

Extreme low-power design

MODERN IMPLANTABLE MEDICAL DEVICES, SUCH AS PACEMAKERS AND CARDIOVERTER DEFIBRILLATORS, USE AGGRESSIVE POWER-MANAGEMENT TECHNIQUES TO REDUCE SOC POWER.

A typical modern pacemaker may consume on average only a few microamperes of current to achieve long battery life. To meet these low power requirements, engineers use many techniques that may be applicable today to other power-conscious designs. The techniques vary from analog to digital and from circuit to system level, all of which are necessary to keep power to such a minimum.

The first implantable pacemaker in 1959 consisted of a two-transistor blocking-oscillator circuit (Reference 1). The 1-Hz oscillator would produce a 2-msec pulse of about 5V that the device would apply directly to the heart with electrodes. A modern pacemaker or ICD (implantable cardioverter defibrillator) has millions of transistors, thanks to VLSI. Physicians can configure hundreds of parameters to meet patients' needs. A pacemaker senses the heart's electrical activity and makes decisions based on its characteristics. It can then deliver appropriate therapy based on programmable values that the physician sets.

Figure 1 depicts a modern pacemaker. The analog chip interfaces with electrodes that go to the heart. It amplifies electrical activity and converts it to digital form. Charge pumps deliver to the electrodes a pacing pulse of varying amplitude and shape. The digital SOC (system on chip) has ROM and RAM for program code and waveform storage. The device requires a large amount of RAM to produce electrocardiograms that a physician can review. The telemetry module allows wireless communication between the implanted device and an external programmer that a physician uses to set operating parameters. A typical pacemaker battery is a 2.8V LiI (lithium-iodine) primary cell.

POWER AND ENERGY

This article concentrates on the power that the digital SOC consumes. Logic circuits can consume power from three sources:

$$P_{TOTAL} = P_{DYNAMIC} + P_{SHORT-CIRCUIT} + P_{STATIC} \quad (1)$$

Each source requires examination to understand how it contributes to the total power. The dynamic component is the switching power, which is the power the device loses when the circuit capacitances charge and discharge. Typically, this contributor is the largest of the three. You can express the dynamic power by:

$$P_{DYNAMIC} = \frac{\alpha CV^2 f_{CLK}}{2}, \quad (2)$$

where α is the activity factor, which is the number of transitions in one clock period and has a value of 0 to 2, and C is the capacitance whose charge is switching at a frequency that the clock frequency determines. For common synchronous circuits, the activity factor is 1, because all transitions occur on only one edge of the clock. All four factors are important, and this article discusses techniques to reduce the effects of each.

Short-circuit power is due to turning on both the NMOS and PMOS elements within a logic gate simultaneously during the switching process. This current is sometimes known as crossover or crowbar current. The slow rise and fall times at the input of logic gates and small load capacitances can exaggerate this condition (Reference 2). The good news here is that crossover current is one of the easiest sources of power consumption to control. Typically, this source is less than 10% of the total power dissipated (Reference 3). Because this value is so low, this article will not dwell on this source. Instead, it will look at two methods of effectively minimizing crossover current.

Proper gate sizing can minimize short-circuit current. Ensuring that the input rise and fall times are shorter than the output rise and fall times significantly reduces short-circuit current. You can constrain the input-versus-output rise and fall times within the cell-characterization data of the logic cells. It is important to meet these constraints during logic synthesis and at

