Submitted to special issue on Power-Aware Embedded Computing:
The Role of Static, Dynamic, and Adaptive Techniques

Statistical Derivation of an Accurate Energy Consumption Model for Embedded Processors[*]

Sheayun Lee[1] Andreas Ermedahl[2] Sang Lyul Min[1] Naehyuck Chang[1]

Abstract

The energy consumption of software is becoming an increasingly important issue in designing mobile embedded systems where batteries are used as the main power source. As a consequence, recently, a number of promising techniques have been proposed to optimize software for reduced energy consumption. Such low-power software techniques require an energy consumption model that can be used to identify the factors contributing to the overall energy consumption. We propose a technique to derive an accurate energy consumption model by abstracting the energy behavior of the target processor. The proposed approach combines empirical measurement with a statistical analysis technique to approximate the actual energy consumption, whose result is a model equation that can be used to estimate software energy consumption. The model equation also provides insightful information that can be used in program optimization for low energy, by identifying the factors affecting the energy consumption of software. Experimental results show that the model equation can accurately estimate the energy consumption of a random instruction sequence, with an average error of 2.5 %.

[*] An earlier version of this paper appeared in the Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems, 2001 [16].

[1] School of Computer Science and Engineering, Seoul National University, Seoul 151-742, Korea. (TEL) +82-2-880-1831, (FAX) +82-2-885-7296, (EMAIL) sylee@archi.snu.ac.kr, symin@dandelion.snu.ac.kr, naehyuck@snu.ac.kr.

[2] Dept. of Information Technology, Uppsala University, SE-751 05 Uppsala, Sweden. (TEL) +46-18-4713172, (FAX) +46-18-550225, (EMAIL) ebbe@docs.uu.se.

1 Introduction

Energy consumption of software has recently emerged as an important metric of system performance with the growing requirement for low-energy computing. Especially for embedded systems, there is a high demand for optimization techniques that enable energy reduction for software, since an increasing number of applications are powered by batteries. Therefore, several promising techniques have been proposed for reducing the energy consumed by software, focusing on program optimization [12, 18, 24]. Such program optimization techniques require a detailed program cost model represented in terms of energy consumption that can guide the decisions on program transformation.

We propose a technique to derive an accurate energy consumption model by abstracting the energy behavior of the target processor, which can be used to identify the important factors contributing to software energy consumption and thus provide insightful information to a program optimizer. At the same time, the energy consumption model resulting from the proposed technique can be used to estimate the energy consumption of software at a high level of accuracy, based on a simple model equation. For the energy consumption model to be applicable to a program optimization framework, it must have the following properties.

- **Accountability**: The model should be able to identify the factors that affect the energy consumption of software. Moreover, it should be able to indicate the significance of each contributing factor.
- **Accuracy**: The model should be able to accurately estimate the energy cost of instructions, by reflecting the actual energy behavior of the target processor.
- **Simplicity**: The model should be constructed using simple properties accessible from the program's point of view, so that it can be easily incorporated into a program optimizer.
- **Generality**: The model derivation technique should be independent of any specific implementation, so that the technique can be applied to different target processors.

To derive an energy consumption model, we combine an empirical method and a statistical approach. That is, we first assume a hypothetical model equation that is composed of various aspects of instruction execution that can possibly influence the energy behavior of the processor. Then we determine the yet unknown parameters in this equation by applying a *stimulus-response* approach as illustrated in Figure 1, where we run a set of test programs (stimulus) and observe the energy consumption measured from real hardware (response). For the purpose of approximating the model equation to the actual energy consumption data gathered from measurement, a statistical analysis technique called *regression analysis* is used. That is, we derive the energy model equation by investigating a large number of sample combinations of instructions executed and the corresponding energy consumption data.

The major contributions of the proposed technique are as follows. First, using the proposed technique, we can identify the important factors that influence the energy consumption of instruction execution. Second, the technique provides a systematic approach for estimating software energy consumption based on a simple linear equation. Third, the framework used in the model derivation is generally applicable to a variety of different processors, since it does not depend on implementation information of a specific target processor. That is, the technique requires only basic information about the ISA (Instruction Set Architecture) and the instruction pipeline organization. Finally, the availability of such a detailed energy consumption model can direct future research for developing aggressive program optimization techniques geared towards software energy reduction.

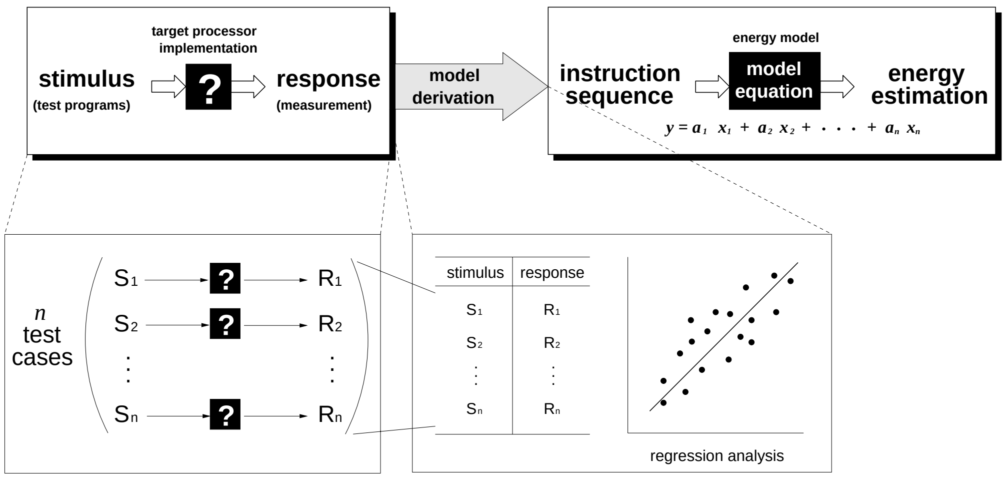


Figure 1: The stimulus-response approach for energy model derivation.

