Fast Software-Level Power Estimation for Design Space Exploration

Carlo Brandolese §*, William Fornaciari §*, Fabio Salice §*, Donatella Sciuto §
§ Politecnico di Milano, DEI, Piazza L. Da Vinci, 32, 20133 Milano (Italy),
* CEFRIEL, via Fucini, 2, 20133 Milano (Italy)

Abstract

Aim of the proposed methodology is to perform design space exploration at a high-level of abstraction based on high-level estimations of different parameters. In particular, this paper presents a methodology for static and dynamic estimation of the power consumption of the software components. This analysis is based on a fast software compilation strategy that allows a fast re-targeting over different microprocessors. The paper focuses on the description of the overall power assessment flow and its application on an industrial application.

1. Introduction

The importance of the power constraints during 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] [4].

According to [5], the methods to estimate the software power consumption can be grouped in three classes: a gate-level processor simulation [6], an 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 [7]. This approach requires a coarser grain model of the processor (ALU, register file, etc.) and the knowledge of the relations between the instructions being executed and the functional units activated. 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], [3] and [4].

The goal of this paper is to describe a system-level power estimation methodology for software components suitable for the typical architectures of HW/SW embedded systems. Such a strategy provides the capability of exploring the architectural design space to early retarget architectural design choices avoiding complete redesigns. Different metrics have been defined and implemented [8] [9] [10], both statically and dynamically computed. In this paper we focus on the important aspect of software power estimation. The basic idea is to provide a methodology allowing the designer to explore different alternative target microprocessors, for which a software compilation flow is provided, along with a fast, but accurate, power estimation, given the base cost of the power consumption of the assembly-level instructions. This allows verification of the degree of acceptability of a given partitioning in terms of power budget, in addition to performance and cost. The proposed metric has been implemented in the TOSCA co-design framework for control-dominated embedded systems [11] [12]. It is built on the internal high-level OCCAM2 model. This formalism has a well funded semantic, derived from process algebra, which allows definition of concurrent, communicating processes [13]. Moving down to the implementation, software-bound parts of the specification are compiled into a pseudo-assembly retargetable intermediate language, called *VIS* (Virtual Instruction Set) [11] [16].

The use of OCCAM2 as the description language does not affect the generality of the proposed approach. The power analysis methodology is composed of four main phases:

1. *Compilation:* the *OCCAM2* model is first translated in *VIS* and then in the target assembler.
2. *Static power estimation:* for each assembler instruction, the average current absorption and the execution time are calculated.
3. *Back-annotation:* assembler-level data are reported to *VIS* and *OCCAM2* levels.
4. *Dynamic power analysis:* static currents and timing figures are combined with profiling results.

Different levels of accuracy are provided during the compilation flow, depending on the level of abstraction.

The proposed approach combines the advantages of static and dynamic analysis: the dynamic behavior accounted for by means of profiling data, while reliable static power consumption figures are computed by moving down into the software compilation steps and then back-annotating the obtained information to the upper levels. The back-annotation phase allows identifying the major components of power consumption at the system level, thus giving the designer the awareness of the consequences of partitioning and target microprocessor alternatives. This is one of the main advantages of the proposed methodology, whose importance is steadily increasing [14] [15].

The paper is organized as follows. Section 2 details the software compilation; in section 3, the three levels of the power estimation process are described; in section 4 a portion of an industrial application example is discussed, and the different transformations, formats and back-annotation results of estimations are reported. Finally, conclusions are drawn in section 5, with an outline on future research.

2. The software compilation

The power analysis operates at *three* different description levels (Figure 1): the high-level OCCAM2 system description language, the intermediate pseudo-assembly *VIS* level, and the target processor assembler level. The software-bound portion of the system specification, described in OCCAM2, is compiled into the intermediate VIS code, which is in turn mapped to the target assembly language. The OCCAM2 syntax allows an easy definition of a formal system representation based on process algebra: in fact OCCAM2 supports both parallel and sequential execution of processes and a synchronization mechanism based on channels [13]. Timing and performance constraints have been introduced within the TOSCA framework, by extending the OCCAM2 syntax [11]. The basic assumption for the analysis is that power and timing characterization of the target system architecture are given through two technology files:

- *Processor Technology File:* The file contains the basic power consumption figures for each instruction and each addressing mode provided by the processor instruction set and a characterization of inter-instruction effects.
- *Memory Technology File:* The file contains power consumption data of the read/write operations for each level of the memory hierarchy (*on*-processor and *off*-processor).

