Energy Estimation of Peripheral Devices in Embedded Systems

Ozgur Celebican, Tajana Simunic Rosing
Mobile and Media Systems Laboratory
HP Laboratories Palo Alto
HPL-2003-251
December 2nd , 2003*

energy, estimation, device drivers, embedded systems

This paper presents a methodology for estimation of energy consumption in peripherals such as audio and video devices. In current embedded systems peripherals can be responsible for significant amount of the energy consumption. We introduce a cycle-accurate energy simulator and profiler capable of simulating peripheral devices. Our energy estimation tool for peripherals can help with hardware and software energy optimization of multimedia applications and device drivers. Our tool uses 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. I/O communication protocols such as polling, I/O interrupts and direct memory access (DMA) are implemented. To demonstrate benefits of our estimation scheme we compared two different types of audio drivers, one using interrupt driven direct memory access and the other using polling. Results show 57% of reduction in total system energy consumption for an audio driver.

* Internal Accession Date Only
Approved for External Publication
© Copyright Hewlett-Packard Company 2003

Energy Estimation of Peripheral Devices in Embedded Systems

Ozgur Celebican and Tajana Simunic Rosing
Hewlett-Packard Labs, Palo Alto

Abstract— **This paper presents a methodology for estimation of energy consumption in peripherals such as audio and video devices. In current embedded systems peripherals can be responsible for significant amount of the energy consumption. We introduce a cycle-accurate energy simulator and profiler capable of simulating peripheral devices. Our energy estimation tool for peripherals can help with hardware and software energy optimization of multimedia applications and device drivers. Our tool uses 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. I/O communication protocols such as polling, I/O interrupts and direct memory access (DMA) are implemented. To demonstrate benefits of our estimation scheme we compared two different types of audio drivers, one using interrupt driven direct memory access and the other using polling. Results show 57% of reduction in total system energy consumption for an audio driver.**

***Index Terms*—device drivers, energy estimation, software optimization, embedded systems.**

I. INTRODUCTION

Energy-efficiency is important for both portable and non-portable embedded systems. In portable systems the goal is to extend battery lifetime. In non-portable systems it is to reduce the cost of cooling. The energy-efficiency of an embedded system must be increased without hindering cost and time to market constraints. Significant ratio of energy consumption in portable systems comes from peripheral devices such as audio, video and wireless link. Up to now optimizing energy consumption of peripheral devices has been done in an ad hoc manner. Adding up datasheet energy values for each component is a commonly used solution. Often optimization is also done using prototypes. Prototype testing gives the exact power and performance analysis of the embedded system but the cost of the prototype and time spent for its development makes it impossible to try all possible solutions for a commercial product. There is considerable number of established tools for performance simulation of embedded systems; but a few include energy simulation. There are no tools to simulate energy performance at system level for peripheral devices.

In our work, we introduce a tool capable of simulating and profiling of cycle accurate energy and performance models for peripheral devices such as audio. Using profiling a software designer can determine which routines in the program flow are consuming a lot of power. Two different types of communication protocols between processor and peripherals are implemented. These are polling and interrupt-based communication. Direct memory access (DMA) is also implemented to enable direct access between memory and peripherals. Each I/O component is characterized with different operation modes. For each mode an equivalent energy per cycle value is calculated from the power and performance values given in the manufacturer datasheets. Models from Simunic et. al [1] are used to create processor, memory and power supply components of the system. Choice of the communication protocol affects the energy consumption of the system up to 57% as shown in the results

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

II. RELATED WORK

Several commercial CAD tools focus on integration of the system components [2][3][4][5], but are limited to performance simulation. Synopsys Power Compiler [6] is a tool for estimating energy consumption for HDL designs Power Compiler is a circuit level simulator, which calculates the energy consumption using switching information of the circuit. Its complexity increases exponentially with the size of the design and this 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, it does not have any energy simulation capabilities.

