A Retargetable Software Power Estimation Methodology

C. Brandolese
Politecnico di Milano, P.zza L. da Vinci, 32 - 20133 Milano, Italy
brandole@elet.polimi.it

Abstract

In the design of mixed hardware/software embedded systems, the early assessment of the power budget is a key factor to concurrently meet time-to-market and product competitiveness. An increasing contribution to the overall power consumption depends on the software portion of the design and is influenced both by the system specification style and by the target microprocessor. The proposed estimation methodology operates at an abstraction level halfway between the system language and the specific assembly language and provides power consumption figures useful for both source code analysis and modification, and for target processor selection.

I. Introduction

The importance of the power constraints in the design of embedded systems has continuously increased in the past years, due to technological trends toward high-level integration and increasing operating frequencies, combined with the growing demand of portable systems. So far, only a few co-design approaches have considered power consumption as a comprehensive system-level metric [1] [2] [3]. According to [4], the methods to estimate the software power consumption can be grouped in three classes: gate-level processor simulation [5], architectural-level processor description and instruction-level models. The gate-level simulation provides the most accurate results at the cost of extremely time demanding simulations. Furthermore, due to the lack of information of the processor gate-level description, this methodology is rarely viable. Architectural-level power estimation is less precise but much faster than gate-level estimation [6] and requires a coarser grain model of the processor (ALU, register file, etc.). Instruction-level power estimates are typically based on stochastic data modeling of the current drawn by the processor for each instruction. Such methodologies have been proposed in [2] and [3]. The goal of this paper is to describe a power estimation methodology for software components suitable for typical hardware/software embedded system architectures. The proposed approach allows exploring the design space either to early retarget architectural design choices, or to redesign the power-critical parts of the system. The proposed technique has been implemented in the TOSCA co-design framework for control-dominated embedded systems [10]. The description language adopted in the framework is OCCAM2, since it allows describing mixed hardware/software systems, and provides constructs for parallel execution and process synchronization with the rendez-vous semantic. It is worth noting that the use of OCCAM2 as description language, does not affect the generality of the proposed approach, the key point being the introduction of an intermediate pseudo-assembly language, called VIS (*Virtual Instruction Set* [7]). The VIS model provides sufficient detail to permit an accurate estimate and a degree of generality allowing easy retargeting towards different processors. The paper is organized as follows. In section II, the TOSCA framework, and in particular software flow, are described; in section III, the procedure adopted to derive the power estimation model is detailed and justified; in section IV, the results obtained on some benchmarks and an industrial design are reported; finally, in section V conclusions are drawn and an outline of the current research is given.

II. Framework and Flow

When designing low-power embedded systems, and, in particular, their software components, two aspects have to be considered. On one hand, the specification can be partitioned between hardware and software in different ways; on the other hand, the same source code results in different power requirements when compiled and executed on different microprocessors. The designer, usually, performs the partitioning on the basis of figures such as cost and performance. With the increasing demand of portable devices, considering the power consumption from the early phases of the design, becomes essential. A satisfactory design-space exploration should consider these aspects concurrently and should be fast enough not to compromise time-to-market constraints. As far as the software portion of the design is concerned, the analysis of different solutions would require a complete development environment (compiler, debugger, instruction-set-simulator, etc.) for each target microprocessor considered. In addition, a modification of the source code implies re-running the complete flow, from compilation, through profiling and power esti-

mation, to back-annotation. Currently only a few microprocessor vendors ship such a complete development toolkit. This means that it is often necessary to complete (or, at least, customize) the vendor toolkits with in-house developed tools and utilities. The TOSCA environment overcomes these problems with the introduction of the intermediate, pseudo-assembly VIS. This language is sufficiently close to an actual assembly to guarantee a good degree of detail, and, at the same time, sufficiently general to permit fast retargeting towards different assembly languages. The power estimation methodology presented in this paper operates at the VIS level, and achieves a good trade-off between accuracy and effectiveness. Figure 1 depicts the part of the TOSCA framework devoted to the software flow. In this figure, rounded boxes indicate input and output files while grayed-out boxes indicate tools. The software power estimation flow is composed of five tools:

- `COMPILER` - An OCCAM2 to VIS compiler.
- `SIMULATOR` - A VIS simulator and profiler.
- `MAPPER` - A library-based translator from VIS to different real target assembly languages.
- `ESTIMATOR` - A library-based power estimator.
- `BACK-ANNOTATOR` - Atool for assembly–to–VIS and VIS– to–OCCAM2 back-annotation.

