Web-Based Energy Exploration Tool for Embedded Systems

Ikhwan Lee
Samsung Electronics

Yongseok Choi, Youngjin Cho, Yongsoo Joo, Hyeonmin Lim, Hyung Gyu Lee, Hojun Shim, and Naehyuck Chang
Seoul National University

Editor's note:
This article describes a Web-based energy estimation tool for embedded systems. An interesting feature of this tool is that it performs real-time cycle-accurate energy measurements on a hardware prototype of the processor. The authors describe the various steps involved in using the tool and present case studies to illustrate its utility.
—Anand Raghunathan, NEC Laboratories

Along with speed and cost, energy consumption is now a primary performance metric for battery-operated embedded systems. A well-designed embedded system should be globally optimized to the target application, from the user interface to the device technology. This type of global optimization over many layers of software and hardware is challenging because most designs require extensive interdisciplinary collaboration. Energy estimation is routine in low-level hardware design. Unfortunately, at this stage, the specific application of most hardware components is unknown, and designers can't perform an application-specific optimization. Another opportunity for optimization is during software and systems design, but these designers are often unfamiliar with hardware-related energy issues. This problem is compounded because traditional energy estimation tools such as Spice and PowerMill target low-level-hardware engineers, which might discourage designers working at higher levels from attempting global optimization.

A simple approach is to analyze actual measurements from a hardware platform. Tools such as Itsy use computer-controlled multimeters or A/D converters to measure energy consumption.[1] However, although these measurement-based systems are not restricted to a specific target, they have certain limitations. The major drawback of Itsy is that it is incapable of performing cycle-accurate analysis because it averages energy consumption over the entire execution time. Other studies adopt a software energy estimation methodology—an important aspect of embedded-systems design since Tiwari, Malik, and Wolfe first introduced this methodology.[2] Their work focused on the possibility of software optimization, but recently proposed tools extend the methodology to support high-level hardware optimization. For example, JouleTrack is a publicly available Web-based software energy-profiling tool for processor cores.[3] SimplePower and Wattch estimate the power consumption of processors, including the on-chip cache, on-chip bus, and on-chip SRAM, but still excluding the off-chip subsystems.[4,5] JouleTrack, SimplePower, and Wattch are suitable for architecture-level analysis because they consider only processors. On the other hand, Simunic, Benini, and De Micheli present a system-level energy estimation framework that includes a processor, an L1/L2 cache, off-chip memory, and a DC-DC converter.[6] Nevertheless, energy models for off-chip memory devices are too simple to support a cycle-level analysis. Their framework takes a basic power-per-mode approach and assumes that devices have only two modes: active and idle.

The increasing trend toward low-power design requires a higher-fidelity, system-level energy estimation environment. Here, we propose a Web-based energy exploration tool, the SNU (Seoul National University) Energy Explorer (SEE) Web—the publicly promoted ver-

0740-7475/04/$20.00 © 2004 IEEE Copublished by the IEEE CS and the IEEE CASS

sion of SEE, our accurate in-house energy estimation tool. SEE Web offers high-fidelity energy estimation based on cycle-accurate energy characterization, overcoming many of the limitations of existing tools. We also developed a finite-state machine (FSM) model that isolates dynamic and leakage energy consumption with the help of a cycle-accurate measurement technique.[7,8] Additionally, we built the instruction set simulator (ISS) in hardware to increase simulation speed. To achieve a wider design space, SEE Web leaves many things available for users to configure themselves. Configurable parameters include various system design parameters such as the processor clock frequency, cache organization, the SDRAM clock frequency, and the SDRAM control policy.

SEE Web

SEE executes real application software on a testbench based on the SNU Energy Scanner.[8] SES focuses on the energy consumption of a microprocessor for software optimization, giving software programmers a perspective on their code's energy consumption. SEE targets an energy-optimal system configuration, given a set of components, such as hardware IP cores and software codes. SEE extends SES by adding a configurable memory hierarchy and an off-chip environment.[7,10]

Both SES and SEE are based on the same hardware support, which provides direct cycle-accurate measurement of the microprocessor's energy consumption. This hardware plays the role of an ISS and comes with a real microprocessor. An innovative technique using switched capacitors captures the hardware's cycle-by-cycle energy consumption.[8] This lets us run after two hares—speed and accuracy—and catch both of them. Engineers can measure energy consumption using a real component, or they can estimate it using an energy FSM. Unlike other FSM models, an energy state machine handles dynamic energy and static power separately.[7]

Although SEE is a useful tool, it has serious limitations for open use because it includes custom hardware. SEE Web overcomes these limitations through Web technology, providing a high-fidelity energy estimation environment to any Internet user. SEE Web currently includes basic components for program execution, but we designed it for easy expansion and customization, and plan to progressively add more features. However, being a Web-based tool, SEE Web encounters many obstacles, such as restricted profile size, limited simulation time for interactive use, and delays in downloads. These obstacles could discourage users from intensive use with large programs. Therefore, we've set some restrictions for normal users, and we currently accommodate "power users" through batch processing and FTP support.

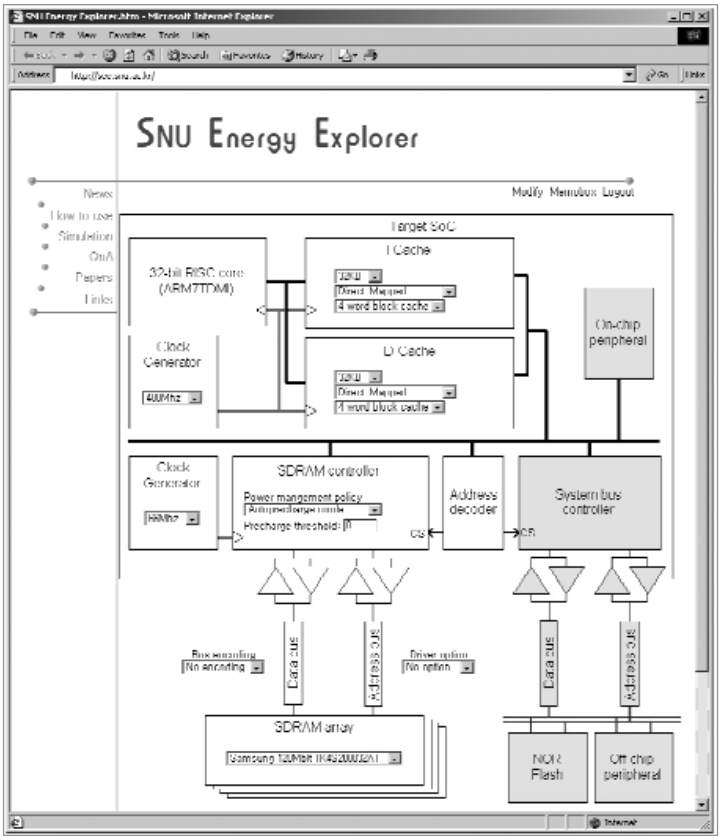

Figure 1. Configuration of SEE Web's target system.

The Web page at http://see.snu.ac.kr gives immediate access to a high-fidelity energy estimation environment. Users can estimate the energy consumption of an application program running on a desired hardware specification in three steps:

- configure the target architecture,
- upload a binary image, and
- retrieve the energy profile.

The SEE Web page also includes a trace formatter, manuals, related articles, and a bulletin board. Designed for compatibility, SEE Web requires only a standard Web browser that can run Java applets.

Configuring the target system

Figure 1 shows SEE Web's home page. The main menus are on the left; the authentication menus are between the title and the system configuration. SEE Web guides users to the energy-optimal configuration

of an embedded system or a SoC, and has sufficient degrees of freedom to support energy exploration of embedded applications. Clicking on the *Simulation* menu opens the configuration screen. The user then sees a typical kernel of a modern embedded system equipped with a 32-bit reduced-instruction-set computing (RISC) processor, a cache memory, and an SDRAM main memory. The configuration affects performance, energy consumption, and thus cost, for the target system with a given application program and user data.

Preparing a binary file

Because we based SEE Web on an ARM7 RISC processor, it requires a cross-development environment. We verified two development environments: the standard GNU Compiler Connection (GCC) environment for Linux users and the ARM Software Development Toolkit (SDT). We provide simulation support files such as linker scripts for both environments. We recommend GCC version 2.95.3 because it's the latest stable version and is easy to obtain.

