an intrusive technique that usually relies on software, whether at the BIOS, operating-system, or application level, to make decisions based on a generic or a custom model of behavior.

Power dissipation from a device's clock tree can represent as much as 50% of the chip's total power, because the clock signal is typically operating at least twice the frequency of any other signal, and it needs to propagate everywhere. Systems that are partitioned to employ separate clock domains for different modules and components eliminate the need for the entire system to operate at the maximum speed and, as a result, reduce the power dissipation. Using slower clock frequencies not only reduces power dissipation, but also reduces the need for the fast edge rates needed for high clock frequencies that drive a system's EMI performance.

Clock gating is a dynamic power-management technique that is independent of and transparent to software; it reduces dynamic power dissipation and EMI by preventing the clock signal from propagating to currently inactive blocks of the system. Clock gating does not remove power from a functional block, so it does not affect static power dissipation, but clock gating does not suffer from a long start-up-time delay, and it is effective on a clock-by-clock basis.

Clock gating, with a properly designed hierarchical clock-distribution scheme, can stop, with a fine level of granularity, the clock from propagating to inactive components, such as buses, cache memories, functional accelerators, and peripherals. The level of granularity down to which clock gating can practically extend is limited, because it relies on control logic to decide when and whether to propagate the clock signal along a clock-distribution branch. Therefore, to be practical, the power dissipation by the clock-gating control logic should be smaller than the resulting power reduction. Asynchronous or clockless circuits are specialized alternatives to balance

ASYNCHRONOUS CIRCUITS

Synchronous processor architectures rely on a clock-distribution system that delivers the timing signals from the clock source throughout the system so that each functional block operates synchronously with the others. In large systems, clock distribution can represent a significant portion of the system's active power consumption. The potential benefits of using asynchronous or clockless circuits, such as from Fulcrum Microsystems, include lower power consumption, faster processing speeds, and lower EMI (electromagnetic interference).

Asynchronous circuits are delay-insensitive circuits that require no global timing and clock-distribution infrastructure, because they rely on a local handshake between each logic block to trigger the movement of data through a pipeline. The handshake circuits exchange completion signals to ensure that the actions at each stage begin only when each block has the required data. If there is no data presented to a block, it sits idle, incurring only static power dissipation, until there is data to process. This situation results in power consumption's being directly proportional to the amount of useful work performed in a fashion that is equivalent to perfect clock gating in synchronous systems. It also means that data processing ripples through each block in a fashion that spreads out the radiation profile and lowers overall EMI.

Like synchronous circuits, asynchronous circuits can support voltage and clock-speed scaling to reduce power consumption. For synchronous systems, you often implement volt-age and clock-speed scaling in a number of discrete steps; designers must fully characterize and verify the system's operation, because changing the voltage or clock speed changes the performance and timing relations between logic blocks. For asynchronous systems, the local handshaking between logic blocks and the delay-insensitive nature of the logic and control functions simplify designers' efforts to characterize and verify the changes in performance and timing relations between logic blocks.

Faster processing using asynchronous circuits is possible, because each functional block can initiate as soon as the prerequisite actions complete, without waiting for a system clock to advance. Therefore, the system's speed depends on the average time of the functional blocks rather than being driven by the slowest functional block. An asynchronous system may, on average, be faster than a synchronous system if a small overhead is required for interblock coordination.

Asynchronous designs can be beneficial for low-power applications, but using them presents challenges. Each functional block sets its own pace, which may vary. Several concurrent actions may finish in a large number of possible sequences, and enumerating all of those sequences to verify a complex design can be difficult. Despite this difficulty, depending on the behavior of the application, asynchronous designs can be the appropriate technology for very-low-power applications.

high performance with very low power dissipation, analogous to perfect clock gating in synchronous systems (see **sidebar** "Asynchronous circuits").

Another common power-management feature is the ability to scale the clock frequency. Clock dividers and integrated low-speed clock sources, usually under software control, are two common mechanisms for dynamically scaling the clock frequency. This type of flexibility allows you to linearly affect your power consumption by slowing the computational speed when you do not need the processor to run at full speed. In some devices, the integrated low-speed clock source can support a dual-speed start-up when restarting both modules and a high-speed clock source. The core or module can begin operation using the internal, fast-starting but lower power and slower clock source, and it can transition to the slow-starting but faster clock source after the circuit becomes stable.