The rest of this section describes the phases composing the TOSCA power estimation flow (Figure 1).

The *software compilation process* moves in the forward direction, from the high-level description to the low-level one, whilst the *back-annotation process* moves in the opposite, backward, direction. The compilation process is structured in two different phases: the *OCCAM2-to-VIS* compilation and the *VIS-to-assembler* mapping. Similarly, the back-annotation process is split in the *assembler-to-VIS* and *VIS-to-OCCAM2* phases, whose purpose is to annotate the power figures on the functional system model. The flow described thus far provides a *static* power estimation. Static figures can be combined with *profiling* results to obtain more significant *dynamic* estimates.

2.1 The *VIS* language

The introduction of the intermediate *VIS* language enables the proposed methodology to be *independent* of the target processor. The *VIS* is close to the assembler but still preserves a good generality. Its twofold goal is to provide an instruction set allowing both the analysis of the characteristics of a broad range of possible target processors and low-level optimizations that usually cannot be easily performed on the source code. Other important features of the *VIS* language are portability, resources independence (code generation should abstract as much as possible from the available resources) and simulatability [16].

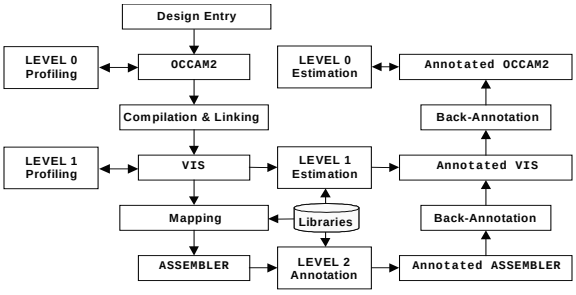

Figure 1. The power estimation design flow.

The architecture of the *virtual VIS machine*, based on a load/store model, is composed of a single execution unit without pipelining, a user defined number of general-purpose registers (GPRs) and some special-purpose registers (SPRs).

The currently used memory model has no *cache* memory and a single addressing space for both data and code; I/O registers are memory mapped. The *VIS* architecture allows arithmetic/logic operations among all GPRs.

The most common addressing modes are supported (immediate, direct, indirect, indexed). The *HW/SW* and *SW/SW* communication, based on the rendez-vous protocol, is implemented using:

- *Channel State Table:* collects the information on the state of channels and processes involved.
- *System API*: a set of basic system routines.

From the architectural point of view, no significant differences exist between HW/SW and SW/SW communication since each channel is mapped to a different memory location.

2.2 The OCCAM to VIS compilation

The basic features of the OCCAM2 to *VIS* compiler are extensibility and modularity. The former guarantees a compiler structure as flexible as possible with respect to future extensions and improvements while the latter allows to easily keep under control the single compilation passes.

Concerning the *VIS* memory management, two different approaches can be adopted: the static or the semi-dynamic allocation. In the first approach, all data segments are allocated statically at link-time and a memory table is created to associate each procedure with the base, stored in the base pointer (BP), of the corresponding data segment. In the second case, the entries of the memory table are either dynamically computed or looked up through API calls each time a procedure is invoked. If an entry of the memory table is valid, i.e. the corresponding procedure is still running, the entry is used as it is, if not, it is computed again. The parallelism of the OCCAM2 language requires passing the parameters to procedures by value-result.