As an embedded-system program running without an operating system, SEE Web is a pure cross-development target. Thus, every target program must include a start-up code, and has limited access to runtime libraries; we recommend newlib-1.9.0. SEE Web supports start-up codes for the SDT and the GCC. Target programs must include only supported function calls and should be linked with the start-up code. This code, init.s, sets up the interrupt vector table and the stack pointer's initial value. Upon initialization, it calls c_entry(), which in turn calls the user's main() function. Users who want to estimate only specific portions of the code can do so by placing only that code in the main() function. For convenience, we also provide special functions, EstimationStart() and EstimationEnd(), to enable and disable the energy estimation process during simulation, thereby considerably reducing the amount of trace data.

SEE Web's memory map is simple. A 16-Mbyte RAM area starts at 0x00000000. SEE Web also supports a linker script, so users need not worry about section divisions. SEE accepts the Motorola S3 S-record format for image uploads. (SEE Web's home page, http://see.snu.ac.kr, contains detailed documentation on supported functions and on building the cross-development environment.)

Users can upload a binary file through a typical Web-style dialog box. Clicking on the *Submit* button sends the image file to the Web server and eventually downloads it to the SES hardware vector memory. After uploading, the energy estimation process starts, and the log window reports progress. Finally, SEE Web completes the estimation process by notifying the time elapsed on the target system, the elapsed ticks of the processor, and the program counter's last value. Because SEE Web has a limited capacity for the profile, the estimation process could stop earlier than the position specified by the control function However, we are accommodating power users for intensive energy simulation using SEE Web. We will provide more capacity to registered users.

Verifying the results

SEE Web presents simulation results in various forms. It generates a text file as a simple simulation report, and it provides Java applets for viewing the energy trace and memory map on a Web browser. We could have used ActiveX control, which provides many useful functions (load and save, screen hard copy, and so on), but that would limit users to Microsoft Windows. Using a Java applet provides wider operating-system compatibility. For power users, bulk energy trace files are also available. SEE Web stores each component's cycle-accurate energy consumption and each cycle's program counter in binary format, making it possible to correlate energy estimation with each input line in the source code. However, this is quite complicated, so we recommend they use EstimationStart() and EstimationEnd() for function-level energy analysis.

Simulation report. After completion, SEE Web prints basic information about the simulation to a text file. This file includes the target system configuration, total elapsed time, cache and memory transactions, and total energy consumption. Users can access this file by clicking on the *view* link in the *Simulation report* column. This file can serve as data for basic analysis and as a reference for later use.

Trace formatter. The result of energy estimation is a large binary file that remains in the Web server's file system. All users can download the file by clicking on the *Down* link in the *Profile data* column. However, because of the potentially long download times and unfriendly data formats, we don't recommend this.

SEE Web provides a trace formatter that visualizes the energy values like a storage oscilloscope. Users can selectively activate the energy waveforms for the processor core, cache memory, SDRAM controller, off-chip bus and drivers, or SDRAM. The trace formatter supports

pan and zoom. When the zoom scale is 1×, each time instant corresponds to a tick of the processor clock. As the zoom scale increases, the energy values for each time instant become the accumulation of the cycle-by-cycle energy values.

SDRAM viewer. For users wishing to verify their program's result, we also provide a memory map viewer. SEE Web saves the entire main memory after program execution so that users can dump the memory contents later. The SDRAM viewer shows the memory contents in both ASCII and hexadecimal formats. It's also possible to download the entire memory image, which is 16 Mbytes.

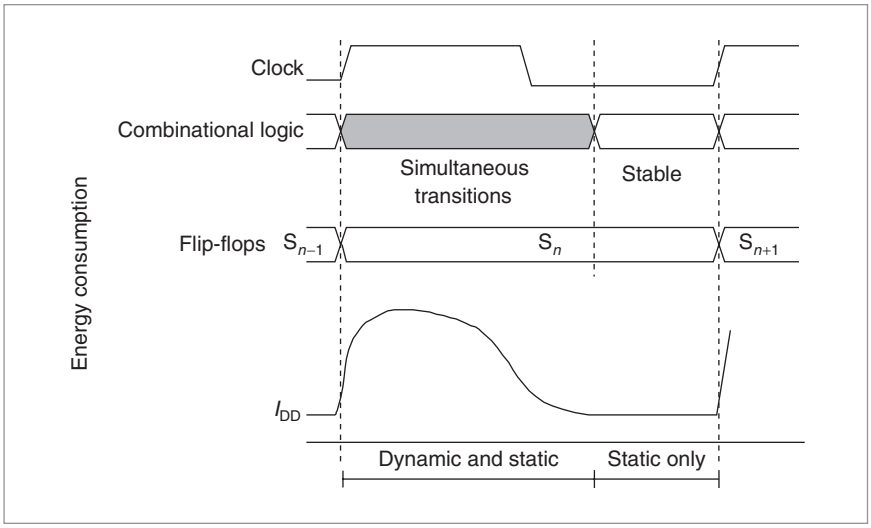

Figure 2. Energy consumption for a single state transition.

Multiuser issues

Users may not interrupt the hardware ISS during a simulation request. Currently, the Web server has one hardware ISS card, and a simple job queue schedules multiple requests. If a user submits a new binary file while another user is occupying the system, the new request goes into the queue, and that user receives a message about this file's rank in the queue. The queue rank number is decremented internally as the preceding jobs finish; users can check progress by occasionally refreshing their Web browser. The hardware ISS's processing speed is relatively fast—one request usually takes less than a minute—so users shouldn't have to wait long to use the tool. Moreover, we can easily improve SEE Web's availability by installing multiple hardware ISS cards. We will do this when the number of accesses justifies it. Once SEE Web has begun to service a simulation request, it stores the simulation results in files, where users can access their results at any time.

High-fidelity energy estimation

Here, we discuss the energy consumption of a synchronous finite-state machine (FSM). For cycle-accurate energy annotation of an FSM, we use a special measurement technique with two switched capacitors instead of conventional measurement equipment such as an ammeter.

Energy consumption of a synchronous FSM

High-fidelity energy estimation begins with an accurate energy consumption model. The energy consumption of most logic gates consists of dynamic energy and leakage power. Dynamic energy consumption stems from switching capacitance and short-circuit currents during signal transitions. Leakage (or static) power consumption is due to the logic structure, the physical design, or both. The amount of dynamic energy in a synchronous FSM is independent of the operating clock frequency and depends solely on the number of state transitions. Leakage power is generally constant for a single gate, but appears variable in a system-wide view because of the internal circuits' selective activation. Execution time is a function of the number of clock ticks required to finish a job, and the leakage energy within a clock cycle is inversely proportional to the clock frequency.

Figure 2 shows the energy consumption for a single state transition. When a clock transition occurs, the flip-flops use the new input values to update their output after a certain propagation delay. As new output values appear at the flip-flops, combinational logic connected to the flip-flops can change its output. Logic forming the next state then starts to recalculate its outputs, thus propagating simultaneous transitions. As the combinational logic finishes its calculation, the FSM becomes stable again, introducing no further transitions until the next clock tick. From the initial clock transition until it becomes stable, the FSM consumes dynamic energy. The time that the combinational logic needs for recalculation doesn't vary with the clock frequency. Hence, dynamic energy consumption is constant regardless of the clock period. But the static current flow is steady, so leakage power consumption is proportional to the

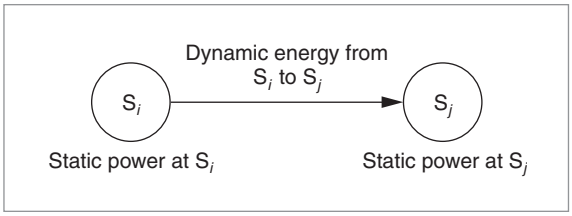

Figure 3. Energy consumption of a synchronous FSM.

length of the clock period. Again, taking a system-wide view, some active logic gates could differ from this state-by-state analysis due to current-mode circuits, bipolar totem-pole outputs, open-drain outputs, selective power-down, and so on.

Figure 3 denotes the power consumption of a synchronous FSM.[7] Logic circuit designers generally characterize combinational logic by its input and output data. In contrast, the energy consumption of most sequential logic depends primarily on the FSM's internal state. We can assume that the static power is constant at a given state. Therefore, the static energy consumption is proportional to the tenure time (the length of time that the static power spends at that state). But dynamic energy is not dependent on the tenure time. Thus, we denote the total energy consumption for a state transition from to S_i to S_j as

$$E(S_i, S_j) = E_d(S_i, S_j) + [P_s(S_j) \times \tau] \quad (1)$$

where E_d is the dynamic energy in joules, P_s is the static power in watts, and τ is the clock period of the FSM in seconds.

