Energy Estimation for Extensible Processors

Yunsi Fei†, Srivaths Ravi‡, Anand Raghunathan‡, and Niraj K. Jha†
† Dept. of Electrical Engg., Princeton University, NJ 08544, Email: {yfei,jha}@ee.princeton.edu
‡ NEC Labs, Princeton, NJ 08540, Email: {sravi,anand}@nec-lab.com

Abstract

This paper presents an efficient methodology for estimating the energy consumption of application programs running on extensible processors. Extensible processors, which are increasingly popular in embedded system design, allow a designer to customize a base processor core through instruction set extensions. Existing processor energy macro-modeling techniques are not applicable to extensible processors, since they assume that the instruction set architecture as well as the underlying structural description of the micro-architecture remain fixed.

Our solution to this problem is an energy macro-model suitably parameterized to estimate the energy consumption of a processor instance that incorporates any *custom instruction extensions. Such a characterization is facilitated by careful selection of macro-model parameters/variables that can capture both the functional and structural aspects of the execution of a program on an extensible processor. Another feature of the proposed characterization flow is the use of regression analysis to build the macro-model. Regression analysis allows for in-situ characterization, thus allowing arbitrary test programs to be used during macro-model construction.*

We validate the proposed methodology by characterizing the energy consumption of a state-of-the-art extensible processor (Tensilica's Xtensa). We use the macro-model to analyze the energy consumption of several benchmark applications with custom instructions. The mean absolute error in the macro-model estimates is only 3.3%*, when compared to the energy values obtained by a commercial tool operating on the synthesized RTL description of the custom processor. Our approach achieves an average speedup of three orders of magnitude over the commercial RTL energy estimator. Our experiments show that the proposed methodology also achieves good relative accuracy, which is essential in energy optimization studies.*

I. Introduction

Extensible processors are increasingly sought by embedded system designers to meet their conflicting requirements of tight performance and power constraints, high flexibility and short turn-around times. An extensible processor combines the high flexibility of a general-purpose processor and the high efficiency of an application-specific integrated circuit (ASIC) by allowing the designer to extend the instruction set of a base processor core through application-specific (custom) instructions. Thus, extensible processors can benefit embedded system design with their ability to simultaneously tune both the underlying hardware and the application software to meet diverse design requirements.

While commercial vendors of extensible processors, such as [1], [2], [3], [4], do offer design tools to take extensible processors from specification to hardware implementation, a large number of issues remain unresolved. One such open problem is energy estimation for extensible processors, which needs to be efficiently addressed since low power dissipation is a pre-requisite for most embedded systems. Note that if the extensions to the base processor instruction set architecture (ISA) have been decided already, a new energy estimation technique is not required. This is because any existing processor energy estimation/power analysis framework can be used to characterize the extended processor and estimate the energy consumed by an application. However, such an approach is impractical for use in power optimization studies done in an application-specific instruction set processor (ASIP) design cycle, since energy characterization has to be performed for every extended processor.

While existing processor energy estimation techniques are not directly applicable to the problem of efficient energy estimation for extensible processors, they offer valuable insights that can aid in the development of a good solution. Macro-modeling, which we adopt in this work, is a commonly used technique in processor energy estimation. It formulates the energy consumed by the processor in terms of parameters that are easily observable (say, during instruction set simulation). Two categories of processor macro-modeling techniques have been successfully employed. Structural macro-modeling approaches express the overall energy consumption in terms of the energy consumption of its constituent hardware blocks, and use the activity statistics of the hardware blocks for a given program trace to estimate energy. Instruction-level macro-modeling approaches, on the other hand, characterize the energy consumption of the processor instructions using carefully constructed test programs and can use fast instruction set simulation to yield efficient energy estimates. Thus, structural approaches offer the benefits of high accuracy especially if they model the structure of the processor at a fine granularity, while instruction-level approaches facilitate fast energy estimation since they do not have to be cycle-accurate or structure-aware. In the case of extensible processors, which have a fixed base processor core and customizable components (due to instruction set extensions), we hypothesize that a hybrid approach that combines the efficiency of instruction-level approaches with the accuracy of structural approaches is best suited.

A. Paper Overview and Contributions

Our methodology involves deriving a composite energy macro-model by characterizing the energy consumed by applications with custom instructions using (i) instruction-level parameters that capture the interplay of the dynamic execution trace of a program and the processor micro-architecture (inclusive of pipeline stalls and other effects, cache misses, *etc.*), and (ii) structural parameters that account for the energy effect of an instruction (base/custom) on the custom hardware. By using both instruction-level and structural parameters in the macro-model, both efficiency and accuracy can be simultaneously targeted. A significant feature of our macro-modeling flow is that characterization is performed using *regression macro-modeling*, which has the following advantages.