Procedures are compiled separately and then linked together. To this purpose, a set of *directives* has been added to the *VIS* language. The compilation phase consists of several passes performing the following, independent, basic operations:

1. *Variables and temporaries analysis.* The amount of memory required for variables and temporaries is computed and allocated.
2. *Addressing mode selection and register allocation.* The *VIS* is translated into an internal formalism to decouple operating code selection from register binding.
3. *VIS Generation.* The model is translated in *VIS*.
4. *Scheduling.* Additional code is generated to conform the execution to the original OCCAM2 semantics. Furthermore, some back-end directives are added to guide linking and mapping.
5. *Linking.* The compiled code of the procedures is linked into a stand-alone, simulatable *VIS* code.

2.3 The *VIS* to target assembler mapping

The mapping phase translates the *VIS* code, generated for a number of registers corresponding to those available on the selected class of microprocessors, into a specific assembler code. The mapper is constituted by a kernel, which has been designed to be computationally efficient, and a set of binary libraries to allow dealing with different assemblers. Each *VIS* instruction is expanded into a sequence of the specific assembler instructions. The mapping may be driven by a user-defined cost function aimed at locally optimizing a number of figures, like execution time, code area, power consumption or a combination of these. This allows an effective exploration of the solution space. The generated assembler code can still be optimized due to the fact that the *VIS* to assembler translation is performed on a local basis. Currently, the available mapping libraries target the Motorola MC68000 and the ARM Ltd. ARM7TDMI in Thumb (low-power) mode.

3. The Power Estimation Methodology

The power estimation methodology consists of three *annotation* phases and two *back-annotation* steps. Each power annotation process applies to a specific level of abstraction of the model and uses the corresponding libraries. On the other hand, the back annotation process propagates the information, collected with annotation, to higher levels of abstraction. In the following sections the different phases composing the complete methodology are detailed.

3.1 The Software Power Estimation

The methodology for estimation of power consumption operates at OCCAM2, *VIS*, and target assembler levels. All data required are collected into the *Processor Technology File* and *Memory Technology File.* The methodology requires that the OCCAM2 code is compiled into *VIS* code, limiting the *family* of the target processors to be used, and then the *VIS* code is mapped to the target assembler. In this context the term *family* means a set of processor architectures with the same number of general purpose registers.

LEVEL 2 Power Annotation. The lowest level of abstraction for the software implementation of an OCCAM2 model description is its translation into the target microprocessor assembler. The power consumption calculation performed at this level of abstraction is referred to as *LEVEL 2 power annotation* and needs a detailed knowledge of the power and timing characteristics of the specific processor and memory subsystem. The *LEVEL 2* annotation is performed by matching each instruction of the compiled and mapped assembler code against its power dissipation value in the specific technology file and, finally, taking into account inter-instruction effects [2].

LEVEL 1 Power Estimation. One level of abstraction above the target assembler code is the *VIS* code, generated for a number of registers suitable for the *family* of microprocessors envisaged for the application. The power evaluation performed at this level is referred to as *LEVEL 1 power estimation.* A power consumption parametric model characterizes each *VIS* instruction. The definition of the analytical parametric model of each *VIS* instruction requires a detailed knowledge of the compilation and mapping phases. In our approach, the compiler allows full control of the compilation process. Furthermore, a set of compiled, mapped and back-annotated benchmarks has

been selected for microprocessor-specific parameters extraction. By means of the model and the parameters available, *LEVEL 1* estimation can be performed in a similar way to that outlined for *LEVEL 2* annotation, the main difference being that the basic power consumption figures are computed rather than looked up in the technology files.

LEVEL 0 Power Estimation. At the OCCAM2 level, the power assessment is achieved through a back-annotation of the power consumption figures calculated at the *VIS* level. The *VIS*-to-OCCAM2 back-annotation technique slightly differs from that adopted for the assembler-to-*VIS* phase for the following reasons. First, at the OCCAM2 level, the overhead due to scheduling (implemented as API calls) is not explicit: its contribution is hidden in the semantics of the container processes. To obtain realistic power estimation, the scheduling cost has to be adequately distributed among the container process itself and its subprocesses. Second, the translation of OCCAM2 into *VIS* introduces an additional amount of code for memory and auxiliary data structures initialization and maintenance. Again these contributions are not logically related to specific OCCAM2 statements. The cost of these operations has been taken into account. The result of the back-annotation is a source OCCAM2 description in which every single line is annotated with a static estimation of the power consumption. To obtain a dynamic power characterization of the code, software profiling is necessary.

