Chapter 8
Power and Energy Awareness

Overview The increasing growth of energy-aware and power-aware computing is driven by the following concerns:

- The widespread use of mobile battery-powered devices, where the available *time-for-use* depends on the power consumption of the device
- The power dissipation within a large system-on-chip that leads to high internal temperatures and hot spots that have a negative impact on the chip's reliability, possibly physically destroying the chip
- The high cost of the energy for the operation and cooling of large data centers, and finally
- The general concern about the carbon emissions of the ICT industry, which is of about the same magnitude as the carbon emissions of the air-transport industry.

In the past, the number of instructions executed by a computer system in a unit of time (e.g., *MIPS* or *FLOPS*) was the important indicator for measuring the performance of a system. In the future, the number of instructions executed per unit of energy (e.g., a joule) will be of equal importance.

It is the objective of this chapter to establish a framework for developing an understanding for energy-efficient embedded computing. In the first section we introduce simple models to estimate the energy consumption of different tasks. This gives the reader an indication of where energy is dissipated and what are the mechanisms of energy dissipation. Since energy consumption depends very much on the considered technology, we assume a hypothetical 100 nm CMOS VLSI technology as the reference for the estimation. The next section focuses on hardware techniques for energy saving, followed by a discussion about the impact of system architecture decisions on the energy consumption. Software techniques that help save energy are treated in the following section. In the final section we look at the energy content and management of batteries and the topic of energy harvesting from the environment.

H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, Real-Time Systems Series, DOI 10.1007/978-1-4419-8237-7_8,

8.1 Power and Energy

The ICT sector is a major consumer of electric energy. At the time of this writing, the carbon footprint of the ICT sector is in the same order of magnitude as the carbon footprint of the air-transportation industry and growing at a rate of 10% per year.

Example: Out of 156 of GW of electric power used for ICT equipment worldwide (about 8% of the worldwide total electricity consumption) in the year 2007, about 28 GW (18%) are used for the operation of PCs, 22 GW (14%) are used for network equipment, 26 GW (17%) are used for data centers, 40 GW (25.5%) are used for TV, and 40 GW (25.5%) are used for other ICT purposes [Her09, p. 162].

8.1.1 Basic Concepts

The concept of energy, initially introduced in the field of mechanics, refers to a scalar quantity that describes the *capability of work* that can be performed by a *system*. The *intensity of the work*, that is the energy used up per unit of time, is called *power*. Energy is thus the integral of power over time. There exist different forms of energy, such as *potential energy*, *kinetic energy*, *thermal energy (heat)*, *electrical energy*, *chemical energy*, and *radiation energy*.

The *first law of thermodynamics*, relating to the *conservation of energy*, states that in a closed system, the total sum of all forms of energy is constant. The transformation of one form of energy to another form of energy is governed by the *second law of thermodynamics* that states that thermal energy can only be partially transformed to other forms of energy. For example, it is *impossible* to convert thermal energy to electrical energy with an efficiency of 100%, while the converse, the transformation of electrical energy to thermal energy can be performed with 100% efficiency.

There exist many different units to measure the amount of energy. We will use the *joule*, which is the unit of energy in the *mks* (*meter-kg-second*) system. One *joule* (*J*) is defined as the amount of work done by a force of one *newton* moving a mass of one kg for a distance of one meter. At the same time a *joule* is the amount of energy that is dissipated by the power of one *electric watt* lasting for *one second*. Thermal energy, i.e., heat, is often measured in *calories or BTUs* (*British Thermal Units*). One calorie is defined as the heat needed to increase the temperature of one gram of water at room temperature by one degree *celsius*. One *calorie* corresponds to about 4.184 *joule*, while one *BTU* corresponds to about 1,055 *joule*.

Example: A gram of gasoline contains chemical energy in the amount of about 44 kJ. An automotive engine converts about one third to one fourth of this chemical energy to mechanical energy, i.e., gram of gasoline provides about 12 kJ of mechanical energy. The rest of the chemical energy is converted to heat. The mechanical energy of 12 kJ is sufficient to lift a car with a mass of 1,000 kg (or 1 t) about 1.2 m (the potential energy is

mgh, where g is the gravitational acceleration of 9.8 m/s^2 and h the relative altitude in *meters*) or to accelerate the car from zero m/s to a speed of 5 m/s (or 18 km/h), considering that the kinetic energy of the car is mv^2/2. If we stop the car by braking, this kinetic energy is converted into heat by the brakes and tires. In an electric car, a substantial part of this kinetic energy can be transformed back into electrical energy that can be stored in a battery for further use.

A battery is a storage device for electric energy. The voltage that exists at the terminals of a battery can drive electric current through a resistor. In the resistor, electric energy is converted to heat. If an electric current I flows through a wire with a *resistance* R, the voltage drop on the wire will be, according to Ohm's law

$$U = IR$$

and the dissipated electric power is given by

$$W = UI$$

where W denotes the dissipated electric power (in *watt*), I the current (in *ampere*), and R the resistance (in *ohm*). If a constant voltage of U is applied to a system with a resistance of R over a time of t s, the energy that is dissipated equals

$$E = tU^2/R$$

Where E is the dissipated energy in *joule*, t denotes the time in *seconds* and U and R denote the voltage and the resistance, respectively.

8.1.2 Energy Estimation

The energy that is needed by a computing device to execute a program can be expressed as the sum of the following four terms:

$$E_{total} = E_{comp} + E_{mem} + E_{comm} + E_{IO}$$

where E_{total} is the total energy needed, E_{comp} denotes the energy needed to perform the computations in a CMOS VLSI circuit, E_{mem} denotes the energy consumed by the memory subsystem, E_{comm} denotes the energy required for the communication, and E_{IO} is the energy consumed by I/O devices, such as a screen. In the following, we investigate each of these terms in some detail by presenting simplified energy models. The numerical values of the parameters depend strongly on the technology that is used. We give some approximate value ranges for the parameters for a *hypothetical* 100 nm *CMOS* technology which is operated with a supply voltage of 1 V in order to enable the reader to develop an understanding for the relative importance of each one of the terms.

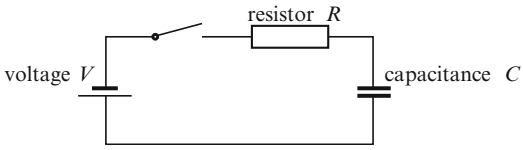

Fig. 8.1 First-order RC network

Computation Energy. The energy E_{comp} required for the computations consists of two parts: (1) the dynamic energy $E_{dynamic}$ that denotes the energy needed to perform the switching functions in a CMOS VLSI circuit and (2) the static energy E_{static} that is dissipated due to a leakage current drawn from the power supply irrespective of any switching activity.

