Energy Estimation of Peripheral Devices in Embedded Systems

Ozgur Celebican
Center for Research on Embedded Systems and Technology,
Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA, 30332-0250, USA
(1) 404 3851722
celebi@ece.gatech.edu

Tajana Simunic Rosing
Hewlett-Packard Labs
Palo Alto, CA, 94304-1126, USA
(1) 650 725 3647
tajana@stanford.edu

Vincent J. Mooney III
Center for Research on Embedded Systems and Technology,
Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA, 30332-0250, USA
(1) 404 3850437
mooney@ece.gatech.edu

ABSTRACT

This paper introduces a methodology for estimation of energy consumption in peripherals such as audio and video devices. Peripherals can be responsible for significant amount of the energy consumption in current embedded systems. We introduce a cycle-accurate energy simulator and profiler capable of simulating peripheral devices. Our energy estimation tool for peripherals can be useful for hardware and software energy optimization of multimedia applications and device drivers. The simulator and profiler use cycle-accurate energy and performance models for peripheral devices with the cycle-accurate energy and performance models for computing, storage and power devices created in previous work. We also implemented I/O communication protocols such as polling, I/O interrupts and direct memory access (DMA). Using our energy simulator and estimator, we optimized an audio driver for an MP3 (MPEG-2 Layer 3) audio decoder application. Our optimization results show 44% reduction in the total system energy consumption for the MP3 audio decoder when optimized audio driver is used.

Categories and Subject Descriptors

C.4 [**Computer System Organization**]: Performance of Systems – *design studies, modeling techniques*

General Terms

Performance, Design, Experimentation, Verification.

Keywords

Device drivers, audio, energy estimation, software optimization

1. INTRODUCTION

Demand for more powerful embedded systems is increasing everyday. This demand propels companies to create faster, smaller and more capable products. With increasing speed and performance requirements, energy consumption of the system also increases. Increasing energy consumption requires more effort for heat dissipation and also may require larger batteries for a portable system. While increasing the energy capacity of the system is one approach to solve the problem, another approach is to optimize the energy consumption of the system. A significant ratio of the energy consumption for a state-of-the-art embedded system comes from peripheral devices such as audio, video or network devices. Up to now, energy optimization for peripheral devices has been done with restricted methods. One method is to add up datasheet values for each component. This method can give a rough estimate but cannot show effects of software. Another method is using prototypes. Prototypes can give exact energy and performance numbers, but the cost of the prototype and time spent building the prototype is huge. There are established tools for performance and energy simulations of embedded systems but none of them includes energy simulation of peripheral devices at the system level.

In this work we introduce a system level energy simulation and profiling tool for peripheral devices in an embedded system. With profiling, a software designer can discover which software routines consume most of the power. Also, our tool is capable of simulating two different types of protocols. These are polling and interrupt-based communication. Furthermore, Direct Memory Access (DMA) functionality is added for direct access between memory and peripherals. Each peripheral device is defined with a number of energy modes. For each mode an equivalent energy per cycle value is calculated from the power and performance values given in the manufacturer datasheets. Models introduced in a previous work from [1] are used to create energy models for processor, memory and power supply components of the system.

The rest of the paper is organized as follows. Section 2 gives an overview of related work. Section 3 describes our methodology. Section 4 presents the simulation results. Finally, we summarize our findings in Section 5.

2. RELATED WORK

There are some commercial system level simulation CAD tools for embedded systems [2][3][4][5], but all are limited to simulation of execution time. For energy simulation, the Synopsys Power Compiler [6] can be used for HDL designs but it is not practical for system level simulation. SimOS [7] is a system level performance simulation environment for both uniprocessor and multiprocessor systems. SimOS can simulate a computer hardware system, which can run a commercial operating system. However, SimOS does not have any energy simulation capabilities.

Much of the work on energy-driven optimization only considers the energy consumption of the processor alone [8][9][10][11]. In current embedded systems, the processor accounts for a limited ratio of the total energy budget. Energy optimization of memory and

GLSVLSI'04, April 26-28, 2004, Boston, Massachusetts, USA

communication systems between processor and memory are presented in [12][13][14][15]. SimplePower [16] and [1] present cycle-accurate energy simulators consisting of processor and memory modules but do not include peripheral devices. Peripheral devices such as video and wireless links have a considerable impact on energy consumption. This impact can be up to 60% of the total system energy consumption for the wireless link [17].