- Variables in a regression macro-model can be chosen from instruction-level or structural domains, or both. Thus, regression macro-modeling is naturally applicable to our hybrid formulation.
- Regression macro-modeling significantly simplifies the process of constructing test applications or programs used in characterization. Conventional instruction-level approaches require test programs that contain isolated instructions, selected instruction sequences, *etc.*, wrapped in loops, in order to infer the average energy consumption of a given instruction under various scenarios. However, test program construction becomes cumbersome in most cases (for example, if test programs need to target instructions such as branch). Regression macro-modeling, through its in-situ character-

Acknowledgments: This work was supported by DARPA under contract no. DAAB07-00-C-L516.

ization, only requires that the test programs have diversity in their instruction statistics so as to cover the instruction space. Thus, arbitrary test programs can be used for regression macro-modeling.
- Construction and use of regression models are efficient, and the tools for building a regression model are widely available.

With the energy macro-model of the extensible processor built in the above manner, energy consumption of an application incorporating any custom instructions can simply be determined by instruction set simulation to capture instruction-level execution statistics, and dynamic resource usage analysis to derive custom hardware activation data needed by the macro-model. Note that energy estimation with this energy macro-model does not require the custom processor to be synthesized. Thus, our methodology is easily usable for evaluating energy-performance trade-offs among different candidate custom instructions. We applied the proposed methodology to characterize the Xtensa extensible processor core from Tensilica Inc. [1]. We then used the energy macro-model of the Xtensa processor to evaluate the energy consumption of several applications with custom instructions. Our experimental results show that the mean absolute error in the macro-model estimates, when compared to the energy values computed by a commercial tool operating on the actual hardware description of extended processors, is only 3.3%, while the average speedup is three orders-of-magnitude.

B. Related Work

Various techniques have been developed to estimate and optimize power or energy consumption of hardware throughout the design hierarchy [5], [6]. Recently, attempts have also been made for energy estimation and optimization of software running on embedded processors. As mentioned before, these approaches can be classified, based on the macro-modeling employed, into *structural* and *instruction-level* techniques.

Structural techniques for energy estimation of software utilize the architectural description of the processor to collect the dynamic activity information for each architectural block using simulation, calculate the energy consumption for each component, and finally sum them up to compute the overall energy consumption. Early work [7] characterized the power consumption of each architectural block as a single number. The power profiler in [8] calculates the energy consumption of functional units based on the switching activity between consecutive cycles. Wattch [9] and SimplePower [10] estimate the energy consumption at each cycle at the architecture level. Commercial tools such as WattWatcher from Sente [11] can also be used for energy estimation, once the RTL hardware description of the processor and the binary image of the program become available. However, RTL simulation on a processor is extremely slow for even small programs and methods for reducing the simulated trace become necessary [12].

Instruction-level macro-modeling techniques compute the energy consumption of a program based on its instruction profile. They primarily rely on the energy consumption characterization of each instruction of the processor and also estimate the energy consumption of special cases (such as cache misses) that can occur during the execution of a program. Characterization of each instruction can be performed by actual current measurements for a processor chip executing carefully created test programs [13]. The techniques in [14] measure the instantaneous processor power to build a software energy estimation model. The accuracy of instruction-level modeling is improved further by the techniques in [15], [16], [17], which are cognizant of variations due to instruction encoding, addressing mode, register fields, operands values, bit toggling on internal and external busses, *etc*. Since the added accuracy comes at the cost of additional CPU time, efficiency is targeted in [16], [18] which perform measurement only on a subset of the instructions (for base energy) and instruction sequences (for inter-instruction effects). Measurement-based approaches are accurate because data are acquired from an actual chip implementation. The same reason, however, makes measurement based techniques infeasible for power trade-off studies early in the design cycle, especially if the hardware architecture is not fixed.

Recent work has focused on building energy/power prediction models for very large instruction word (VLIW) and reduced instruction set computer (RISC) processors using statistical analysis. Instruction-level approaches, such as [18], [19], [20], decompose an instruction into its constituent pipeline functions (for example, fetching and decoding, execution, load and store, write back, *etc.*), and calculate the energy coefficients for these functions, while structural models such as [21] have variables corresponding to the fraction of total instructions executed by each functional unit. The technique proposed in [20] examines instructions with a finer granularity by considering, for example, instruction fetch address, instruction bit encoding, register numbers and immediates, data values, *etc.*

The rest of this paper is organized as follows. Section II briefly examines the Xtensa processor core used in this work. Section III examines the energy macro-modeling requirements for an extensible processor. Section IV describes the proposed energy estimation methodology. Section V presents the results of applying the proposed methodology to build the energy macro-model of the Xtensa processor core and using it to evaluate the energy consumption of applications with different custom instructions. Section VI concludes.