The rest of the paper is organized as follows. Section 2 discusses previous works related to our research. In Section 3, we explain the basics of energy consumption in CMOS processors, and describe the method used in gathering the cycle-level energy consumption data used in the model derivation. Section 4 details the proposed approach for deriving an energy consumption model by abstracting the energy behavior of the processor. Our experimental setup and results are presented in Section 5. Finally, Section 6 concludes the paper by outlining possible future extensions to this work.

2 Related Work

Recently, attempts have been made to construct energy consumption models for software. The previous approaches are based either on software simulation of the processor, or on direct measurement of energy. In *simulation-based* methods, the energy consumed by software is estimated by calculating the energy consumption of various components in the target processor through simulations at different levels. The benefit of these simulation-based approaches is that they provide high flexibility, i.e., we can easily add, remove, or modify components of the simulation, or even retarget the simulator to different processors. Besides, these methods have the advantage that they can cover a large spectrum of abstraction level. For example, we can develop a circuit-level or gate-level simulator when accurate details are required, while a behavioral function simulator can be used to abstract the processor's energy consumption.

On the other hand, in the *measurement-based* approaches, the energy consumption of software is characterized by examining the data obtained from real hardware. The major advantage of the measurement-based approaches is that the resulting energy model is close to the actual energy behavior of the processor, because the data is acquired from the hardware itself. Another good point is that the complexity of the model derivation is much lower than in the simulation-based approaches, because the measurement-based approaches generally do not require the detailed information about the hardware implementation.

Both simulation-based methods and measurement-based methods can be classified according to their level of abstraction used in modeling energy behavior of processors. At the highest level of abstraction, efforts are focused on accounting for the energy costs of instructions. Typically, attempts have been made to characterize the energy consumption of instructions based on measuring the average current drawn by the processor. In this class of modeling techniques, Tiwari et al. [23] describe a seminal approach, where they model the software energy consumption using a power cost table that records the unique base cost for each instruction and the inter-instruction effects. The base cost of an instruction is defined as the average current drawn by that instruction executed repeatedly in a tight loop, multiplied by the number of cycles taken by each instance of the instruction. On the other hand, the inter-instruction effect is defined as the additional power cost incurred by executing different instructions sequentially. However, recording this inter-instruction effect requires $O(N^2)$ space where N is the number of instructions in the instruction set. To rectify this problem, a technique is proposed to group the instructions into common classes [15]. These techniques provide a simple framework for software energy estimation by summarizing the energy consumption of instructions in the form of a table. However, by relying on the average current, they largely ignore the detailed impacts of various factors that affect the energy consumption at the instruction level. In contrast to these approaches based on the average current, Russell and Jacome [19] present a software energy estimation model based on instantaneous power measured by a digitizing oscilloscope. In this work, they propose a simple constant parameter model, which is validated by a statistical inference technique.

The next highest level of abstraction is centered around functional unit modeling, where analytical models are typically used for each component comprising the processor. For example, Mehta et al. [17] propose a power profiler that records the information of the previous and the current states of functional units, as well as the correlated switching capacitance. As an extension to this approach, Chen et al. [8] present a technique that can be used to estimate the cycle-level energy consumption data based on hierarchical decomposition of the architectural features of the target processor. A more generalized form of an RT (register transfer) level energy simulator called SimplePower is proposed in [26]. SimplePower provides detailed energy information about the instruction execution on a pipelined RISC processor using accurate architecture-level simulation of internal components.

At the lowest level of abstraction, an energy model is constructed using circuit-level/gate-level simulation or cycle-level measurement. Typically, efforts are made to characterize the energy behavior of the target processor by describing the low-level implementation of the target processor or by measuring detailed cycle-level energy consumption data. Klass et al. [14] analyze the effect of sequential execution of different instructions, using a gate-level analysis tool. In their approach, the inter-instruction effect is modeled by additional energy consumption observed when each instruction is executed after a NOP instruction. While the above approach is based on an analytical model, Chang et al. [6] present a technique to derive a fine-grained energy model by measuring the cycle-level energy consumption using special measurement hardware. They also analyze the impact of various properties of instructions on the energy consumption, showing that software energy consumption is dependent on the factors such as register numbers, immediate operands, etc.