3.2 Software profiling for power analysis

The power analysis carried out so far is *static*. *Dynamic* analysis is mandatory to increase the accuracy and significance of the final estimates, taking into account the influence of input patterns on the overall power consumption.

Software profiling can be performed at two levels. At *LEVEL 0* profiling is purely *functional* while at *LEVEL 1* it is more *structural* since the details of the *VIS* representation are already accounted for.

For *LEVEL 0* profiling, the OCCAM2 source code is simulated by using the in-house developed OSTE simulator [12]. The API calls and scheduling overhead are neglected at this level. The profiling carried out at this level results in a limited accuracy, but it requires short execution times.

For *LEVEL 1* profiling, the linked *VIS* code is simulated by using an instruction set simulator developed on purpose. Profiling at this level results in a higher accuracy with respect to the *LEVEL 0*, at the cost of longer execution times.

3.3 Software back-annotation

The back-annotation process occurs at two levels: assembler-to-*VIS* and *VIS*-to-OCCAM2.

The basic idea behind both levels is to keep track, during the forward flow from the OCCAM2 specification down to the target assembler code, of the links connecting instructions between the different abstraction levels. In general, a single instruction of a higher level language translates into many instructions of a lower level one.

The back-annotation process can start either at assembler or at *VIS* level. The only assumption made here is that each line of the assembler code has already been annotated with area, timing and power values. The assembler-to-*VIS* back-annotation is performed by scanning the assembler code, adding up annotated values coming from the same line of the *VIS* code and associating the collected information to the corresponding *VIS* line. The back-annotation from *VIS* to OCCAM2 is performed in a likewise manner.

Inter-instruction effects are considered at the assembly level, since they are strongly processor dependent, and suitably propagated backward at the upper levels of abstraction.

4. Application Example

The proposed methodology has been applied to the ILC16 component, commercialized by Italtel and modeled in OCCAM2. This design has been selected because it was the test vehicle for system-level design space exploration during the SEED Esprit project [12].

The ILC16 is a data-link controller for sixteen asynchronous/synchronous data streams based on the HDLC protocol. From a functional point of view, the ILC16 is characterized by a set of communication links, on which data are received and transmitted, and by a control unit managing links and the external environment interaction. Data are transmitted/received according to the HDLC protocol and they are stored in a system memory. In particular, ILC16 is constituted by 5 macro-modules: the Buffer Description Manager (ESRAM control module), the RISC (device microcontroller), the Link Interface (links-RISC interface), the Bus Interface (ILC16-External Bus Interface) and 16 RX/TX Links. Figure 2 reports an example of the software synthesis flow applied to a module (CRC computation) of the ILC16 device. During the compilation from the OCCAM2 source code to the target assembler through the *VIS*-level, the correspondence among the statements at the different description levels is maintained. The back-annotation of energy and timing data at the *VIS* level can be easily performed by adding the contributions associated with the different target assembler statements corresponding to the same VIS instruction. A similar back-annotation process can be applied at OCCAM2 level. In particular, figure 2 shows the different representations, files and steps involved in both the compilation and power estimation process. The *vir.ref* file specifies, for each *VIS* instruction, the corresponding OCCAM2 statement in the source file by using line identifiers. Similarly the *asm.ref*

file relates the target assembler file *asm* with the *VIS* code, stored in the file *vis*, originating it. The file *asm.sa* contains actual data for each instruction of the target microprocessor (in this example the Motorola MC68000) in terms of average current and number of clock cycles. Similarly, the file *vis.sba* collects data for each instruction of the *VIS* code. The total clock cycles are calculated adding up the contributions of assembler instructions associated with the same *VIS* line, while the average current is obtained dividing by the total clock cycles the sum of total currents of assembler instructions associated with the same *VIS* line. Finally, the file *occam.sba* is obtained in a likewise manner.