II. The Extensible Xtensa Processor

We use the extensible Xtensa processor from Tensilica [1] as the target processor for macro-modeling and energy estimation. Xtensa's ISA consists of a basic set of instructions which exists in all Xtensa implementations, plus a set of configurable and extensible options [22].

The base ISA defines approximately 80 instructions, and the basic hardware implementation of the Xtensa core is built around a traditional five-stage RISC pipeline, with a 32-bit address space. The configurable options include a wide range of architectural settings. For example, the designer can configure the base processor to include floating point co-processors, customize the memory/cache architecture and register file, and set up interruption/exception mechanisms and levels. Extensibility is achieved by specifying application-specific functionality through custom instructions (also called Tensilica Instruction Extension or TIE). The behavior of the custom instructions is described using a subset of the Verilog hardware description language. Custom instructions can access the general-purpose register file of the base processor or additional custom registers/register files for their computations. They can be used to perform complex computations, which can take multiple clock cycles to complete. The TIE compiler processes the custom instruction specification and facilitates seamless integration of the added custom hardware with the base processor configuration. Control logic, such as the TIE instruction decoder, bypass logic, interlock detection, and immediate-generation logic required by the custom instructions are automatically generated. After the custom instructions are incorporated, a processor generator automatically generates the enhanced processor, and the corresponding software development environment. In this way, both the hardware and software are tuned for specific applications.

III. Extensible Processor Energy Macro-model Requirements

In this section, we illustrate with an example the different factors which must be considered in building an energy macro-model for an extensible processor.

*E*xample 1: Fig. 1(a) shows a portion of the architecture of an extended processor, wherein the base processor datapath has been

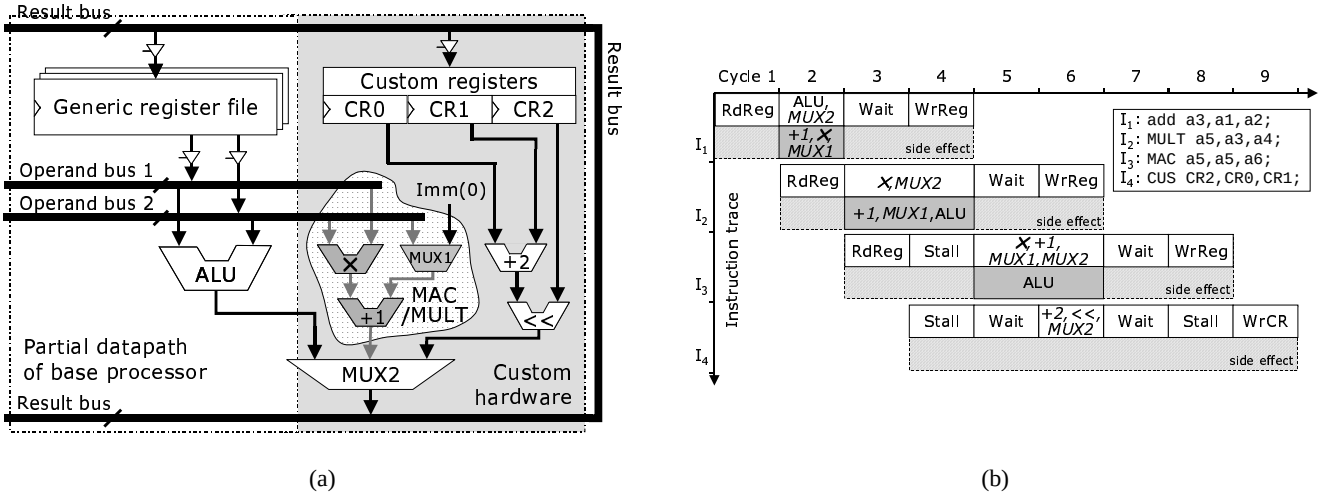

Fig. 1. (a) An extended processor, and (b) a snapshot of dynamic execution of a program on the processor

augmented with custom hardware needed to implement three custom instructions: $MULT$, MAC and CUS. Base processor arithmetic instructions execute on the datapath portion shown with a generic register file, an ALU, two operand buses and one result bus. Custom instructions $MULT$ and MAC perform their respective functionality (multiply and multiply-accumulate) on data values off the operand buses using shared custom hardware, while custom instruction CUS accesses custom registers $CR0, \cdots, CR2$ for its operands.

A snapshot of the dynamic execution of an application is captured in Fig. 1(b). Four instructions $I_1, \cdots, I_4$ are shown in the trace, which correspond to base processor instruction add and custom instructions $MULT$, MAC and CUS, respectively. For each instruction, the top horizontal bar lists the sequence of processor events dictated by its execution. For example, instruction I_1, which is an add instruction, executes by first reading data off the generic register file onto the operand buses ($RdReg$), then performing the add operation on the ALU and writing onto the result bus (ALU, $MUX2$), and writing back to the register file ($WrReg$) after a latency of one cycle due to pipeline effects ($Wait$). Stalls, if any, are also indicated in the figure.