The dynamic energy consumption of a switching action of a transistor can be modeled by a first-order RC network as shown in Fig. 8.1. In such a network, consisting of a power supply with voltage V (measured in volt), a capacitor C (measured in farad), and a resistor R (measured in ohm), the voltage on the resistor R as a function of time t (measured in seconds) after the switching event, is given by

$$V_{res}(t) = Ve^{-t/\tau}$$

Where $\tau = RC$, which has the dimension of *second*, is called the time-constant of the RC network. This time constant characterizes the speed of a switching operation.

The total energy dissipated during one switching operation is given by the integral of the product *voltage* $V(t)$ and *current* $I(t)$ through the resistor over the time period from zero to infinity.

$$E = \int_0^\infty V(t)I(t)dt = \frac{V^2}{R}\int_0^\infty e^{-2t/RC}dt = \frac{1}{2}CV^2$$

The current that flows into the capacitor builds up an electric charge in the capacitor and does not contribute to the energy dissipation.

Let us set a hypothetical effective capacitance C_{eff} such that the term $C_{eff}V^2$ covers the energy needed to switch all output transitions of an average instruction of a processor and the *short circuit energy* E_{short} that is dissipated by the brief current flow from the voltage source to the ground during the switching actions.

$$C_{eff} = C/2 + E_{short}/V^2$$

Then the dynamic energy needed for the execution of a single instruction can be expressed by the simple equation

$$E_{dynamic-instruction} = C_{eff}V^2$$

and the dynamic energy needed to run a program with N instructions is given by

$$E_{dynamic-program} = C_{eff} V^2 N$$

In the following the term *IP-core* (intellectual property core) is introduced to refer to a well-specified functional unit of a system on chip (SoC). An IP-core can be a complete processor with local scratchpad memory or some other functional unit, such as an MPEG decoder. Let us now assume that an IP-core executes on the average one instruction per clock cycle and this IP-core is driven with a frequency f. Then, the dynamic power dissipation of the IP-core equals

$$P_{dynamic} = C_{eff} V^2 f$$

Example: Assume a program of 1,000 million machine instructions that is executed on a low-power IP-core with a voltage of 1 V and where an average instruction is characterized by an effective capacitance of 1 nF. In this example we only consider the dynamic energy. The execution of one instruction requires an energy of 1 nJ. The dynamic energy that is needed for the execution of this program is then 1 J. If the IP-core is driven with a frequency of 500 MHz, then the power dissipation is 0.5 W and the program execution will take 2 s.

There are different mechanisms that contribute to a steady flow of current (the *leakage current*) between the terminals of a transistor, even if the transistor is not performing any switching action. According to [Ped06], the most important contribution to the leakage current in a technology which is below 100 nm is the *subthreshold leakage current* I_{sub}, which increases with decreasing threshold voltage and with increasing temperature. Another important effect is the *tunnel current* flowing from the gate into the channel, although these are separated by the non-conducting gate oxide. A quantum-mechanic effect causes electrons to *tunnel* through the very thin isolation, and this current grows exponentially as the oxide becomes thinner with smaller feature sizes. The exponential growth of the leakage current with rising temperature is of major concern in a submicron device with a low threshold voltage. This increase of the leakage current can lead to run-away thermal effects that, if not properly controlled, end with the thermal destruction of the device.

In submicron technologies, the energy dissipation caused by the static leakage current can approach the same order of magnitude as the energy needed for the dynamic switching actions [Ped06]. From a system perspective it is therefore advantageous to structure a hardware system in such a way that subsystems can be switched off completely if their services are not needed during short intervals. This is the topic of power gating discussed in Sect. 8.3.3.

Communication. The energy requirement for the transmission of a bit stream from a sender to a receiver is highly asymmetric. While the sender needs to generate signals that are strong enough to reach the intended receiver, the receiver must only sense the (weak) incoming signals.

The energy needed by a sender E_{tx} to transport a bit string of k bytes can be approximated by

$$E_{tx}(k) = E_o + k(E_c + 8E_{tr})$$

while the energy needed by the receiver E_{rx} can be approximated by

$$E_{rx}(k) = E_o + k(E_c)$$

where E_o is the energy needed to set up the sending or receiving of a message and E_c is the energy dissipated by the communication controller to process 1 byte in the sender's and receiver's circuitry. If we assume that the setup of a transmission requires ten instructions and the DMA processing of one bit requires the energy-equivalent for one instruction then E_o is 10 nJ and E_c is 8 nJ in our simple reference architecture. In addition to the energy required for the reception of a message, a receiver needs energy during the standby period of waiting for a message to arrive. If the *receive/wait ratio* is small, the standby energy consumption can be substantial.

The term E_{tr} denotes the energy needed to transmit a single bit. This term is different for wired and wireless communication.

For a wired interconnect, this term depends on wire length d, the effective capacitance of the interconnect per unit length $C_{effunit}$, and the square of voltage V:

$$E_{tr} = dC_{effunit}V^2$$

A typical value for the effective capacitance $C_{effunit}$ per unit length of an interconnect on a die is in the order of 1 pF/cm [Smi97, p. 858].

If we connect two IP-cores on a System-on-Chip (SoC) by a Network-on-Chip (NoC), then we have to find a value for the network capacitance C_{eff} such that $C_{eff}V^2$ denotes the average energy consumption for the sending, transmission, and reception of a message across the network. This network capacitance C_{eff} depends on the implementation technology, the size, the topology, and the control of the NoC. In our simple model, the energy needed for accessing the data from the sender's memory, for the delivery of the message in the receiver's memory, and for the transport of a message through the NoC is considered. We assume that the sending, transmission, and receiving of a 32 bytes message across the NoC requires in the order of 500 nJ.

For wireless communication, the transmit energy per bit can be approximated by

$$E_{tr} = d^2 b$$

where d is the distance between the sender and the receiver. This square relationship between distance and energy is a gross approximation and must be adapted to the energy characteristics of the concrete antenna. In many portable devices, such as mobile phones, the sender adjusts the transmit energy dynamically in order to find the most appropriate energy level for the actual distance to the receiver. A typical value for the parameter b is given by [Hei00] as 100 pJ/(bit/m^2). If we send a 32 bytes message from a sender to a receiver which is 10 m away, then we need a transmit energy in the order of about 10 nJ/bit or about 2,500 nJ/bit per message. Due to partial wave reflections, the transmit power can decrease with the fourth power of distance in terrestrial communication systems.

