Available online at www.sciencedirect.com

SCIENCE DIRECT®

Computer Standards & Interfaces 28 (2005) 150–158

COMPUTER STANDARDS & INTERFACES

www.elsevier.com/locate/csi

Developing an environment for embedded software energy estimation

S. Nikolaidis[a], A. Chatzigeorgiou[b], T. Laopoulos[a,*]

[a]*Department of Physics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece*
[b]*Department of Applied Informatics, University of Macedonia, 54006 Thessaloniki, Greece*

Available online 14 March 2005

Abstract

The paper presents the results of a novel method for the instruction-level energy consumption measurement and the corresponding modeling approach for embedded microprocessors. According to the proposed method the base and inter-instruction energy costs of the ARM7TDMI embedded processor as well as the energy cost due to different values in the instruction parameters are modeled. These models can be used in the estimation of the energy consumed by the processor to execute real software programs. A software tool has been developed to automate energy estimation.

Keywords: Embedded microprocessors; Low power systems; Power consumption; Software modeling tools

1. Introduction

A large number of embedded computing applications are power or energy critical, that is power constraints form an important part of the design specification. Early work on processor analysis had focused on performance improvement without determining the power–performance tradeoffs. Recently, significant research in low power design and power estimation and analysis has been developed.

Embedded software power modeling techniques are distinguished into two main categories: a) physical *measurement-based* and b) *simulation-based* ones. In *simulation-based* methods, energy consumed by software is estimated by calculating the energy consumption of various components in the target processor through simulations. The main drawback of these simulation-based techniques is the need of information about the circuit level design of the processor, which is usually not available. In *measurement-based* approaches [1–6], the energy consumption of software is characterized by data obtained from real hardware. The advantage of measurement-based approaches is that the resulting energy model proves to be closer to the actual energy behavior of the processor.

In measurement techniques, a common practice is to associate instructions running on the processor with their corresponding energy cost. The majority of work published on the field of measurement-based techni-

* Corresponding author.
E-mail address: laopoulos@physics.auth.gr (T. Laopoulos).

0920-5489/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.csi.2005.01.016

ques refers to the Tiwari method [1,2] as a base point. By this method only average power estimates can be utilized for modeling tasks, since the measurements are taken with a standard digital ammeter. Loops of hundreds of instances of the same instruction are performed on the processor and the average drawn current is used to model the energy consumption of the instruction. However, such experiments do not correspond to real processor execution conditions.

In Ref. [3], physical measurements for the processor current are also obtained by a precise amperemeter. However, power modeling effort is more sophisticated, as architectural-level model parameters are introduced and integrated within the power model. These consist of the weight of instruction fields or data words, the Hamming-distance between adjacent ones, and basic costs for accessing the CPU, external memory and activating/deactivating functional units.

The above techniques acquire the current drawn by the processor on instruction execution. A complex circuit topology for cycle-accurate energy measurement is proposed in Ref. [4], which is based on instrumenting charge transfer using switched capacitors. The switches repeat on/off actions alternately. A switched capacitor is charged with the power supply voltage during a clock cycle and is discharged during the next cycle powering the processor. The change in the voltage level across the capacitors is proportional to the square of the consumed energy and this value is used for the calculation of energy in a clock cycle. However, this method employs a very complex measuring instrumentation.

A new instruction-level energy modeling methodology for simple in-order pipelined processors has been proposed by the authors in Ref. [5] aiming at the creation of highly accurate models. The derived energy models are used for software energy consumption estimation. The proposed methodology is based on measurements of the instantaneous current of the processor. A simple measuring environment was established for this purpose [6].

In this paper the results of our experiments are presented and the achieved accuracy of our method is given. A short description of the proposed modeling approach is also given. In addition, the implementation of a software tool for the estimation of the energy consumed during the execution of a given program is described.