Since the execution of an instruction can activate other portions of the processor (side effects), the bottom bar for each instruction depicts the side effects in either the base processor or the custom hardware. For example, the execution of the base processor instruction add activates custom hardware ($+1$, X, $MUX1$) in the second cycle. This occurs because the custom hardware and the ALU share the same operand buses. Similarly, the execution of some custom instructions (I_2 and I_3) can activate the base processor hardware, while the execution of other custom instructions (I_4) may be independent of base processor hardware.

In order to compute the energy consumption of this instruction trace, we need an energy macro-model for the extensible processor which can (a) account for the energy consumed by a base processor (custom) instruction on the base processor (custom) hardware, (b) model inter-instruction dependencies, pipeline effects and other non-idealities which manifest as stalls, cache misses, *etc.*, and (c) capture the activation of custom (base processor) hardware by base processor (custom) instructions.

IV. Macro-modeling and Estimation Methodology

In this section, we present the proposed methodology for estimating the energy consumption of an application with (any) custom instruction enhancements running on an extensible processor. Section IV-A presents an overview of our methodology, while Section IV-B details the constituent steps.

A. Overview

Fig. 2 shows the different steps involved in performing energy estimation for an extensible processor. The basic flow involves (a) characterizing the energy consumption of the extensible processor through regression macro-modeling (steps **1-8**), and (b) profiling an application with custom instruction extensions dynamically to determine the statistics associated with the macro-model variables, thus, computing the energy consumption (steps **9-11**).

Building an energy macro-model involves first selecting the extensible processor parameters on which the energy consumption of an instruction trace depends and constructing a template that expresses the energy consumption (dependent variable) as a function of those parameters (independent variables) (step **1**). We use a linear macro-model template in our analysis, since construction and use of linear macro-models is efficient. The linear macro-model template [1] expresses the energy consumed, E, as a linear function of $X_1, X_2, \cdots, X_n$, which are characteristic variables of a program running on the extensible processor. In other words,

$$E = \alpha_0 + \alpha_1 X_1 + \alpha_2 X_2 + \cdots + \alpha_n X_n \quad (1)$$

where $\alpha_0, \alpha_1, \cdots, \alpha_n$ are constants called the energy coefficients. Variables $X_1, X_2, \cdots, X_n$ are chosen from both instruction-level and structural domains. Instruction-level parameters are employed for characterizing instruction effects on the fixed base processor core, while structural parameters are used to characterize instruction effects on custom hardware due to any base or custom instructions (see Section IV-B).

The energy coefficients in the macro-model are determined using regression analysis (step **8**). Since any test program can be used for building the regression model, we merely ensure that the different instructions in the base processor ISA are covered (step **2**). The input space of custom instructions that can be added to an extensible processor ISA is, however, exponential for a given choice of custom hardware library components. Therefore, the test program suite also incorporates custom instructions so as to cover all

[1] Note that linear macro-models require linearity in the coefficients, not in the parameters . Hence, polynomial and other non-linear functions of macro-model parameters can be present in the template.