Dynamic voltage scaling is a power-management feature that increasing numbers of power-sensitive processor architectures are supporting. The relationship between clock frequency and supply voltage allows you to lower the supply voltage in conjunction with lowering the clock frequency. Dynamically scaling the voltage relies on software control, and it represents the most dramatic global savings in power outside partially powering down the system, because of the quadratic relationship between power dissi-

APPLICATION SOFTWARE — DPM-AWARE APPLICATION · GENERIC APPLICATION · APPLICATION IN DPM WRAPPER

MIDDLEWARE — POWER-POLICY MANAGER

SYSTEM SOFTWARE — DRIVER · DRIVER · DRIVER · OPERATING SYSTEM · DPM SUBSYSTEM

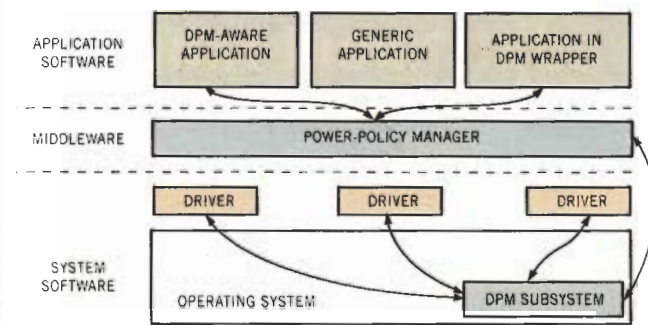NOTE: DPM=DYNAMIC POWER MANAGEMENT.

Figure 1

A power-policy manager is software that usually runs between the operating system and the application code. It can apply a customized power policy to coordinate system resources with multiple application requirements (courtesy MontaVista).

pation and supply voltage. The set of frequency and voltage pairs that a given device can support is determined during characterization; it ensures that there is sufficient margin for the processing performance under all supported operating conditions.

The sequence for dynamically selecting a higher clock frequency and the corresponding supply voltage consists of configuring the system for the new voltage, waiting for the higher supply voltage to stabilize, and then switching to the new frequency. The sequence for dynamically selecting a lower clock frequency is the same except that you need not wait for the voltage to stabilize, because the current supply voltage is already higher than necessary to support the new lower clock frequency.

OTHER FEATURES

Although the number of cycles it takes a device to process an interrupt is not a power-management feature, interrupt handling represents a sequence of actions that consumes power and time and produces no application-level work. Processing an interrupt may mean flushing the pipeline. Saving the system state may involve pushing the state data onto the stack or switching to another register set. To begin execution of the interrupt, the processor or the operating system needs to determine the interrupt vector and call the interrupt routine. After the interrupt routine is complete, you need to restore the previous system state, and you may need to again flush the pipeline.

The number of clock cycles that this sequence of housekeeping tasks requires can represent a significant source of power dissipation, especially for systems that are periodically waking on interrupt from a low-power standby mode to perform a small sequence of instructions. The system may also have to restart other portions of the system when waking from a standby mode and then again shut them down to resume the standby mode.

Properly sizing on-chip memory, including register files and caches, to your application needs can significantly affect power dissipation by minimizing expensive off-chip memory accesses. Connecting to off-chip resources, such as external memory, generally requires greater capacitance than connecting to analogous on-chip resources. The capacitance of memory is lower the closer it is to the core, so register files and caches do more than just speed data and instruction accesses; they also contribute to lower pow-

er dissipation. Cache-locking is a feature that can force a block of code to run entirely from cache to avoid external memory accesses. Including too much memory in your design may mean you are wasting power by incurring more leakage currents than necessary unless you can power down the unneeded portions of the memory.

If you partition the memory into banks and your system supports low-power modes when a bank of memory would otherwise be idle, further power savings are possible. Note that memory is idle only when it contains no useful data—a different situation from when an application is currently not accessing the memory. The optimal size and number of memory banks is application-specific; it depends, for example, on application size, data structures, and access patterns. The availability of on-chip nonvolatile memory, such as flash or EEPROM, can enable a system to employ lower power sleep modes for the memory banks if the amount of state data to save is small enough and the processing idle periods are long enough.

An approach that enables you to use lower clock frequencies is by being able to perform more work per clock cycle. Application-specific instructions, accelerators, coprocessors, or even redundant or heterogeneous cores are common mechanisms to increase instruction parallelism per clock cycle and reduce the peak clock-frequency requirement for higher system performance. The redundant or custom-hardware block may consume more power than the basic processor blocks per clock cycle, but it may be able to complete the same task in less time with a net reduction in power consumption.

SOFTWARE CONSIDERATIONS

Power-reducing features and techniques, such as using multiple clock domains, clock gating, and slower, low-leakage transistors, are independent of and transparent to software; however, you need power-aware software to harness the full potential of most hardware power-management and related features. Power-aware software can reside as part

ANALYSIS TOOLS CAN INDIRECTLY ASSIST IN POINTING OUT WHERE IN YOUR CODE YOU CAN REDUCE POWER CONSUMPTION.

of or as a combination of the BIOS, peripheral drivers, operating system, power-management middleware, and application code.