2. Instruction-level energy modeling

The proposed method is based on the measurement of the instantaneous current drawn by the processor during the execution of the instructions. The instrumentation set-up for measuring the instantaneous current has been presented in Ref. [6]. The current sensing circuit is a high performance current mirror, which copies the drawn current on a resistor. By using the current mirror, supply voltage fluctuation problems, which degrade measurements are eliminated. A high-speed digitized oscilloscope is used for monitoring the voltage drop on the resistor. Taking the voltage waveform and making the appropriate calculations, the energy consumption at each clock cycle can be derived. The method is developed for pipelined processors like the ARM7 (three-stage pipeline).

The energy consumed during the execution of instructions can be distinguished in two amounts. The *base cost*, which is the energy amount needed for the execution of the operations which are imposed by the instructions, and the *inter-instruction cost* which corresponds to an energy overhead due to the changes in the state of the processor provoked by the successive execution of different instructions [1]. Measurements for determining these two energy amounts for each instruction of the ARM7TDMI processor have been performed and presented in Ref. [7]. However the base costs in Ref. [7] were for specific operand and address values (zero operand and immediate values and specific address values to minimize the effect of 1s). This base cost is called *pure base cost*.

We have observed in our measurements that there is a strong dependency of the energy consumption of the instructions on the values of their parameters (operand values, addresses) [3,4]. To create accurate models this dependency has to be determined. Additional measurements have been performed to satisfy this necessity. By incorporating these effects in our models the proposed method keeps its promised accuracy while it becomes very attractive since it can be easily implemented in software as an estimation tool.

Through appropriate experiments we have observed that the effect of each energy-sensitive factor on the energy cost of the instruction is independent of the effect of the other factors [3–5]. The distortion of our

results from this conclusion is, most of the time, less than 2–3% and only in some marginal cases becomes more than 7%. According to this conclusion, the effect of the energy-sensitive factors can simply be summed to give the total energy amount.

Other sources of energy consumption are conditions of the processor, which lead to an overhead in clock cycles because of the appearance of idle cycles. This is the case of the appearance of pipeline stalls. The effect of such cases on the energy consumption was measured and modeled.

According to the above, the energy, E_i, consumed during the execution of the i instruction can be modeled as:

$$E_i = b_i + \sum_i a_{i,j} N_{i,j} \quad (1)$$

where b_i is the pure base cost of the i instruction, $a_{i,j}$ and $N_{i,j}$ is the coefficient and the number of 1s of the j energy-sensitive factor of the i instruction, respectively.

Having modeled the energy cost of the instructions, the energy consumed for executing a program of n instructions can be estimated:

$$PE = \sum_{i=1}^{n} E_i + \sum_{i=1}^{n-1} O_{i,j+1} + \sum_{\#\ of\ pipeline\ stalls} \varepsilon \quad (2)$$

where $O_{i,j}$ is the inter-instruction cost of the instructions i and j, and ε is the cost of a pipeline stall.

3. Pure base cost and inter-instruction cost models—results

For measuring the pure base costs, loops with instances of a reference instruction (the NOP was used) and the one test instruction were executed. The condition of the pipeline structure of the ARM7TDMI processor during the execution of such loops is shown in Table 1. The energy of the test instruction was calculated as the sum of the energy consumed in the clock cycles required for this instruction to be executed minus two times the energy budget of the NOP instruction [5]. Due to the pipeline structure, two NOP instructions are also executed in the clock cycles needed for the execution of a test instruction as it can be observed in Table 1 for n+1, n+2 and n+3 cycles. Complex instructions which need more cycles are modeled in the same way. They use more than three cycles (multi-cycle instructions) but only two NOP are also performed during their execution. It should be noted that, although the proposed method is applied for the ARM7TDMI, its application to processors with more pipeline stages is straightforward.