As an example, consider line 36 of the OCCAM2 source code (figure 2). Its translation is spread over lines 125-129 of the *VIS*. The total number of clock cycles is 12+12+4+8+10=42 and the corresponding average current is (505×12+505×12+314×4+290×8+400×10)/42 =469.

The complete flow has been tested on the ILC16 OCCAM2 specification. The dimensions of the files (in terms of line count) are summarized in table 1.

File	*occam*	*vis*	*asm*
Lines	2.380	45.154	87.332

Table 1. ILC16 design: file sizes.

The computational time required for the complete processing has been 129 seconds on a Sun Ultra 1 workstation.

5. Concluding Remarks

The paper discussed a methodology to estimate the contribution of software in the power budget of mixed hardware/software embedded systems. The most relevant features of the proposed approach are summarized in the following:

- it is integrated into a more general co-design environment for control-dominated embedded systems;
- it addresses both static and dynamic estimation of power consumption;
- it defines power evaluation metrics for software, applicable for hardware/software partitioning;
- it is independent of the system specification language and of the target processor; thus it is re-targetable towards several input formalisms and commercial microprocessors.

Current effort is in the direction of extending the set of supported assembler instruction sets, thus allowing a larger set of microprocessors for embedded applications to be supported.

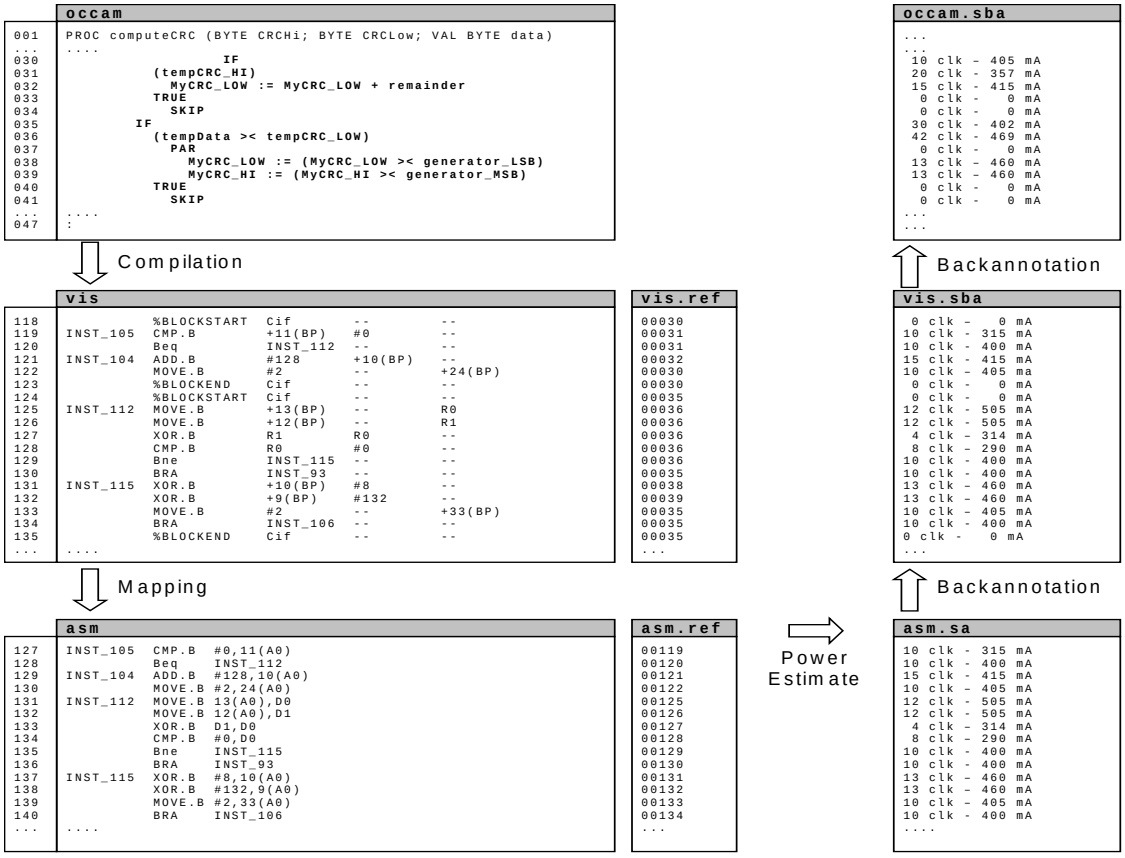

Figure 2. Example of the software synthesis flow applied to the CRC module of the ILC16 device.