Wang et al. [18] define an automation method for device driver development in embedded systems. They define device driver behaviors using event driven finite state machines, with constraints and synthesis patterns. Their method optimizes device drivers for high reliability, productivity, reusability and fast time-to-market. There are no system level energy simulators that take peripherals into account.

Our work introduces an energy-driven optimization methodology using cycle-accurate energy simulation for peripheral devices. Datasheets provided by component manufacturers are used to create energy models for peripherals. Our cycle-accurate simulator enables simulation of real applications such as MP3 audio playback or MPEG video on cutting-edge embedded systems. Our work also includes energy profiling. The energy profiler shows the energy consumption in each software routine (e.g., device driver) by each hardware component (e.g., memory) including peripherals. The profiler can also be used to profile total system energy consumption. In the next section we describe our methodology for energy estimation.

3. SYSTEM MODEL

We can classify typical devices found in an embedded system into five groups. These are computing, storage, power, interface and peripheral devices. Figure 1 shows a sample embedded system classified into these five groups. In our simulation tool we are going to use this same classification. Energy models for computing, memory and power supply devices are implemented in previous work [1].

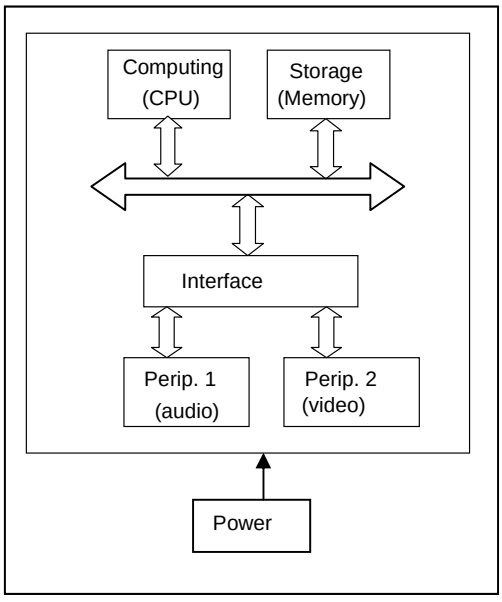

Figure 1. A typical embedded system and architecture of the simulator

In this work, we incorporate our energy and performance models for peripheral devices into ARMulator [19], which is a commercial cycle-accurate performance simulator for ARM processors. ARMulator itself is designed to accept such extra models provided that they use proper ARMulator Application Program Interfaces (APIs). Application software then can be cross-compiled with the ARM compiler provided and can be loaded to the simulator to obtain performance results. In our methodology, we add cycle-accurate energy models for each device in a target embedded system such that after performance simulation, energy consumption of individual devices and the entire system can also be obtained. Energy models introduced by [1] are used for processor, memory and power supply units. A simple representation of our energy simulator architecture is shown in Figure 1. In each cycle, ARMulator and external performance models for devices send state information of the devices plus address and data bus information to our energy simulator. Using state, address and data bus information, the energy simulator calculates energy consumption of each device in each cycle. In our energy models the cycle frequency is equal to system data bus speed. This is the maximum speed with which devices in the system such as the processor and the memory can communicate. For the rest of the paper we will refer to this as simulator cycle frequency. Each device has a distinct clock frequency (at least at the interface, in the case of an analog device, after the ADC). Energy consumed by busses between devices is also calculated using bus capacitance and switching activity; for each bus, this energy consumption value is added to the energy consumption of the device which is driving the bus. The energy consumption of the components in the power supply devices such as battery and voltage converters are calculated using their efficiency tables provided by the component manufacturers. System energy for a particular cycle is calculated using Equation 1.

$$E_{Cycle} = E_{Computing} + E_{Storage} + E_{Power} + E_{Interface} + E_{Peripheral} \quad (1)$$

Another function included in our simulator is an energy profiler. ARMulator has a performance profiler, which we modify for energy profiling. The energy profiler uses energy values found in our energy simulator. To operate the profiler, a user must define two distinct inputs before starting the energy simulator. One of these inputs is the time difference between each profiler step in microseconds. The second input decides which energy to profile. It is possible to profile the system energy consumption or the energy consumption of one of the devices in the system, namely, processor, memory, interface, peripheral or power supply unit energy consumption.