Table 1
Pipeline states during the execution of instructions for measuring base costs

Pipeline stages	3-Stage pipeline operation				
IF	NOP	NOP	Instr	NOP	NOP
ID	NOP	NOP	NOP	Instr	NOP
EX	NOP	NOP	NOP	NOP	Instr
Clock cycles	$n-1$	n	n+1	n+2	n+3

Table 2
Pipeline states during the execution of instructions for measuring inter-instruction costs

Pipeline stages	3-Stage pipeline operation				
IF	NOP	Instr1	Instr2	NOP	NOP
ID	NOP	NOP	Instr1	Instr2	NOP
EX	NOP	NOP	NOP	Instr1	Instr2
Clock cycles	n	n+1	n+2	n+3	n+4

Since the energy budget of a program corresponds to the sum of the energy of each instruction, such a model seems to lead to an overestimation because instead of modeling the circuit state changes from one instruction to another, we model every instruction as a circuit state change (change from test to NOP instruction), which results in counting the number of circuit state changes twice. To overcome this problem the inter-instruction effect is also modeled in a similar way, and it is expected to take also negative values compensating this overhead. Consequently, the calculated inter-instruction energy costs, according to the proposed method, are a correcting factor rather than a real value of the corresponding energy amount, and they are expected to take also negative values.

In the measurements for the inter-instruction costs, program loops featuring appropriate instruction pairs were formed. The condition of the pipeline states is shown in Table 2. In this case, in four clock cycles one *Instr1*, one *Instr2* and two reference instructions are executed.

The total energy consumed when a program is executed can be estimated by summing the base costs of the instructions and the inter-instruction costs. Doing that, it is observed that the inter-cycle costs arisen by the followed measurement approach cancel each other and finally a sum including only the actual energy components of each instruction and the corresponding inter-instruction effects appears.

The complete models for the instruction-level energy consumption of the ARM7TDMI created according to the proposed methodology can be found in Ref. [7]. Some thousands of experiments according to the execution model presented in Table 1 have been performed. Pure base costs of all the instructions and for all the addressing modes are given. Since the number of the possible instruction pairs (taking into account the addressing modes) is enormous, groups of instructions and groups of addressing modes according to the resources they utilize, have been formed and inter-instruction costs have been given only for representatives from these groups [1]. In this way we keep the size of the required model values reasonable without significant degradation of the accuracy (less than 5% in the inter-instruction cost by using only representative instructions).

Results for the pure base cost of data processing instructions for all the addressing modes of the processor are given in Table 3. The type of the addressing mode, which corresponds to the referred number in table, is given in Ref. [7]. Most of the values of the pure base costs present a difference less than 20% in the energy of the instructions which are executed in the same number of cycles.

Most of the values of the inter-instruction costs have a negative sign as it was expected. The contribution of the inter-instruction costs, as they are calculated according the proposed method, remains