The evaluation of power consumption of a given OCCAM2 source code begins within the compilation in VIS code. The VIS code can then be debugged and validated, executing it on the virtual *VIS Machine* implemented in the simulator. When the functionality of the VIS code has proven correct, the simulator can be run in batch mode to obtain profiling information, i.e. the number of times that each VIS instruction has been executed. The VIS code is then *mapped* to a specific, commercial assembly language, according to a fixed set of *rules* and its power consumption is evaluated on the basis of power figures collected into *annotation libraries*. Exploring different solutions

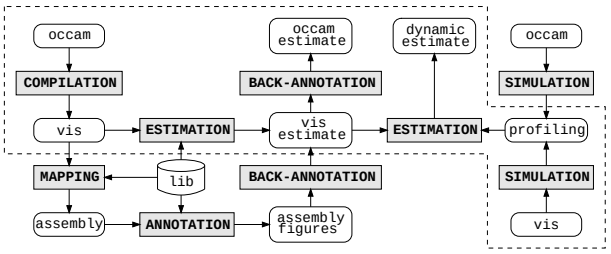


Fig. 1. The TOSCA software flow

targeted to different microprocessors would require re-mapping and re-estimating each and every alternative. This can be avoided noting that the same VIS instruction (same op-code, same operands type and value) is always mapped to the same assembly portion of code. This knowledge, along with an accurate statistical analysis of the results (see sections III and IV), allows characterizing directly each VIS instruction in terms of power consumption, once the target microprocessor is selected. This approach shortens the design flow, since avoids the mapping, the assembly annotation and one of the back-annotation steps (see figure 1). Furthermore, when a dynamic analysis is desired, a single session of profiling at VIS level is necessary. It is worth noting that, being each VIS instruction power-characterized for different architectures, the power consumption impact of the choice of a specific target microprocessor can be estimated by simply switching to the suitable annotation library, without simulating or mapping. Furthermore, the static or dynamic power consumption figures gathered at VIS level can be conveniently back-annotated to the OCCAM2 source code, giving a clearer and more useful view of the overall power budget of the software components. Typically, the back-annotated information are valuable hints to the designer on what parts of the source code should possibly be modified or moved to the hardware partition.

III. Methodology

The methodology presented in this section can be applied to derive the power characterization of the VIS instruction set for any target processor. A VIS instruction is defined by:

- *op-code:* the type of operation to perform;
- *addressing mode:* the type of operand used;
- *operand value:* the value of the operands;

The *class* of an instruction is defined by its op-code and the addressing mode of its operand, but ignoring the value of operands. For each instruction *class*, thus, a set of instructions can be built, varying the value of the operands. As an exemple, consider the VIS instruction `MOVE.W #16, +5(R0)`: the op-code is `MOVE.W`, the addressing mode of the two operands are *immediate* (`#16`) and *indirect* (`+5(R0)`) and the values are 16 for the first operand and the couple (5, R0) for the second. It is not uncommon that instructions of the same *class* map to different assembly codes and are thus characterized by different power consumption [9] [11]. As an example consider the ARM7TDMI microprocessor: an immediate constant can be loaded into a 32-bit register directly if and only if it falls in the range 0–255; when the immediate value is greater than 255, its low and high bytes must be loaded into the register separately, suitably shifting the register content after the first load. To properly account for these differences *all* the possible instructions of a given instruction *class* must be

analyzed. Let $\mathcal{I}$ be an instruction *class* and $i_j \in \mathcal{I}$ a generic instruction with specific operands values. Using the estimation flow described in section II, all instructions $i_j \in \mathcal{I}$ can be annotated with the actual timing $t_j = t(i_j)$ and average current $c_j = c(i_j)$ leading to two vectors of measures $\mathbf{T} = \{t_j\}$ and $\mathbf{C} = \{c_j\}$. To derive single values $t(\mathcal{I})$ and $c(\mathcal{I})$ for the VIS instruction *class* $\mathcal{I}$, four different approaches have been adopted:

- **Different translations only.** The timing $t(\mathcal{I})$ and current $c(\mathcal{I})$ are computed by averaging only the values t_j and c_j corresponding to different translations. This method is easily applicable because the mapping *rules* are known and thus it is possible, for each instruction *class*, to build all the instructions necessary to exercise all the cases that the specific *rule* considers. This strategy is supported by the hypothesis that a generic VIS code exploits in a statistically uniform way all the possible alternatives of all the *rules*.
- **Different figures only.** The timing and current are computed by averaging only the values t_j and c_j that are different. This approach is justified by the consideration that different translations may lead to identical total power figures.
- **Complete uniform.** The timing and current are computed averaging *all* the values t_j and c_j. This choice is statistically justified considering that, when a sufficiently long VIS program is considered, and the semantics of the instructions is neglected, the distribution of instructions within an instruction *class* tends to be uniform. This is mostly due to the fact that there is no reason to suppose non-uniform the distribution of values of the operands.
- **Complete weighted.** The timing and current are computed by averaging *all* the values t_j and c_j, *weighted* with relative frequencies obtained from an analysis of a sufficiently large set of benchmarks. The previous case, as mentioned, ignores the semantics of the instructions. This hypothesis can be removed considering the fact that some values are more likely to be used than others. The measured relative frequency is thus considered an estimate of the probability.