The processor can communicate with peripheral devices using one of two distinct methods. One method is using special communication instructions, and the other is memory-mapped communication. In this work we focus on memory-mapped peripherals but our methodology can also work with special processor instructions for peripherals. For memory-mapped peripherals, communication can be handled with two distinct protocols. These are polling-based and interrupt-based protocols. In this paper we study the energy advantages and disadvantages of both protocols. The Direct Memory Access (DMA) method can be used to handle the communication between memory and interface units. The processor stays in sleep state while such an access is processing, and at the end the processor is woken up by an interrupt. We also implement this type of communication in our simulator.

We next show the sample peripheral devices implemented to represent our methodology. These are a coprocessor as an I/O controller and an audio module consisting of an audio interface and an audio device.

3.1. I/O Controller

The I/O controller receives data from the processor and converts the data to the format required by the peripheral device. The I/O controller can be a coprocessor, an FPGA or an ASIC circuit designed specifically for this task. Alternatively, in some systems the processor itself handles the data conversion. The I/O controller can also directly communicate with memory via DMA. A high-level model of the I/O controller is shown in Figure 2. In this paper we model an SA-1111 coprocessor as the I/O processor.

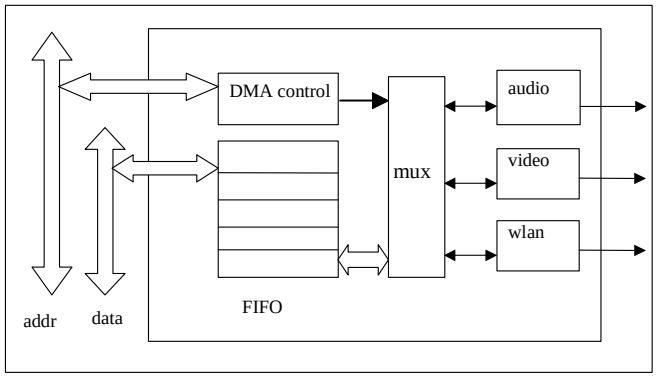

Figure 2: An I/O controller model

In our model, the I/O coprocessor has a queue and communication busses for communication with the processor, memory and peripheral devices. Also in our model the I/O controller handles DMA control for peripherals.

We create I/O controller energy models using datasheet values given by the manufacturer. There are two power states for the I/O controller, an active state and an idle state. The I/O processor is active when there is data transfer to/from the I/O processor, or when there is data stored in the I/O processor. If there is neither a data transfer nor any stored data, the I/O processor enters the idle state. An equivalent capacitance value for each state can be calculated using the supply voltage and the state current values as shown in Equation 2.

$$C_{coproc,state} = \frac{I_{coproc,state}}{V_{dd,coproc} * f_{coproc}} \qquad (2)$$

We can calculate in each system bus cycle the energy consumption for each state of the I/O controller from its equivalent capacitance value using Equation 3. N_{coproc} is the ratio of the simulator cycle frequency to the coprocessor clock frequency, f_{coproc}. By dividing the energy result with N_{coproc} in Equation 3, we distribute the energy consumption during one coprocessor cycle evenly among every simulator cycle; hence we find the energy consumption per one simulator cycle.

$$E_{coproc,state} = \frac{C_{coproc,state} * V_{dd,coproc}^2}{N_{coproc}} \qquad (3)$$

Distributing the energy consumption of one coprocessor cycle over multiple simulator cycles is deviating from the strict definition of cycle-accurate simulation because there is no guarantee that real energy consumption in a coprocessor cycle is evenly distributed over simulator cycles; unfortunately it is impossible to obtain the real energy distribution in one coprocessor cycle from datasheet information. Thus, we have to sacrifice some accuracy at particular clock cycles due to the use of datasheet modeling. However, [1] showed that macroscopic energy results (e.g., execution of thousands of software assembly instructions) are nonetheless fairly accurate (within 10% in [1]) when compared empirically with actual hardware measurements. On the positive side, this deviation from exact cycle accurate energy modeling is not an issue for the energy profiling of the software because if a device is slow, it will slow down the software routine using it too. The same issue will be seen in all low frequency peripherals and will be handled in a similar way.