Table 1 compares the synchronous energy FSM with two popular energy characterization schemes, where n is the number of accesses, and t is the time the FSM spends in active mode. A power-per-mode analysis distinguishes a device's power consumption by its operating modes (active, idle, doze, power-down, and so on). However, it ignores the dynamic energy required for mode changes. More precisely, this analysis averages out the dynamic energy portion for the clock period and includes it in the static energy; that is, $E = (P_s + E_d/\tau)t$. Thus, by relying on characterization, the FSM might underestimate the dynamic energy at a faster clock speed and overestimate it at a slower speed. Moreover, power-per-mode characterization cannot describe the energy variation caused by control-sequence changes such as $S_1 \rightarrow S_2 \rightarrow S_3$ and $S_1 \rightarrow S_3 \rightarrow S_2$, even if each state's tenure time is the same.

The characterization scheme for energy consumption per access can't represent the energy variation that a clock frequency change causes, because a portion of the static energy for a clock period is added to the dynamic energy; that is, $E = (E_d + P_s\tau)n$. This analysis is suitable for asynchronous devices, although the FSM might overestimate the static energy at faster clock speeds and underestimate it at slower clock speeds. However, the synchronous-energy FSM denotes the static and dynamic energy separately. It can thus can characterize exact energy consumption while reflecting energy variations caused by control-sequence and clock-frequency changes; thus, $E = P_s t + E_d n$.

Now, assume we want to find the energy variation of a synchronous memory system by changing the memory clock frequency while fixing the processor clock frequency, as Figure 4 shows. Although the example in Figure 4b uses a slower clock than in Figure 4a, the number of accesses within a given period is the same; hence, there is no performance change. However, in Figure 4c, the faster memory clock lets one more memory transaction take place during that period. This example demonstrates the modeling superiority of the energy FSM over conventional energy characterization methods, such as the power-per-mode and energy-per-access models widely used in system-level energy simulators.

As an example, assume that the idle-to-idle dynamic energy is 1 nJ, the idle-to-active dynamic energy is 10 nJ, the active-to-idle energy is 5 nJ (usually precharge, which consumes significant energy, occurs at this point), the

Table 1. Energy characterization schemes for a synchronous finite-state machine (FSM), where E is total energy consumption in joules, P_s is the static power in watts, E_d is the dynamic energy in joules, τ is the clock period in seconds, t is the time the FSM spends in active mode in seconds, and n is the number of accesses.

	State		Transition	
Energy characterization	**Annotation**	**Energy equation**	**Annotation**	**Energy equation**
Power per mode	Energy	$E = (P_s + E_d/\tau)t$	Probability	NA
Energy per access	NA	NA	Energy	$E = (E_d + P_s\tau)n$
FSM energy	Static power	$E = P_s t$	Dynamic energy	$E = E_d n$

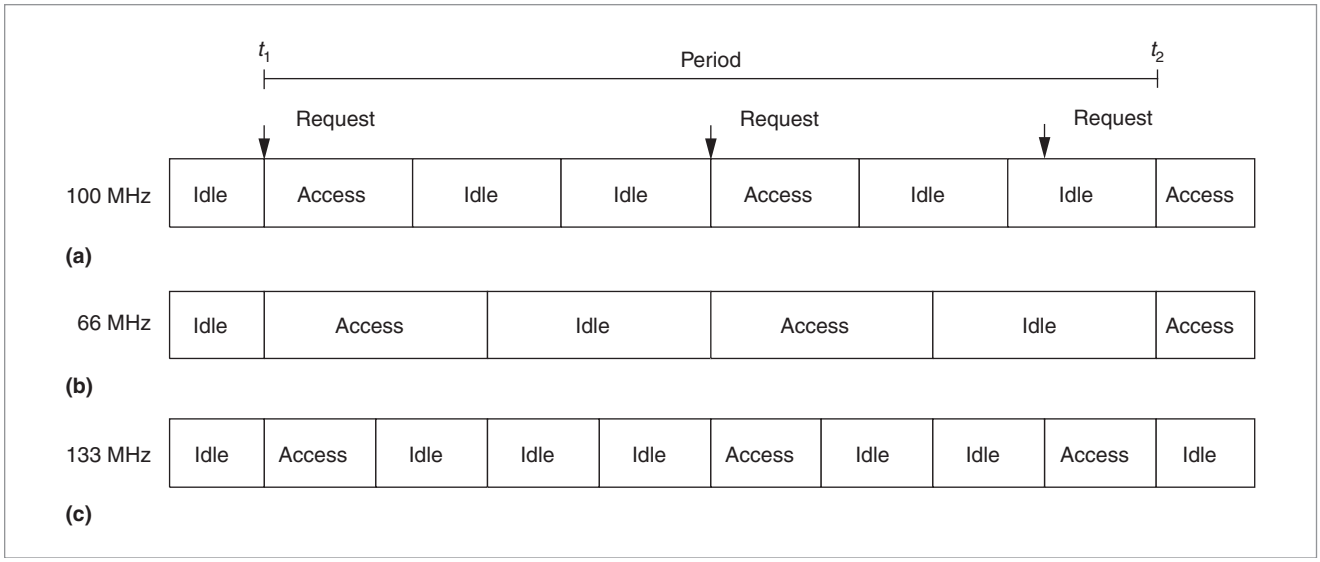

Figure 4. Energy consumption of different access styles between times t_1 and t_2: standard access (a), slow clock (b), and fast clock and one more transaction (c).

idle-mode leakage is 0.01 nJ/ns, and the active-mode leakage is 0.2 nJ/ns. We can directly apply these values to our FSM model, but we must convert them to fit them into the other two energy characterization schemes. Assuming a 100-MHz operation, the power-per-mode model describes the memory system energy as 0.11 nJ/ns for idle mode and 1.6 nJ/ns for active mode. The energy-per-access model describes the memory system energy as 1.1 nJ for an idle cycle and 16 nJ for an active cycle. Consequently, the exact energy consumptions that the energy FSM reports are 36.4, 36.3, and 47.9 nJ for 100, 66, and 133 MHz. The power-per-mode model reports 36.4, 51.3, and 40.1 nJ. The energy-per-access model reports 36.4, 34.2, and 53.5 nJ. These results indicate that the two traditional techniques do not guarantee correct results when operating at different frequencies.

Once we can manage a devices' functional FSM model, the associated energy FSM also becomes feasible. Unfortunately, state explosion often limits an FSM's utility, but many valuable techniques such as macromodeling and hierarchical FSMs can extend its potential applications. Consequently, we use energy FSMs only for buses, peripheral devices, and memory devices, which have relatively few states.

Cycle-accurate energy annotation

Although a synchronous-energy FSM is ideal for high-fidelity characterizations of energy consumption for synchronous digital systems, we need a special technique to annotate energy values for transitions and states. As

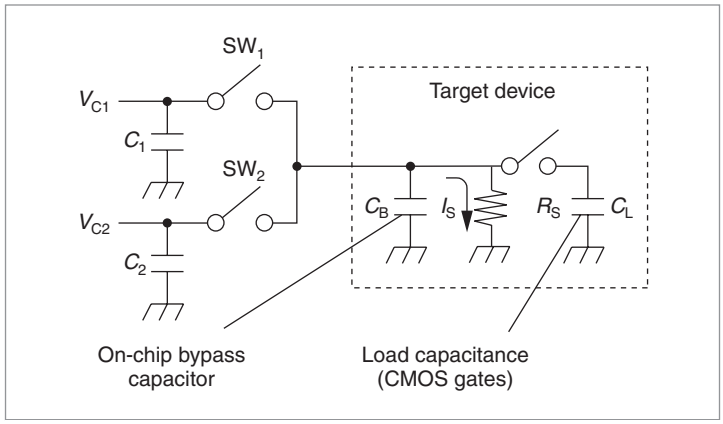

Figure 5. Cycle-accurate energy measurement using switched capacitors C_1 and C_2, where V_{C1} and V_{C2} are the voltages across those capacitors, SW_1 and SW_2 are switches, C_B is the on-chip bypass capacitance, I_S is the leakage current, R_S is the static resistance corresponding to this leakage current, and C_L is the device's load capacitance.