Table 3
Pure base costs (nJ) for the data processing instructions

Instruction	Addressing mode										
	1	2	3	4	5	6	7	8	9	10	11
ADC	2.07	0.94	2.09	0.98	2.06	0.92	2.05	0.92	0.89	0.90	0.94
ADCS	2.11	0.99	2.16	1.03	2.10	1.00	2.11	1.00	0.93	0.94	0.98
ADD	2.14	1.06	2.17	1.08	2.14	1.05	2.13	1.05	0.99	0.93	0.98
ADDS	2.09	1.00	2.10	1.02	2.06	1.00	2.05	1.00	0.94	0.92	0.91
AND	2.10	0.87	2.09	0.91	2.11	0.91	2.09	0.93	0.86	0.83	0.91
ANDS	2.19	0.97	2.21	0.98	2.20	0.99	2.19	0.96	0.90	0.88	0.95
BIC	1.90	0.71	1.96	0.72	1.91	0.70	1.92	0.70	0.64	0.61	0.68
BICS	1.99	0.75	2.03	0.77	1.97	0.73	1.96	0.75	0.69	0.66	0.64
CMN	2.11	1.03	2.13	1.06	2.12	1.02	2.11	1.03	0.96	0.95	1.01
CMP	2.03	0.91	2.06	0.93	2.03	0.90	2.04	0.91	0.83	0.83	0.90
EOR	2.16	1.01	2.02	1.07	2.02	1.05	2.05	1.06	0.96	0.94	1.02
EORS	2.09	1.11	2.06	1.11	2.08	1.11	2.05	1.07	1.02	1.02	1.10
MOV	2.10	1.01	2.15	1.04	2.11	1.01	2.10	0.99	0.94	0.93	0.98
MOVS	2.15	1.03	1.98	1.11	1.19	1.06	2.17	1.06	1.01	0.99	1.05
MVN	1.96	0.83	1.99	0.85	1.94	0.81	1.95	0.81	0.75	0.74	0.82
MVNS	2.03	0.87	2.02	0.93	2.03	0.88	2.00	0.88	0.82	0.83	0.87
ORR	2.06	0.98	2.08	1.00	2.06	0.96	2.08	0.97	0.91	0.88	0.96
ORRS	2.11	1.02	2.14	1.03	2.12	1.05	2.14	1.04	0.97	0.94	1.01
RSB	2.14	1.08	2.21	1.12	2.16	1.12	2.15	1.11	1.05	1.00	1.15
RSBS	2.25	1.16	2.32	1.22	2.22	1.14	2.24	1.15	1.11	1.10	0.19
RSC	2.23	1.15	2.28	1.18	2.22	1.14	2.25	1.16	1.10	1.10	1.14
RSCS	2.28	1.21	2.34	1.24	2.32	1.22	2.34	1.24	1.17	1.19	1.23
SBC	2.00	0.87	2.05	0.88	2.00	0.85	2.01	0.86	0.80	0.81	0.85
SBCS	2.05	0.93	2.09	0.94	2.07	0.90	2.09	0.89	0.85	0.88	0.91
SUB	1.98	0.88	2.01	0.91	1.98	0.88	1.99	0.89	0.82	0.80	0.88
SUBS	2.06	0.95	2.09	0.98	2.07	0.95	2.06	0.96	0.89	0.87	0.96
TEQ	2.04	1.05	2.07	1.08	2.05	1.05	2.03	1.05	0.98	1.00	1.04
TST	2.14	0.91	2.15	0.95	2.14	0.90	2.13	0.90	0.85	0.85	0.91

small. As it can be observed by our models most of the inter-instruction costs are less than 5% of the corresponding pure base costs while almost all the cases are covered by a 15% percentage. Also, there is no symmetry in the inter-instruction cost for a pair of instructions. For example, the execution of the LDR after the ADD presents a cost of 0.064 nJ while the execution of ADD after LDR presents a cost of −0.122 nJ. This seems more reasonable from the case of symmetric inter-instruction costs as Tiwari method supposes.

To determine the accuracy of the method a number of programs with various instructions were created. In these instructions the effect of energy sensitive factors has not been taken into account. Also, the accurate measurements of the instructions and not that of their representatives in the groups were used. The error was found to be up to 1.5%.

4. Energy dependency on instruction-level parameters—results

The dependency of the energy of the instructions on the values of the instruction parameters and the operands, called energy sensitive factors, was also studied. Energy depends on the number of 1s in the word structures of these entities. The Hamming distance between the corresponding word fields of successive instructions is not considered here since according to the followed modeling methodology (Table 1), where NOP is used as a reference instruction, Hamming distance equals to the number of 1s of the corresponding fields of the instructions. This is an advantage of our method compared to Ref. [3] since it leads to simpler models. The energy-sensitive factors are the register numbers, the register values, the immediate values, the operand values, the operand addresses and the fetch addresses of the instructions [3,4].

This energy dependency can be approximated with sufficient accuracy by linear functions. Coefficients were derived for each instruction for every energy sensitive factor. However, appropriate grouping of the instructions was used to keep the number of required coefficients reasonable, in order to increase the applicability of the method without significant loss in the accuracy.

The grouping of the instructions for the derivation of the coefficients and the corresponding measurements are presented in Ref. [8]. According to the results the linear dependency mentioned above is obvious. Some results are presented here. In Fig. 1 the effect of the register number for data-processing instructions in immediate addressing mode (noted by (9) in Ref. [8]) is presented. The actual physical measurements versus estimated energy values for the ADC instruction in scaled register offset addressing mode is shown in Table 4 where the achieved accuracy for the selected coefficient is also given. The same number of 1s in different rows just corresponds to different positions of the 1s in the word. The influence of this position is very small. From Table 4, it is observed that the error is less than 3%. Such error values characterize all the selected coefficients.