There is a considerable amount of work on energy-driven optimization of embedded systems. Much of the work just considers the energy consumption of the processor alone [8][9][10][11]. In current embedded systems, processor accounts for a limited ratio of the total energy budget. Energy optimization of memory and communication systems between processor and memory are presented in [12][13][14][15]. Simunic et. al. [1] and SimplePower [17] present cycle-accurate energy simulators consisting of processor and

memory modules However, current portable embedded systems often run multimedia applications, which require multiple peripheral devices. The peripheral devices such as video and wireless link have a considerable impact on energy consumption. This impact can be up to 60% of the total system energy consumption for the wireless link [19].

A method for optimizing peripheral devices and their drivers is defined in Wang et. al.[16]. In this work device driver behaviors are specified using event driven finite machines, with constraints and synthesis patterns. A device driver is synthesized automatically using the constraints given. The goal is to automatically create platform independent device drivers, which can be mapped to a specific platform with little effort. There are no system level energy simulators that take peripherals into account.

Our work presents an energy-driven optimization methodology using cycle-accurate energy simulation for peripheral devices. Energy models for such devices are created from datasheets provided by manufacturers. Cycle-accurate simulator enables simulating real applications such as MP3 audio playback or MPEG video on cutting-edge embedded systems. Another advantage of our work is the energy profiling. Energy profiler shows how much energy is spent in each software routine (e.g. device driver) by each hardware component (e.g. processor) including peripherals. Total system energy consumption can also be profiled. In the next section we describe our methodology for energy estimation.

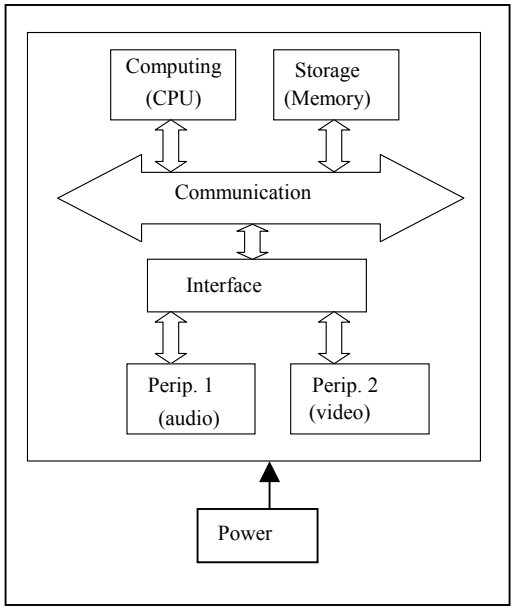

Figure 1: Simulator architecture

III. SYSTEM MODEL

A typical embedded system consists of computing, storage, peripheral and power devices as shown in Figure 1. In this work we introduce general cycle-accurate energy and performance models for the peripherals. Previous work implemented cycle-accurate models of computing, storage and power elements [1].

In current systems there are two commonly used styles of peripheral operation [20]. One of these methods is using special I/O instructions in CPU to activate the peripheral. The other method is using memory-mapped I/O. In this case portions of address space are assigned to I/O device and processor communicates with the device by reading from or writing to those addresses. The abstract behavior of a peripheral device for different communication schemes is shown in Figure 2.