Figure 2 shows, dynamic energy consumption, represented by power-supply current I_{DD}, occurs only during the propagation time, which is usually a matter of nanoseconds. The physical design, not the operating frequency, determines the propagation delay, so the power spectrum of I_{DD} reaches well beyond several hundred megahertz, regardless of the operating frequency. This seriously discouraged us from trying to distinguish I_{DD} from the cycle-by-cycle dynamic energy using conventional equipment such as an ammeter.[8]

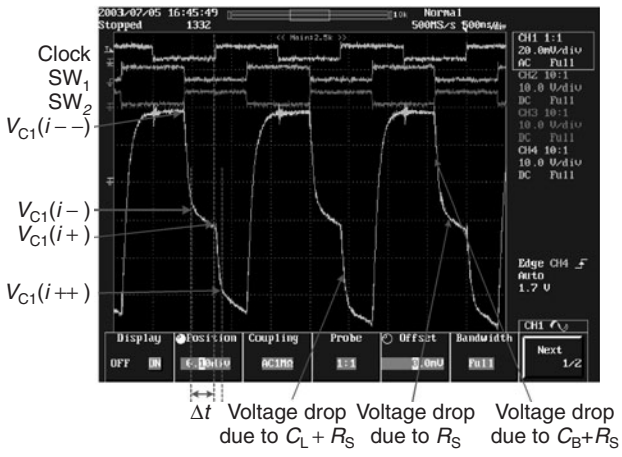

Figure 6. Voltage across switched capacitor C_1 in Figure 5.

Because cycle-accurate energy measurement is essential for annotating a synchronous-energy FSM, we developed a special technique to handle the cycle-by-cycle energy measurement of high-speed digital systems.[8] Figure 5 shows the measurement setup. We transfer charges to the capacitor and operate the target device using these charges. By simply measuring the capacitor's initial and final voltage, we can derive the exact energy that the target device consumed.

Most modern devices mitigate power-supply fluctuation using on-chip bypass capacitors, but this complicates energy calculations. We denote the on-chip bypass capacitance as C_B and determine its value using the charge-sharing rule.[11] In addition, modern high-performance devices are not generally free from leakage current. Therefore, we add static resistance R_S into the device model to represent the leakage current. Although most system-level energy simulators are primarily concerned about load capacitance C_L, we use fairly realistic energy models for both measurement and characterization.

Figure 6 shows a waveform we captured with a digital storage oscilloscope (DSO) using high-speed, pipelined A/D converters. The voltage drop varies, depending on the target device. We minimize the A/D converters' quantization error by adjusting the capacitance of switched capacitors C_1 and C_2 to correspond to the A/D converters' full-scale voltage swing. This demonstrates why it's necessary to customize the measurement tool to a target device, one of the motivations for our Web-based tool.

Dynamic energy consumption causes the major voltage drop that appears on the switched capacitors in Figure 6. The continuous voltage drop's slope corresponds to the leakage power consumption. $V_{C1}(i\cdot)$ denotes the voltage across capacitor C_1 for the ith clock cycle, where argument $(\cdot)$ denotes the four different states of the capacitor supplying power to the target circuit:

- $(--)$, meaning fully charged;
- $(-)$, meaning connected to on-chip bypass capacitor C_B;
- $(+)$, meaning discharged by leakage energy consumption; and
- $(++)$, meaning discharged by dynamic energy consumption.

At the same time, C_2 is discharged during $(--)$ and remains in a fully charged state during $(-)$, $(+)$, and $(++)$.[11]

The waveform's slope is the static or leakage energy consumption.[11] We denote the ith clock cycle's static power as

$$P_s(i) = 1/2(C_1 + C_B)[V_{C1}(i-)^2 - V_{C1}(i+)^2]/\Delta t \quad (2)$$

We eliminate Δt by converting the static power to static energy consumption over clock period τ. We denote the ith clock cycle's dynamic energy as

$$E_d(i) = 1/2(C_1 + C_B)[V_{C1}(i+)^2 - V_{C1}(i++)^2] \quad (3)$$

Finally, the total energy consumption is

$$E = \sum_{i=0}^{n}\left[E_d(i) + \tau P_s(i)\right] = \sum_{i=0}^{n} E_d(i) + n\tau P_s \quad (4)$$

It's not easy to determine the exact time that delimits the period for the dynamic energy through delimiter $V_{C1}(i++)$. Improper division into dynamic and static energy can cause severe errors if the clock frequency changes significantly. To avoid this, we measure the cycle-accurate energy at various clock frequencies. We cross-check the dynamic energy values measured at different clock frequencies and thus confirm the dynamic energy values.[11]

SEE Web architecture

SEE Web is expandable, and we can add a new IP core in the form of a behavioral-level C code function. Such a function would include an energy consumption

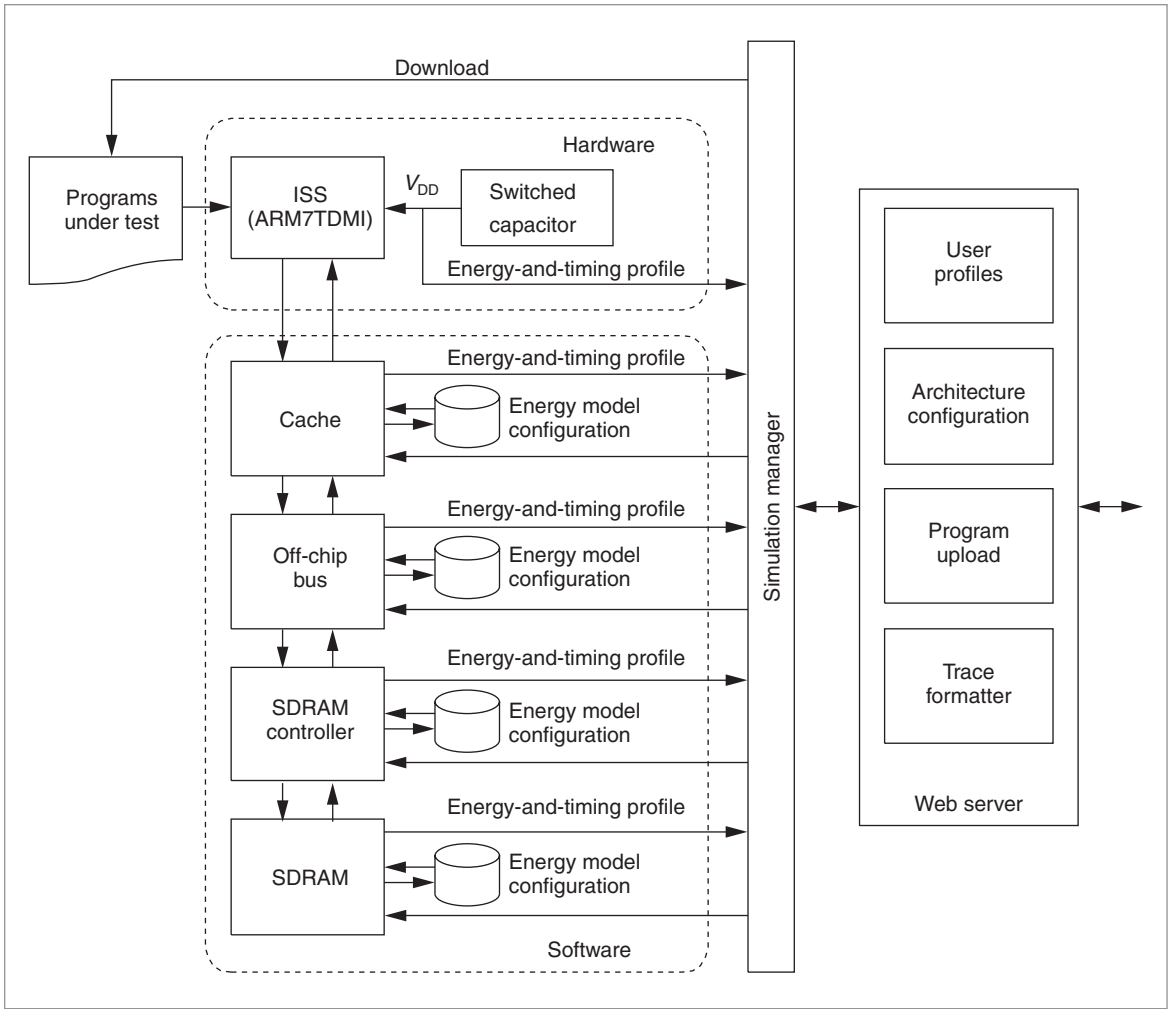

Figure 7. Architecture of SEE Web.

model at the clock-cycle level in the form of the energy FSM. This model would enable design-space exploration with complete freedom.

Figure 7 shows SEE Web's architecture. The simulation manager connects a typical embedded system containing a CPU, cache, off-chip bus, SDRAM controller, and SDRAM to a Web server. Users can configure the system architecture, upload a program to simulate, and see the simulation results through the Web interface.

