Dynamic Modeling of Inter–Instruction Effects for Execution Time Estimation

G. Beltrame[†], C. Brandolese[§], W. Fornaciari[§], F. Salice[§], D. Sciuto[§], V. Trianni[§]

[§] Politecnico di Milano, Piazza L. da Vinci, 32 - 20133 Milano, Italy

{brandole,fornacia,salice,sciuto,trianni}@elet.polimi.it

[†] CEFRIEL, Via R. Fucini, 2 - 20133 Milano, Italy

beltrami@cefriel.it

ABSTRACT

The market for embedded applications is facing a growing interest in power consumption issues: this work is intended to provide a new model to estimate software–level power consumption of 32-bit microprocessors. This model extends previous ones by considering dynamic inter–instruction effects that take place during code execution, providing a static means to characterize the energy consumption associated with them. The model is formally sound: it is conceived for a generic architecture and it has been preliminary validated on the Intel486™ architecture.

1. INTRODUCTION

While there has been a significant research effort in power estimation techniques and low power design tailored for hardware systems, no EDA tools are available to help hardware/software embedded systems designers [4]. The main obstacle is an efficient analysis of the CPU power consumption, necessary to take into account also the software components during design–space exploration, while avoiding to rely with architectural or even layout–level simulation of the microprocessor. To fill such a gap, strategies working at the instruction–level recently appeared in literature. In fact, having a power model of assembly instructions is a value added for designers, since the increasing complexity of embedded systems software is evident and the need of early prototyping of embedded systems, in particular in terms of power consumption, is becoming a must. In [12][13][11] *a priori* knowledge of the current drawn by an instruction is obtained by executing an infinite loop of the target instruction in order to average out fine–grained fluctuations. These approaches are strongly processor–dependent and normally the statistical significance of power figures is not taken into account. A different approach, working on the concept of early *virtual* prototyping of the software for different target CPU cores has been proposed in [1]. This methodology abstracts from the architectural level and focuses on the *functionalities* involved during instruction execution. The resulting functional model decouples the execution time of an instruction from its average power consumption. This allows a *static* (data independent) characterization of each instruction in terms of the energy consumed together with a statistical validation of the resulting software power model. These assumptions are the basis for the work presented in this paper, which concentrates on timing aspects. To consider also the presence of *dynamic* inter–instruction effects such as pipeline interlocks or cache misses, which typically lead to an additional energy consumption not to be neglected in a overall system–level perspective, the previous approaches should be extended. In fact, even if estimates are accurate for the static microprocessor model, the introduction of dynamic inter–instruction effects may cause severe strays from reality for the entire system. The importance of this problem has been recognized in [10], where an instruction–level power model that considered dynamic effects is presented. The solution is based on a modification of the model proposed in [12][13] to obtain a more precise estimate for the base costs. Basically, the authors separated instructions with the same opcode but different addressing modes and added a statistical analysis of cache and pipeline interlock overheads. Unfortunately, this solution is still not general, in the sense that it needs measures for every processor it has to be applied to. The goal of this paper is to overcome the above limitations, providing a general model to describe interlock overheads for different types of processor cores, to complete the information provided via *static* analysis. This model is going to be implemented in a co-design flow in order to obtain a truly accurate and efficient software power estimation tool [9]. This paper is organized as follows: Section 2 defines the problem of considering inter–instruction effects in the scope of a static analysis; Section 3 introduces the mathematical and statistical models; Section 4 describes the methodology for model tuning, and Section 5 presents some experimental results. Finally, some conclusions are drawn in Section 6.

2. PROBLEM DEFINITION

In this work, the model proposed in [1] is extended, taking into account inter–instruction effects related to pipelining. Interlocks are generally related to the execution of a particular sequence of instructions, thus they correspond to *dynamic* events. Taking into account such effects implies either a dynamic analysis, which requires an excessive computational complexity, or an accurate characterization of the