When an energy model is constructed at a high level of abstraction, we have the benefit of being able to relate the energy consumption behavior of the processor to program execution information, thus providing useful information to program optimization techniques. However, compared to the lower-level energy models, these high-level techniques lack the ability to give detailed information about the variation caused by the actual behavior of the processor, which is often masked by abstraction. On the other hand, the advantage of a low-level energy model is that it provides a better mechanism for energy estimation in terms of accuracy. Since the techniques are based on detailed and fine-grained energy consumption information, the resulting models accurately reflect every small element that contributes to the total energy consumption, which cannot be provided by high-level modeling techniques. However, such low-level techniques have the disadvantage that the model derivation is too complex to be practically applied to software optimization techniques. In other words, by putting more effort to model the details of hardware implementation, we gain model accuracy at the cost of degraded accountability.

The technique described in this paper can be viewed as a hybrid approach, where we try to provide highly abstract information while keeping the model as accurate as possible by relying on detailed measurement data. It is distinguished both from the simulation-based approaches in that we do not describe the internal implementation, and from approximation methods based on average current, where various aspects of the processor's energy behavior are largely ignored. To derive an abstract model from the detailed cycle-level measurement data, we incorporate a well-defined modeling technique called linear regression analysis. Such statistical analysis techniques have been used in previous works for energy modeling methods, since the technique is in nature suitable for approximating the relationship between observed data and the factors assumed to influence them. Gebotys et al. [10, 11] propose an energy estimation and optimization technique for VLIW processors, incorporating a statistical method for analyzing the functional unit usage patterns of instructions. This approach tries to predict the energy consumption of software using regression analysis. The prediction is used to minimize the energy consumption with respect to the average current drawn. Recent studies due to Brandolese et al. [3] and Sami et al. [20] present techniques to estimate the software energy consumption using a combination of functional decomposition and statistical analysis techniques. These approaches are focused on modeling the energy consumption in terms of the usage of various functional units, mainly targeted for VLIW processors.

3 Processor Energy Consumption and Measurement

In Section 3.1, we present the basics of the energy consumption in CMOS processors, to form a basis for modeling the energy behavior using a linear equation. Section 3.2 briefly explains the design and implementation of a measurement scheme used to provide cycle-level energy consumption information, which is used in our model derivation.

3.1 Energy Consumption of CMOS Processors

The total energy consumption of a CMOS (Complementary Metal Oxide Semiconductor) circuit, which is commonly used for building microprocessors, consists of the following three components [9].

- **Switching power**: The energy consumed when a gate with output capacitance of C is charged and discharged, is given by
$$E = \frac{1}{2} \times C \times V_{dd}^2 \,, \tag{1}$$
where V_{dd} is the supply voltage of the circuit.
- **Short-circuit power**: A CMOS circuit consists of both p-type and n-type transistors. When the input of a gate is at an intermediate level, both the p-type and n-type networks can conduct, causing a short-circuit path from V_{dd} to ground.
- **Leakage current**: The transistor networks conduct a small current even when the circuit is not active. The current is in general negligibly small as compared with the above two components, but becomes significant when transistors are operated at a low voltage [21].

In a well-designed active CMOS circuit, the switching power dominates, with the short-circuit and the leakage accounting for only a small portion of the total energy consumption [9]. Therefore, the power dissipation, hence the energy consumption of a CMOS circuit largely depends on the switching activity of the circuit, which is a function of the current inputs and the previous states of the circuit.

This suggests that the energy consumption is linearly dependent on the number of state transitions, i.e., the Hamming distances between the current and the previous states of the circuit. However, when dynamic CMOS circuits are used in the implementation, which is the case for high-performance datapaths used in a number of modern embedded systems, the total energy consumption has also a component that is not proportional to the Hamming distance mentioned above. This is due to the *precharge-and-evaluation* scheme, as described in the following. The dynamic CMOS circuits precharge before every evaluation, for the sake of fast operation. Therefore, the circuit draws a large current causing high energy consumption if it has been discharged in the previous evaluation, i.e., in the previous clock cycle. On the other hand, the circuit draws only a small current if the circuit has not been discharged, i.e., when it has been left charged in the previous clock cycle. Accordingly, the energy consumption of this dynamic CMOS circuit is proportional to the weight of the current data, i.e., the number of 1's (or the number of 0's, depending on the circuit structure). This observation motivates us to abstract the energy behavior of a processor using a model equation that is a linear combination of Hamming distances and weights of a number of variables, as will be explained in Section 4.1.

3.2 Cycle-Level Energy Measurement Hardware

In previous approaches for energy model construction, standard instrumenting equipment has been used to measure the energy consumption of processors. However, these methods have limitations that they do not provide accurate energy consumption information with high precision. For example, digital multimeters can only measure average current drawn when the target processor executes the same instructions repeatedly [23], mainly due to its slow response. To rectify this problem, a high-bandwidth digitizing oscilloscope can be used to measure instantaneous power [19] instead of the average current, by capturing the voltage envelope. This method, however, has a drawback that the measurement data can be corrupted by current spikes, which is common in digital systems.

