Instruction-Level Power Dissipation in the Intel XScale Embedded Microprocessor

Ankush Varma[†‡], Eric Debes[‡], Igor Kozintsev[‡] and Bruce Jacob[†]

[†]University of Maryland, College Park, MD;
[‡]Intel Labs, Santa Clara, CA

ABSTRACT

We present an instruction-level power dissipation model of the Intel XScale® microprocessor. The XScale implements the ARM™ISA, but uses an aggressive microarchitecture and a SIMD Wireless MMX™co-processor to speed up execution of multimedia workloads in the embedded domain.

Instruction-Level power modelling was first proposed by Tiwari et. al. in 1994. Adaptations of this model have been found to be applicable to simple ARM processors. Research also shows that instructions can be clustered into groups with similar energy characteristics. We adapt these methodologies to the significantly more complex XScale processor.

We characterize the processor in terms of the energy costs of opcode execution, operand values, pipeline stalls etc. through accurate measurements on hardware. This instruction-based (rather than microarchitectural) approach allows us to build a high-speed power-accurate simulator that runs at MIPS-range speeds, while achieving accuracy better than 5%.

The processor core accounts only for a portion of overall power consumption, and we move beyond the core to explore the issues involved in building a SystemC simulation framework that models power dissipation of complete systems quickly, flexibly and accurately.

Keywords: XScale, Power, SystemC, Instruction-Level

1. INTRODUCTION

Power is a primary design constraint for embedded systems, and accurate power estimation tools are required to ensure that power envelope and battery life constraints are satisfied. Such estimates are useful feedback tools to allow code optimizations for minimization of energy consumption.

The Intel *XScale* [1] is a family of microprocessors that implement the ARM ISA, use deep pipelines and microarchitectural optimizations for high performance, and feature a WMMX (Wireless MMX) SIMD co-processor optimized for wireless and multimedia applications in mobile platforms. To the best of our knowledge, the 8-stage XScale is the most complex processor to have been characterized for instruction-level power modelling so far. We present a fast instruction-level power model of the XScale microprocessor that takes a wide variety of energy effects into account in order to provide highly accurate power estimates of processor power consumption. We implement this power model in SystemC, and integrate it into an accurate system simulation framework in order to achieve power estimates that are accurate within 5%, yet provide simulation speeds over 1 MIPS.

A variety of tools exist for power analysis of microprocessors at the microarchitectural level. *Wattch* [2] is built on the *SimpleScalar* [3] processor simulation framework and it provides power models for register files, caches and other microprocessor components. *SimplePower* [4], also based on *SimpleScalar* provides a tool for microarchitectural and circuit-level power estimation of on-chip buses, caches and other components. Chen

Further author information, send correspondence to:
Ankush Varma (ankush@eng.umd.edu)
Eric Debes (eric.debes@intel.com)

†‡This work was done by Ankush Varma as an intern at the Intel Labs, Santa Clara, CA.

et. al. [5] also describe a microarchitectural scheme for microprocessor power estimation, while [6] describes a detailed microarchitectural power model of the XScale microprocessor core. Microarchitectural power models rely on the availability of detailed knowledge of the internal microarchitecture of a processor. This information changes rapidly with each processor generation, and is usually not in the public domain. The vast amount of detail required for modeling modern processors also makes such analysis slow (typically KIPS).

Instruction-level microprocessor power consumption was first explored by Tiwari et. al. [7, 8], who described how instruction-level power can be characterized from hardware measurements. Sinha et. al. [9] perform energy profiling of ARM processors and also describe how leakage power can be estimated by plotting processor power at various frequencies. This provides a route to building power models that are suited for high-speed simulation and do not necessarily rely on microarchitectural information.

Instruction-level power studies of ARM ISA processors have been performed earlier[9–14]. However, none of these accurately characterize or model external memory accesses or stalls. While this is acceptable for small benchmarks, a real OS and application can spend a significant number of cycles in such states. We accurately model both instructions and *events* such as stalls and memory accesses, in order to create a power model that is sufficiently accurate for modeling an full-featured OS and complete applications running over billions of clock cycles.