The asymmetry of the energy requirements of a sender and a receiver has a decisive influence on the design of communication protocols between a base station that is connected to the electric utility and an energy-constrained mobile battery-powered device.

Memory. In memory-intensive applications, the energy that is required to access the memory subsystems can be larger than the energy that is needed for the computations. The energy that is consumed by a memory subsystem consists of two terms. The first term is the product of the power that is consumed by an idle memory subsystem and the time the memory subsystem is powered up. This term depends on the type and size of the memory and is different for SRAM, DRAM and non-volatile FLASH memory. The second term consists of the product of the number of memory accesses and the energy requirement for a single memory access. In an MPSoC that consists of a number of IP-cores that are connected by a network-on-chip (NoC) three types of memory accesses can be distinguished:

- Scratchpad memory that is within an IP-core.
- Shared on-chip memory in a separate IP-core of the SoC. It is accessed via the NoC.
- Off-chip memory, e.g., a large DRAM, that is accessed via the NoC and a memory gateway to the memory chip.

The energy requirements for accessing each of these three types of memory are substantially different. Whereas the access for the instruction and data in the scratchpad memory is considered in the effective capacitance C_{eff} of an instruction, the access to the on-chip and off-chip memory requires the exchange of two messages via the NoC: a *request message* containing the address to the memory and a *response message* with the data from the memory. We make the gross estimate that a memory access for a 32 bytes block of on-chip memory requires an energy in the order of 1000 nJ and access to the off-chip memory requires about twenty times as much, i.e., 20 μJ.

I/O Devices. The energy consumed by I/O devices such as screens, sensors and actuators is application specific (e.g., size of a screen). It can be substantial.

8.1.3 Thermal Effects and Reliability

The dissipation of electric energy in a device causes a heating of the device. The temperature rise of the device depends on the amount of energy dissipated, the heat capacity of the device, and the heat flow between the device and its environment.

Example: Let us assume an MPSoC with an area of 100 mm^2 housing 32 IP-cores connected by a Network-on-Chip. One of the IP-cores executes the program of the above example. It is physically contained in a block of 1 mm × 1 mm of a 0.5 mm thick silicon die. If we assume that this block of silicon, measuring 0.5 mm^3, is thermally isolated and

no heat can be transferred to the block's environment, then the temperature of our silicon block will increase by 1°C if about 815 μJ of energy are dissipated in this block. (Silicon has density of 2.33 g/cm^3, and a specific heat of 0.7 J/g°C.) The execution of the program of the above hypothetical example – generating a heat of 1 J – would thus lead to a temperature rise of about 1,200°C and result in a *hot spot* on the die. In reality, such a temperature rise will not occur, since any temperature difference between two bodies forces a heat-flow from the warmer body to the colder body that reduces the temperature difference.

In a VLSI device, the heat flow from a hot spot to the chip's environment takes place in two steps. In a first step, the heat of the hot spot flows to the die and increases the temperature of the entire die. In a second step the heat flows from the die through the package to the environment. Taking the above example further let us assume that the complete die is thermally isolated. Then the execution of the above program, which dissipates 1 J, will lead to an increase of the die temperature by about 12°C.

The heat flow from the hot spot to the entire die and the exact temperature profile of the die can be calculated by solving the partial differential equation for heat transfer. In order to get some gross insight into this phenomenon of heat transfer from the hot spot to the die, we look at a *simple heat flow model* where a bar of a cross-section A and length l is connecting a heat source with temperature T_{source} with a heat sink with temperature T_{sink}. The stationary heat flow P_{heat} across the bar between these two bodies can then expressed by

$$P_{heat} = H_\Theta A(T_{source} - T_{sink})/l$$

where H_Θ is the heat conductivity of the bar.

Example: Let us assume a bar of silicon with a cross section of 1 mm^2 and length of 10 mm is linking the heat source with the heat sink. The thermal conductivity H_Θ of silicon is 150 W/m°C. If the temperature difference between the heat source (the hot spot) and the heat sink (the rest of the die) is 33°C then a steady heat flow of about 500 mW will develop (this is the dynamic power dissipated by the execution of the program in the above example). If the bar has a cross section of 0.25 mm^2, then the temperature difference for transporting 500 mW must be 132°C. In reality, the temperature difference will be smaller, since the hot spot is much better embedded in the substrate of the die than expressed by this simple heat-flow model.

What can we learn from this simple example? Hot spots will only develop if a substantial amount of power is dissipated in a very small area. For example, let us assume that a temporary low impedance path between the power rails of a transistor, a *latch-up* (which can be corrected by a power-cycle) develops in a small part of a circuit due to a fault caused by a neutron from ambient cosmic radiation. The current that dissipates in this path will result in a hot spot that can physically destroy the circuit. It is therefore expedient to monitor the current drawn by a device and switch off the power quickly (e.g., within less than 10 μs) if an unexpected current surge is observed.

The temperature difference that develops between the die and the environment is determined by the power dissipation in the die and the thermal conductivity

$H_{package}$ of the package. This thermal conductivity $H_{package}$ of a typical chip package is between 0.1 and 1 W/°C, depending on package geometry, size and material. Plastic packages have a significantly lower thermal conductivity than ceramic packages. The temperature difference ΔT between the environment and the die can be calculated by

$$\Delta T = P_{die}/H_{package}$$

where P_{die} is the total power dissipated in the die. If the heat flow through the package is more than 10 W, then a fan should cool the package. The introduction of fans has a number of disadvantages, such as the additional energy required to operate the fan, the noise of the fan and the reliability of the mechanical fan. If a fan fails, overheating might destroy the circuit.

A high substrate temperature has a negative effect on the reliability of a device and can cause transient and permanent failures. High substrate temperatures change the timing parameters of the transistors and the circuits. If the specified timing patterns are violated, transient and data-dependent device failures will occur.