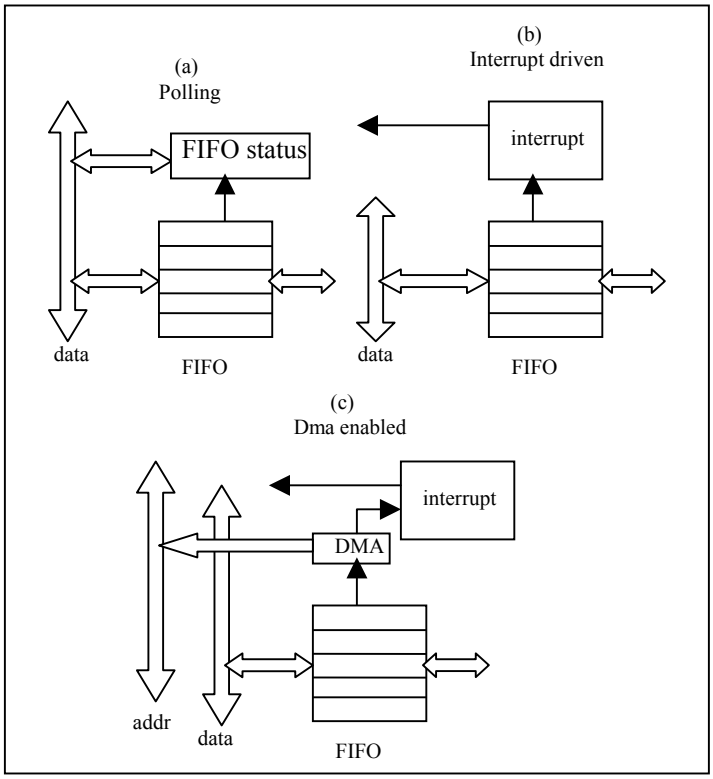

Figure 2: I/O peripheral controller model.

Peripheral devices have two ways to signal back to the CPU: polling and interrupts. In polling, peripheral device writes the data into a status register and the processor periodically checks the status register. Advantages of polling are, it is simple to implement and the processor is in control all the time. The disadvantage is polling crates overhead in CPU time and energy consumption. Polling is implemented in a cycle-accurate simulator as a memory read and a status check in a loop, which runs until the necessary peripheral status is reached. CPU will be active all the time.

The other method is using I/O interrupts. In this case peripheral device creates an I/O interrupt when it requires processor cycles. The advantage is that the processor is occupied only when required. On the other hand, special hardware is needed to create and detect an interrupt. In addition, for each interrupt processor needs to save its state. This is implemented on a cycle-accurate simulator with an interrupt routine and a delay in simulator. Processor is in idle state or runs any other task until it is interrupted.

A common method used with interrupts is direct memory access (DMA). DMA is very helpful if the processor is just a

medium for communication between memory and the peripheral. In such case DMA enables memory and peripheral to communicate directly. DMA controller is the master and the communication is done external to the processor. DMA is implemented in a cycle-accurate simulator with direct data transfer between two memory regions. The processor is idle while transfer is done .

We implemented a general memory-mapped peripheral device energy model. We focused on memory-mapped I/O because it is very common in the current system implementations. Both polling and interrupt-based communication between processor and peripherals are implemented. Also we include DMA capability. The energy and performance models for peripherals are added to ARMulator [18], a commercial performance simulator for ARM processors. To add a new module to the ARMulator, designer must provide a simple cycle-accurate functional and performance model for each system component. Then application software can be cross-compiled with provided compiler and compiled software is loaded to the simulator to get performance statistics. Cycle-accurate energy models for each component have been added. Energy models are based on datasheets provided by component manufacturers. Energy models introduced by Simunic et. al. [1] are included in our work.

Simulator architecture is shown in Figure 1. In each cycle ARMulator sends information about state of the processor and data and address bus values to the external modules. Processor cycles can be classified in two groups; active cycles where processor is operating, and idle cycles where processor is waiting for memory or peripheral access. Using the processor state, address and data bus information from ARMulator, each module determines its state and its energy consumption in that state. Switching activity on each bus is calculated for each cycle. Energy loss on the bus is calculated using bus capacitance and switching activity. Two power devices are represented: the DC/DC converter and the battery. Both calculate the energy loss from the efficiency tables. Total energy consumption per cycle is sum of component energy consumption values as shown in Equation 1.

$$E_{Cycle} = E_{Computing.} + E_{Storage} + E_{power} + E_{peripherals} \quad (1)$$

The energy and performance profiler is used to obtain the distribution of the energy consumption per software routine of a specific hardware unit as well as the overall system. In each cycle, the profiler determines the currently executing procedure and adds the energy increment between this and previous cycle to energy consumption of that software routine. Profiler cycle frequency is determined before the execution of the code by the user. At the end, the profiler reports energy distribution with percentages.

We next show a sample implementation of a peripheral system in our simulator consisting of a coprocessor as I/O controller and an ADC converter as an audio module. They combine to drive a sound device such as microphone or speaker. These devices are typical example of peripherals found in embedded systems such as portable MP3 players.