Once all VIS instructions have been characterized, according to one of these strategies, in terms of number of clock cycles and average current consumption, power estimates of a generic VIS program can be derived. The estimation flow produces, for each instruction i_k of the VIS program, the number of clock cycles required for execution t_k, the average current absorbed per clock cycle c_k and the number of times n_k the instruction has been executed during a simulation session driven by typical input streams. The energy absorbed by all executions of the k-th instruction is $e_j = n_k V_{dd} t_k c_k$ and the total energy and cycles are:

$$E = \sum_{k=1}^{M} e_k = V_{dd} \sum_{k=1}^{M} n_k t_k c_k \quad (1)$$

$$T = \sum_{k=1}^{M} n_k t_k \quad (2)$$

where V_{dd} is the power supply of the core and M is the number of instructions of the considered VIS program. The average power consumption is thus:

$$\overline{W} = \frac{E}{T} = V_{dd} \cdot \frac{\sum_{k=1}^{M} n_k t_k c_k}{\sum_{k=1}^{M} n_k t_k} \quad (3)$$

In the next section the four proposed characterization approaches are compared. o

IV. Experimental Results

A. Methodology tuning

The four modeling approaches presented in the previous section have been applied to the entire VIS instruction set for the two commercial processors ARM7TDMI in Thumb mode (ARM Ltd.) and MC68000 (Motorola) and the results for the first are reported in figure 2 (similar results have been obtained for the Motorola processor, but are omitted here for brevity). The figure reports relative errors with respect to the *complete uniform* method, and shows that the four methods, for each instruction *class* lead to significantly different figures (up to $\pm 50\%$). To select the most accurate method, benchmarking is necessary. Some sample OCCAM2 sources have been run through the complete compilation, mapping, power annotation and back-annotation flow and the power measures obtained have been compared against the four estimation methods, leading to the results summarized in table I. Note that these errors are much smaller than those plotted in figure 2: this is due to cancellation effects that occur over real VIS instruction sequences.

Method	Error
Different figures only	7.21%
Different translations only	4.31%
Complete uniform	3.14%
Complete weighted	2.71%

TABLE I
Relative errors (absolute values)

The comparison of the results shows that the two best methods are the *complete uniform* and the *complete weighted*, the latter being slightly more accurate. In the next paragraph the methodology is applied to an industrial example and results are discussed.

B. Test case

To validate the presented methodology and framework, a set of benchmarks has been built. This set

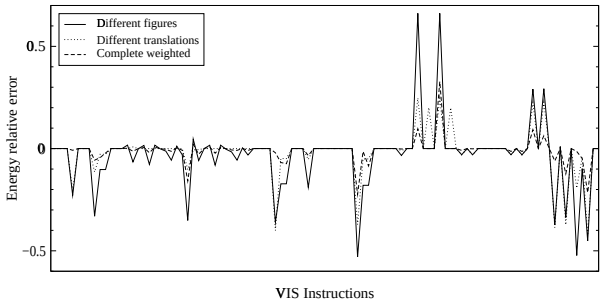


Fig. 2. Energy errors relative to the *complete uniform* method

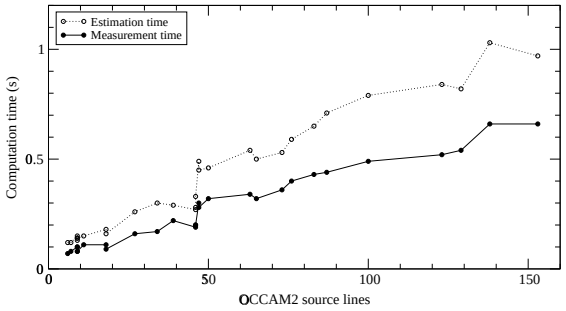


Fig. 3. Run times of measure and estimation flows

comprises OCCAM2 programs 10 to 230 lines long. The result of compilation of these codes are VIS files with lengths ranging from 60 to 2,600 lines, approximately. The benchmarks have been used: to decide for one of the strategies for VIS instruction set characterization (see table I); to measure the accuracy of the VIS-level model and to measure the speed-up obtained with the estimation flow with respect to the complete flow. Figure 3 reports the run times of the complete flow and the estimation flow on the benchmarks. The speed-up varies from 30% to 75% with an average value of 63%, confirming the suitability of the proposed methodology as an effective design-space exploration strategy. To determine the accuracy of the proposed model, the methodology has been applied to a real industrial design called ILC16, a 16-channel link controller developed at Italtel R&D Labs [12], and the results, summarized in tables III and II, have been obtained.