The *Arrhenius equation* gives a gross estimate for the acceleration of the failure rate caused by an increase of the temperature of the silicon substrate:

$$AF = \exp\left(\frac{E_a}{k}\left(\frac{1}{T^{normal}} - \frac{1}{T^{high}}\right)\right)$$

where AF is the acceleration factor of the failure rate, k is the Boltzmann constant (8.617×10^{-5} eV/K), T^{normal} is the normal substrate temperature (expressed in *Kelvin*), T^{high} is the high substrate temperature (expressed in *Kelvin*), and E_a is a failure-mechanism-specific activation energy (see Table 8.1).

From this equation we can deduce that the *failure rate* of a device increases exponentially with the increase of the substrate temperature of the device.

Example: If the temperature of the substrate of a device increases from 50°C (i.e., 323K) to 100°C (i.e., 373K), and a failure mechanism with an activation energy of 0.5 eV is assumed, then the failure rate of the device will increase by a factor of about 11.

Table 8.1 Activation energy for different failure mechanisms (Adapted from [Vig10])

Failure mechanism	Activation energy E_a (eV)
Oxide defects, bulk silicon defects	0.3–0.5
Corrosion	0.45
Assembly defects	0.5–0.7
Electromigration	0.6–0.9
Mask defects/photoresist defects	0.7
Contamination	1.0
Charge injection	1.3

8.2 Hardware Power Reduction Techniques

8.2.1 Device Scaling

The most effective way to reduce the power consumption of CMOS devices is the scaling of the device parameters, i.e., making the transistors smaller [Fra01]. Table 8.2 depicts the effect of *ideal scaling* on the different parameters of a CMOS device. The scaling factor α from one micro-electronic generation to the next is normally $1/\sqrt{2}$, i.e., about 0.7, such that the area of a scaled version of a design is reduced by a factor of 2, the power requirement is reduced by a factor of 2, the speed is increased by a factor $\sqrt{2}$ and the energy needed for the execution of an instruction (the *energy performance*) is reduced by $2\sqrt{2}$. Note from Table 8.2 that *ideal device scaling* has no effect on the power density that is dissipated in a given area of the die. It follows that ideal scaling will not result in a temperature increase of the die.

Example: Let us assume that an IP-core scales down ideally by a factor of $1/\sqrt{2}$ every 2 years. At the start, the IP-core has a size of 16 mm^2 and executes 125 MIPS, consuming a power of 16 W. Eight years later, after four generations of shrinking, this IP-core has a size of 1 mm^2, executes 500 MIPS and consumes a power of 1 W. The energy needed for the execution of an instruction has been reduced by a factor of 64, while the time performance has increased by a factor of four.

Device scaling has made it possible to place up to one billion transistors on a single die. It is thus within the capabilities of the semi-conductor industry to place a complete system, including processor, memory, and input/output circuitry on a single die, resulting in a system-on-chip (SoC). Spatial and temporal closeness of subsystems that are involved in a computation leads to a significant improvement of the energy efficiency. Spatial locality reduces the effective capacitances of the switching actions, which implies lower energy needs and faster operations. Temporal locality reduces the number of cache misses. If subsystems residing on different chips are integrated on a single die, the significant amount of energy needed to exchange data and control among chips can be saved.

Example: According to Intel [Int09], the 1996 design of the first *teraflop super computer*, consisting of 10,000 Pentium Pro Processors, operated with an energy efficiency of 2MegaFlops/J or 500 nJ per instruction. Ten years later, in 2006, a *teraflop research chip*

Table 8.2 The effect of ideal device scaling on device parameters

Physical parameter	Scaling factor
Channel length, oxide thickness, wiring width	α
Electric field in device	1
Voltage	α
Capacitance	α
RC delay	α
Power dissipation	α^2
Power density	1
Time performance in MIPS	$1/\alpha$
Energy performance	$1/\alpha^3$

of Intel containing 80 IP-cores on a single die connected by a Network-on-Chip achieved an energy efficiency of 16,000 MegaFlops/J or 62 pJ/instruction. This is an increase in the energy-performance by a factor of 8,000 within 10 years. If we assume that in 10 years five generations of scaling are taking place, the increase in the energy-performance in one generation is not only factor of $2\sqrt{2}$, the value stipulated by ideal scaling, but by a factor of more than 4. This additional improvement is caused by the integration of all subsystems on a single die.

Over the last 25 years, device scaling has also had a very beneficial effect on device reliability. The failure rates of transistors have been reduced even *faster* than the increase in the number of transistors on a die, resulting in an increase in chip reliability despite the fact that many more transistors are contained in a scaled chip. The MTTF w.r.t. permanent failures of industrial state-of-the art chips is significantly lower than 100 FIT [Pau98].

Scaling cannot continue indefinitely because there are limits due to the discrete structure of matter and quantum mechanical effects, such as *electron tunneling*. The reduction in the number of dopants in a transistor increases the statistical variations. The thermal energy of noise limits the reduction of the supply voltage. If the supply voltage is higher than stipulated by *ideal scaling*, then scaling will lead to an increased thermal stress. In submicron technologies, we have reached the point where these effects cannot be neglected any more and lead to an increase in the transient failure rates of chips. Although the permanent failure rate per transistor may still decrease, this decrease is not compensating the increase in the number of transistors on the die anymore, causing an increase of the chip failure rate. The International Technology Roadmap on Semiconductors 2009 [ITR09, p. 15] summarizes these challenges in a single sentence: *The ITRS is entering a new era as the industry begins to address the theoretical limits of CMOS scaling*.

8.2.2 Low-Power Hardware Design

Over the past few years, a number of hardware-design techniques have been developed that help to reduce the power needs of VLSI circuits [Kea07]. In this section we give a very short overview of some of these techniques.

Clock Gating. In many highly integrated chips, a significant fraction of the total power is consumed in the distribution network of the execution clock. One way to reduce power in the execution clock network is to turn execution clocks off when and where they are not needed. Clock gating can reduce the power consumption of chips by 20% or more.

Transistor Sizing and Circuit Design. Energy can be saved if the transistor and circuit design is optimized with the goal to save power instead with the goal to get the optimal speed. Special sizing of transistors can help to reduce the capacitance that must be switched.

Multi-threshold Logic. With present day micro-electronic design tools it is possible build transistors with different threshold voltages on the same die. High-threshold transistors have a lower leakage current but are slower than low-threshold transistors. It is thus possible to save dynamic and static power by properly combining these two types of transistors.

8.2.3 Voltage and Frequency Scaling

It has been observed that, within limits characteristic for each technology, there is a nearly linear dependency of frequency on voltage. If the frequency for the operation of the device is reduced, the voltage can be reduced as well without disturbing the functionality of the device [Kea07]. Since the power consumption grows linearly with frequency, but with the square of the voltage, combined voltage and frequency scaling causes not only a reduction of power, but also a reduction of energy required to perform a computation.