A. I/O controller

I/O controller receives data from processor and converts it to the format needed by peripheral hardware. In some systems a coprocessor is used as an I/O controller. In other cases an FPGA or ASIC is used or even in some cases main processor itself handles I/O controller functionality. I/O controller can communicate with direct memory access (DMA) to decrease interconnect switching activity and also to free the main processor for other tasks. A high-level model of the I/O controller is shown in Figure 3. In our model, coprocessor has a queue, which can communicate with the processor or memory and a communication channel between the queue and audio/video device. When memory access is finished, coprocessor creates an interrupt to activate the processor. If the operation is recording to memory, DMA control waits until FIFO is filled to a user-specified threshold value and then uses burst access to write into memory. When the system is playing audio/video from memory using DMA, memory writes a burst of data to the FIFO. Using burst accesses decreases the time the bus is busy and also decreases energy consumption on the address bus.

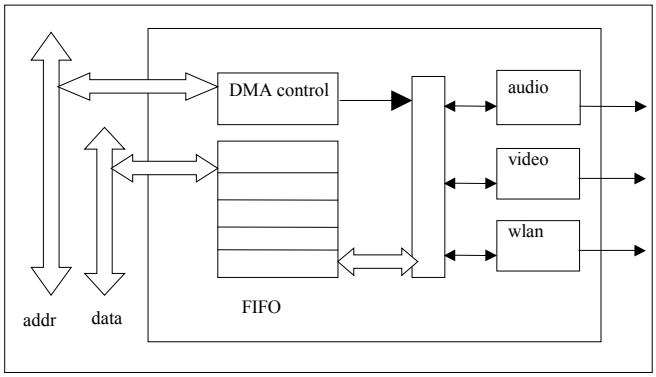

Figure 3: An I/O controller model

I/O controller energy model is created from datasheet information given by the manufacturer. There are two power modes for coprocessor: active and idle. Using the supply voltage and current information given in datasheets equivalent capacitance values for each mode can be found from Equation 2.

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

Using this capacitance, the energy consumption per cycle in active mode can be calculated from Equation 3.

$$E_{coprocactive} = \frac{C_{coprocactive} * V_{dd.coproc}^2}{N_{coproc}} \quad (3)$$

Coprocessor is modeled as idle when there is no access to peripheral devices. In such case energy consumption of coprocessor per coprocessor cycle can be calculated from Equation 4.

$$E_{coproc,idle} = \frac{C_{coproc,idle} * V_{dd.coproc}^2}{N_{coproc}} \quad (4)$$

N_{coproc} is the ratio of bus frequency to coprocessor frequency.

B. Audio Module

Audio module converts digital information to appropriate analog voltage level and sends it to the sound device, or receives analog data and converts it to digital before transferring it to the system. In our model we combine the audio controller with the audio device itself. To model energy consumption of an audio device per cycle we use the information given in datasheets.

Three operation modes for audio device are defined: standby, digital to analog and analog to digital. For each mode an equivalent capacitance is calculated using source voltage, current and clock frequency of the device. Equations used for capacitance calculations are similar to Equation 2. Energy per cycle for each mode is calculated using Equation 5. N_{aud} is the ratio of bus frequency to audio device clock frequency.

$$E_{aud,mod} = \frac{C_{aud\,mod} * V_{dd.aud}^2}{N_{aud}} \quad (5)$$

Another factor in energy consumption is the sound device itself. It can be a speaker, headphone or microphone. A capacitance or resistance or both in parallel model each of these devices. Equation 6 shows the energy consumption per audio sample for the sound device. R_{dev} and C_{dev} are resistance and capacitance parameters of the sound device, f_{audio} is audio frequency, ΔV is the voltage difference between consecutive samples and V_{sample} is the voltage level of the sample. Equation 7 shows how to calculate V_{sample}.

$$E_{dev} = \frac{V_{sample}^2}{R_{dev} * f_{audio}} + C_{dev} * V_{sample} * \Delta V \quad (6)$$

In D/As or A/Ds analog voltage level is proportional to the digital data value and maximum digital data is generally equal to supply voltage of the converter. In Equation 7 analog voltage level equivalent of a digital sample with value d_{sample} is given. Number of bits used to represent digital data is n.