Therefore, to provide cycle-level energy consumption information, we propose a novel technique, where the energy measurement is based on instrumenting the *charge transfer*. First, using a separate power source, we charge a capacitor, which is then used to supply power to the target processor for one clock cycle. As the processor consumes energy, the capacitor is discharged by the amount of energy it has supplied to the processor. Therefore, by measuring the voltage level observed at the output of the capacitor before and after it is discharged, we can calculate the energy consumption as the difference between the energy initially charged in the capacitor and the energy remaining after it has been discharged. Using Equation 1, we can calculate the energy difference by

$$\Delta E = \frac{1}{2}CV_+^2 - \frac{1}{2}CV_-^2 \, , \tag{2}$$

where C denotes the capacitance used, whereas V_+ and V_- denote the voltage level before and after the capacitor is discharged, respectively.

Figure 2 illustrates the mechanism used to implement the measurement technique explained above. Note that we have two separate capacitors controlled by two pairs of alternating switches, which are synchronized to the same clock signal that is used by the target processor. This is required because, in a clock cycle when one capacitor is being charged, the other capacitor must supply power to the target processor. In the next clock cycle, the capacitor discharged in the previous cycle is recharged, while the other is supplying power to the processor and thus discharged. For example, assume that the switch pair $\{S_1, S_4\}$ is closed and the other pair $\{S_2, S_3\}$ is open at the current clock cycle. Then the capacitor C_1 is charged with V_s, while C_2 is connected to the voltage input of the target processor to supply power, being discharged. In the next clock cycle, the switch pair $\{S_2, S_3\}$ becomes closed and $\{S_1, S_4\}$ becomes open, charging C_2 and discharging

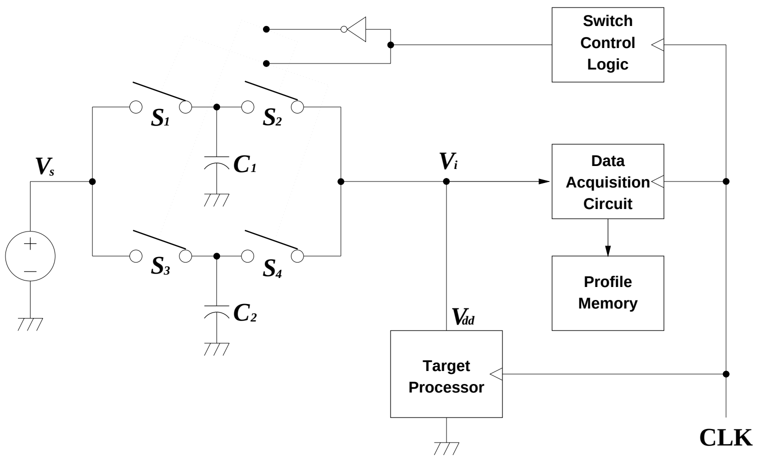


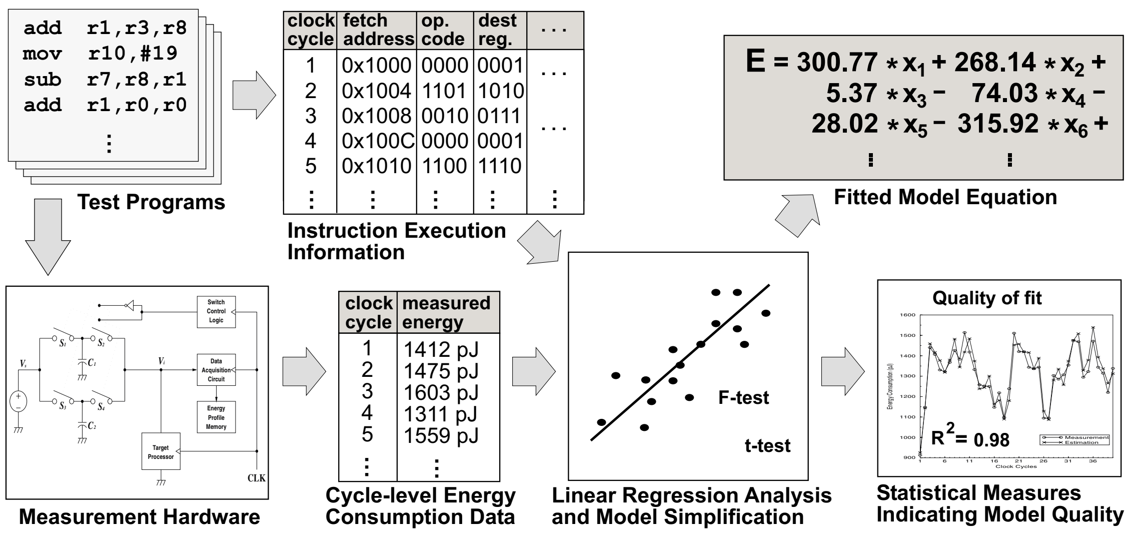
Figure 2: Functional block diagram for the measurement hardware.

C_1. Like this, the two capacitors are repeatedly charged and discharged, as controlled by the switch pairs, alternately providing a power source for the target processor. To measure the voltage output of the capacitors, the data acquisition circuit is equipped with a high-precision AC/DC converter, which records V_i at every rising edge and falling edge of the clock signal. This information is stored in the profile memory and later used to calculate the cycle-level energy consumption information, based on Equation 2.

