POET

Power Optimization for Embedded sysTems

Demostrators	
	Carlo Brandolese and Daniele Paolo Scarpazza
Institution:	Cefriel / Politecnico di Milano
Address:	Via Fucini 2, I-20133 Milano (Mi), Italy
Phone:	++39-02-23954.325
Fac-simile:	++39-02-23954.254
E-mail:	{brandole,scarpazza}@cefriel.it

Software authors	
	Giovanni Beltrame, Carlo Brandolese, Luca Ceresoli, Francesco Curto, Daniele Paolo Scarpazza
Institution:	Cefriel / Politecnico di Milano
Address:	Via Fucini 2, I-20133 Milano (Mi), Italy
Phone:	++39-02-23954.325
Fac-simile:	++39-02-23954.254

Abstract

POET is an energy consumption estimation and optimization flow for embedded system software. It is able to determine the energy consumption of a given project written in C. It internally relies on three estimation flows: an assembly-level estimation flow, a source-level estimation and optimization flow, and a library estimation flow.

The *assembly-level flow* comprises a C++ programmable framework, which makes it possible to model an arbitrary architecture in terms of its functional units, and to simulate the execution of a binary program over it, thus obtaining synthetic information, such as average functional unit currents, amount of stalls and amount of instruction-level parallelism per instruction class.

The *source-level flow* estimates the energy consumption, the execution time and the occupied size of each C syntax element (from primary expressions up to statements, lines of code and whole functions), relying on the above lower-level estimates provided by the assembly-level flow. It is worth noting that feeding the flow with pre-production architectural estimated data, allows to optimize source code for executors which have not been manufactured yet. The ability to work at the level of abstraction of the C source code increases the simulation speed of orders of magnitude with respect to assembly-level simulations. The flow internally contains time/energy gain models for a number of source code optimizing transformations and allows to interactively perform a fast optimization space exploration on the critical code.

The *library flow* allows to smoothly integrate third-party binary libraries (for which sources could not be available), and provide statistically-accurate estimates of the cost of each library function call. The estimates rely on a set of rich semantic modellings of function signature data types.

An *integrated GUI* allows to automate the interaction between the flows of the methodology, thus automatically analyzing a project, indicating critical code sections and providing gain estimates for a number of source-code transformations.