2. CHARACTERIZATION METHODOLOGY

Tiwari et. al. [7] describe how the energy consumption for each instruction, and for inter-instruction effects, can be obtained by running a repeated instruction sequence in a loop and measuring the energy consumed by the processor. For accuracy, the loop must be short enough to fit entirely in the instruction cache, yet long enough that the branch at the end of the loop has a negligible effect on accuracy. In practice, loops of 100–1000 instructions are typically used. We use such programs (which we refer to as *stimuli*) to characterize a variety of energy-related instruction-level effects. A stimulus sets up the processor into a desired initial state and then runs a long loop of containing repeated instances of a short instruction sequence. The loops can be run for an arbitrary number of iterations.

We ran the stimuli on hardware to obtain the parameters for the XScale power model. However, this approach is not limited to post-silicon characterization. A variety of RTL and micro-architectural power modeling tools exist, and stimuli can be easily run on these instead of hardware to extract power model parameters. Based on these, we create a power model incorporating the energy consumption patterns observed, and implement it in SystemC to create an execution-driven power model of the complex Intel XScale processor, its WMMX coprocessor and L1 caches.

3. THE XSCALE MICROPROCESSOR POWER MODEL

We create an accurate,instruction-level power model of the XScale microprocessor. It incorporates a number of energy-relevant *effects*, with the energy consumed by a given instruction being the sum of the energies due to all the applicable effects. We describe all effects and the stimuli used to characterize them in this section.

3.1. Leakage Power and Voltage-Frequency Scaling

The XScale processor provides a large number of voltage/frequency settings. We run a given stimulus at a fixed voltage and vary the frequency, obtaining a linear plot. Static power dissipation is estimated by extending this curve to obtain power consumption at zero frequency, as described in [9]. Static power is the sum of leakage power and bias currents, which are small for the XScale. Power is then given by:

$$P = P_{static} + P_{dynamic} = V I_{leakage} + \frac{1}{2} C_L V_{dd}^2 f \quad (1)$$

3.2. Low-Power States

We also characterize power consumption of the processor in various low-power modes, such as *idle, deep idle, standby, sleep* and *deep sleep*. Each of these states is described in [1] and the power consumption in each state can be characterized easily through an instruction sequence that puts the processor into the desired low-power state.

3.3. Instruction Opcode

Based on functionality, the instructions were divided into 11 different types (*add, b, cdp, cmp, dummy, ldr, logical, mov, mul, str* and *sub*), similar to [9]. *Dummy* refers to conditionally predicated instructions that were fetched but not committed. Each energy cost was measured using straightforward stimuli running the same instruction repeatedly with zero operands.

3.4. Operand Value

The effect of the value of the operands on the energy consumed to execute an instruction. Energy tends to increase roughly linearly with the operand value and the number of "1"s in the operand. Such effects are also observed in [6, 13]. This effect can be observed and characterized by comparing the average power consumptions of stimuli that differ in the values of the operands they are adding, but are identical in other respects. Stimuli that yield power parameters based on their differences rather than the actual values of power observed are referred to as *differential stimuli.*

3.5. Bypass Paths

A rather interesting pattern of bypass path behavior was observed, with three different cases:

1. The base case is when there are no inter-instruction dependencies and all source operands are obtained through register file reads. For example, running *add r6, r7, r8* repeatedly.
2. When all source registers for an instruction are the destination registers for a previous instruction, the source operands are obtained from bypass paths and 4% *less* energy than the base case is used. An example of this is executing *add r6, r6, r6* over and over.
3. When both the bypass paths and the register file are used to get source operands, 5% *more* energy than the base case is used. An example of this would be executing *add r6, r6, r7* repeatedly.

To the best of our knowledge, this effect has not been characterized before.

3.6. Sign Update and Conditional Flags

Instructions which updated or used the conditional flags consumed more energy than corresponding instructions which did not. For example an *adds* or *addeq* instruction consumes more energy than a simple *add.* However, this increase is under 0.5% and so it has not been made part of the power model.

3.7. Register Switching

When two consecutive instructions use different source or destination registers, an energy overhead is incurred depending upon the number of registers switched. This can exceed 10% of instruction energy and can be expected to be incurred often. To the best of our knowledge, this effect has been not been used in instruction-level power models before. This effect is characterized by running a series of differential stimuli, each of which changes a different number of registers between consecutive instructions, with all other factors being constant.

3.8. Cache Accesses

Caches are modeled as on-chip SRAM. From the instruction-set point of view, the energy cost of a load or store depends on the number of bytes accessed. We characterize and model this. The relevant stimuli load or store a given number of bytes to a given address. We observed negligible address and data dependance in cache accesses.