By using this measurement hardware synchronized to the same clock signal that is used by the target processor, we have the following advantages. First, we can provide the cycle-level accurate energy consumption information without necessitating a high sampling rate, which is commonly required by traditional approaches depending on instantaneous power measurement. That is, we need to take only two samples per clock cycle, since energy is consumed by signal transitions caused by the clock ticks [5]. Second, because the measurement data is synchronous to the processor clock cycle, we can easily correlate the energy consumption data to the instruction execution information, which is crucial in modeling energy behavior of microprocessors at the software level. The detailed design and implementation of the measurement hardware is described in [5, 6], along with applications of the hardware for instruction-level energy characterization. Moreover, in [5], the accuracy of the measurement system is verified against the average power measured by a digital multimeter.

4 Statistical Derivation of an Energy Model

The energy consumption of a microprocessor is dependent on the internal implementation of the processor. This implies that one possible approach for estimating the energy consumption is to describe the details of hardware implementation of the target processor, which is essentially equivalent to simulating the processor at the gate level or at the circuit level. However, this simulation-based energy estimation is not practical for the purpose of program optimization for a number of reasons. For example, evaluating the impact of even a slight change in the instruction sequence would require re-running the whole simulation procedure, which has substantial complexity. This high complexity of the models relying on the implementation data makes the simulation-based methods not applicable to program optimization, where frequent cost evaluation of optimization candidates is required. Moreover, gate-level or circuit-level implementation data for a commercial off-the-shelf microprocessor is not generally available, which makes the simulation-based


```
add  r1,r3,r8
mov  r10,#19
sub  r7,r8,r1
add  r1,r0,r0
       ⋮
```

clock cycle	fetch address	op. code	dest reg.	...
1	0x1000	0000	0001	...
2	0x1004	1101	1010	
3	0x1008	0010	0111	...
4	0x100C	0000	0001	
5	0x1010	1100	1110	
⋮	⋮	⋮	⋮	⋮

$$E = 300.77 * x_1 + 268.14 * x_2 + 5.37 * x_3 - 74.03 * x_4 - 28.02 * x_5 - 315.92 * x_6 + \vdots$$

clock cycle	measured energy
1	1412 pJ
2	1475 pJ
3	1603 pJ
4	1311 pJ
5	1559 pJ
⋮	⋮

Figure 3: The overall approach for deriving an energy consumption model.

approaches more difficult to apply.

Therefore, we propose a novel approach for deriving an energy consumption model that enables accurate estimation of the energy consumption of software by investigating high-level information related to instruction execution. Figure 3 summarizes our approach for deriving the energy consumption model. Instead of relying on the implementation information of the target processor, we use an empirical method, where the model construction is based on the actual energy consumption data measured from real hardware. That is, we regard the circuit-level implementation of the target processor as a black box, whose internal structure is unknown, and assume that the only accessible information is the response from this black box for a set of stimuli. In other words, we observe the energy consumption (responses) of the target processor when it executes a number of test programs (stimuli). Then we try to derive an energy consumption model by reasoning about the relationship between the instructions executed and the corresponding energy consumption.

This empirical method has the following advantages over describing the detailed hardware implementation. First, the resulting model is self-validating. That is, the derived energy model accurately reflects the actual behavior of the target processor, since the model is based on real measurement. Second, the empirical model remarkably simplifies the process of software energy estimation, by providing the abstract form of an equation instead of a complex implementation model for calculating energy consumption. Finally, the technique is generally applicable to a variety of different processors because it is not dependent on a specific implementation. Of course, the whole measurement procedure should be repeated when the energy consumption model is to be constructed for a different processor, which means we need measurement equipment such as the one described in 3.2. Note, however, that our methodology is not confined to measurement-based construction of an energy model. That is, the proposed technique can also be used to derive an energy consumption model from the information provided by a cycle-level energy simulator, when such a simulator is already available. In this case, the proposed technique can abstract the energy behavior of the target processor, and thus correlate the energy consumption data to the instruction execution information, which is not possible by the simulator alone.

To derive the energy consumption model by summarizing the data gathered from measurement (or simulation), we use a statistical modeling technique called regression analysis [7]. First, we assume a hypothetical model equation, which is expressed in terms of the factors that are assumed to influence the energy consumption of software. This model equation is a linear combination of the model variables defined in terms of various aspects of instruction execution, with a number of unknown parameters yet to be determined. Then we extract the values of these model variables from the test programs by examining its instruction sequence. Using the model variable values combined with the corresponding cycle-level energy information given by measurement, the regression analysis determines the values of the unknown parameters of the model equation. This procedure is called model fitting [7], whose result is a fitted model equation that explains the relationship between the model variables and the energy behavior of the target processor. Using this fitted model equation, we can accurately estimate the energy consumed by a given instruction sequence executed on the target processor. In addition, the regression analysis will produce a set of statistical measures, which can be used in refining the model as well as in assessing the quality of the fitted model equation.

In Section 4.1, we show how the hypothetical model equation is constructed, by introducing a set of energy formulas. Then in Section 4.2, we explain how we can derive the parameters of the model equation using linear regression analysis.