> **Example:** The Intel XScale® processor can dynamically operate over the voltage range of 0.7–1.75 V and at a frequency range of 150–800 MHz. The highest energy consumption is 6.3 times the lowest energy consumption.

Voltage scaling can be performed in the interval $\langle V_{threshold}, V_{normal} \rangle$. Since V_{normal} is reduced as a device is scaled to lower dimensions (see Sect. 8.2.1), the range that is available for voltage scaling in sub-micron devices is reduced and voltage scaling becomes less effective.

The additional circuitry that is needed to perform software-controlled dynamic voltage and frequency scaling is substantial. In order to reduce this circuitry, some designs support only two operating modes: a *high-performance operating mode* that maximizes performance and an *energy-efficient operating mode* that maximizes energy efficiency. The switchover between these two modes can be controlled by software. For example, a laptop can run in the high-performance operating mode if it is connected to the power grid and in the energy-efficient operating mode if it runs on battery power.

Given that the hardware supports voltage and frequency scaling, the operating system can integrate power management with real-time scheduling of time-critical tasks to optimize the overall energy consumption. If the *Worst-Case Execution Time* (the WCET) of a task is known on a processor running at a given frequency and the task has some slack until it must finish, then the frequency and voltage can be reduced to let the task complete just in time and save energy. This integrated real-time and power-management scheduling has to be supported at the level of the operating system.

8.2.4 Sub-threshold Logic

There is an increasing number of applications where ultra-low power consumption with reduced computational demands is desired. Take the example of the billions of standby circuits in electronic devices (e.g., television sets) that are continuously draining power while waiting for a significant event to occur (e.g., a start command from a remote console or a significant event in a sensor network). The technique of sub-threshold logic uses the (normally unwanted) sub-threshold leakage current of a sub-micron device to encode logic functionality. This novel technique has the potential to design low time-performance devices with a very low power requirement [Soe01].

8.3 System Architecture

Next to device scaling, the following system architecture techniques are most effective in reducing the energy requirement significantly.

8.3.1 Technology-Agnostic Design

At a high level of abstraction, an application requirement can be expressed by a platform-independent model (PIM) (see also Sect. 4.4). A PIM describes the functional and temporal properties of the requested solution without making any reference to the concrete hardware implementation. For example, when we specify the functionality and timing of the braking system of a car, we demand that the proper braking action will start within 2 ms after stepping on the brake pedal. We say that such a high-level description of an application is *technology agnostic*. A PIM can be expressed in a procedural language, e.g., System C, augmented by the required timing information, e.g., by UML MARTE [OMG08]. The system implementer has then the freedom to select the implementation technology that is most appropriate for his/her purpose.

In a second step, the PIM must be transformed into a representation that can be executed on the selected target hardware, resulting in the platform-specific model (PSM). The target hardware can be either a specific CPU with memory, a Field Programmable Gate Array (FPGA), or a dedicated Application Specific Integrated Circuit (ASIC). Although the functional and temporal requirements of the PIM are satisfied by all of these implementation choices, they differ significantly in their non-functional properties, such as energy requirements, silicon real-estate, or reliability. Figure 8.2 gives a gross indication of the energy required to execute a given computation in the three mentioned technologies. CPU-based computations have a built-in power overhead for instruction fetch and decoding that is not present in hardwired logic.

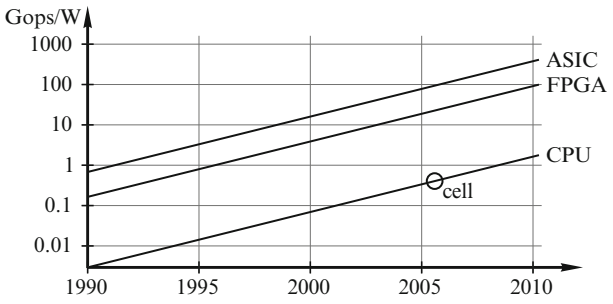

Fig. 8.2 Power requirement of different implementation technologies (Adapted from [Lau06, slide 7])

The *technology-agnostic design* makes it possible to change the target hardware of a single component, e.g., replace a CPU-based component by an ASIC without having to revalidate the complete system. Such an implementation flexibility is of particular importance for battery-operated mass-market devices where the initial test version of the functionality of a component can be realized and tested on a CPU-based implementation and later transferred to an ASIC for the mass-market production.

The *technology-agnostic design* makes it possible to address the *technology-obsolescence problem* as well. In long-lived applications, such as the control system on an airplane, the services of the control system must be provided for a long time-span, e.g., 50 years. During this time-span, the original hardware technology becomes outdated. Technology-agnostic design makes it possible to change the hardware and the related transformation of the PIM to the PSM, without having to change the interfaces to the other subsystems.

8.3.2 Pollack's Rule

Over the past 20 years we have seen a tremendous performance increase of single processor systems. New architectural mechanisms, such as pipelining, out-of-order execution, speculative branching, and many levels of caching have made it possible to significantly reduce the execution time needed for a sequential program without having to invest in alternative system and software architectures that support a highly parallel execution environment. However, this performance increase of the sequential processor has its (energy) price. Fred Pollack from Intel looked at the integer performance increase of a new micro-architecture against area and power of the previous micro-architecture, implemented in the same process technology [Bor07]. Pollack found, that over a number of Intel architectures, starting with the i386 in 1986, the performance of every subsequent micro-architecture increased only with the square root of the power or silicon area. This relationship is normally referred to as *Pollack's Rule*.

Embedded systems are characterized by an *application-inherent parallelism*, i.e., they consist of many concurrent, nearly independent processes. In order to establish a viable software execution environment for these nearly independent parallel processes, a complex operating system that provides spatial and temporal partitioning must be implemented on top of a sequential processor. From the energy perspective, this is yet another setback. At first, energy is wasted by the execution of the powerful sequential machine, and then energy is wasted again to provide the encapsulated parallel execution environments for the support of the parallel processes running on this sequential machine.

Example: According to Pollack's rule, the speed improvement of an IP-core achieved by advanced micro-architectural mechanisms scales by the square root of two, while the required energy and the silicon area increase by factor of 2. After 4 generations of micro-architecture evolutions, an IP-core would have grown 16 times its original size, would consume 16 times as much energy as the original, and achieve a time-performance improvement of four. The micro-architecture evolution has degraded the energy efficiency by 400%.