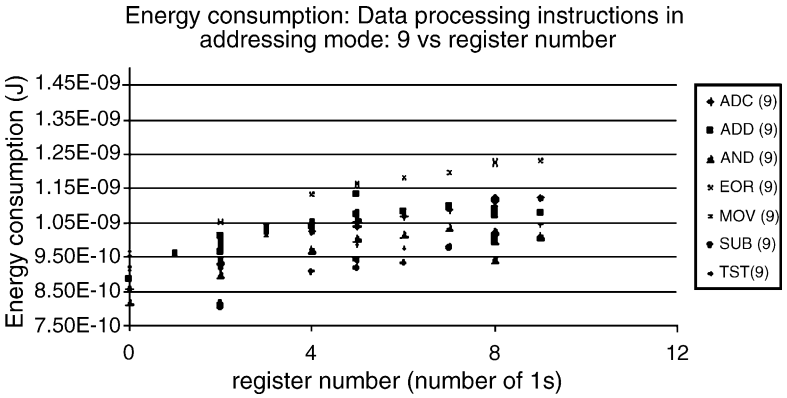

Fig. 1. The effect of register number for data-processing instructions.

Table 4
Actual physical measurements versus estimated energy values (nJ) for the ADC instruction

#1s	Estimation	Measurement	% Error
0	0.874	0.855	2.20
2	0.936	0.929	0.74
2	0.936	0.924	1.23
5	1.028	1.040	1.19
6	1.059	1.067	0.77
8	1.121	1.119	0.16
8	1.121	1.124	0.28
9	1.151	1.124	2.47
7	1.090	1.088	0.17
5	1.028	1.054	2.46
8	1.121	1.114	0.61
4	0.997	1.023	2.51

To evaluate the absolute accuracy of our modeling approach, real programs were used as benchmarks. The corresponding assembly listings were extracted from C programs by utilizing the facilities of the *armcc* tool, shipped with the ARM ADS software distribution. The energy consumption at each clock cycle is measured and estimations for the instructions are produced based on the derived models. The overall energy dissipation is calculated by Eq. (2) in order to sum up all the individual contributions that relate to variations in the energy consumption at the instruction level. In Table 5 the measured and estimated energy consumption for five common software kernels are presented. According to our results the error of our approach in real life programs was found to be less than 6%.

5. Software energy estimation framework

This section describes the main features of the software power estimation framework that has been developed for the ARM7TDMI processor. The software (hereafter "Energy Profiler") employs the instruction-level power models that have been presented, enabling exploration of various alternatives of a given program, in order to optimize its power consumption. Such alternatives for a given algorithm are often found in the domain of motion estimation, in order to decrease for example the energy consumption in the memory hierarchy [9]. The Energy Profiler receives as input the trace file of executed assembly instructions and estimates the base and inter-instruction energy cost of the program.

The kernel of the program, which is written in the ANSI C programming language [10], contains the derived instruction-level energy models. An overview of the energy estimation process is shown in Fig. 2.

Energy estimation is initiated by compiling the source file of the program under study with one of the available compilers of the ARM processor family [11]. Compiling the program provides both the code size and the minimum RAM requirements for the data memory. Next, the execution of the code using the Debugger generates the trace file and provides the number of executed assembly instructions.

The trace file which is then parsed serially by the Energy Profiler, contains two kind of entries: *memory lines* that indicate an access to the data or instruction memory for fetching data or opcodes, respectively, and *instruction lines* which indicate the conditional execution of an assembly instruction.