4.1 Energy Model Equation

In general, the energy consumed in a CMOS processor is dependent on the number of state transitions as well as the current charge/discharge states of the internal signals, as previously explained in Section 3.1. Specifically, the energy consumed in a clock cycle is proportional to (1) the number of bit flips in the internal signals in that clock cycle, and (2) the number of logical 1's (or the number of logical 0's, alternatively) in the signals. Since the internal signals are intrinsically controlled by the execution of instructions, we conjecture that the energy consumption can be modeled by properties derived from the instructions executed. This motivates us to define the energy model equation in terms of the currently executing instruction and the previously executed instruction.

For a pipelined processor, in general, the energy consumed in a clock cycle is determined by the instructions that currently occupy the pipeline stages and those that previously occupied them. Assuming that the energy consumed in a clock cycle is the sum of energy consumed at all the pipeline stages in that cycle, we calculate the energy consumption in a clock cycle by a simple equation. Let S be the set of all the pipeline stages and $I_s(i)$ the instruction occupying stage s at clock cycle i. Then the energy consumed at cycle i can be calculated by

$$E_i = \sum_{s \in S} e_s(I_s(i), I_s(i-1)) \ , \tag{3}$$

where $e_s(X, Y)$ denotes the energy consumed in pipeline stage s when instruction X is executed in that stage, preceded by instruction Y.

For example, consider the pipelined execution scenario shown in Figure 4, where we have three pipeline stages: IF (instruction fetch), ID (instruction decode), and EX (execute). Assume that an instruction sequence of A, B, C, D, E, and F is executed sequentially. The energy consumption at clock cycle 4, for instance, is calculated by

$$E_4 = e_{\mathrm{IF}}(\mathsf{D}, \mathsf{C}) + e_{\mathrm{ID}}(\mathsf{C}, \mathsf{B}) + e_{\mathrm{EX}}(\mathsf{B}, \mathsf{A}) \ . \tag{4}$$

We model the energy consumed at pipeline stage s by instruction X executed after instruction Y, denoted by $e_s(X, Y)$, in terms of the resources controlled by the execution of instructions X and Y. Specifically, we define a set of resources that possibly have influence on the energy consumption of instruction execution and construct the model equation as a combination of these resources. Note that we can capture different aspects of instruction execution by defining different sets of resources. For example, we can model the

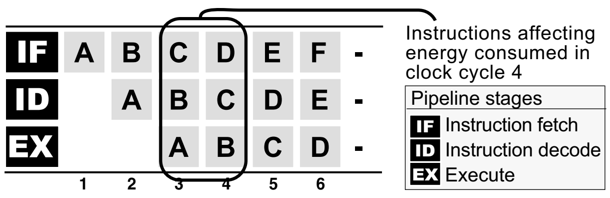


Figure 4: Example of pipelined execution scenario.

instruction properties visible in the binary representation of instructions by defining resources such as the opcode, register numbers, etc. On the other hand, we can describe architecture-level behavior by defining the functional units as the resources. In other words, the model equation is general enough to handle various levels of abstraction in energy consumption modeling, by incorporating the concept of resources.

Assume that we have identified all the resources that have significance in energy consumption. If we let R be the set of all the resources, $e_s(X,Y)$ is given by

$$e_s(X,Y) = B_s^X + \sum_{r \in R} f_s^{r/X}(\beta_r(X), \beta_r(Y)) \ , \tag{5}$$

where $\beta_r(X)$ and $\beta_r(Y)$ denote the binary representation of resource r defined by instructions X and Y, respectively. The term B_s^X gives the base cost for instruction X at pipeline stage s, which corresponds to the portion of energy consumed by instruction X at stage s, regardless of the current and the previous states of the resources. On the other hand, the function $f_s^{r/X}(\beta_r(X), \beta_r(Y))$ gives the variation of energy consumption according to the resource r controlled by the execution of instruction X at stage s, preceded by instruction Y.

Based on the observation that the energy consumption is proportional to the number of bit flips and the number of logical 1's in the internal signals, we define the energy variation function as

$$f_s^{r/X}(\beta_r(X), \beta_r(Y)) = H_s^{r/X} \cdot h(\beta_r(X), \beta_r(Y)) + W_s^{r/X} \cdot w(\beta_r(X)) \ , \tag{6}$$

where $h(i,j)$ denotes the Hamming distance between two binary numbers i and j, while $w(i)$ denotes the weight (number of 1's) of a binary number i. In Equation 6, $H_s^{r/X}$ and $W_s^{r/X}$ are unknown coefficients that characterize the energy consumption of instruction X at pipeline stage s, with regard to resource r.

Combining Equations 3, 5, and 6, we have an equation system connecting the cycle-level energy consumption with the resources defined by instruction execution. Note that this model equation is a linear combination of model variables defined in terms of various resources, with a number of unknown parameters, i.e., $H_s^{r/X}$'s, $W_s^{r/X}$'s, and B_s^X's. We call these parameters *characterizing parameters*, because they describe the characteristics of the energy behavior of the target processor. When we gather the energy consumption data for a large number of cycles and extract the values of model variables, we have a number of simultaneous equations with the characterizing parameters as unknowns. One might be tempted to solve these equations to obtain the exact solution to the equations. However, due to the erroneous nature of the measurement and possible incompleteness of the model equation (e.g., we might not have completely listed all the resources that actually affect the energy consumption), it does not make much sense to exactly solve the equations. Instead, we use a statistical analysis technique called *regression analysis* to derive the best estimates of these characterizing parameters, as will be explained in Section 4.2.