Instruction set simulator

A 32-bit RISC processor, the ARM7TDMI, serves as the ISS for SEE Web. Users can freely vary the ISS's clock frequency between 33 MHz and 1 GHz, because the energy measurement circuit measures energy per clock cycle rather than the average power over a clock period. (The ARM7TDMI actually operates with a fixed clock frequency). The ISS captures the ARM7TDMI's cycle-by-cycle energy consumption and bus transactions. The ARM7TDMI test chip does not include a memory management unit or cache.

A hardware ISS has many advantages. One of its most beneficial features is execution speed acceleration. A software ISS generally has a speed bottleneck due to complex parsing. But only the energy measurement circuits limit the ARM7TDMI's speed. Simple board-level analog-circuit techniques permit clock speeds of several megahertz. Moreover, we saved manpower in designing and implementing the ISS. It wasn't much different from making an ordinary PC card with a microprocessor; it took only a few person-months to finish the first version of the hardware. The short design and implementation time encourages us to upgrade the microprocessor, if necessary. Most importantly, we dramatically reduced verification time because we didn't have to pursue bugs in the ISS. This is a critical issue in

modern systems design. We can also reduce the time required to port a program under test.

One of the most serious drawbacks of using a hardware ISS is the difficulty of duplicating or distributing the tool. This was the motivation for SEE Web, and Web technology has now overcome this problem. Another drawback is the processor core's reduced flexibility, but this also occurs when using a software ISS. In fact, despite the fixed ARM7TDMI core, SEE Web is suitable for several diverse systems, including the following examples:

- Embedded-systems designers are often interested in the entire system, not just the microprocessor. For example, the proportion of energy that the processor core consumes is generally less than 10% of the total system energy.
- Most RISC processors have a similar number of registers, similar code densities, and the same word length, so you can expect similar memory transactions between the processor core and the first-level cache.
- Sometimes the average power estimation, as reported on data sheets, is sufficient. Pipelined operations maximize each component's use, so power fluctuations are often minor.

We implemented the rest of the hierarchy in software.

Cache

The cache module has a 32-bit data word, a 32-bit address space, and separate cache memories for instructions and data. The cache simulator differs from well-known simulators, which often ignore data behavior. The instruction cache and the data cache are independently configurable with considerable flexibility. Their capacity is selectable between 1 and 32 Kbytes. Set associativity is configurable between direct-mapped and 32-way. Line size is selectable between four and eight words. The replacement policy is currently least recently used (LRU), but we will soon provide other policies, such as pseudorandom and round-robin. The cache module and the processor core use the same clock frequency for synchronization.

We set up the cache energy model based on XCacti,[12] which provides an analytical model for a cache's energy. First, we composed an FSM model of a cache in terms of the microprocessor's view, as Figure 8 shows, but including synchronization with an SDRAM controller whose clock frequency was different from that of the cache. Second, we decomposed XCacti's energy-per-access model into dynamic energy and leakage power in terms of the cache's FSM. XCacti provides not only the total energy for each cache operation—such as read hit, write hit, write back, and line fill—but also the energy for each cache component. Among the cache components, we assigned *sense amp tag* and *sense amp data* energy to leakage power, and the others to dynamic energy. Finally, we had to resolve idle-to-idle leakage and dynamic-energy values because XCacti doesn't distinguish them. We interpolated these values using the power consumption figures given on the data sheets corresponding to different types of synchronous SRAM.

The dynamic energy and leakage power depend on the cache's size and configuration. Table 2 shows the energy values corresponding to Figure 8, for a typical cache configuration. The FSM includes energy values for all the configurations that the simulator supports.

Off-chip bus

The off-chip bus driver specification is a low-voltage technology (LVT) for either a 3.3-V transistor-transistor logic (TTL) or 3.3-V BiCMOS bus with a 2.7-pf transmission line capacitance. Soon, we will provide other bus drivers such as low-voltage CMOS (LVCMOS), gunning transceiver logic plus (GTL+) and stub-series termination logic (SSTL).[10] To prevent the data bus from floating during a high-impedance state, we made bus-hold logic selectable. Because of excessive static current, passive pull-up is not suitable for quality systems, but we plan to include it anyway. Users can also select bus-invert coding using the transition activities or the logical-low state.[10]

The energy state machines shown in Figure 9 denote the energy consumption of synchronous off-chip buses. In Figure 9a, states S_0 and S_1 represent driven-low and driven-high states. Figure 9b includes S_2 and S_3 to represent bus-hold states. Researchers have already studied the power consumption of the LVT and GTL+ bus and bus drivers.[10] We compose an energy state machine for an LVT bus (a commonly used high-performance memory bus for embedded systems) by converting the power values to cycle-accurate energy values. The bus-hold logic acts like a small capacitance (typically 0.5 pf) and consumes negligible DC current, and we allow no state change, such as $S_0 \rightarrow S_3 \rightarrow S_1$. This guarantees that $E_4 = E_5 = E_6 = E_7 = 0$, and $P_2 = P_3 = 0$.

In Figure 9a, we used a Fairchild 74LVT245 transceiver to compose a 2-inch bidirectional bus with $E_0 = E_1 = 0.55$, $E_2 = E_3 = 0$, $P_0 = 0.0053$, and $P_1 = 0$. The units of

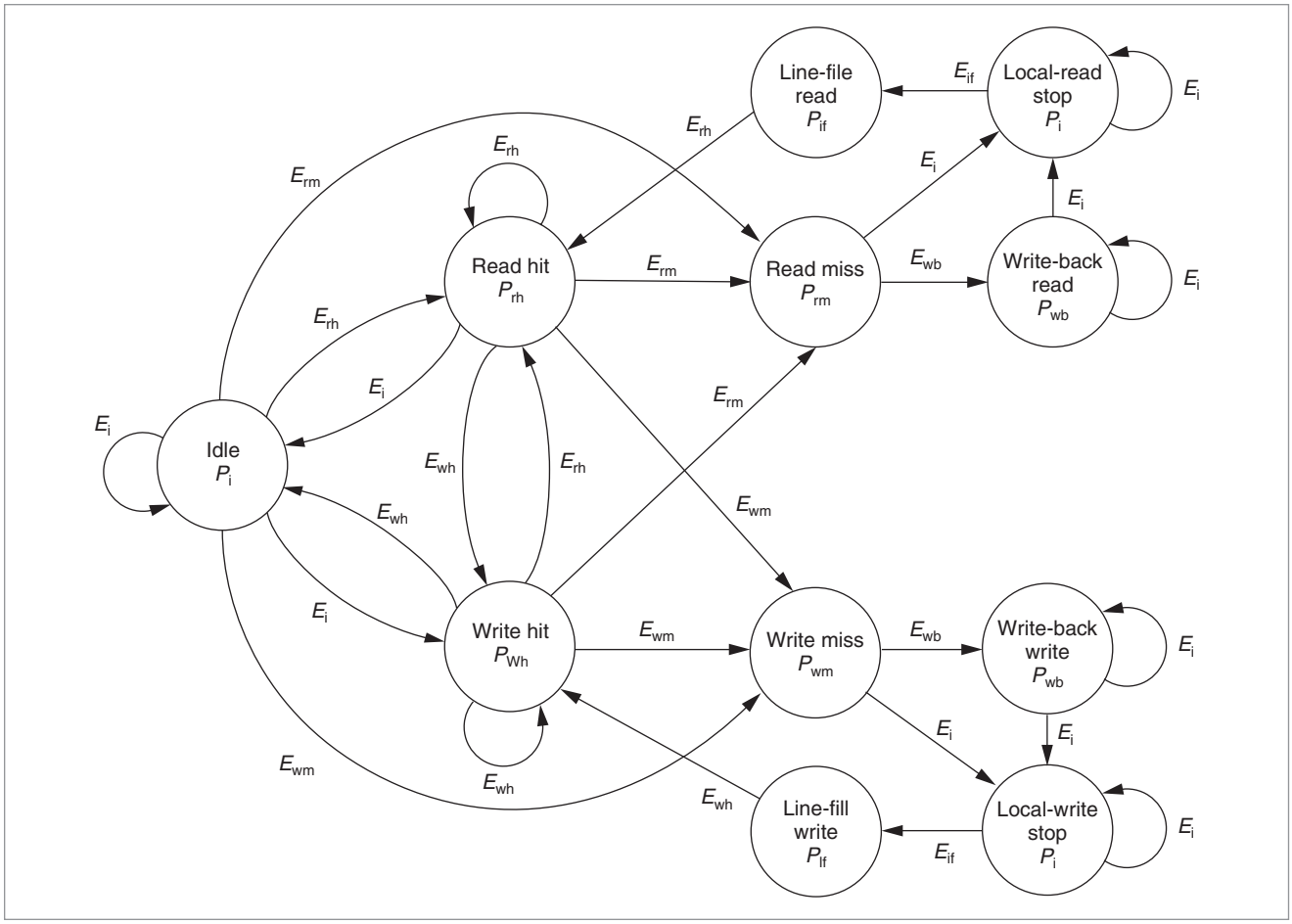

Figure 8. Synchronous energy FSM of the cache, where E_i is the idle-state dynamic energy; P_i is the idle-state leakage power; E_{rh} is the read-hit dynamic energy, P_{rh} is the read-hit leakage power, E_{wh} is the write-hit dynamic energy, P_{wh} is the write-hit leakage power, E_{rm} is the read-miss dynamic energy, P_{rm} is the read-miss leakage power, E_{wm} is the write-miss dynamic energy, P_{wm} is the write-miss leakage power, E_{lf} is the line-fill dynamic energy, P_{lf} is the line-fill leakage power, E_{wb} is the write-back dynamic energy, and P_{wb} is the write-back leakage power.