The recent introduction of multi-core systems on chip (MPSoC), where simple IP-cores are connected by a network-on-chip (NoC), will thus revolutionize the execution environment for embedded systems. Considering the size of the embedded system market, it can be expected that in the future energy-efficient multi-core systems that focus on this market will become dominant. The potential for energy savings of these systems is significant.

An important issue in the design of MPSoCs is related to the structure of the interconnect among the IP-cores. There are basically two alternatives (1) a message-based communication infrastructure and (2) the provision of a large shared memory. Poletti et al. [Pol07] have investigated the energy efficiency of these two alternatives and come to the conclusion that message-based systems are preferable if there is a high computation/communication ratio while shared memory outperforms message passing if the computation/communication ratio is low. The comparative analysis of memory models for chip multiprocessors comes to a similar conclusion [Lev08]. In many industrial embedded systems the high computation/communication ratio suggests that message passing is the preferred alternative.

Example: In a premium car of today one can find up to one hundred *Electronic Control Units* (ECU) that are connected by a number of low-bandwidth CAN busses. Aggregating some of these ECUs on a single die of an MPSoC will put very little load on the high-bandwidth NoC.

Message passing has many other advantageous properties over shared memory, such as function encapsulation, fault containment, error containment, the support of *implementation-agnostic design methods*, and the support of *power gating*.

8.3.3 Power Gating

In a multi-core SoC, consisting of a set of heterogeneous IP-cores interconnected by a real-time network-on-chip, a well-defined application functionality can be implemented in a dedicated IP-core, i.e., a component (see Chap. 3). Examples for such an application functionality are resource management, security, MPEG processing, input–output controllers, external memory manager, etc. If the components interact with each other via message passing only and do not access a shared memory, then it is possible to encapsulate a component physically and logically in a small area of silicon that can be powered down when the services of the component are not needed, thus saving the dynamic and static power of the component. Normally, the *state* (see Sect. 4.2) that is contained in the component will be lost on power-down. It is therefore expedient to select a power/down power/up point when the state of the component is empty. Otherwise, the state of the component must be saved. There are two ways of saving the state, either by hardware techniques or by sending a message containing the state to another component such that this other component can save and update the state.

The hardware effort for saving the state transparently can be substantial [Kea07]. On the other side, a distributed architecture that supports robustness must support the dynamic restart of components in case a component has failed due to a transient fault

(see Chap. 7). This restart must be performed with a temporally accurate component state. This software-based state restoration mechanism can be used to support the state restoration required by power gating as well without any additional overhead.

In an MPSoC architecture that consists of a plurality of components that are connected by a network on chip (NoC), power gating is a very effective technique to save power. A component that is not in use can be shut down completely, thus not only saving dynamic power but also the static power. Since static power is increasing substantially as we deploy below 100 nm technology, power gating becomes an extremely important power-saving technology.

In many devices it is useful to distinguish between two major modes of operation: *service mode* and *sleep mode*. In service mode, the full set of device services is provided and the dynamic power dissipation is dominant. In sleep mode, only the minimal functionality for activating the device upon arrival of a wake-up signal is required. In sleep mode the static (leakage) power is of major concern. Power gating can be very effective in reducing the power demand in sleep mode. Alternatively, the sleep mode can be implemented in a completely different technology, e.g., in subthreshold logic, that starts up the service mode as soon as a relevant wake-up signal is recognized. In this case, all components that are involved in the service mode can be shut down completely while in sleep mode, thus not consuming any power at all.

8.3.4 Real Time Versus Execution Time

It is important to stress the fundamental difference between *real-time* and *execution time* in a distributed real-time system. There is no close relation between these two time bases (See also Sect. 4.1.3 on Temporal Control, which is related to real-time and Logical Control, which is related to execution time). In an MPSoC, the granularity of the real-time will be one or two orders of magnitude larger (and correspondingly, the frequency lower) than the granularity of the execution time at the chip level. Since the power consumption is proportional to the frequency – see Sect. 8.1.2 – the global *real-time clock distribution network* will only consume a small fraction of the power that would be needed for a global *execution time clock distribution network*. Establishing a single global real-time base for the whole MPSoC, but many local asynchronous execution time bases, one in each IP core of an MPSoC, can itself save a significant amount of energy and furthermore increase the energy savings potentials of a chip, as explained in the following paragraphs.

The *real-time base* makes the nodes of a distributed system aware of the progression of real-time and provides the basis for the generation of temporal control signals (see also Sect. 4.1.3). The local real-time clocks should be incremented according to the international standard of time TIA. If no external clock synchronization is available, real-time is established by a real-time reference clock that forms the source of the distributed real-time base. The granularity of the global real-time depends on the precision of the clock synchronization and will be different at different integration levels. For example, at the chip level, where the IP-cores of a SoC communicate via a NoC, the local real-time clocks in the IP-cores

will have a better precision (and consequently a smaller granularity) than the global real time at the device level, where devices communicate via a local area network (see Chap. 4 on clock synchronization). At the chip level, the establishment of the global real-time can be realized by a stand-alone real-time clock distribution network or it can be integrated into the NoC.

The *execution time base* drives the algorithmic computations of the nodes and thus determines the speed of the computation (logical control – see Sect. 4.1.3). In a large SoC, the energy dissipation of a global execution time clocking system of a SoC can form a large part of the chip's energy consumption. In addition, the location-dependent delay of the central high-frequency timing signals results in a clock skew that is difficult to control. Furthermore, the individual control of the voltage and frequency of an IP-core is not possible if all IP-cores operate with the same clock signal. It makes therefore sense to design each IP-core and the NoC as an *island of synchronicity* that generates its clocking signal for the execution time locally. If the voltage of an IP-core can also be controlled locally, then the IP-core is an encapsulated subsystem with the capability for local voltage-frequency scaling and power gating. In the architecture model outlined in Chap. 4, clock-domain crossing occurs in the message-interface between an IP-core and the NoC, which must be carefully designed in order to avoid meta-stability problems.

8.4 Software Techniques

The equation $E = C_{eff} V^2 N$ of Sect. 8.1.2 gives the dynamic energy E required for the execution of a program. There are three parameters in this equation, the effective capacitance C_{eff}, the supply voltage V, and the number of instructions N. Reducing the effective capacitance C_{eff} and reducing the number of instructions per task reduces the time needed to complete a computational task. There is thus no inherent conflict at the software level between designing for energy-performance and designing for time-performance.