3.9. Shifts

The ARM instruction set [15] allows the last operand of an instruction to be bit-shifted by either an immediate or a register value. This shift causes an additional result latency of one cycle and consumes additional energy. We divide shifts into three distinct types: *RRX* (Rotate Right eXtended) with takes no operands, shifts by an integer operand, and shifts by a register operand. The incremental cost of a given type of shift can be extracted using differential stimuli that compare the energy cost of an instruction with and without the shift.

3.10. Stalls

Stalls can be divided into *instruction stalls*, which are due to inter-instruction data dependencies, *event stalls*, such as stalls on a double-word load, *branch stalls*, or the pipeline flush penalty, and *memory stalls* on accesses to external memory. Energy costs of all stall types were characterized and modeled.

3.11. The WMMX Coprocessor

The XScale processor family has an on-die Wireless MMX coprocessor [16] for fast SIMD processing. We divided the WMMX instructions into 17 types based simply on functionality, in a manner similar to that for the main processor. WMMX instructions clusters that were close in functionality and energy cost were then merged, reducing the number of instruction clusters to 12: *bit, pack, unpack, logical, arith, cmp, mul, maxmin, read, write, load* and *store*. Base costs for WMMX instructions were then characterized and built into the power model.

4. IMPLEMENTATION

We create an execution-driven implementation of this power model. In this approach, an existing performance simulator is modified to provide information about the instruction stream, cache accesses, stalls etc. to a power model as they occur.

We use *Xsim*, a C-based cycle-count accurate performance simulator for the XScale family. It models all XScale instructions, the L1 caches and the WMMX coprocessor. The fetch and retire times of each instruction are computed by tracking dependencies and resource constraints instead of detailed pipeline modeling. Xsim has been validated to be cycle-accurate within 2% of actual hardware and achieves speeds over 10 MIPS when run as a stand-alone processor simulator. We modify Xsim to enable its use as a SystemC component which can be used to execute instructions on a cycle-by-cycle basis or a specified number of cycles at a time.

In addition, we use SystemC models of SDRAM, buses and arbitration to ensure that system performance is simulated accurately, and all latencies are correctly modeled. An accurate system-wide performance simulation is needed to realistically model the system's behavior (bus latencies, arbitration etc.) *even if only microprocessor power is being studied*. This is because microprocessor activity, such memory stall cycles, interrupts etc., depends on the latency and behavior of other system components, which must be modeled correctly for the processor simulation to be accurate. The XScale power model plugs directly into this to maintain instruction-by-instruction computation of core power consumption. However, an existing performance model can be easily leveraged to build in power models of other components too. We are currently investigating such a system modeling approach, and preliminary results suggest that this is highly feasible.

5. EXPERIMENTAL SETUP

For validation, we use a reference platform (Figure 1) featuring an XScale-based PXA271 SoC which contains an XScale processor, its WMMX coprocessor, L1 instruction and data caches (32KB each) and other system components [1]. The platform also has 64MB on-board SDRAM, 32MB synchronous Flash and a variety of peripherals. The main board is instrumented with 100mΩ resistors in series with the power supply on each module, which enable power measurements of individual components.

We measure the power consumption of the processor using an NI-DaQ data acquisition card, sampling at up to 20KHz. Post-processing of acquired data is done using LabView virtual instruments. Processor frequency is varied, while the memory controller runs the off-chip bus at 91MHz.

We use Windows CE as the operating system, and run identical benchmarks on the hardware and the simulator. The simulator runs a complete OS boot routine followed by the application. Each benchmark is run in a loop, with average power measured physically on hardware over a period of one second and compared with the estimate obtained from the simulator.

To validate our results, we use the following benchmarks:

- *Autocorrelation* and *Huffman Decoding* benchmarks from the EEMBC benchmark suite.