Note that the number of such characterizing parameters can be very large, depending on the number of pipeline stages, the number of different instructions, and the cardinality of the set of resources defined.

Moreover, the set of characterizing parameters possibly contains those parameters that are insignificant or even irrelevant to energy consumption, which possibly degrades the accuracy of the model. Therefore, to rectify this problem, the regression analysis uses two techniques based on statistical model testing, in addition to deriving the values of the characterizing parameters. First, we identify the parameters that have little or no significance and eliminate them, to reduce the number of characterizing parameters and enhance the model accuracy at the same time. Second, we merge certain characterizing parameters for all the instructions or for a selected group of instructions, when the behavior of different instructions in the same group is similar with respect to one or more resource model variables. These techniques, together with the process of deriving the values of the characterizing parameters, are presented below.

4.2 Regression Analysis for Energy Model

Regression analysis is a statistical method for investigating functional relationships among variables [7]. The relationship is expressed in the form of an equation or a model connecting a *response variable* with one or more *predictor variables*. That is, when we denote the response variable by y and the set of predictor variables by $x_1, x_2, \ldots, x_p$, the true relationship between y and $x_1, x_2, \ldots, x_p$ can be approximated by a regression model

$$y = f(x_1, x_2, \ldots, x_p) + \varepsilon \ , \tag{7}$$

where ε is assumed to be an error representing the discrepancy in the approximation.

When the relationship between the response and the set of predictors is assumed to be linear, the method for constructing the regression model is called *linear regression analysis*. That is, a linear regression model is expressed by an equation

$$y = \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_p x_p + \varepsilon \ , \tag{8}$$

where α_0, α_1, ..., α_p are constants and are called *regression coefficients*. The best estimates of these regression coefficients, i.e., the ones that lead to the model equation that best explains the relationship between the response and the predictors, are determined by investigating a number of sample combinations of the response and the predictors. The most common method used in finding the regression coefficients is called the *least square method* [7], which is the one that we use in our analysis. The least square method tries to approximate the model equation to the true relationship between the response and the set of predictors by minimizing the sum of the deviations squared (least square error) from a given set of data.

In our case of deriving the energy model, the response variable corresponds to the energy consumption measured in each clock cycle, while the predictor variables are:

1. the Hamming distances between the binary representations of resources controlled by execution of two adjacent instructions, i.e., $h(\beta_r(X), \beta_r(Y))$'s,

2. the weights of the binary representations of resources controlled by execution of an instruction, i.e., $w(\beta_r(X))$'s, and

3. indicator (or dummy) variables[1] introduced to express the base costs of instructions, for each of the pipeline stages.

Note that, with the above settings, the structure of our energy model equation presented in Section 4.1 can be exactly mapped to a linear regression model, with the characterizing parameters being the regression

[1]An indicator or dummy variable is a predictor variable that can only take a boolean (0 or 1) value [7].

coefficients. That is, finding the best estimates of the regression coefficients corresponds to determining the values of the characterizing parameters in the energy model equation, and thus approximating our model to the actual energy behavior observed by the measurement hardware.

Compared to many other applications of statistical analysis, where the sample data corresponds to observation from uncontrollable natural environment, our case lends us the benefit of being able to freely set the values of the predictor variables. For example, when we define the model variables in terms of instruction-level resources such as instruction fetch address and register numbers, we can control the values of the predictor variables by using carefully written test programs. This means that we can determine a subset of characterizing parameters independently of the others, and use the values of these parameters later in the analysis for the remaining parameters yet to be determined. For example, we can change the instruction fetch address while fixing all the other factors such as register numbers and data values, by repeatedly executing the same instructions. By doing this, we can derive the impact of the instruction fetch address, separately from those of the other resources. Then we can determine the impact of register numbers, for example, by executing the same instructions with different register numbers but with all the other factors fixed. This stepwise derivation of characterizing parameters not only enhances the accuracy of the model by reducing the possible errors incurred in regression, but also keeps the complexity of the regression analysis low by decomposing the whole problem into several subproblems.

In addition to deriving the values of the characterizing parameters, we identify and remove the predictor variables that have little or no significance in the energy consumption, based on statistical inference. This is done by testing a null hypothesis $H_0 : \alpha_i = 0$ against the alternative $H_1 : \alpha_i \neq 0$ for each regression coefficient α_i. The best model is selected using a model testing technique called the t-test [7]. Intuitively, to assess the significance of each predictor variable, we compare the model with a specific characterizing parameter set to zero with another model with the parameter set to the value derived from the model fitting procedure. Removing the predictors that are insignificant or even irrelevant to the energy consumption not only increases the accuracy of the resulting model equation, but also maintains the model complexity at a reasonable level.

Also, we can reduce the complexity of the model equation further by merging specific characterizing parameters for all the instructions, or for a selected group of instructions. This is done by testing a null hypothesis $H_0 : \alpha_i = \alpha_j = \cdots = \alpha_k$ against the alternative $H_1 : \alpha_i \neq \alpha_j \neq \cdots \neq \alpha_k$ for a combination of selected regression coefficients α_i, α_j, ..., α_k. The model that best explains our measured energy value is selected based on a model testing technique called the F-test [7]. Intuitively, to check if a group of instructions should be merged or not with respect to a specific type of resource, we compare a model with the same characterizing parameters for a certain subset of instructions and another model with different characterizing parameters for those instructions. For example, we intuitively assume that the impact of the instruction fetch address and that of register numbers are identical for all the instructions, while the impact of data values is different from one instruction to another. In Section 5.1, this assumption is validated for our target processor, using the techniques explained above.