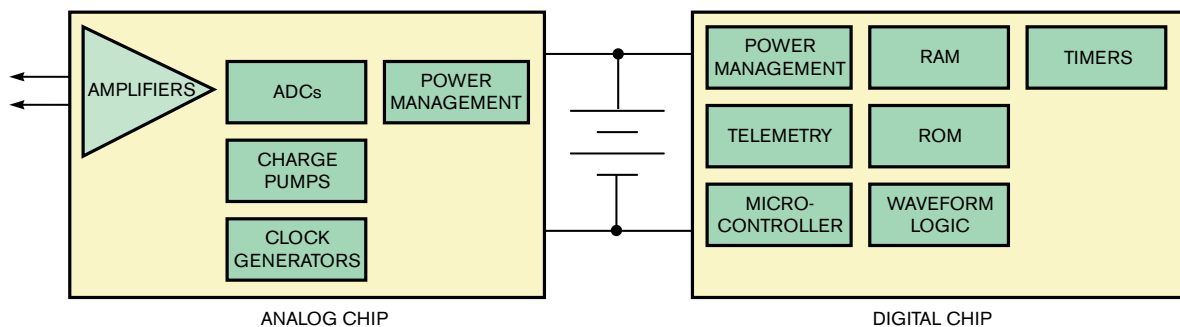


Figure 1 A modern pacemaker comprises two chips to separate analog and digital functions.

the in-place-optimization stage during place and route. With proper cell characterization and design-rule checking, you can keep short-circuit current under control. As you lower the supply voltage to close to the sum of the NMOS and PMOS threshold voltage, short-circuit current is minimal, because the NMOS and the PMOS transistors cannot be simultaneously on.

The circuit consumes static or leakage power while the clocks are stopped. In older CMOS technologies, this power was negligibly small, but, in modern technologies, you can no longer neglect this value. Experts estimate that leakage current increases by a factor of 10 for each process generation (Reference 4). For many designs, static power is a significant contributor to power consumption. For modern processes, the main source of this power is subthreshold leakage in the transistor, which is the drain-to-source current flow while the transistor is off.

Power reduction can occur at each design stage, bringing varying advantages. The greatest benefit comes at the system level. The least benefit is at the circuit level. You can view these results as a pyramid with power reduction at various stages of design (Figure 2). The exact power reduction at each level varies from design to design. As a guideline, the system-level and architecture-level reduction can be orders of magnitude larger than any other level, with the system level offering slightly more benefit. You can achieve power reduction of 10 to 90% at the logic level, and you can expect a reduction of 15 to 20% at the circuit level (Reference 5). There are plenty of exceptions to this simplification of power reduction. For example, at the circuit level, varying the thickness of the gate oxide that determines the threshold voltage of the transistor can profoundly affect leakage current. But, in general, this template provides a good way to start thinking about power reduction in an SOC.

One of the most fundamental concepts of conserving power is to shut it off when you don't need it. Figure 3 depicts this concept: power gating. Power gating is the limit of voltage

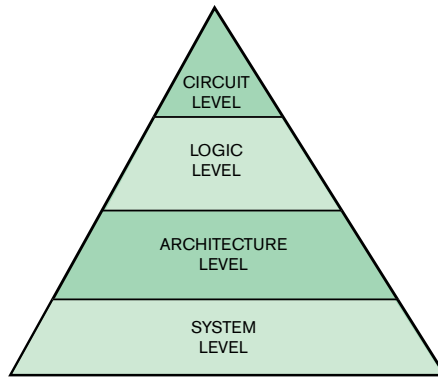


Figure 2 Each stage of the design phase offers varying degrees of power-reduction benefits.

scaling in which you scale the voltage to zero, thereby reducing the total power to zero. Retaining data is an issue in many cases, so, a typical chip cannot completely lose its voltage supply. But you can shut off sections of the chip when it's not in use. Configuration registers and data-retention flip-flops typically are on a separate uninterrupted power source.

A close examination of Equation 2 shows that power scales quadratically as a function of supply voltage; hence, scaling the voltage can greatly impact dynamic power. This concept forms the basis of voltage scaling. From a power-conservation standpoint, running a circuit at the lowest voltage

possible is ideal. The lower limit is generally the sum of the NMOS and the PMOS threshold voltages with some noise margin. For this reason, lower geometry IC technology can further lower the voltage. This process is technology-driven voltage scaling. Unfortunately, transistor speed decreases with a decrease in voltage. So you base the final decision of where to set the voltage on circuit performance, reliability, and power consumption. An adjustable on-chip voltage regulator allows you to either statically (based on testing or characterization) or dynamically (based on circuit demand) adjust this voltage.