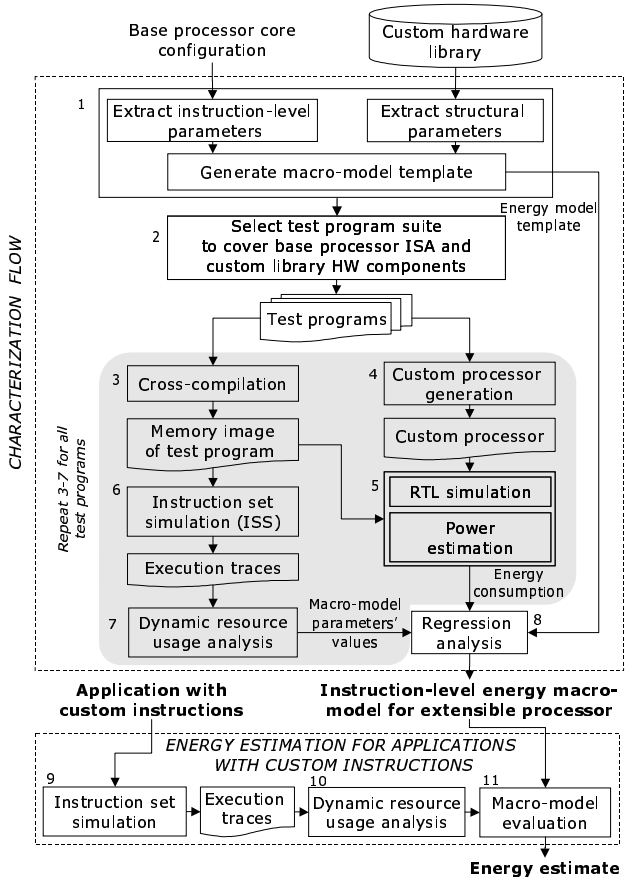

Fig. 2. Extensible processor energy estimation flowchart

the custom hardware library components. Instruction set simulation of a test program (step **6**) and dynamic resource usage analysis (step **7**) are used to determine the values of the independent variables in the macro-model template, while energy estimation of the test program executing on the RTL description of the processor (step **5**) is used to measure the value of the dependent variable. If the test program incorporates custom instructions, the processor used in the above step corresponds to a custom processor that has been augmented with the custom instructions. Note that, while custom processors are generated during characterization, they are not needed for using the macro-model to compute application energy consumption. Steps **3-7** are repeated for all the test programs to gather data needed by regression macro-modeling to find estimates of the energy co-efficients, thus completing characterization.

When the energy consumption of an application with custom instructions has to be estimated (step **9-11**), instruction set simulation (step **9**) is first performed to gather execution statistics (values of instruction-level macro-model variables) as well to generate the dynamic execution trace. Since the integration of custom hardware (due to the custom instructions) with the processor architecture is defined in an extensible processor design flow, we analyze the resource usage (step **10**) of each instruction in the execution trace to determine the activation (if any) of custom hardware. This analysis yields the values of the structural macro-model variables. The parameter values are fed to the energy macro-model to yield the energy consumed by the application.

B. Details

In this section, we focus on the implementation of salient steps of our methodology. Section IV-B.1 discusses energy macro-model template generation, while Section IV-B.2 examines macro-model fitting using regression analysis.

B.1 Energy Macro-model Template Generation

The energy macro-model template used in our analysis is linear, with two components as shown below.

$$E = E_{ins}(X_1, \cdots, X_m) + E_{struc}(X_{m+1}, \cdots, X_n) \quad (2)$$

where E_{ins} and E_{struc} are linear functions of instruction-level parameters (X_1,$\cdots$,X_m) and structural parameters (X_{m+1},$\cdots$,X_n), respectively.

Instruction-level Macro-model Variables

The instruction-level macro-model variables are chosen to reflect the usage of the base processor core due to either base processor or custom instructions. In the process, we also select parameters to consider the effect of non-ideal cases such as instruction cache miss, data cache miss, uncached instruction fetching, data or control dependent interlocks, *etc.*, that occur during the execution of a program.

Equation (3) illustrates the use of instruction-level macro-model variables to compute E_{ins}.

$$\begin{aligned} E_{ins} = & E_{arith} \cdot Cyc_{arith} + E_{ld} \cdot Cyc_{ld} + E_{st} \cdot Cyc_{st} + \\ & E_j \cdot Cyc_j + E_{br\_tk} \cdot Cyc_{br\_tk} + E_{br\_utk} \cdot Cyc_{br\_utk} + \\ & E_i \cdot Num_i + E_d \cdot Num_d + E_{uncache} \cdot Num_{uncache} + \\ & E_{interlock} \cdot Num_{interlock} + E_{side\_tie} \cdot Cyc_{side\_tie} \end{aligned} \quad (3)$$

wherein E_{ins} is expressed as a linear sum of the following.

- Energy due to the base processor functionality exercised by an instruction belonging to the base processor ISA. Experimental studies of energy profiles of processor instructions in the literature suggest that instructions in the base processor ISA can be clustered into arithmetic ($arith$), load (ld), store (st), jump (j), branch taken (br_tk), and branch untaken (br_utk) classes [13]. Macro-model variables $Cyc_{arith}, \cdots, Cyc_{br\_utk}$ represent the number of cycles taken by each instruction class in the dynamic execution trace of the program. Such a clustering is convenient (and later seen to be accurate) since macro-model variables do not have to be separately present for individual instructions in the base processor ISA.
- Energy due to dynamic effects manifested as instruction-cache misses (i), data-cache misses (d), uncached instruction fetches ($uncache$) and processor interlocks ($interlock$) [13]. Macro-model variables $Num_i, \cdots, Num_{interlock}$ denote the number of times each non-ideal case occurs during program execution.
- Energy due to side-effects in the base processor imposed by custom instructions ($side\_tie$). The macro-model variable Cyc_{side_tie} accounts for the number of cycles taken by custom instructions which access the generic register file for their operation.

Structural Macro-model Variables

Structural macro-model variables reflect the usage of custom hardware due to the execution of either base processor or custom instructions. The variables are chosen to account for the number of cycles for which a custom hardware component is active during the execution of a program. All the components present in the custom hardware library should, therefore, be covered by these macro-model variables. The following considerations are made in the process.