Table 2. Baseline energy values for the 0.18-μm, 8-Kbyte, four-way, four-word-block cache module in Figure 8.

Energy type	Idle state	Read hit	Write hit	Read miss	Write miss	Write back	Line fill
Leakage power (nJ/ns)	P_i	P_{rh}	P_{wh}	P_{rm}	P_{wm}	P_{wb}	P_{lf}
	0.0004	0.0106	0.0106	0.0106	0.0106	0.0106	0.0004
Dynamic energy (nJ)	E_i	E_{rh}	E_{wh}	E_{rm}	E_{wm}	E_{wb}	E_{lf}
	0.06	0.18	0.23	0.18	0.18	0.21	0.31

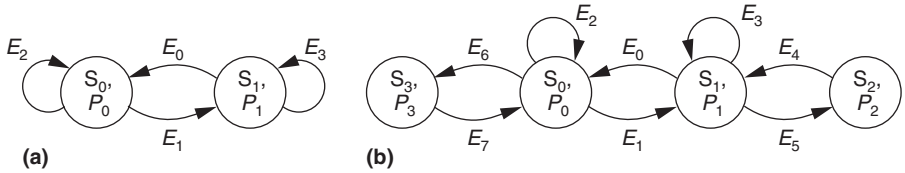

Figure 9. Energy state machine of synchronous off-chip bus driver without (a) and with (b) bus-hold logic.

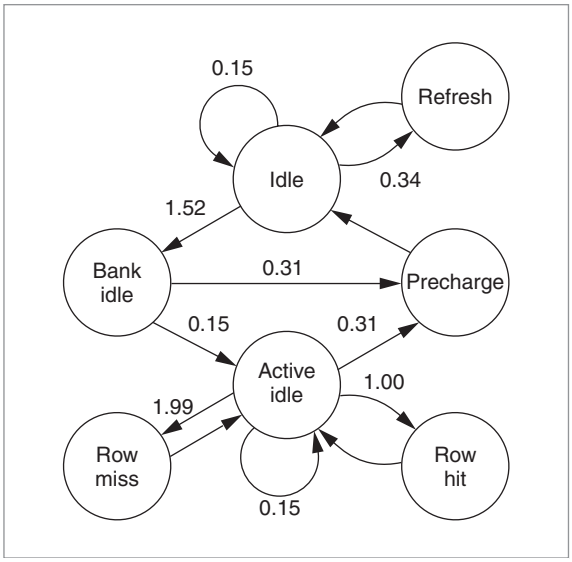

Figure 10. Synchronous-energy FSM of the SDRAM controller implemented with 0.18-micron technology. (All energy values are in nJ.)

E and *P* values are nJ/bit and nJ/(ns·bit), respectively. We apply these energy values to the address and data buses by using the bus models described earlier.[7]

On-chip SDRAM controller

We based SEE Web's SDRAM controller on a typical SDRAM controller but with special embedded functions for power management. These include active-page, auto-precharge, and delayed-precharge policies, as well as automatic idle-mode power-down and wake-up features.[7] The SDRAM's clock frequency is adjustable from 33 to 133 MHz.

We acquired the SDRAM controller's energy model using an FPGA implementation. We use the cycle-accurate measurement technique exhaustively to find the energy consumption of each SDRAM controller operation. Then we extract the leakage and dynamic energy using Equations 2 and 3. Our FPGA implementation consumes 241.4 mW during continuous auto-precharge, burst-mode accesses,[7] whereas a similar SDRAM controller implemented with 0.18-micron CMOS technology consumes 26.6 mW. So we normalize the energy value in the energy FSM using the 0.18-micron CMOS controller's average power consumption. The leakage energy of all the states is the same, 1.33 pJ/ns, which is 5% of the total power.

SDRAM

SEE Web comes with a 32-bit (8 bit × 4 bit), off-chip SDRAM array. Currently, SEE Web supports two devices from two major vendors: Samsung Electronics' 128-Mbit K4S280832B and Micron Technology's 128-Mbit MT48LC16M8A2.[7] Currently, SEE Web allows only 16 Mbytes of memory space for target application programs out of a possible 64 Mbytes.

An SDRAM's energy consumption also varies with the data and address for read and write transactions, but these contribute a very small percentage of the total energy, which mainly depends on the device's internal state. The SDRAM changes its state at each clock transition, according to the command input. Figure 11 illustrates a simplified FSM of an SDRAM. We merged some states together to simplify the figure; in the simulator, we use the exact FSM models for accuracy.[7] For each state, we annotate the leakage energy per clock cycle consumed in the state, and we annotate each transition with the dynamic energy consumed during the transition. We also acquired these values using our cycle-accurate measurement technique and Equations 2 and 3. As Figure 11 shows, the Samsung and Micron devices have different energy consumption behaviors. Data sheets for the two devices tell us only that the Micron device consumes a little more power than the Samsung one, but the FSM model lets us discover more details about their behavior.

Other components

Currently, SEE Web supports only the core components necessary to run application programs: a CPU, a cache, an SDRAM, an SDRAM controller, and an off-chip memory bus. We will continually add other components to allow more system-wide energy exploration. For example, we will soon add

- an asynchronous system interface comprising a programmable address decoder and an off-chip bus interface with a configurable wait cycle;
- a NOR flash memory for both asynchronous and synchronous interfaces, such as Intel StrataFlash, that allows burst-mode transactions;
- on-chip peripherals such as timers and DMA controllers; and
- off-chip peripherals such as an LCD controller (we'll also include a frame-buffer memory and associated buses, a backlight system, and an LCD panel).

Verification of the energy estimation

To verify SEE Web's energy estimation, we compared its simulation results with measured energy values from a real hardware board. The board is a typical