3.2. Audio Module

An audio module contains two devices, an audio interface device and an audio device. The audio interface either converts digital data into an analog voltage level or else converts an analog voltage level into digital data. The audio device either creates sound from electrical voltage or else creates electrical voltage from sound. Datasheet values obtained from the audio interface and the audio device manufacturers are used to create energy models.

For the audio interface, three operation modes are defined: standby, digital to analog and analog to digital. An equivalent capacitance for each mode is calculated using datasheet values of supply voltage and mode current using equations similar to Equation 2. Energy per system bus cycle for each mode in the audio interface is calculated using Equation 4. N_{audio} is the ratio of the simulator cycle frequency to the audio interface device clock frequency.

$$E_{aud,\mathrm{mod}} = \frac{C_{audio,\mathrm{mod}} * V_{dd.audio}^2}{N_{audio}} \qquad (4)$$

The audio device, on the other hand can be a speaker, headphone or microphone. We model all of these different types as a capacitance and a resistance in parallel. Because of the low frequency threshold for hearing in the human ear, the analog voltage level, which creates the sound (or is created from the sound), is nearly constant for one simulator cycle. We assume it is constant and voltage change occurs just at the beginning of each simulator cycle. Equation 5 shows the energy consumption per simulator cycle for the sound device. R_{dev} and C_{dev} are the resistance and the capacitance parameters of the sound device, $f_{simulator}$ is the simulator cycle frequency, ΔV is the voltage difference of the audio signal between current and previous simulator cycles and V_{sample} is the voltage level for the current simulation cycle. Equation 6 shows how to calculate V_{sample}.

$$E_{dev} = \frac{V_{sample}^2}{R_{dev} * f_{simulator}} + C_{dev} * V_{sample} * \Delta V \qquad (5)$$

In A/Ds and D/As analog voltage input or output value is linearly proportional to the digital output or input. The maximum analog voltage value is the equivalent to the digital value 2^n where n is the number of bits used to represent the data. (In some conventions there will be an additional bit for the sign, but it is easy to convert such techniques to the technique used here.) Equation 6 shows how to calculate the analog voltage equivalent of any given digital d_{sample}.

$$V_{sample} = \frac{V_{dd} * d_{sample}}{2^n} \qquad (6)$$

We also need to calculate energy consumption on the interconnect between the I/O controller and the audio interface. This energy can be calculated using capacitance of the interconnect, the voltage level and the switching activity on the interconnect. The capacitance can be calculated using the material properties of the hardware platform if the length of the connection is known. Switching activity (N_{switch}) is obtained from the simulator on every cycle. Equation 7 shows the resulting interconnect energy per audio sample.

$$E_{connection} = C_{line} * V_{dd}^2 * N_{switch} \quad (7)$$

4. RESULTS

To experiment with our energy models we use an embedded system model shown in Figure 3, consisting of one processor, a memory system with three different types of memories, a coprocessor as an I/O processor, an audio interface, a speaker and a headphone. This model is derived from a Linux based development board SmartBadge IV [20]. SmartBadge IV has an SA-1110 processor, FLASH, SRAM and SDRAM for off-chip memory, an SA-1111 coprocessor and a UDA 1341 audio interface chip. Table 1 shows the datasheet values used for energy models for the system components in our experiment. Validation of the energy models used in the simulator are done in [1] for CPU, memory and power supply units. We plan to do the validation of the energy models used for peripheral devices as a future work.