HARDWARE OR SOFTWARE

High-speed signal-processing applications often implement custom logic to meet the timing requirements. The same is true for very-low-power implementations, often leading to the question of whether to implement a function or an algorithm in hardware or software. There are many trade-offs, such as time, cost of development, configurability, and area, with either approach. For many cases, this question involves whether to choose a serial or a parallel implementation. For a software algorithm, you implement the function in a general-purpose microcontroller unit or a DSP, which involves instruction decoders, instruction fetching, and data fetching to execute the desired function. All operations operate at a higher clock rate (the clock frequency in Equation 2) than needed for the actual processing, because a serial process of events must happen before and after the actual computation. A clock can run at a lower rate in a parallel-custom-logic implementation of the same function. In many cases, this rate can be the actual sample rate of the data. This process can lead to a larger but lower power design.

A good example of the hardware-versus-software trade-off is the implementation of a CRC (cyclic-redundancy check). Because software requires so much bit manipulation, this computation can be intense, requiring many clock cycles. In hardware, you can implement a CRC with a simple combination of standard gates and flip-flops. Reference 6 notes a 53-times reduction in clock cycles with a hardware implementation of a CRC compared with an efficient software implementation of the same function. This reduction, too, would directly translate to a huge reduction of power.

You cannot ignore the impact of software implementation.

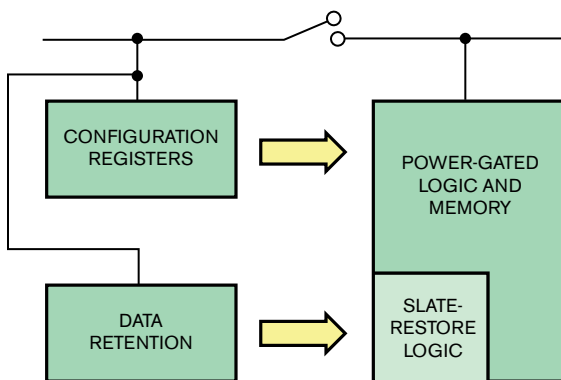


Figure 3 Power gating simply shuts off input voltage when the system does not need it to conserve power.

The efficiency at which the system executes a function can largely influence power. Executing a function in as few clock cycles as possible often leads to assembly coding for the best possible results. However, this process is time-consuming and creates processor-specific code.

Accesses to memories can be especially costly. Likewise, a function that relies heavily on instruction fetches, data loads, and data stores can be costly. Code profiling can help analyze and estimate the amount of memory-access activity. As a simple example of the effect of software coding on memory access, consider **Listing 1** (Reference 7).

For this example, the code performs $f(x)$ and $g(x)$ on data values $A[i]$ and $B[i]$. For the example on the left, the processor stores intermediate value B in memory. For the example on the right, the processor does not store B in memory. Aside from being a more concise coding style, the example on the right requires 100 fewer memory-data stores and 100 fewer data loads. Note that coding style as well as compiler transformation can create this kind of inefficiency.

RESOURCE SPLITTING

Reading and writing to system memories can be a large source of SOC-power consumption. Partitioning is a good approach when it comes to memories. This example shows the trade-off of power for area, because a partitioned memory is larger than a nonpartitioned one. Large memories have long word and bit lines, creating large capacitances. Partitioning

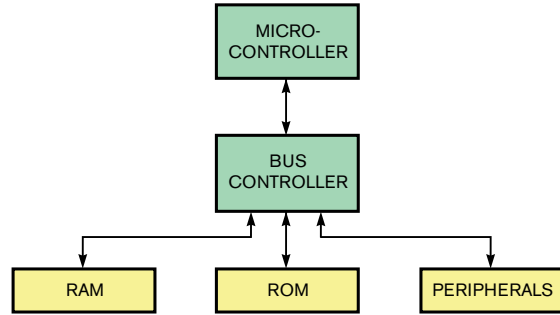


Figure 4 You can split buses to eliminate large capacitances and reduce switching power.

a memory into smaller instances can be beneficial because a smaller memory consumes less power when something accesses it. Understanding this trade-off can help you decide whether to implement a memory in one or multiple instances. Not all memories are the same. Some memories reduce unnecessary toggling by using a memory cache. Many processors have high rates of associativity with program code. Given this situation, a memory cache can offer power-reduction benefits.