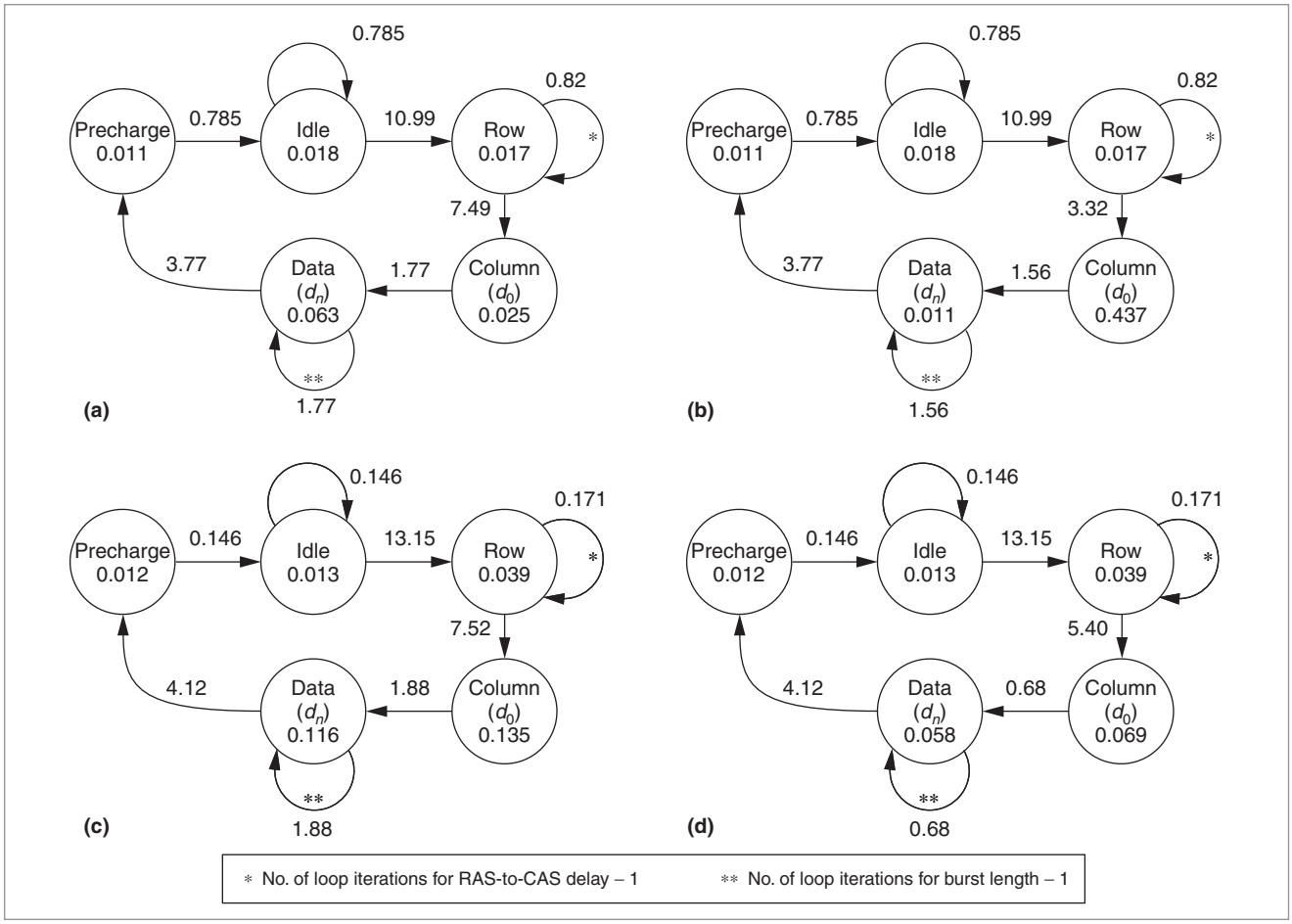

Figure 11. Simplified FSM of SDRAM operations for the Samsung Electronics K4S280832B and the Micron Technology MT48LC16M8A2: burst-read Micron (a); burst-write Micron (b); burst-read Samsung (c); burst-write Samsung (d). Dynamic energy values (numbers annotated at each transition) are in nJ. Leakage power values (numbers annotated in each state) are in nJ/ns.

embedded system equipped with an Intel XScale 80200 processor, a Micron 128-Mbyte SDRAM, an Intel flash memory, Ethernet controllers, and so on. The board has separate power planes for different components, and it's possible to measure each component's energy consumption separately using a data acquisition system. In this experiment, we considered the energy consumption of the CPU and the SDRAM power planes. This includes the energy consumption of the CPU core, on-chip instruction and data caches, address and data buses, and SDRAM devices, comprising an architecture quite similar to that of our simulator. To improve this verification's plausibility, we configured SEE Web as similarly as possible to the board. Table 3 summarizes the configuration of these two environments.

Table 3. SEE Web's configuration compared with a real embedded-system hardware board.

Parameter	SEE Web	Board
CPU family	ARM7	Xscale
Data bus width	32 bits	64 bits
SDRAM capacity	64 Mbytes	128 Mbytes
SDRAM control	100 MHz, active-page policy	
Instruction and data caches	32 Kbytes, 32-way set associative	
CPU frequency	333 MHz to 733 MHz	

We ran four embedded applications on both environments: an MPEG-4 decoder and three MiBench benchmark suite applications. MiBench is a well-known benchmark suite that provides a set of representative embedded programs. Because the board is not designed for this verification, we didn't expect the energy consumption behavior of the two systems to be identical. The

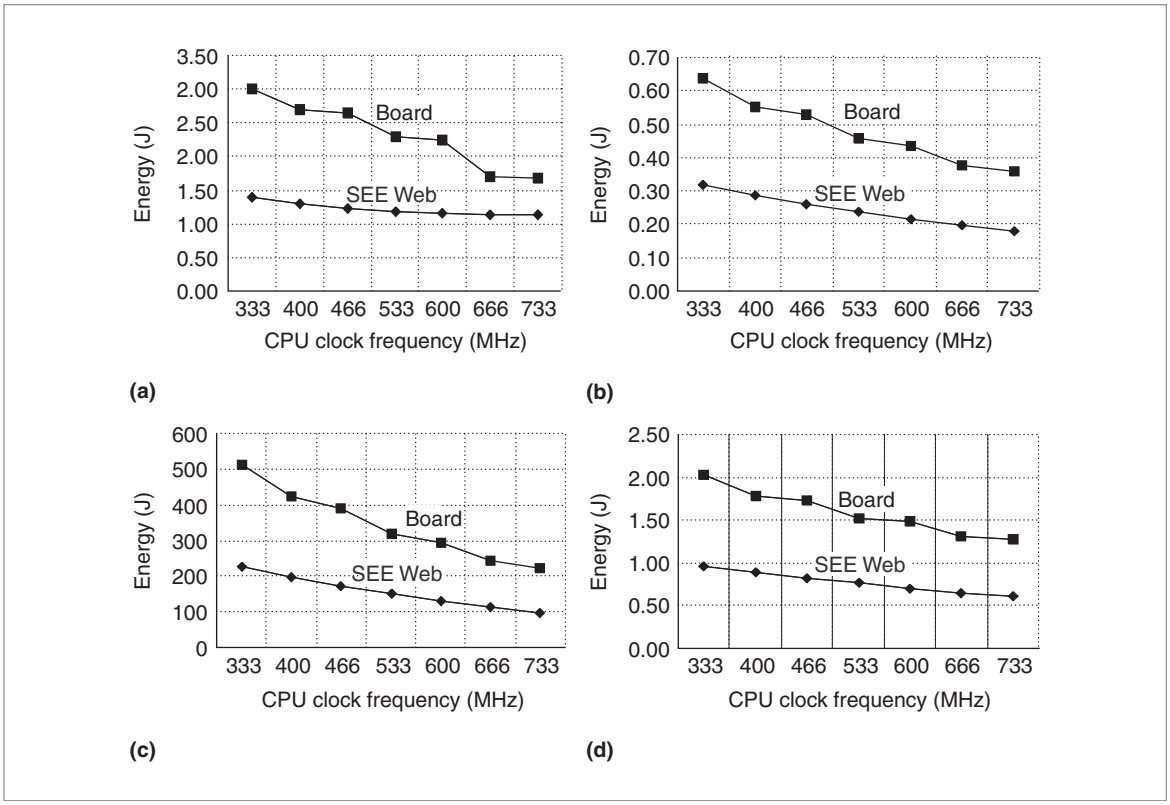

Figure 12. Energy consumptions of SEE Web and a real embedded-system hardware board for an MPEG-4 decoder (a), and for string-search (b), basic-math (c), and CRC32 cyclic-redundancy-check (d) benchmarks.

two systems' energy densities differ, so the simulation results will be somewhat different from the measured data. However, this experiment shows that SEE Web gives reasonable results for an entire embedded system.

Figure 12 shows the energy consumption of SEE Web and the board at different CPU core clock frequencies. In both environments, the CPU's energy consumption tends to remain in a narrow range because we scaled only the clock frequency, not the supply voltage. On the other hand, because the SDRAM's power consumption was almost independent of the CPU core's clock frequency, a higher frequency leads to shorter execution time and thus lower energy consumption by the SDRAM. Therefore, the curves show a decline in energy consumption as the CPU core clock frequency increases. This tendency appears more clearly in the board because it has a larger memory and a wider bus. The difference between the two curves is the consequence of different silicon technologies, different components, and so on. In summary, SEE Web gives reasonable results, and we hope this encourages users to employ it for a wider range of embedded systems.

Example application

Figure 13 gives an idea of how the tool displays the simulation result for an example application—a JPEG compressor. This application has the following configuration:

- a 266-MHz microprocessor;
- four 66-MHz SDRAMs;
- an 8-Kbyte instruction cache and data cache with two-way set associativity and four-word blocks;
- an SDRAM controller with an auto-precharge policy; and
- the Samsung K4S280832B as the SDRAM device.

The simulation report can provide a short summary, if that's all a user needs. The trace formatter shows each system component's cycle-accurate energy consumption, using graphs that plot time against power. SEE Web also supports the SDRAM viewer, which lets users access the system's main memory. Users can employ the SDRAM viewer for verification or debugging purposes.