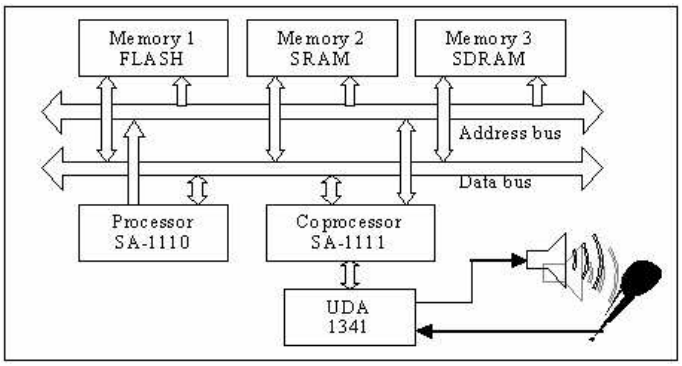

Figure 3: Simulated Embedded System

We use an MP3 audio decoder application to experiment with our energy simulator. In this example our aim is not to suggest a new energy optimization technique but to show how our energy simulator can help a designer to optimize driver code more easily. Our test input is a five second audio sample. Before testing the application we also design an audio driver. Although it is possible to evaluate operating system behavior on the simulator, we have not included such behavior in our experiments.

We start our audio application testing by first designing an audio driver. We base our initial design on the default audio device driver that is part of the SmartBadge IV Linux distribution [21]. An audio device driver controls the audio record and play operations. When recording (playing) an audio sample, the device driver receives (sends) data from (to) the peripheral using polling. Polling is done to determine if the queue in the I/O controller is not empty (not full). If it is not empty (not full) the processor reads (writes) data from the queue. If the queue is empty (full), the processor continuously checks the queue until there is some data (space) in the queue. This continues until all samples are recorded (played).

Table 1: Datasheet information about components

Type	Device	Modes and datasheet values
CPU	SA-1110	active = 500mW @ 200MHz
		idle = 85 mW @ 200MHz
		sleep = 5 uA
I/O Controller	SA-1111	active = 50mA @ 3.3V
		sleep = 50 uA @ 3.3V
Flash Mem.	28F800C3	read = 18 mA @ 5 MHz
		standby = 7 uA
SRAM	TC55V400FT−70	active = 50 mA @ 3V
		standby = 2 mA @ 3V
		low standby = 0.5 uA @ 3V
SDRAM	KMM466S924T	active = 480mA @ 3.3V
		standby = 120 mA @ 3.3V
		idle = 20mA @ 3.3V
Audio CODEC	UDA1341TS	playback = 20mA @ 3V
		record = 19.55mA @ 3V
		standby = 10.05mA @ 3V
Sound device	Speaker and microphone	R= 5K C=25pF (including termination resistor)

We tested our audio driver using a sample application. Our application first records a 0.1 second audio clip with 48KHz audio frequency. Then the same audio clip is played back. We average effects of both functions by testing both the record and the playback functions in the same application. The results from our simulator, which are shown in Table 2, show that most of the device-driver related system energy is wasted in continuous checking of the queue status. Profiling shows that while 96% of the device-driver related system energy is consumed during polling, only 3% of the device-driver related system energy is used for the audio data transfer in the driver. In addition, using the processor as a medium to transfer data between the memory and the device creates extra bus switching. As a result, we decided to redesign this device driver.

Table 2: Energy profile of polling based device driver

Routine	Energy %
check_fifo	96.29
to_fifo	1.30
from_fifo	1.29
main	0.29
flsbuf	0.02

Our new device driver uses DMA to communicate between the memory and the I/O controller. The processor goes to sleep while data is exchanged between the audio device and the memory. In the sleep mode all processor activities stop except the clock tree [22]. There is no useful computation performed in the CPU in this mode. Code for waking from sleep is stored in RAM to enable fast wake up. Interrupts are used to wake the processor when the data transfer is finished or when the size limit for DMA access is reached. Our simulation based profiling results shown in Table 3 show that nearly all the device-driver related system energy is consumed for the actual data transfer with DMA.

Table 3: Energy profile of DMA based device driver

Routine	Energy %
dma_transfer	98.78
flsbuf	0.49
fprintf	0.11
freopen	0.06
fputc	0.04

We can use the results of our energy simulator in these experiments to show the system energy saving between discussed audio drivers. Table 4 shows the performance and the total system energy consumption for both audio drivers. There is a 58% decrease in device-driver energy consumption with the DMA-based audio driver compared to the polling-based audio driver. As expected, there is little change in the execution time.