A similar argument holds for multiple buses in a system. One large bus can have a large fan-out and large capacitances that toggle with bus activity. Consider splitting buses to mitigate unwarranted bus switching (**Figure 4**). For example, having separate buses for RAM, ROM, and peripherals reduces the expense of toggling all the devices when communicating to one of them.

The general goal is to reduce unnecessary switching. One of the most effective ways of achieving it is to reduce clocks to the minimum frequency at which they need to run or to shut off clocks when not in use. This objective is the purpose of clock scaling and clock gating. From **Equation 2**, power scales linearly with frequency. Reducing the clock to the minimum it needs to run to achieve performance minimizes the power consumption. For this reason, a low-power circuit may have multiple clock domains, thus allowing circuitry to run only at the minimum frequency. If the clocks synchronize with each other, the designer need not be concerned with clock-domain crossing.

Some EDA tools can automatically insert clock gating, but you must exercise care when employing them. Gated clocks are notorious for creating problems with test insertion and static-timing analysis. Ideally, you gate the clock as close to the source as possible, thereby shutting off as many clock nets as possible. For this reason, clock gating at RTL (register-transfer level) can be the most effective method. A good flow is first to understand where the system is consuming the power in the clock nets and then to find an effective place to gate the clock and choose the appropriate signal for gating. The manual insertion

LISTING 1 CODING SAMPLES

```

for (i=0; i<100; i++) {           for (i=0; i<100; i++) {
    B[i] = f(A[i]);                B[i] = f(A[i]);
}                                   C[i] = g(B[i]);
for (i=0; i<100; i++) {           }
    C[i] = g(B[i]);
}

```

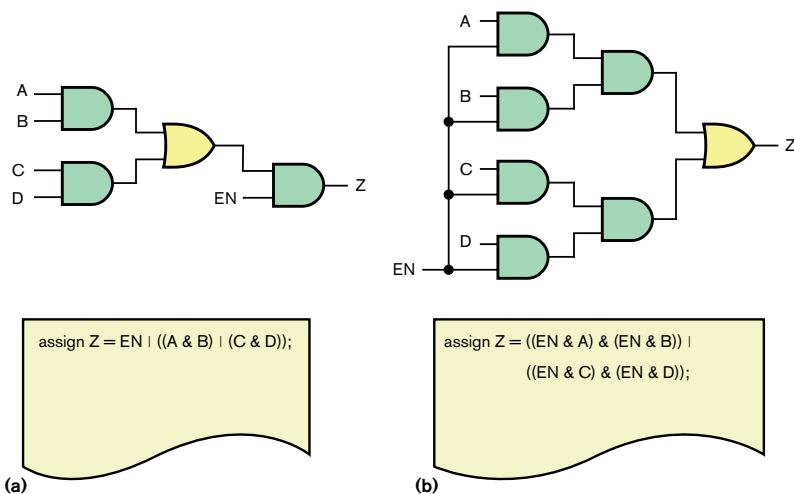


Figure 5 Toggling on operands A, B, C, and D can propagate through two layers of logic before operand EN gates it (a). The logic operand gates the input operands (b).

of clock gating is more time-consuming but often leads to better results with fewer gated-clock nets.

Excessive signal toggling is not limited to clock nets. Combinatorial paths can create unnecessary toggling. Consider the following implementations of the same function in **Figure 5**. With the implementation in **Figure 5a**, toggling on operands A, B, C, and D can propagate through two layers of logic before operand EN gates it. With the implementation in **Figure 5b**, the input operand, EN, gates the input operands. Logic synthesis may optimize a circuit according to its constraints. Therefore, the actual logic implementation from a given RTL model may vary. It is always a good approach to evaluate the results after each logic transformation. Also note that some EDA tools during logic synthesis isolate operands either automatically or through RTL-pragma directives (**Reference 8**).