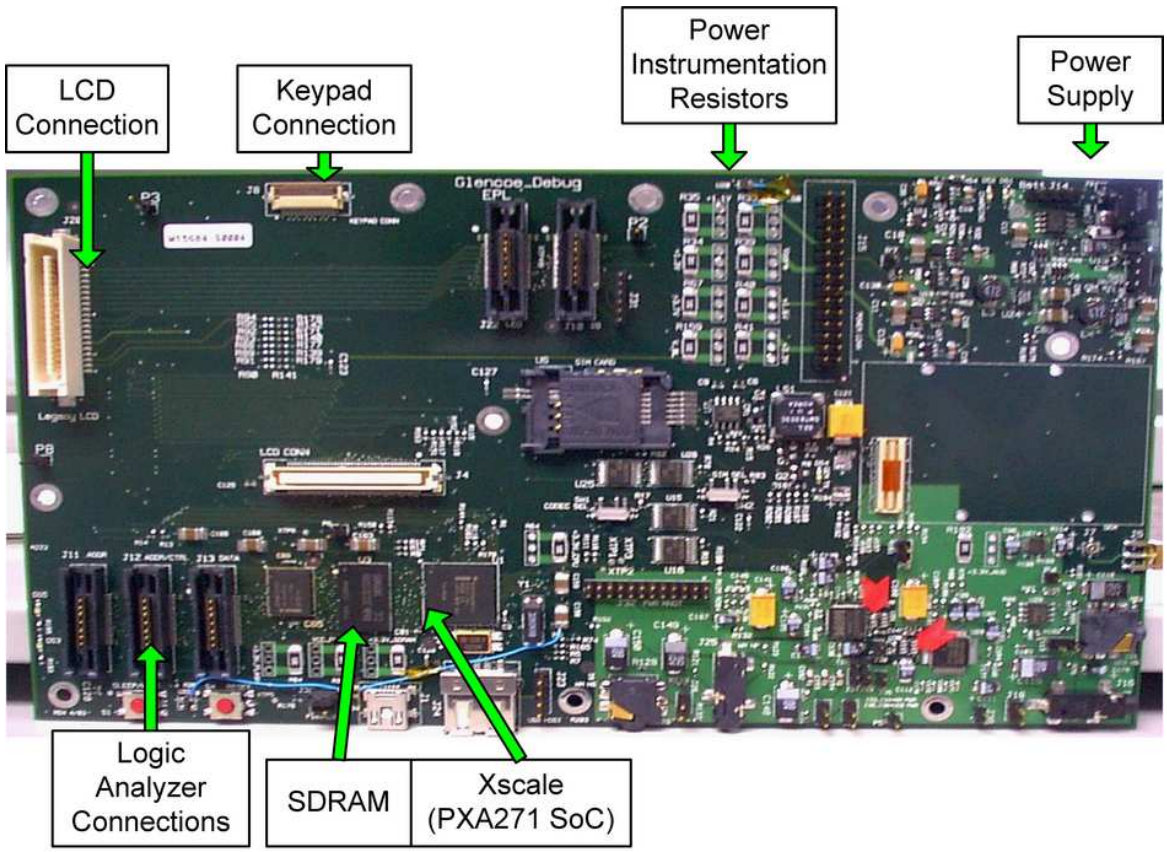


Figure 1: The reference platform used for physical experiments. The XScale processor, the WMMX unit and the L1 caches are on the PXA271 SoC. The Logic Analyzer connections allow bus signals and timing to be observed, while an array of power instrumentation resistors allows the power supply of various components to be studied.

- The *Motion Estimation* kernel from an H.264 video encoder.
- A video filter (*vidsp*) from an H.264 video decoder. We evaluate three versions of this filter: plain C, hand-optimized assembly, and hand-optimized assembly with additional WMMX optimizations.
- *FFT*(for 10,000 samples), JPEG Forward DCT (*JFDCT*) and Matrix Multiply (*MatMul*) benchmarks from SNU-RT benchmark suite from Seoul National University. *MatMul* is a simple cache-resident benchmark.

6. RESULTS

Figure 2 compares the power consumption physically measured on the core power consumption for all benchmarks. Variations in power across benchmarks were tracked accurately. The *worst-case* error was 4.46%, for the *Motion Estimation* benchmark. In particular, the three versions of the *vidsp* benchmark were correctly ranked, indicating good relative accuracy as well.

One of the advantages of a simulation-based approach is the ability to study the processor in great detail and obtain information that cannot be revealed through simple hardware measurements. Figure 3 shows the various contributors to XScale power consumption for two benchmarks. Note that *Huffman Decoding* is far more memory intensive than *MatMul*, and spends comparatively more energy stalled on memory accesses and less on cache accesses or opcode execution. In contrast, hardware measurements can only reveal *net* power, not the fine level of detail seen here.