- For efficiency, the components in the library are classified into several categories based on empirical studies of their average energy consumption. In the context of the custom hardware library used by TIE instructions, we classified the basic primitives into five

categories (1) multiplier, (2) adder, subtractor and comparators, (3) bit-wise logic, reduction logic, and multiplexers, (4) shifter, and (5) custom registers. Additional categories correspond to specialized modules available for TIE instructions, namely, (6) TIE_mult, (7) TIE_mac, (8) TIE_add, (9) TIE_csa, and (10) table.

- The energy consumption of a hardware component depends significantly on its bit-width (or the number of entries and bit-width of each entry in the case of a table). We use C to represent the complexity of a hardware module and n_b the bit-width. The dependence on bit-width is linear ($C \propto n_b$) in the case of hardware components such as adders, multiplexers, *etc.*, while the dependence is quadratic in the case of a multiplier ($C \propto n_b^2$).

Equation (4) expresses the custom hardware energy consumption, E_{struc}, where macro-model variable $Cyc_{i,j}$ (i = 1, 2, ..., 10) denotes the number of cycles in which the jth functional block belonging to component category i is active, and $C_{i,j}$ represents the dependency of this functional block on the bit-width (number of entries).

$$E_{struc} = E_1 \cdot \sum_j C_{1,j} Cyc_{1,j} + E_2 \cdot \sum_j C_{2,j} Cyc_{2,j} + \cdots + E_{10} \cdot \sum_j C_{10,j} Cyc_{10,j} \quad (4)$$

B.2 Macro-model Fitting through Regression Analysis

Regression analysis is used to determine the energy coefficients in the macro-model template shown in Equation (2) (with E_{ins} and E_{struct} as given in Equations (3) and (4), respectively). Test programs are used to gather both energy consumption values and execution statistics that correspond to the different macro-model variables in the equation. For a set of n test programs, the energy consumption data are grouped into an $n \times 1$ column vector, E, while the values corresponding to the macro-model variables are grouped into an $n \times 21$ matrix, M. In such a case, model fitting through regression analysis involves solving the linear matrix equation, $M \times C = E$, in which C is the energy coefficient vector corresponding to [E_{arith}, E_{ld}, E_{st}, E_j, E_{br_tk}, E_{br_utk}, E_i, E_d, $E_{uncache}$, $E_{interlock}$, E_{side_tie}, E_1, E_2, E_3, E_4, E_5, E_6, E_7, E_8, E_9, E_{10}].

In the above formulation, let $\widetilde{C}$ represent the estimate of C, and $\widetilde{E}$ represent the estimate of E. Solution of the matrix equation using the pseudo-inverse method [23] yields the values for the energy coefficients vector C as shown in Equation (5), such that the square error $||E - \widetilde{E}||^2$ is minimized.

$$\widetilde{C} = (M^T \times M)^{-1} \times M^T \times E \quad (5)$$

The macro-model with these energy coefficient values best fits the data acquired using the test programs.

V. Experimental Results

We have implemented the proposed flow described in Section IV using commercial tools and scripts to perform several key tasks in the methodology. The target extensible processor used in our experiments is the Xtensa processor from Tensilica [1]. The base processor is a T1040.0 version of the processor running at 187 MHz (0.18μ technology), and the configuration includes a 32-bit multiplication instruction, 4-way 16 KB instruction and data caches, 32-bit wide system bus and a generic register file with 64 32-bit registers.

Characterization in our set-up proceeds as follows. The GNU-based cross-compiler and instruction set simulator of the Xtensa software development kit are used to cross-compile and simulate test programs to gather execution statistics (cycle count, cache misses, *etc.*) needed by the macro-model (steps **3**, **6**, **9** in Fig. 2). Test programs are Tensilica benchmarks written in C, while custom instructions are written in TIE. Test programs instantiate TIE instructions intrinsic in their description, which are cross-compiled to generate executables for instruction set simulation. The Xtensa processor generator is used to generate the RTL description of the base/custom processors needed during characterization. The RTL description is simulated with the memory images of the test programs using ModelSim [24] to generate the simulation traces needed by the RTL power estimator WattWatcher [11] (steps **5**). The energy values and the execution statistics thus obtained are used by the tool Splus [25] to perform regression and derive the energy macro-model. The results of characterization are presented in Section V-A, while the results of applying the macro-model to evaluate the energy consumption of different applications are presented in Section V-B.

A. Energy Macro-model for the Xtensa Processor

The energy macro-model template for the Xtensa processor is shown in Equation (2), with the instruction-level and structural macro-model terms given in Equations (3) and (4), respectively. There are 21 macro-model variables in the template, whose coefficients are determined through regression analysis. Table I presents these coefficients which indicate the per-cycle estimates of the base processor energy consumption for each base processor instruction category ($E_{arith}, \cdots, E_{br\_utk}$), the per-miss/per-fetch/per-interlock estimates of the energy consumptions for execution-time effects ($E_i, \cdots, E_{interlock}$), and the per-cycle estimates of the energy consumption of the side-effects of custom instructions on the base processor (E_{side_tie}), as well as the unit energy consumption (per-cycle, per-bit) for the different custom hardware library components ($E_1, \cdots, E_{10}$). The fitting errors corresponding to test programs are shown in Fig. 3. The maximum error is under 8.9%, and the root mean square error is 3.8%.