The lower the level of power-aware code, the more general it usually is. APM (Advanced Power Management) and ACPI (Advanced Configuration and Power Interface) are industry-standard specifications for operating-system-directed configuration and power management on laptops, desktops, and servers. There is no analogous industry-standard specification for embedded systems. You can employ a power-policy manager between the operating system and the application code for systems running applications over an operating system (**Figure 1**).

The power manager might employ processor usage to dynamically scale the clock frequency and supply voltage. As the processor usage decreases, the power manager decreases the clock frequency and supply voltage to reduce power. As the processor usage increases, the power manager increases the clock frequency and supply voltage in an attempt to provide adequate performance for the workload. This type of power manager may be inappropriate for hard, real-time-control systems that have tight response windows unless the power-policy customization could adequately capture those requirements.

The closer the power-aware code is to the application code, the more application-specific the decisions it can make, and the more power-efficient it can be. Few, if any, directly power-aware tools now exist for embedded-software development. Analysis tools, such as code profilers, pipeline viewers, and cache viewers are predominantly performance-analysis tools, but they can indirectly assist in pointing out where in your code you can reduce power consumption, because they can help you identify where pipeline flushes and cache misses are wasting clock cycles and, hence, power. They do not point out how restructuring your code sequence could allow you to keep specific functional blocks idle longer and with less power cycling. Incorporating power awareness into your software is now a mostly manual effort.

A primary area of effectiveness for power-aware software is in as quickly and as often as possible powering down or placing into a low-power mode all inactive resources or components, especially external devices and peripherals. The availability and granularity of power-down and low-power modes that the target hardware supports limits or expands the potential effectiveness of any power-aware software.

Interrupt-driven event management is more efficient than polling loops, so use them instead when you can. If the processor is truly idle between interrupt events, shut it down and wake it with an interrupt. Additionally, explore whether changing your algorithm or reordering the execution order of loop iterations allows you to adjust the lengths of idle periods so you can take better advantage of power-management modes. An example of this approach includes a modification that would allow a memory bank to go into a low-power mode more frequently and for longer periods.

Another power-management philosophy to consider is running the processor as fast as it can go and shutting everything off when there is nothing more to do. A contrasting philosophy for managing power is to scale the clock frequency and supply voltage to operate the processor at the slowest frequency and lowest voltage that meet the performance requirements. The merit of each approach is application-dependent and should be determined on a case-by-case basis.

Memory is a performance and power bottleneck, so minimize the total memory your system needs. Consider using hardware-supported code-compression features to minimize the total amount of memory you need. Examine your instruction sequence to avoid needlessly moving data across buses and between internal and external memory, because those actions can consume significant power without adding value.

PUTTING IT TOGETHER

Your choice of processor architecture is a high-level-design decision that affects your ability to both directly and indirectly wring out the power and cost in your design with software power-reduction techniques. Each class of processor architecture implements instruction-execution units, data-flow structures, integrated peripherals, and power-management mechanisms in a way that is appropriate for different types of application behavior.

General-purpose-microprocessor-design decisions stress software flexibility and performance. Microprocessors often employ off-chip memories and peripheral chip sets for flexibility, and they rely on speculative execution and branch prediction to improve performance. However, they do so at the expense of power consumption and cost. Microcontroller architectures, targeting real-time control processing, focus on balancing performance, power, and cost by offering many integrated peripheral and on-chip memory options within a device family. Microcontrollers rely on fast interrupt handling to rapidly respond to non-deterministic events. DSP architectures target processing for streaming data by using specialized execution units and data-flow structures to optimize application-specific signal-processing functions. Unlike microcontrollers, DSPs generally stress data movement and loop efficiency over context-switching speed.

Choosing the processor architecture with the integrated peripherals, memory, and power-management features that best suit your application algorithms, behavior, and requirements should enable you to better optimize instruction sequences, data, and memory-access patterns; off-chip activity; and functional-unit-usage patterns that result in lower power consumption and EMI. Designing from the beginning with attention to power reduction may reduce your overall system size and cost by enabling you to reduce the power-supply requirements, increase component density, decrease the parts count, and reduce the bill-of-material costs.□

> **DESIGNING FROM THE BEGINNING WITH ATTENTION TO POWER REDUCTION MAY REDUCE YOUR OVERALL SYSTEM SIZE AND COST.**

You can reach Technical Editor Robert Cravotta at 1-661-296-5096, fax 1-661-296-1087, e-mail rcravotta@edn.com.