6. References

[1] E.Macii, M.Pedram, F.Somenzi, "High-Level Power Modeling, Estimation and Optimization," *DAC-34: ACM/IEEE Design Automation Conference*, Anheim, CA, June 1997.

[2] M.T.C.Lee, V.Tiwari, S.Malik, and M.Fujita, "Power Analysis and Minimization Techniques for Embedded DSP Software," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, Vol. 5, No. 1, pp. 123-135, March 1997.

[3] V.Tiwari, S.Malik, and A.Wolfe, "Compilation techniques for low energy: An overview," *Proceedings of Symposium on Low Power Electronics*, San Diego, Oct. 1994, pp.38-39.

[4] M.T.C.Lee and V.Tiwari, "A memory allocation technique for low-energy embedded DSP software," *Proceedings of Int. Symposium on Low Power Electronics*, San Jose, CA, Oct. 1995, pp. 24-25.

[5] K.Roy and M.C.Johnson, "Software Design for Low Power," *Low Power Design in Deep Submicron Electronics*, NATO ASI Series, Series E: Applied Sciences, Vol. 337, Kluwer Academic Publisher, 1997, pp. 433-460.

[6] T.Chou and K.Roy, "Accurate Estimation of Power Dissipation in CMOS Sequential Circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 1996.

[7] T.Sato, Y.Ootaguro, M.Nagmatsu and H.Tago, "Evaluation of Architecture-Level Power Estimation for CMOS RISC Processors," *1995 Symposium on Low-Power Electronics*, pp. 44-45, October 1995.

[8] A.Allara, C.Brandolese, W.Fornaciari, F.Salice and D.Sciuto, "System-Level Performance Estimation Strategy for Sw and Hw," *IEEE-ICCD'98 Int. Conference on Computer Design: VLSI in Computers & Processors*, Austin, Texas, October 5-7, 1998, pp.48-53.

[9] W.Fornaciari, P.Gubian, D.Sciuto and C.Silvano, "Power Estimation of Embedded Systems: A Hardware/Software Co-design Approach," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol.6, n.2, June, 1998. pp. 266-275.

[10] A.Balboni, W.Fornaciari and D.Sciuto, "Partitioning of Hw-Sw Embedded Systems: a Metrics-Based Approach," *Integrated Computer-Aided Engineering*, J.Wiley Interscience Journal, IOS press, vol. 5, n.1, 1998, pp.39-55.

[11] A.Balboni, W.Fornaciari, and D.Sciuto, "TOSCA: A Pragmatic Approach to Co-Design Automation of Control Dominated Systems," *Hardware/Software Co-design*, NATO ASI Series, Series E: Applied Sciences, Vol. 310, pp. 265-294, Kluwer Academic Publisher, 1996.

[12] SEED (Software-hardware Exploration, a European Demonstration project) project:

www.cefriel.it/projects/seed/mainmenu.html

[13] C.A.R.Hoare, "OCCAM2 Reference Manual", Prentice Hall, Hempstead, Hertfordshire, 1998.

[14] J.Staunstrup and W.Wolf, "Hardware/Software Co-Design: Principles and Practice," Kluwer Academic Publisher, October, 1997.

[15] F.Balarin et Al., "Hardware-Software Co-Design of Embedded Systems: The POLIS approach," Kluwer Academic Publishers, 1997.

[16] A.Balboni, W. Fornaciari and D.Sciuto, "Co-Synthesis and Co-simulation of Control Dominated Embedded Systems," *Int. Journal of Design Automation for Embedded Systems*, Vol. 1, No. 3, Kluwer Academic Publisher, Norwell, MA, Jul. 1996.