TABLE I
Energy coefficients of the characterized Xtensa processor

Energy coefficient	Description	Value
E_{arith}	arithmetic instruction	$0.638nJ$
E_{ld}	load instruction	$0.512nJ$
E_{st}	store instruction	$1.092nJ$
E_j	jump instruction	$0.603nJ$
E_{br_tk}	branch taken	$0.504nJ$
E_{br_utk}	branch untaken	$0.345nJ$
E_i	instruction cache miss	$2.93nJ$
E_d	data cache miss	$4.34nJ$
$E_{uncache}$	uncached instruction fetch	$2.42nJ$
$E_{interlock}$	processor interlock	$0.98nJ$
E_{side_tie}	side effects due to custom instructions	$0.616nJ$
E_1	*	152.0 fJ
E_2	+/-/comp	70.0 fJ
E_3	log/red/mux	12.0 fJ
E_4	shifter	377.0 fJ
E_5	custom register	177.0 fJ
E_6	TIE_mult	165.0 fJ
E_7	TIE_mac	190.0 fJ
E_8	TIE_add	69.0 fJ
E_9	TIE_csa	37.0 fJ
E_{10}	table	27.0 fJ

B. Applying the Energy Macro-model: Accuracy Results

In this section, we present the results of applying the energy macro-model described in Section V-A.

Our first experiment involves determining the energy consumption of several applications (incorporating different custom instruc-

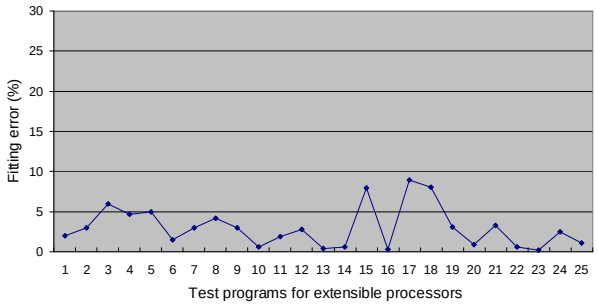

Fig. 3. Fitting error of the test programs for extensible processor core

tions) in two ways: (i) using the derived energy macro-model, and (ii) using the RTL power estimation tool WattWatcher [11] on the synthesized RTL description of the corresponding extended processor. Table II summarizes these results. For the ten application benchmarks shown (different from the test programs used in macro-modeling), the maximum estimation error is 8.5%, while the average absolute error is only 3.3%. The proposed energy estimation methodology is very fast. It takes only a few seconds for application energy estimation using our approach, while the average time taken by WattWatcher to determine the energy consumption of a small application is several hours (an average speedup of three orders of magnitude).

TABLE II
APPLICATION ENERGY ESTIMATES: ACCURACY RESULTS

Application	Estimate (μJ)	WattWatcher (μJ)	Error (%)
Ins_sort	336.9	344.5	-2.2
Gcd	736.5	723.5	1.8
Alphablend	106.9	105.7	1.1
Add4	595.0	583.9	1.9
Bubsort	131.5	126.7	3.8
DES	45.6	43.7	4.3
Accumulate	37.6	35.4	6.2
Drawline	9.9	9.7	2.0
Multi_accumulate	23.8	26.0	-8.5
Seq_mult	13.5	13.7	-1.5

We also performed an additional experiment to study the relative accuracy of the macro-model, when used in energy optimization studies for an application with many custom instruction choices. Fig. 4 shows the energy estimates obtained by our macro-model and WattWatcher for a single application (implementing the Reed-Solomon decoding/encoding algorithm) with four different custom instruction choices. The results show that the energy estimates returned by both these approaches are comparable, while the two profiles track one another. Thus, good relative accuracy is achieved.

VI. CONCLUSIONS

In this work, we presented an efficient framework for characterizing the energy consumption of an extensible processor. Our characterization flow facilitates the construction of an efficient macro-model by using parameters drawn from both instruction-level and structural domains, and by leveraging the benefits of regression analysis. Since application of the macro-model only requires instruction set simulation based analysis of the application, energy estimation using our approach is very fast. We characterized the energy consumption of a state-of-the-art extensible processor and used the macro-model so obtained to derive highly accurate energy estimates for several applications.