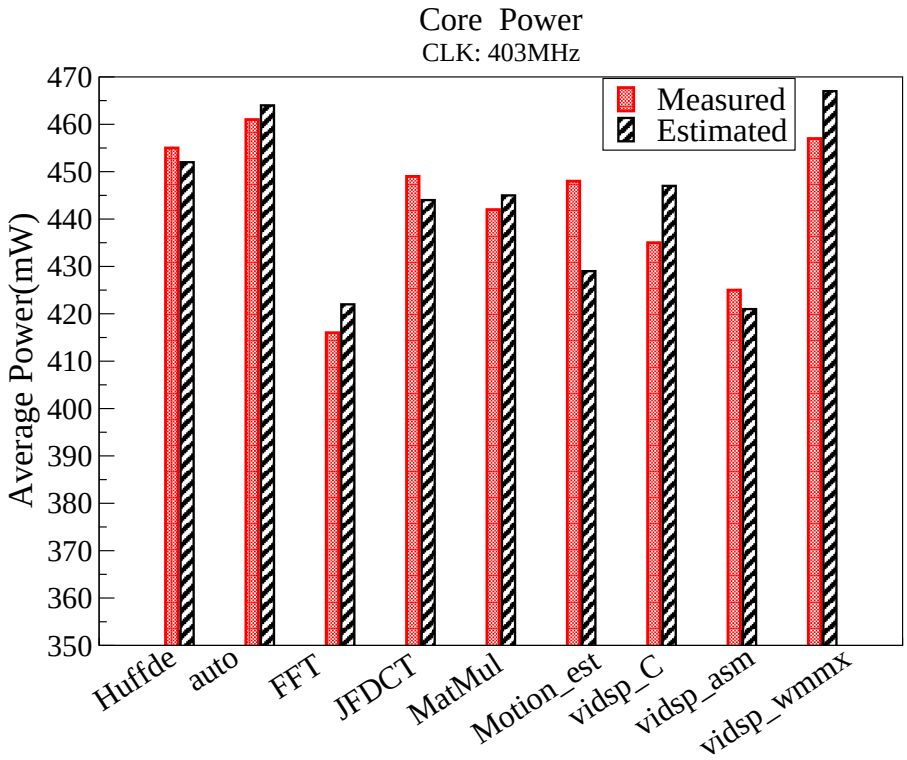


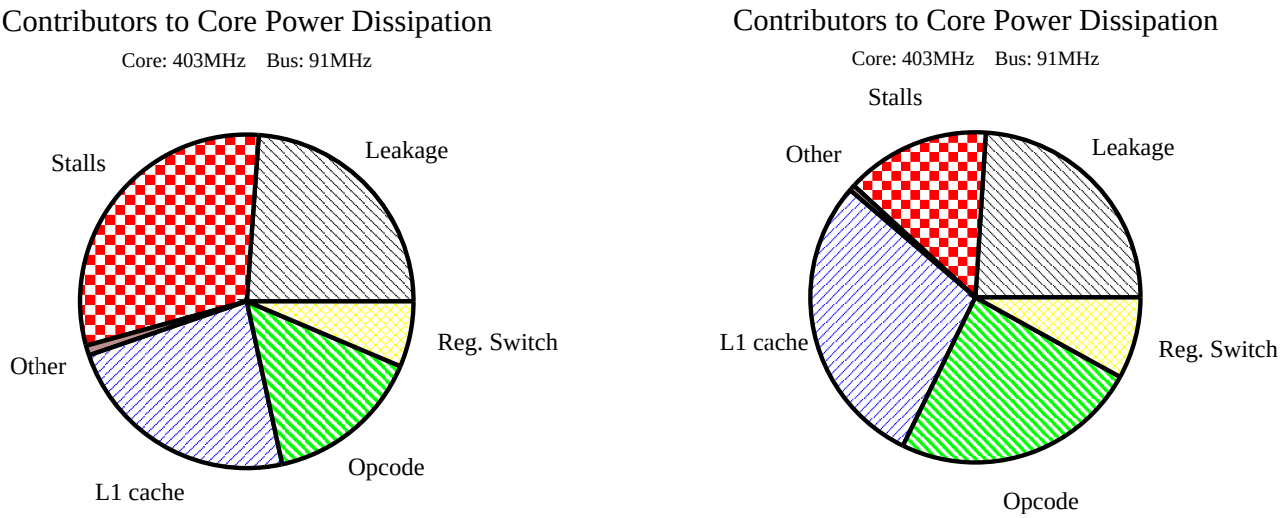
Figure 2: Power consumption at 403MHz for various benchmarks. Bus frequency is 91MHz. Note that power is tracked accurately, with variations between the three versions of the *vidsp* benchmark correctly estimated by the simulator.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we describe a detailed and accurate power model of the XScale processor core, including the WMMX co-processor and L1 caches. We obtained excellent agreement between the estimates obtained from the power model and physical measurements on hardware, with worst-case errors under 5%. The simulation speeds we obtained were in excess of 1 MIPS. We are now focusing on extending this approach by modelling the power consumption of other system components in order to obtain system-wide power-performance co-simulation.