SEE WEB is an active project, and we will regularly maintain and upgrade it. The current version includes the essential components to execute a program that doesn't require a specific peripheral. For applications that need a peripheral device, we will progressively add new components. Adding new peripherals to SEE Web requires the following straightforward steps:

- Derive a control-oriented, clock-cycle-accurate FSM.
- Measure the energy consumption using SNU Energy Characterizer (SEC) and SEC for FPGAs (SECF).
- Compose an energy FSM.
- Write a cycle-driven simulation code.
- Add the new device code.

The SEC and SECF cooperative tools are cycle-accurate energy measurement platforms based on the measurement technique explained in the "Cyclic-accurate energy annotation" section.[7,11] SEC handles memory and other peripheral devices. SECF is for Xilinx FPGA devices. Through cycle-by-cycle measurements, these tools derive dynamic energy and leakage power to annotate the energy FSMs.

SEE Web (http://see.snu.ac.kr) is now available for public use. ■

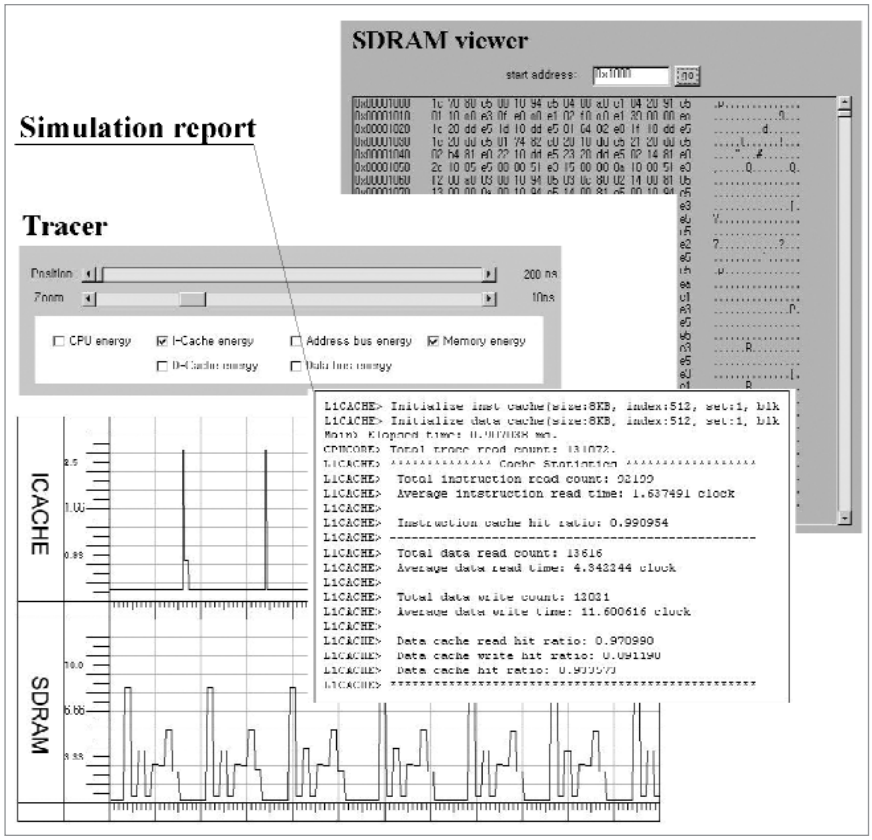

Figure 13. Snapshot of detailed energy consumption of a microprocessor, an instruction cache, and SDRAM devices.

Acknowledgments

The Research Institute of Computer Technology at Seoul National University provided research facilities for this study. The Brain Korea 21 Project also supported this work.

References

1. W.R. Hamburgen et al., "Itsy: Stretching the Bounds of Mobile Computing," *Computer*, vol. 34, no. 4, Apr. 2001, pp. 28-37.
2. V. Tiwari, S. Malik, and A. Wolfe, "Power Analysis of Embedded Software: A First Step Towards Software Power Minimization," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 2, no. 4, Dec. 1994, pp. 437-445.
3. A. Sinha and A. Chandrakasan, "JouleTrack: A Web Based Tool for Software Energy Profiling," *Proc. 38th Design Automation Conf.* (DAC 01), ACM Press, 2001, pp. 220-225.
4. W. Ye et al., "The Design and Use of SimplePower: A Cycle-Accurate Energy Estimation Tool," *Proc. 37th Design Automation Conf.* (DAC 00), ACM Press, 2000, pp. 340-345.
5. D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," *Proc. 27th Int'l Symp. Computer Architecture* (ISCA 00), ACM Press, 2000, pp. 83-94.
6. T. Simunic, L. Benini, and G. De Micheli, "Energy-Efficient Design of Battery-Powered Embedded Systems," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 1, Feb. 2001, pp. 15-28.
7. H. Shim et al., "Low-Energy Off-Chip SDRAM Memory Systems for Embedded Applications," *ACM Trans. Embedded Computing Systems* (TECS), vol. 2, no. 1, Feb. 2003, pp. 98-130.
8. N. Chang, K.-H. Kim, and H.G. Lee, "Cycle-Accurate Energy Measurement and Characterization with a Case Study of the ARM7TDMI," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 10, no. 2, Apr. 2002, pp. 146-154.

9. D. Shin et al., "Energy Monitoring Tool for Low-Power Embedded Programs," *IEEE Design & Test*, vol. 19, no. 4, July-Aug. 2002, pp. 7-17.
10. N. Chang et al., "Bus Encoding for Low-Power High-Performance Memory Systems," *Proc. 37th Design Automation Conf.* (DAC 00), ACM Press, 2000, pp. 800-805.
11. H.G. Lee, S. Nam, and N. Chang, "Cycle-Accurate Energy Measurement and High-Level Energy Characterization of FPGAs," *Proc. 4th Int'l Symp. Quality of Electronic Design* (ISQED 03), IEEE CS Press, 2003, pp. 267-272.
12. M. Huang et al., "L1 Data Cache Decomposition for Energy Efficiency," *Proc. Int'l Symp. Low Power Electronics and Design* (ISLPED 01), IEEE Press, 2001, pp. 10-15.

Ikhwan Lee is an assistant engineer at Samsung Electronics in Suwon, Korea. He performed the work described in this article while he was a graduate student at Seoul National University. His research interests include low-power systems design and embedded-systems design. Lee has a BS and an MS in computer science and engineering from Seoul National University.

Yongseok Choi is a PhD student in the School of Computer Science and Engineering at Seoul National University. His research interests include embedded-systems design and low-power systems design. He has a BS and an MS in computer science and engineering from Seoul National University.

Youngjin Cho is a PhD student in the School of Computer Science and Engineering at Seoul National University. His research interests include embedded-systems design and low-power systems design. He has a BS in naval architecture and ocean engineering, and a BS in computer science and engineering, both from Seoul National University. He is a student member of the IEEE.

Yongsoo Joo is a PhD student in the School of Computer Science and Engineering at Seoul National University. His research interests include embedded-systems design and system-level energy simulation. He has a BS and an MS in computer science and engineering from Seoul National University. He is a student member of the IEEE.

Hyeonmin Lim is a graduate student in the School of Computer Science and Engineering at Seoul National University. His research interests include embedded-systems design and low-power systems design. He has a BS in civil, urban, and geosystems engineering, and a BS in computer science and engineering, both from Seoul National University. He is a student member of the IEEE.

Hyung Gyu Lee is a PhD student in the School of Computer Science and Engineering at Seoul National University. His research interests include device-level energy measurement and characterization, system-level low-power design, and low-power FPGA design. He has a BS in computer engineering from Dongguk University, Seoul, Korea, and an MS in computer science and engineering from Seoul National University. He is a student member of the IEEE.

Hojun Shim is a PhD student in the School of Computer Science and Engineering at Seoul National University. His research interests include embedded-systems design and low-power systems design. He has a BS in computer science and engineering from Seoul National University. He is a student member of the IEEE.

Naehyuck Chang is an associate professor in the School of Computer Science and Engineering at Seoul National University. His research interests include system-level low-power design and embedded-systems design. He has a BS, an MS, and a PhD in control and instrumentation engineering from Seoul National University. He is a member of the IEEE and the ACM.

■ Direct questions and comments about this article to Naehyuck Chang, 506 Engineering Bldg., School of Computer Science and Engineering, College of Engineering, Seoul National University, Silim Dong, Kwanak Gu, Seoul, 151-010, Korea, naehyuck@cslab.snu.ac.kr.