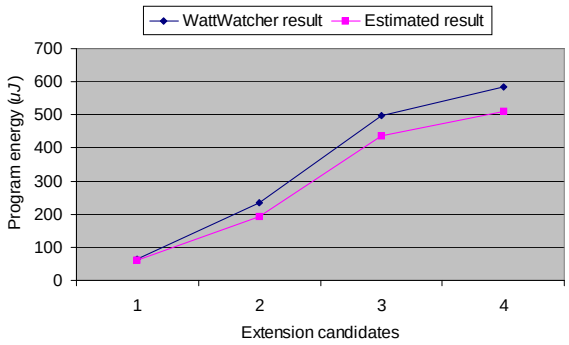

Fig. 4. Energy consumption estimates of an application for various custom instruction choices derived using our macro-model and WattWatcher

References

[1] *Xtensa microprocessor*, Tensilica Inc. (http://www.tensilica.com).
[2] *ARCtangent processor*, Arc International. (http://www.arc.com).
[3] *Jazz DSP*, Improv Systems Inc. (http://www.improvsys.com).
[4] *SP-5flex DSP core*, 3DSP Corp. (http://www.3dsp.com).
[5] J. Rabaey and M. Pedram (Editors), *Low Power Design Methodologies*, Kluwer Academic Publishers, Norwell, MA, 1996.
[6] A. Raghunathan, N. K. Jha, and S. Dey, *High-level Power Analysis and Optimization*, Kluwer Academic Publishers, Norwell, MA, 1998.
[7] P. W. Ong and R. H. Yan, "Power-conscious software design - A framework for modeling software on hardware," in *Proc. Int. Symp. Low Power Electronics & Design*, Oct. 1994, pp. 36–37.
[8] H. Mehta, R. M. Owens, and M. J. Irwin, "Instruction level power profiling," in *Proc. Int. Conf. Acoustics Speech & Signal Processing*, 1996.
[9] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," in *Proc. Int. Symp. Computer Architecture*, June 2000, pp. 83–94.
[10] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "The design and use of SimplePower: A cycle-accurate energy estimation tool," in *Proc. Design Automation Conf.*, June 2000, pp. 340–345.
[11] *WattWatcher manual*, Sente Inc. (http://www.sequencedesign.com).
[12] C-T. Hsieh, M. Pedram, G. Mehta, and F. Rastgar, "Profile-driven program synthesis for evaluation of system power dissipation," in *Proc. Design Automation Conf.*, June 1997, pp. 576–581.
[13] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: A first step towards software power minimization," *IEEE Trans. VLSI Systems*, vol. 2, no. 4, pp. 437–445, Dec. 1994.
[14] J. Russell and M. Jacome, "Software power estimation and optimization for high-performance 32-bit embedded processors," in *Proc. Int. Conf. Computer Design*, Oct. 1998, pp. 328–333.
[15] N. Chang, K. Kim, and H. G. Lee, "Cycle-accurate energy consumption measurement and analysis: Case study of ARM7TDMI," in *Proc. Int. Symp. Low Power Electronics & Design*, Aug. 2000, pp. 185–190.
[16] A. Sama, M. Balakrishnan, and J. F. M. Theeuwen, "Speeding up power estimation of embedded software," in *Proc. Int. Symp. Low Power Electronics & Design*, Aug. 2000, pp. 191–196.
[17] S. Steinke, M. Knauer, L. Wehmeyer, and P. Marwedel, "An accurate and fine grain instruction-level energy model supporting software optimizations," in *Proc. Int. Wkshp Power & Timing Modeling, Optimization & Simulation (PATMOS)*, Sept. 2001.
[18] C. Brandolese, W. Fornaciari, F. Salice, and D. Sciuto, "An instruction-level functionality-based energy estimation model for 32-bit microprocessors," in *Proc. Design Automation Conf.*, June 2000, pp. 346–351.
[19] M. Sami, D. Sciuto, C. Silvano, and V. Zaccaria, "Instruction-level power estimation for embedded VLIW cores," in *Proc. Int. Wkshp. Hardware/Software Codesign*, Mar. 2000, pp. 34–38.
[20] S. Lee, A. Ermedahl, S. L. Min, and N. Chang, "An accurate instruction-level energy consumption model for embedded RISC processors," in *Proc. ACM SIGPLAN Wkshp. Languages, Compilers, & Tools for Embedded Systems*, June 2001, pp. 1–10.
[21] C. H. Gebotys, R. J. Gebotys, and S. Wiratunga, "Power minimization derived from architectural-usage of VLIW processors," in *Proc. Design Automation Conf.*, June 2000, pp. 308–311.
[22] R. E. Gonzalez, "Xtensa: A configurable and extensible processor," *IEEE Micro*, vol. 20, no. 2, pp. 60–70, Mar.-Apr. 2000.
[23] S. Chatterjee, A. S. Hadi, and B. Price, *Regression Analysis by Examples, 3rd ed.*, John Wiley & Sons, 2000.
[24] *ModelSim*, Model Technology (http://www.model.com).
[25] W. N. Venables and B. D. Ripley, *Modern Applied Statistics with S-PLUS*, Springer-Verlag, 1998.