Processor	Measure	Estimation	Speed-up
ARM7TDMI	37.57 s	22.07 s	58.8%
MC60000	53.17 s	29.96 s	56.3%

TABLE II
RUN TIMES FOR THE ILC16 APPLICATION

Processor	Measure	Model	Error
ARM7TDMI	224.48 mW	230.65 mW	2.77%
MC68000	26.20 mW	27.41 mW	4.47%

TABLE III
POWER ESTIMATES FOR ILC16

The ILC16 design is composed of 54 OCCAM2 procedures, for an overall line count of 2200 lines. The VIS code resulting from compilation has 55000 lines and the target assembly codes for the ARM and Motorola processors are 75000 and 89000 lines long, respectively. It is worth noting that the average currents for the ARM7TDMI processor—used to characterize the VIS instruction set—have been obtained from actual measurements [11], while those for the MC68000 processor are relative to the power consumption of a reference instruction [8].

V. CONCLUSIONS

The paper presented a new methodology for embedded software power estimation. The main focus of the proposed approach is fast design-space exploration with respect to different coding styles and different target processors. To validate the methodology, an industrial example, used as a test vehicle during two ESPRIT projects, has been analyzed using an integrated environment developed to this purpose within the TOSCA co-design framework. The techniques described, though currently based on the OCCAM2 system modeling formalism, are general and applicable to different high-level languages such as C/C++. The results obtained are encouraging under both accuracy and effectiveness points of view. The current effort of the research is aimed at properly considering dynamic effects such as the presence of a memory hierarchy and pipelining.

REFERENCES

[1] E. Macii, M. Pedram, F. Somenzi, *"High-Level Power Modeling, Estimation, and Optimization,"* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 17, No. 11, 1998.

[2] T.Sato, M.Nagamatsu and H.Tago, *"Power and performance simulator: ESP and its application for 100MIPS/W class RISC design,"* Proceedings of IEEE Symposium on Low Power Electronic, pp. 46-47, 1994.

[3] P.W.Ong and R.H.Yan, *"Power-conscious software design: a framework for modeling software on hardware,"* Proc. of 1994 IEEE Symposium on Low Power Electronic, pp. 36-37, San Diego, CA, Oct. 1994.

[4] V. Tiwari, S. Malik and A. Wolfe, *"Power Analysis of Embedded Software: a First Step towards Software Power Minimization,"* IEEE Transactions on VLSI Systems, Vol. 2, No. 4, pp. 437-445, Dec. 1994.

[5] V. Tiwari and M.T.-C. Lee, *"Power analysis of a 32-bit Embedded Microcontroller,"* VLSI Design Journal, 1996.

[6] J.Russell, M.F.Jacome, *"Software Power Estimation and Optimization for High Performance, 32-bit Embedded Processors,"* Proc. of ICCD'98, International Conference on Computer Design, Austin, Texas, USA, October, 1998.

[7] C. Brandolese, W. Fornaciari, F. Salice, D. Sciuto *"Fast Software-Level Power Estimation for Design Space Exploration,"* Politecnico di Milano, Tech. Report 99.62, 1999.

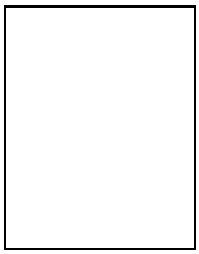
[8] C. Brandolese, W. Fornaciari, F. Salice, D. Sciuto *"An Energy Estimation Model for 32-bit Microprocessors,"* Politecnico di Milano, Tech. Report 99.63, 1999.

[9] PEOPLE ESPRIT project n.26769, Deliverable D1.3.2.

[10] PEOPLE ESPRIT project n.26769, Deliverable D1.3.3.

[11] PEOPLE ESPRIT project n.26769, Deliverable D1.2.1.

[12] A.Allara, M.Bombana, W. Fornaciari, F.Salice, *"A Case Study in Design Space Exploration: The TOSCA Environment Applied to a Telecom Link Controller,"* IEEE Design & Test of Computers, 2000, (to appear).



Carlo Brandolese received his degree in Electronic Engineering with a specialization in Microelectronics and Optoelectronics in 1995 from the Politecnico di Milano, Italy, working on florplanning of analog integrated circuits. He has been working from 1995 to 1997 at Italtel Central R&D Labs as a CAD engineer and focused on the FPGA design flow and methodology. He got his MS in Information Technology in 1997 from CEFRIEL, Politecnico di Milano, with a thesis on hardware/software co-design. He will receive his Ph.D. in Information Technology in october 2000 and is currently working on software power estimation and optimization for embedded systems.