Each time the profiler identifies an instruction line it calculates the energy consumption by proceeding to the following main steps:

Step 1 Identification of instruction category
Step 2 Identification of specific instruction
Step 3 Identification of addressing mode
Step 4 Base energy cost calculation
Step 5 Inter-instruction energy cost calculation
Step 6 Modification of base cost according to energy sensitive factors

Table 5
Comparison of estimated and measured energy consumption for various real kernels

Benchmark	Program energy consumption		Error %
	Estimated (nJ)	Measured (nJ)	
Cadd	32.78	33.43	1.94
Cmul	13.17	12.73	−3.46
Fir	46.51	44.70	−4.05
Sad	52.04	52.01	−0.06
Ablend	22.35	23.76	5.93

Benchmark description:
1. Cadd: complex addition.
2. Cmul: complex multiplication.
3. Fir: FIR filter.
4. Sad: sum of absolute differences (used in motion estimation algorithm of MPEG video coding).
5. Ablend: alpha blending (an image compositing algorithm).

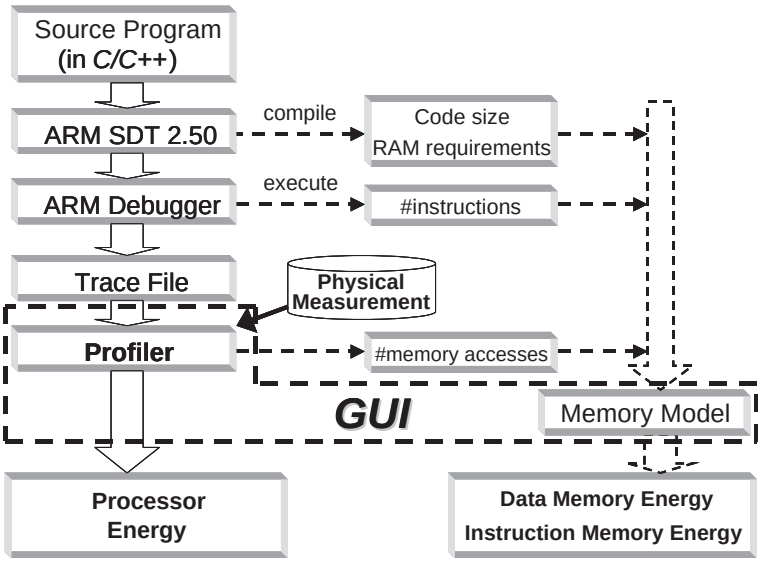

Fig. 2. Energy estimation process.

One of the key aspects in developing an energy estimation framework that is based on an underlying model, is the anticipation of future changes or enhancements [12], since new and more accurate measurements might be performed in the future. If the physical measurement values, such as currents, coefficients for the energy factors, or groups were hardcoded ("hidden in the code"), it would be extremely difficult for users or even developers to upgrade the existing software tool. For this reason, several implementation decisions that have been taken with the above issue in mind, are listed below.

Concerning the first of the above steps, the profiler looks for specific patterns in the operation code bits. To facilitate the organization and retrieval of information, the search for the specific instruction category, is performed by traversing a binary tree, in which each node corresponds to one bit of the opcode. Each node has two descendants: the path to the left subtree is followed if the corresponding bit of the parent node is 0, while the right subtree is followed otherwise. The leaves of the tree contain the information that is being sought. For example, the binary tree for deciding the instruction category based on the values of several bits is shown in Fig. 3.

Concerning the implementation of the binary trees in C, each node is a structure *(struct)* [10] with four fields: a) *data* which holds the node's data, b) *bit* which holds the value of the corresponding bit in the opcode, c) *two pointers* to the left and right subtrees.

Constructing the trees is not a trivial task, since in many cases the trees consist of a few tens of nodes, each of which should receive the correct values for data and bit variables. Hard-coding the information by hand is an error-prone process. For this reason, the proposed approach separates the process of implementing the trees from entering data into them. In this way, the process can partly be automated and data can be easily verified since information is not 'hidden' into the code. The information that is to be organized in each binary tree is stored in a *string*, which represents the *pre-order traversal* (root-left-right) of the tree. The algorithm that constructs a binary tree from the information stored in a string essentially traverses in pre-order the binary tree that is symbolically represented in the string and constructs in parallel the binary tree as linked nodes.