Lesser known techniques have a highly application-specific benefit. For example, sign-magnitude data representation has fewer lines toggling near the zero value than the two's complement (**Reference 9**). Gray coding and bus-inversion encodings may result in fewer lines transitioning on high-capacitance nets, such as address and data buses (**references 10 and 11**). During physical implementation, the goal is to minimize wire length, because it is directly proportional to capacitance. Keeping clock nets as localized as possible with endpoint registers as close as possible to the clock source or gating point is desirable. Good IC floorplanning and a hierarchical place-and-route approach lead to better results.

As noted, threshold voltage can profoundly affect leakage

power; the higher the threshold voltage, the lower the leakage. But when performance dictates, the system may require lower-threshold-voltage transistors. As a result, a design can use a mixture of voltage libraries. For paths that easily meet timing constraints, you can use the higher threshold-voltage cells. For timing-critical paths, you can use the lower threshold-voltage cells. Biasing the substrate of the transistor can effectively alter the threshold voltage. In some cases, this technique has produced a 100-times reduction in leakage current (**Reference 12**).

You can use any or all of these techniques to reduce the power of an SOC, such as a pacemaker. Think about all the power sources of **Equation 1** and understand how to control them. You can achieve power reduction at all phases of the design cycle, but you'll see the largest benefit at the system level, so make sure you think about power early. **EDN**

REFERENCES

- 1 Greatbatch, W, *The Making of the Pacemaker*, Prometheus Books, Amherst, NY, 2000.
- 2 Veendrick, HJM, "Short-Circuit Dissipation of Static CMOS Circuitry and Its Impact on the Design of Buffer Circuits," *IEEE Journal of Solid-State Circuits*, Volume SC-19, pg 468, 1984.
- 3 Chandrakasan, AP, and RW Brodersen, *Low Power Digital CMOS Design*, Kluwer Academic Publishers, Norwell, MA, 1996.
- 4 Semiconductor Industry Association, "International Roadmap for Semiconductors 2005 Edition Executive Summary," Austin, TX, International Sematech, 2005, www.itrs.net.
- 5 Keutzer, K, and P Vanbekbergen, "The Impact of CAD on the Design of Low Power Digital Circuits," 1994 IEEE Symposium on Low Power Electronics, pg 42.
- 6 Simsic, L, "Accelerating Algorithms in Hardware," *Embedded System Design*, Volume 17, No. 2, February 2004.
- 7 Catthoor F, et al, "Global Communication and Memory Optimizing Transformations for Low Power Signal Processing Systems," Proceedings IEEE Workshop on VLSI Signal Processing, pg 178, 1994.
- 8 "Power Compiler User Guide," Synopsys Inc, 2004.
- 9 Chandrakasan, AP, "Ultra Low Power Digital Signal Processing," Proceedings of Ninth International Conference on VLSI Design, pg 352, 1996.
- 10 Su, C, C Tsui, and A Despain, "Low-Power Architecture Design and Compilation Techniques for High-Performance Processors," pg 489, Compcon, 1994.
- 11 Stan, M, and W Burleson, "Limited-weight Codes for Low-power I/O," 1994 International Workshop on Low-power Design, pg 209, April 1994.
- 12 Imai, K, et al, "Device Technology for Body Biasing Scheme," IEEE International Symposium on Circuits and Systems, pg 13, May 2005.

AUTHOR'S BIOGRAPHY

Dean P Andersen is a senior principal engineer at St Jude Medical Inc (Sunnyvale, CA). He holds a bachelor's degree in electrical engineering from San Jose State University and a master's degree in electrical engineering from Stanford University. Andersen's interests are low-power signal processing and embedded systems for medical devices. He enjoys spending time with his kids and biking in the hills around the Silicon Valley.