Table 4: Performance and total energy consumption

Device driver	Time(sec)	System Energy(mJ)
DMA-based	0.2017	173.02
Polling-based	0.2011	411.19

Our tool is also capable of showing energy consumption distribution for each device. The energy distribution for audio driver tests mentioned above is shown in Table 5. The energy consumption of the processor, the memory, the system bus and the coprocessor decrease with the DMA-based driver because of the increased sleep time for the processor, reduced number of instruction reads in the memory, and reduced amount of data transfer activity on the bus which also affects coprocessor pin power. The energy consumption for the audio device, which is composed of the audio interface and a microphone or a speaker, is nearly unchanged. The power source modules (battery and DC/DC converter) spend less energy because of the decrease in current amount demanded by the rest of the system.

Table 5: Energy consumption per module

	DMA-based		Polling-based		
	Energy (mJ)	%	Energy (mJ)	%	% diff.
Proc.	1.15	0.67	67.68	16.46	98.30
Mem.	22.61	13.06	100.66	24.48	77.56
SA1111	33.12	19.15	38.20	9.28	13.22
Sys. Bus	0.18	0.09	0.50	0.13	68.36
Audio D.	101.99	58.94	101.74	24.74	-0.24
DC_DC	12.67	7.33	21.24	5.17	40.31
Battery loss	1.33	0.76	81.18	19.74	98.38

To see how much the improvement in the energy consumption of the audio driver affects energy consumption of a real application we integrate our two audio drivers into an MP3 audio decoder application. The total system energy consumption and energy consumption of the individual components for MP3 decode with each audio driver is shown in Table 6. Because of the decreased audio driver energy consumption, total energy consumption of processor, memory, system bus and power supply components (DC/DC converter and battery) are also decreased for MP3 decode implemented on our hardware (Figure 3). Our DMA-based audio driver achieves a 44% reduction in the total system energy consumption for an MP3 decoder. The percentage energy reduction for the MP3 decoder is less than for the device-driver because the energy consumption of the system and the components are the same for both the polling-based and the DMA-based drivers when the decoder is processing audio data.

Table 6: Energy consumption of MP3 decoder with different audio drivers

Audio driver	Polling-based		DMA-based		% diff.
	Energy (J)	%	Energy (J)	%	
Proc.	2.59	18.43	0.86	10.95	66.74
Mem.	3.30	23.47	1.27	16.15	61.49
SA1111	1.06	7.55	0.93	11.81	12.44
Sys. Bus	0.02	0.16	0.01	0.19	36.18
Audio D.	3.25	23.09	3.25	41.33	-0.20
DC_DC	0.76	5.42	0.54	6.84	29.26
Battery loss	3.08	21.89	1.00	12.73	67.45
System	14.06		7.87		44.03

It is also desirable to show the distribution of energy consumption for MP3 decode per software function. In this paper we try to observe the effect of peripheral devices and their drivers, thus it is reasonable to compare the energy consumption of an audio driver with the remaining functions which are used to decode the MP3 data. As shown in Table 7 for MP3 decode with a polling-based driver, 76% of the total system energy consumption is spent in the audio driver. For MP3 decode with a DMA-based driver, on the other hand, this ratio drops to 57%.

Table 7: Energy profile of MP3 decoder with different audio drivers

	Polling-based	DMA-based
Device driver (%)	76.02	57.16
Data processing (%)	23.98	42.84

5. CONCLUSION AND FUTURE WORK

We presented an energy consumption estimation methodology for peripheral devices such as audio devices in embedded systems. We designed a cycle-accurate simulator and profiler for energy consumption estimation of the peripherals. We also added a profiler, which can be helpful during software optimization, to our simulator. We tested our idea with memory-mapped I/O devices using both polling and interrupts for communication. We further modeled DMA type access to memory to enable direct communication between memory and peripherals. The results of our simulations show that up to 50-55% of the total system energy consumption can be spent in peripheral devices. We also show that our tool can help optimize energy consumption with an example MP3 decoder application. We optimized the audio driver in the MP3 decoder and therefore, reduced the total energy consumption of the MP3 decoder by 44%.