ACKNOWLEDGMENTS

The Xsim simulator was built by Brett Gaines and modified for SystemC by Bhaktha Keshavachar. SystemC integration was done in collaboration with Paul Klein. The reference board used for physical experiments was provided by Jerzy Kolinski and Thao Xiong. Windows CE ports were provided by Sai Prasad, Peter Adamson, Thomas Cronin and Josh Miller. The video filter for H.264 decoding was created by Bob Reese.



(a) Huffman Decoding

(b) Matrix Multiply

Figure 3: Contributors to core power consumption for *Huffde* and *MatMul* benchmarks. Note that *Huffde* causes more memory traffic, and so spends more power and time stalled than *MatMul*, which spends more energy on instruction opcode execution and cache accesses.

REFERENCES

1. Intel, *Intel PXA27x Processor Family Developers Manual*, 2004.
2. D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architecture-level power analysis and optimization," in *Intl. Symp. on Computer Architecture*, 2000.
3. T. Austin, E. Larson, and D. Ernst, "SimpleScalar: An infrastructure for computer system modeling," *IEEE Computer* **35**, pp. 59–67, Feb. 2002.
4. W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "The design and use of SimplePower: A cycle-accurate energy estimation tool," in *Design Automation Conference*, 2000.
5. R. Y. Chen, M. J. Irwin, and R. S. Bajwa, "Architecture-level power estimation and design experiments," *ACM Transactions on Design Automation of Embedded Systems* , 2001.
6. G. Contreras, M. Martonosi, J. Peng, R. Ju, and G.-Y. Lueh, "XTREM: A power simulator for the Intel XScale," in *Languages, Compilers, and Tools for Embedded Systems*, 2004.
7. V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: A first step towards software power minimization," *IEEE Transactions on VLSI Systems* , 1994.
8. V. Tiwari, S. Malik, A. Wolfe, and M. T.-C. Lee, "Instruction-level power analysis and optimization of software," in *IEEE International Conference on VLSI Design*, 1996.
9. A. Sinha and A. P. Chandrakasan, "JouleTrack - a web based tool for software energy profiling," in *Design Automation Conference*, 2001.
10. N. Chang, K. Kim, and H. G. Lee, "Cycle-accurate energy consumption measurement and analysis: Case study of arm7tdmi," in *Intl. Symp. on Low-Power Electronics and Design*, 2000.

11. S. Lee, A. Ermedahl, and S. L. Min, "An accurate instruction-level energy consumption model for embedded risc processors," in *ACM SIGPLAN workshop on Languages, compilers and tools for embedded systems*, 2001.
12. S. Nikolaidis, N. Kavvadias, T. Laoppoulos, L. Bisdounis, and S. Blionas, "Instruction level energy modeling for pipelined processors," in *International Workshop on Power And Timing Modeling, Optimization and Simulation*, 2003.
13. G. Sinevriotis, A. Leventis, D. Anastasiadou, C. Stavroulopoulos, T. Papadopoulos, T. Antonakopoulos, and T. Stouraitis, "SOFLOPO: Towards systematic software exploitation for low-power designs," in *Intl. Symp. on Low-Power Electronics and Design*, 2000.
14. S. Steinke, M. Knauer, L. Wehmeyer, and P. Marwedel, "An accurate and fine grain instruction-level energy model supporting software optimizations," in *International Workshop on Power And Timing Modeling, Optimization and Simulation*, 2001.
15. ARM, *ARM Instruction Set Quick Reference Card v2.1*, 2003.
16. N. Paver, B. Aldrich, and M. Khan, *Programming with Intel Wireless MMX Technology: A Developer's Guide to Mobile Multimedia Applications*, Intel Press, 2004.