The voltage depends primarily on the hardware technology and can be controlled by software if dynamic voltage and frequency scaling is supported by the hardware. The effective capacitance C_{eff} can be reduced by *spatial and temporal locality,* particularly in the memory system. The number of instructions, the *instruction count N*, needed to achieve the intended result depends entirely of the software. The instruction count is the sum of the instructions executed by the *system software* and the *application software*.

8.4.1 System Software

System software consists of the operating system and the middleware. The objectives of a flexible system-software infrastructure versus minimal energy consumption drive the design process in different directions. Many operating systems of the past have considered flexibility as the key design driver, ignoring the topic of

energy-performance. In these systems, a long sequence of system-software instructions must be executed in order to finalize a single application command, such as the sending of a message.

In battery-operated embedded systems, the energy efficiency of the system software can be improved by off-line tailoring of the operating system functions to the specific requirements of the given application. Furthermore, an integrated resource management strategy that considers timeliness requirements and energy requirements in a holistic manner is followed.

Whenever hardware resources (processors, memory, caches) are shared among nearly independent processes, the implicit interactions (e.g., arbitration, cache reloading, processor switching) increase the energy consumption and make time and energy estimation more difficult.

The non-availability of a generally accepted architectural style for embedded systems leads to many property mismatches at the interfaces among subsystems (e.g., *big-endian* vs. *little-endian*) that must be reconciled by the system software, leading to necessary but unproductive glue software which consumes valuable energy when executed.

8.4.2 Application Software

Algorithm Design. An algorithm that has been developed from the point of view of optimal time performance will be different from an algorithm that is developed from the point of view of optimal energy performance. In contrast to best-effort systems, where the average execution time of an algorithm is optimized, in real-time systems, the worst-case execution time of an algorithm is of relevance.

In many real-time applications, such as multimedia or control applications, there is not always a need for precise results – good approximate results will suffice. Finding algorithms that give a good approximate result under energy constraints is a relevant research topic (see also Sect. 10.2.3 on *anytime algorithms*).

In *cloud computing*, where some tasks are processed in the cloud (the servers in the cloud run on power available from the electric utility) and some tasks are computed locally in a battery-operated mobile device, the tradeoff between the energy requirement of the algorithms for the local computations and the energy requirement for the transmission of data to and from the cloud is an important architectural design issue. Appropriate design tools must support the design exploration of different task allocation strategies to the cloud or to the mobile device.

Algorithm Analysis. Many embedded applications contain a computationally intensive algorithmic segment, called a *computational kernel*. Profiling the execution of a program that implements an algorithm can identify the computational kernel. If a computational kernel is isolated into a self-contained component that puts all elements of the computational kernel (e.g., processing engine, memory) physically and temporally close together in an IP-core, then the effective capacity C_{eff} of the execution environment can be reduced, thus saving energy. If an identified

computational kernel is mature, does not change, and is used by many applications, then it can be transformed to an ASIC IP-core of an MPSoC, resulting in orders of magnitude energy savings.

Data Structures. The significant energy cost for memory access can be reduced by the design of data-structures that are optimized from the point of view of energy efficiency for the specified use cases. For example, a strictly binary time-format, which can be implemented with a binary counter requires much less energy to operate than a time-format that is based on the Gregorian Calendar.

8.4.3 Software Tools

Next to the system software, compilers have an important role to play in the design of low-energy systems. Energy aware compilers can select the instructions for the target code on the basis of their energy requirements. The energy-aware allocation of registers is another important issue, since the register file of modern processors is quite energy intensive

System designers need tools to estimate the power consumption at an early stage of the design. These tools must be flexible to support different target architectures in order that *design explorations* can be carried out. They should be smoothly integrated in the design environment.

8.5 Energy Sources

There are three main sources of energy that drive embedded systems: energy from the power grid, energy from a battery, or energy harvested from the environment.

8.5.1 Batteries

Table 8.3 depicts the nominal energy content of different disposable and rechargeable batteries. The actual amount of energy that can be retrieved from a battery depends on the discharge pattern. If the requested battery power is highly irregular, the battery efficiency is reduced and the actual energy that can be drawn out of a battery can be less than half of the nominal energy [Mar99]. The discharge level of a rechargeable battery has an influence on the lifetime of the battery, i.e., the number of times the battery can be recharged before it breaks down.

The efficiency of a rechargeable battery, i.e., the relation *energy-input-for-charging* and *energy-output-for-use* of well-managed batteries is between 75% and 90% under the assumption that during the *charging phase* and the *use phase* the power that flows into and out of the battery is carefully controlled and matched to the electrochemical parameters of the battery. If an application has highly

Table 8.3 Energy content of batteries

Battery type	Voltage (V)	Energy density (J/g)	Mass (g)	Energy (J)
AA (disposable)	1.5	670	23	15,390
AAA (disposable)	1.5	587	11.5	6,750
Button cell CR2032 (disposable)	3	792	3	2,376
NiCd (rechargeable)	1.2	140		
Lead acid (rechargeable)	2.1	140		
Lithium ion (rechargeable)	3.6	500		
Ultra-capacitor		Up to 100		
(Gasoline)		44,000		

irregular power characteristics, then an intermediate energy storage device, such as an *ultra-capacitor*, must be provided to smoothen the power flow in and out of the battery. An ultra-capacitor is an electro-chemical capacitor that has a high energy density when compared to conventional capacitors. It can be used as an intermediate energy storage device to buffer large power-spikes over a short time.

> **Example:** Let us consider an electric car (example of Sect. 8.1.1) that is driven at a speed of 30 m/s (108 km/h). All of a sudden the car has to brake abruptly and come to a stop within 5 s. If regenerative brakes transform the kinetic energy of 450 kJ to electric energy with an efficiency of 60%, a power spike of 54 kJ/s will develop for 5 s – corresponding to a current of 135 A in a 400 V system. If the battery cannot absorb this power spike in such a short time, then an ultra-capacitor must be placed between the brakes and the battery to smoothen the flow of power back into the battery. It is up the embedded control system to control the power-flow in and out of the ultra-capacitor and into the battery.

In the last line of Table 8.3 we denote, for comparison purposes, the chemical energy content of gasoline. Although only at best a third of this chemical energy can be converted to electrical energy – e.g., by a motor generator set – it is seen that the energy density of gasoline per unit weight is still more than an order of magnitude higher than that of the most efficient battery.