Future work will include adding video and wireless link device models to the simulation system. Also, validation of the simulation results will be done using SmartBadge IV. Finally, operation system issues will also be addressed.

6. REFERENCES

[1] T. Simunic, L. Benini and G. De Micheli, "Cycle-Accurate Simulation of Energy Consumption in Embedded Systems," *Proceedings of the 36th Design Aautomation Conference*, pp. 867-872, June 1999.

[2] CoWare, HTTP:http://oradev.coware.com

[3] Mentor Graphics, HTTP: http://www.mentor.com/codesign/
[4] Synopsys, System-level design, HTTP: http://www.synopsys.com/sps/sld.html
[5] Cadence, HTTP: http://www.cadence.com
[6] Synopsys, Power Compiler, HTTP: http://www.synopsys.com/products/power/power.html
[7] SimOS, HTTP: http://simos.stanford.edu/
[8] V. Tiwari, S. Malik and A. Wolfe, "Power Analysis of Embedded Software: a First Step Towards Software Power Minimization," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 2, Issue 4, pp. 437-445, 1994.
[9] A. Bona, M. Sami, D. Sciuto, C. Silvano, V. Zaccaria and R. Zafalon, "Energy Estimation and Optimization of Embedded VLIW Processors Based on Instruction Clustering," *Proceedings of the 39th Design Automation Conference*, pp. 886-891, June 2000.
[10] J. T. Russell and M. F. Jacome, "Software Power Estimation and Optimization for High Performance, 32-bit Embedded Processors," *Proceedings of the Int. Conference on Computer Design: VLSI in Computers and Processors (ICCD'98)*, pp. 328-333, October 1998.
[11] V. Srinivasan, D. Brooks, M. Gschwind, P. Bose, V. Zyuban and P. N. Strenski, P. G. Emma, "Optimizing Pipelines for Power and Performance," *Proceedings of the 35th Annual ACM/IEEE International Symposium on Microarchitecture*, pp. 333-344, November 2002.
[12] N. Vijaykrishnan, M. Kandemir, M. J. Irwin, H. S. Kim and W. Ye, **"**Energy-driven Integrated Hardware-software Optimizations Using SimplePower," *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pp. 95-106, June 2000.
[13] Erik Brockmeyer, Arnout Vandecappelle and Francky Catthoor**, "**Systematic Cycle Budget Versus System Power Trade-off: A New Perspective on System Exploration of Real-time Data-dominated Applications," *Proceedings of the 2000 International Symposium on Low Power Electronics and Design*, pp. 137-142, July 2000.
[14] Tao Li and Lizy Kurian John, "Run-time Modeling and Estimation of Operating Power Consumption," *Proceedings of the International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'03)*, pp. 160-171, June 2003.
[15] T. Simunic, L. Benini, G. De Micheli and M. Hans, "Source Code Optimization and Profiling of Energy Consumption in Embedded Systems," *Proceedings of the 13th International Symposium on System Synthesis*, pp 193-198, September 2000.
[16] W. Ye, N. Vijaykrishnan, M. Kandemir and M. J. Irwin, "The Design and Use of SimplePower: A Cycle-Accurate Energy Estimation Tool," *Proceedings of the 37th Design Automation Conference*, pp. 340-345, June 2000.
[17] A. Acquaviva, T. Simunic, V. Deolalikar and S. Roy, "Server Controlled Power Management for Wireless Portable Devices," HP Labs Technical Report (HPL-2003-82), April 2003.
[18] S. Wang, S. Malik and A. Bergamaschi, "Modeling and Integration of Peripheral Devices in Embedded Systems," *Proceedings of the Design, Automation and Test in Europe Conference*, pp. 136-141, March 2003.
[19] Advanced RISC Machines Ltd. (ARM) ARM Software Development Toolkit Version 2.11, 1996.
[20] SmartBadge 4, HTTP: http://www.it.kth.se/~maguire/badge4.html
[21] SmartBadge 4 audio driver, HTTP: http://www.it.kth.se/~maguire/badge4-audio.html
[22] Intel Strong ARM SA-1110 manual, HTTP: http://www.intel.com/design/strong/manuals/278240.htm