$$V_{sample} = \frac{V_{dd} * d_{sample}}{2^n} \quad (7)$$

Finally energy lost on the interconnect between the audio module and the I/O controller are calculated using the capacitance of the interconnect, the voltage level and the switching activity. Capacitance can be calculated if the length of the connection is known using the material properties of the hardware platform. Switching activity (N_{switch}) per sample is obtained from the simulator on every cycle. Equation 8 shows the resulting interconnect energy per audio sample.

$$E_{connection} = C_{line} * V_{dd}^2 * N_{switch} \quad (8)$$

IV. RESULTS

To demonstrate our energy models we used a Linux based embedded system developed by Hewlett-Packard Labs, called SmartBadge IV. SmartBadge IV consists of an ARM processor (SA-1110), three different types of memory (FLASH, SRAM and SDRAM), a coprocessor (SA-1111) and an audio interface (UDA 1341). Table 1 shows the datasheet values used for energy models for the components.

Table 1: Datasheet information about components

Type	Device	Mode and characteristic of the mode
CPU	SA-1100	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)

To test our methodology we started with a default audio device driver that is a part of the SmartBadge IV Linux distribution. The device driver can both record and play audio. When recording (playing) 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) processor reads (writes) a data from the queue. If it is empty (full) processor continuously check the queue until there is some data (space) in it. This continues until all samples are recorded (played).

Our profiling tool highlights energy wasted in continuous checking of the queue status in Table 2. Profiling shows that with the default audio driver only 3% of the energy is used for the audio data transfer, while 96% of the energy is wasted for polling. In addition, using processor as a medium to send data between the memory and the device creates extra bus switching. As a result, we redesigned 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

New device driver uses DMA to communicate between memory and I/O controller. Interrupts are used to inform the processor when the operation is done or when the size limit for DMA access is reached. Profiler results in Table 3 show that nearly all 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 further compare the two device drivers with a simple application. A 0.1 sec audio clip with 48KHz audio frequency is sent from an audio device and stored into memory. Same audio is then played back. Table 4 shows the performance and total system energy consumption.. There is a 58% decrease in energy consumption with DMA-based audio driver. As expected, there is little change in execution time, which is nearly the same as the time spent to receive and send audio sample.

Table 4: Performance and total energy consumption

Device driver	Time(sec)	System Energy(uWhr)
DMA-based	0.2017	48.06
Polling-based	0.2011	114.22
% difference	0.34	57.93

Energy consumption for each device is shown in Table 5 and Figure 4. DMA-based driver decreases the energy consumption of processor, memory, system bus and coprocessor. Because we used the same sample clip for testing both audio drivers, the energy consumption of microphone and speaker are unchanged. Power source modules (battery and DC/DC converter) spend less energy because of the decrease in current demanded by the rest of the system. CPU energy consumption decreases because processor is in the idle state during the DMA access. Memory power is decreased because of the reduced number of instruction reads. As can be seen from these results, with our tool we can not only detect potential problem areas in hardware and software design of the peripherals but can also solve them.

Table 5: Energy consumption per module

	DMA		Polling		
	Energy (uWhr)	%	Energy (uWhr)	%	% diff.
Proc.	0.32	0.67	18.80	16.46	98.30
Mem.	6.28	13.06	27.96	24.48	77.56
SA1111	9.20	19.15	10.61	9.28	13.22
Sys. Bus	0.05	0.09	0.14	0.13	68.36
Audio D.	28.33	58.94	28.26	24.74	-0.24
DC_DC	3.52	7.33	5.90	5.17	40.31
Battery loss	0.37	0.76	22.55	19.74	98.38