Once the specific instruction category (represented by a unique number) is found, it is used as index in an array of pointers to trees, and the selected tree is employed to obtain the specific instruction that has been executed. Next, a third binary tree (specific for each instruction) is traversed, in order to extract the addressing mode of the executed assembly instruction. Finally, the number corresponding to the

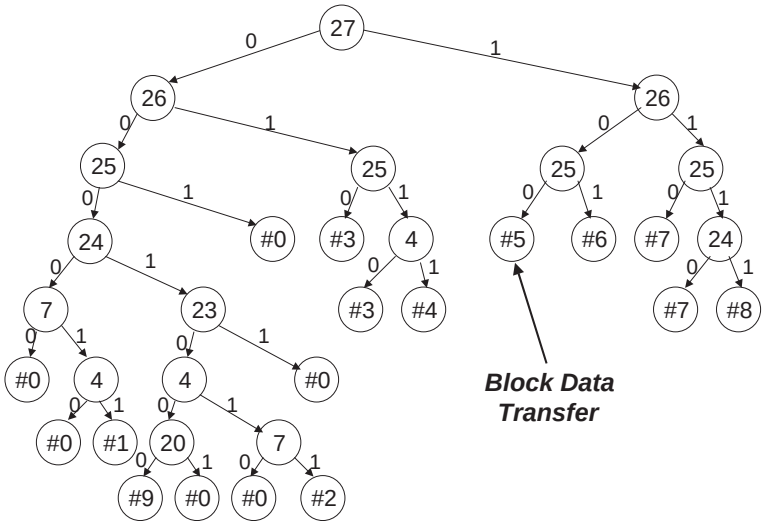

Fig. 3. Binary tree for instruction categories.

addressing mode and the instruction category are used as indices in a two-dimensional table that contains the physical measurements for the base energy costs, in nanoJoules.

Concerning the inter-instruction energy cost calculation, which is associated with the execution of adjacent assembly instructions, the groups that have been determined during the derivation of the corresponding models have been employed in the profiler. The inter-instruction costs have been placed in two-dimensional arrays, while the information extracted during base cost calculation (instruction category, type, addressing mode) has been used both for selecting the appropriate array as well as for the indices that specify an element.

The final step consists of the modification of the pure base cost of each instruction according to the energy sensitive factors described earlier. The factors that have been implemented are: register numbers, immediate values, operand addresses and instruction fetch addresses. A separate function for each energy sensitive factor receives as input the number of 1s of the word space and returns the amount of energy that has to be added, considering the corresponding coefficient. The kernel sums up the results for all energy factors and modifies accordingly the pure base cost for each instruction line.

In parallel, the number of executed instructions and the code size are used as input to a memory power model [13] in order to calculate the energy consumption of the instruction memory (Fig. 2). In the same way, the number of data memory accesses and the minimum RAM size are used to compute the energy consumption of the data memory. These calculations are performed once the complete trace file has been parsed.

The program includes a Graphical User Interface (written in Java) that displays the generated results in multiple tables and graphs. Among others, profiling results are displayed as dissipated energy distribution among system components, instruction categories and main instruction types. The GUI is also capable of comparing results for two or more trace files, thus enabling a comparative analysis of several programs, which aids in exploring the optimum solution of the design space.

6. Conclusions

In this paper, an embedded software energy estimation methodology has been evaluated. All factors that affect the energy consumption for the execution of a software program have been taken into account. By comparisons to actual physical measurements an error up to 6% has been found for the proposed methodology. A software energy estimation framework has been developed based on the derived

models, which can aid in the selection of the most energy efficient solution among several alternatives.

Acknowledgments

This work was performed within the project IST-2000-30093-EASY, funded by the European Union.