8.5.2 Energy Harvesting

The term *energy harvesting* refers to techniques that transform ambient energy (e.g., light, temperature gradients, electric fields, mechanical motion, vibrations, wind, etc.) to electric energy that can be used to drive low-power electronic devices, such as a wearable computer or a node in a sensor network. The harvested energy is stored in a battery that provides a steady flow of current to power the electronic device.

For example, under optimal sunlight conditions photovoltaic cells can produce 15 mW/cm^2. Small thermocouples that convert body heat into electricity can generate 40 μW at 3 V with a 5°C temperature gradient. Piezoelectric transducers can convert mechanical energy, e.g., vibration or acoustic noise, to electric energy. In an RFID tag, the receiver circuit is powered by energy harvested from the electric field of the sender. A typical RFID tag has a power budget of 10 μW (see Table 13.1).

An electronic device that is powered by energy from its environment becomes nearly autonomous and can provide a useful service for a long unattended period. Wireless sensor nodes that operate robustly and autonomously with energy harvested from their environment without any battery are useful in many applications: industrial control, environmental monitoring, surveillance, medical devices that are implanted into the body, and many more. The development of efficient transducers to capture the ambient energy and the design of low-power electronics that manages the energy harvesting process is an active research topic.

Points to Remember

- *Energy* is defined as the capability of performing work. *Power* refers to the intensity of work. Energy is the integral of power over time. Although power and energy savings are closely related, they are not the same.
- Energy that is needed by a computing device to execute a program can be expressed as the sum of the following four terms: $E_{total} = E_{comp} + E_{mem} + E_{comm} + E_{IO}$ where E_{total} is the total energy needed, E_{comp} denotes the energy needed to perform the computations in a CMOS VLSI circuit, E_{mem} denotes the energy consumed by the memory subsystem, E_{comm} denotes the energy required for the communication, and E_{IO} is the energy consumed by I/O devices, such as a screen.
- The dynamic energy needed to run a program with N instructions is given by $E = C_{eff}\, V^2 N$ where C_{eff} is the effective capacitance of an instruction, V is the supply voltage, and N denotes the number of instructions that must be executed.
- The exponential growth of the leakage current with rising temperature is of major concern in a submicron device with a low threshold voltage.
- The asymmetry of the energy requirements of a sender and a receiver has a decisive influence on the design of communication protocols between a base station connected to the power grid and an energy-constrained mobile battery-powered device.
- In memory-intensive applications, the energy that is required to access the memory subsystems can be larger than the energy needed for the computations.
- A high substrate temperature has a negative effect on the reliability of a VLSI device and can cause transient and permanent failures. Around half of all device failures are caused by thermal stress.
- The most effective way to reduce the power consumption of CMOS devices is the scaling of the device parameters, i.e., making the transistors smaller.
- The scaling factor α from one micro-electronic generation to the next is normally 0.7 such that the area of a scaled version of a design is reduced by a factor of 2, the power requirement is reduced by a factor of 2, the speed is increased by a factor $\sqrt{2}$ and the energy performance increases by $2\sqrt{2}$.
- The positioning of all subsystems onto a single die (SoC) leads to a significant reduction of the distances between the transistors in the diverse subsystems (and

in consequence to a reduction of the capacities of the signal lines) which results in major energy savings.
- Device scaling cannot go on indefinitely because there are limits that have their cause in the discrete structure of matter and quantum mechanical effects, such as *electron tunneling*.
- If the frequency for the operation of the device is reduced, the voltage can be reduced as well without disturbing the functionality of the device, resulting in substantial energy savings.
- Real-time and execution time are two different time bases that are not closely related.
- A technology-agnostic design methodology makes it possible to move functionality from software to hardware with a substantial gain in energy efficiency.
- Pollack's rule states that micro-architectural advances from one generation to the next increase the performance of sequential processors only with the square root of the increase in power or silicon area.
- Spatial locality of computational subsystems reduces the effective capacitance and thus increases energy efficiency.
- The most important contribution of software to energy efficiency is a reduction of the number and types of statements that must be executed to achieve the desired results.
- The actual amount of energy that can be retrieved from a battery depends on the discharge pattern. If the requested battery power is highly irregular, the battery efficiency is reduced and the actual energy that can be drawn out of a battery can be less than half of the nominal energy.
- *Energy harvesting* refers to techniques that transform ambient energy (e.g., photovoltaic, temperature gradients, electric fields, mechanical motion, vibrations, wind, etc.) to electric energy that can be used to drive low-power electronic devices, such as a wearable computer or a node in a sensor network.

Bibliographic Notes

The tutorial survey by Benini and Micheli [Ben00] gives an excellent overview of system-level design methods that lead to energy-efficient electronic systems. Pedram and Nazarian [Ped06] provide models to investigate the thermal effects in submicron VLSI circuits. The limits of device scaling are the topic of [Fra01]. Pollack's rule and the future of SoC architectures are the topic of [Bor07]. Issues of energy awareness in systems-on-chip are discussed in [Pol07].

Review Questions and Problems

8.1 Explain the difference between power and energy. Give an example where a reduction of the power leads to an increase of energy needed to complete a computation.

8.2 How many Joule are contained in a calorie or in a kWh?
8.3 Calculate the dynamic energy of a program execution if the program contains 1,000,000 instructions, the supply voltage is 1 V, and the effective capacitance of an average instruction is 1 nF.
8.4 *What is static energy*? How does static energy change with temperature?
8.5 How much energy is required for access to the scratchpad memory, the on-chip memory, and the off-chip memory in the reference architecture introduced in this chapter?
8.6 A sensor node executes 100,000 instructions per second (supply voltage is 1 V and effective capacitance on an instruction is 1 nF) and sends every second a message with a length of 32 bytes to its neighbor node, which is 10 m away. The voltage of the transmitter is 3 V. How much power is needed to drive the sensor node? How many hours will the sensor node operate if the power supply contains two AAA batteries?
8.7 A processor has two operating modes, a time-performance optimized mode characterized by a voltage of 2 V and a frequency of 500 MHz and an energy-optimized mode characterized by a voltage of 1 V and a frequency of 200 MHz. The effective capacity of an instruction is 1 nF. What is the power requirement in each of the two modes?
8.8 A Lithium-Ion laptop battery weighs 380 g. How long will a battery-load last if the laptop has a power demand of 10 W?