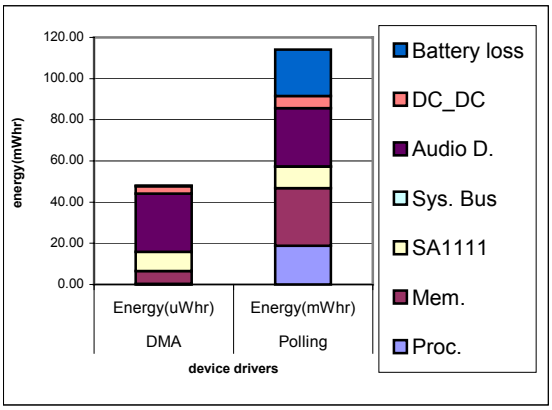

Figure 4: Distribution of energy consumption to different modules

V. CONCLUSION

In this work we present a methodology for estimating energy consumption of peripherals such as audio device in embedded systems. We develop a cycle-accurate simulator and profiler to estimate the energy consumption of the peripherals. Profiler enables optimization of the driver software. A memory-mapped I/O device is modeled to test our method. Communication of peripheral device and CPU is implemented both using polling and interrupts. Also DMA is modeled to enable direct communication between storage devices and peripherals. Our simulation shows that power consumption in such peripheral devices and their communication interface with other parts of embedded system can count for up to 70-75% of total system energy consumption. We also show using our tool that it is possible to optimize energy consumption in device drivers. We decreased system energy consumption of the audio driver 57% by just changing the communication protocol between I/O controller, main CPU and memory.

VI. REFERENCES

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

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

[3] Mentor Graphics, http://www.mentor.com/codesign/
[4] Synopsys, System-level design http://www.synopsys.com/sps/sld.html
[5] Cadence, http://www.cadence.com/products/incisive_unified_simulator.html
[6] Synopsys, Power Compiler http://www.synopsys.com/products/power/power.html
[7] SimOS http://simos.stanford.edu/
[8] V. Tiwari, S. Malik, A. Wolfe, "Power Analysis of Embedded Software: a First Step Towards Software Power Minimization," *IEEE Transaction on Very Large Scale Integration (VLSI) Systems*, 1994, v. 2, no. 4, pp 437-445
[9] A. Bona, M. Sami, D. Sciuto, C. Silvano, V. Zaccaria, R. Zafalon, "Energy Estimation and Optimization of Embedded VLIW Processors Based on Instruction Clustering*," Proceedings of 39th Design Automnation Conference*, 2002, pp 886-891.
[10] J. T. Russell, 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)*, 1998, pp. 328-333.
[11] V. Srinivasan, D. Brooks, M. Gschwind, P. Bose, V. Zyuban, P. N. Strenski, P. G. Emma, "Optimizing Pipelines for Power and Performance," *Proceedings of the 35th Annual ACM/IEEE International Symposium on Microarchitecture*, 2002, pp.333-344.
[12] N. Vijaykrishnan, M. Kandemir, M. J. Irwin, H. S. Kim, W. Ye, **"**Energy-driven Integrated Hardware-software Optimizations Using SimplePower," *Proceedings of the 27th Annual International Symposium on Computer Architecture*, 2000, pp. 95-106.
[13] Erik Brockmeyer, Arnout Vandecappelle, 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*, 2000, pp 137-142.
[14] Tao Li, 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)*, 2003, pp 160-171.
[15] T. Simunic, L. Benini, G. De Micheli, M. Hans, "Source Code Optimization and Profiling of Energy Consumption in Embedded Systems," *Proceedings of the 13th International Symposium on System Synthesis*, 2000, pp 193-198.
[16] S. Wang, S. Malik, A. Bergamaschi, "Modeling and Integration of Peripheral Devices in Embedded Systems," *Proceedings of Design, Automation and Test in Europe Conference,* 2003, pp. 136-141.
[17] W. Ye, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, "The Design and Use of SimplePower: A Cycle-Accurate Energy Estimation Tool," *Proceedings of 37th Design Automation Conference, 2000, pp. 340-345.*
[18] Advanced RISC Machines Ltd. (ARM) ARM Software Development Toolkit Version 2.11, 1996.
[19] Acquaviva, Andrea; Simunic, Tajana; Deolalikar, Vinay; Roy, Sumit, "Server Controlled Power Management for Wireless Portable Devices," *HP Labs Technical Report (HPL-2003-82)*, April 2003.
[20] David A. Patterson and John L. Hennessy, "Computer Organization and Design: The Hardware/Software Interface," Second Edition, August 1997, publisher Morgan Kaufmann.