References

[1] V. Tiwari, S. Malik, A. Wolfe, Power Analysis of Embedded Software: A First Step Towards Software Power Minimization, IEEE trans. on VLSI systems, 1994 (Dec.), pp. 437–445.
[2] Mike Tien-Chien Lee, Vivek Tiwari, Sharad Malik, Masahiro Fujita, Power Analysis and Minimization Techniques for Embedded DSP Software, IEEE transactions on very large scale integration (VLSI) systems, 1997 (March), pp. 123–135.
[3] S. Steinke, M. Knauer, L. Wehmeyer, P. Marwedel, An Accurate and Fine Grain Instruction-Level Energy Model Supporting Software Optimizations, Proc. of PATMOS, Springer Verlag, Switzerland, 2001 (Sept.).
[4] N. Chang, K. Kim, G. Lee, Cycle-Accurate Energy Consumption Measurement and Analysis: Case Study of ARM7TDMI, IEEE trans. on VLSI systems, 2002 (Apr.), pp. 146–154.
[5] S. Nikolaidis, N. Kavvadias, T. Laopoulos, L. Bisdounis, S. Blionas, Instruction Level Energy Modeling for Pipelined Processors, Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), Springer Verlag, Turin, Italy, 2003 (Sept.).
[6] T. Laopoulos, P. Neofotistos, K. Kosmatopoulos, S. Nikolaidis, Measurement of current variations for the estimation of software-related power consumption, IEEE Trans. Instrum. Meas. 52 (4) 2003 (Aug.), pp. 1206–1212.
[7] S. Nikolaidis, N. Kavvadias, P. Neofotistos. Instruction level power measurements and analysis , IST-2000-30093/EASY project, deliverable D15, Sept. 2002.
[8] S. Nikolaidis, N. Kavvadias, P. Neofotistos. Instruction level power models for embedded processors, IST-2000-30093/EASY project, deliverable D21, Dec. 2002. Web site:easy.intranet.gr.
[9] S. Wuytack, J.-P. Diguet, F. Catthoor, Formalized methodology for data reuse, exploration for low-power hierarchical memory mappings, IEEE Trans. VLSI Syst. 6 (4) 1998 (Dec.), pp. 529–537.
[10] B. Kernighan, D. Ritchie, The C Programming Language, Prentice-Hall, Upper Saddle River, NJ, 1988.
[11] ARM Developer Suite, http://www.arm.com/devtools/ads?OpenDocument.
[12] C. Ghezzi, M. Jazayeri, D. Mandrioli, Fundamentals of Software Engineering, Prentice-Hall, Upper Saddle River, NJ, 2003.
[13] P. Landman, Low-power architectural design methodologies, (1994) Doctoral Dissertation, U.C., Beckerly.

Spiridon Nikolaidis received the Diploma and PhD degrees in electrical engineering from Patras University, Greece, in 1988 and 1994 respectively. He is an Assistant Professor at the Electronics Laboratory of the Department of Physics of the Aristotle University of Thessaloniki, Greece. His research interests include design of low power high speed reconfigurable processor architectures (ASIP, RISP), CMOS gate propagation delay and power consumption modeling, high speed and low power CMOS circuit techniques, power estimation techniques. He has published as author and coauthor more than 80 papers in international scientific journals and conferences. He is also involved in research projects funded by European and National resources.

Alexander Chatzigeorgiou is a Lecturer in Software Engineering in the Department of Applied Informatics at the University of Macedonia, Thessaloniki, Greece. He received the Diploma in electrical engineering and the Ph.D. degree in computer science from the Aristotle University of Thessaloniki, Greece, in 1996 and 2000, respectively. From 1997 to 1999 he was with Intracom S.A. Greece, as a Telecommunications Software Designer. His research interests are in software engineering, object-oriented design and metrics and low-power hardware/software design.

Theodore Laopoulos, is Associate Professor at the Electronics Lab., Physics Department, Aristotle University of Thessaloniki, Greece. His research interests are in the fields of Instrumentation Circuits and Systems, Measurement Systems Analysis, Micro-processing and Automation, Sensor Interfacing, and I&M Education. His educational activities are on the subjects of Circuits Design, Instrumentation Systems, and Measurement Techniques. Dr. Laopoulos is serving as Associate Editor of the IEEE Instrumentation & Measurement Transactions, as member of the Editorial Board of the IEEE I&M Magazine, and as chairman of the International Advisory Board of the IDAACS—International Workshop on Intelligent Data Acquisition and Advanced Computing Systems.