Besides the fitted model equation, the regression analysis also produces a set of statistical measures that can be used to assess the soundness of the model. The most common measure of the quality of fit, i.e., the accuracy of the resulting model equation, is the *coefficient of determination*, denoted by R^2 [7]. Intuitively, it is interpreted as the proportion of the total variability in the response variable that is accounted for by the set of predictor variables in the model equation. In the rest of this paper, we will use this R^2 value for evaluating the quality of our energy model.

5 Experimental Results

To demonstrate the validity of our approach for deriving an energy consumption model by abstracting the behavior of the hardware, we performed a set of experiments. In Section 5.1, we apply the proposed technique to derive an energy model equation for our target processor ARM7TDMI [1]. Then, in Section 5.2, we estimate the energy consumption of a randomly-generated instruction sequence using the model equation derived from our analysis.

5.1 Case Study: ARM7TDMI

We applied our proposed technique for energy model derivation to the ARM7TDMI processor core. The target processor has a three-stage pipeline similar to the one shown in Figure 4. We derived an energy consumption model for the ARM data-processing instructions that have one of the following two instruction formats [9].

```
op   Rd, Rn, Rm
op   Rd, Rn, #imm
```

Here, `op` specifies the opcode for the instruction, while `Rd`, `Rn`, and `Rm` specify the destination operand register, the first source operand register, and the second source operand register, respectively. Instead of the second source operand register, an immediate value can be given as the second source operand with the `#imm` field specified.

We defined a set of model variables in terms of instruction-level resources. They are:

- *instruction fetch address*, which is assumed to affect the energy consumption of the internal address bus when the instruction is fetched,
- *instruction bit encoding*, which is assumed to affect the energy consumption of the instruction register and the pipeline latches,
- *operand specifiers*, such as register numbers and the immediates, which are assumed to affect the energy consumption of instruction decoding and execution, and
- *data values*, which are assumed to affect the energy consumption of arithmetic and logical execution units.

That is, the model variables are chosen in such a way that the model equation essentially captures the influence of the binary representation of instructions on the energy behavior of various functional units inside the processor, without describing the internal implementation.

As previously mentioned in Section 4.2, we decompose the problem of energy model derivation into several subproblems and derive the characterizing parameters in a number of steps, for the sake of model accuracy and analysis simplicity. Figure 5 illustrates this problem decomposition, showing sample instruction sequences similar to the ones actually used in our derivation of the characterizing parameters. In the first set of test programs, we executed the same instructions at different program locations with the same operand specifiers and the same data values, to derive the parameters for the instruction fetch address (denoted by FA). That is, we give variation to the fetch address while fixing all the other variables, so that the only variability of the energy consumption comes from the changes in the fetch address. The regression analysis derived $H_{\mathrm{IF}}^{\mathrm{FA}/*} = 2.72$ (pJ) and $W_{\mathrm{IF}}^{\mathrm{FA}/*} = -3.44$ (pJ), which could be merged for all the instructions under

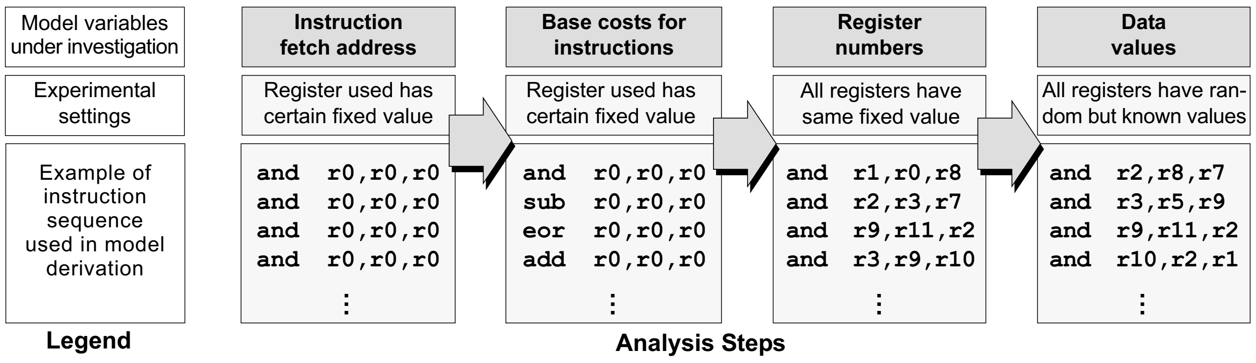

Model variables under investigation
Experimental settings
Example of instruction sequence used in model derivation
Legend
Instruction fetch address
Register used has certain fixed value
and r0,r0,r0
and r0,r0,r0
and r0,r0,r0
and r0,r0,r0
Base costs for instructions
Register used has certain fixed value
and r0,r0,r0
sub r0,r0,r0
eor r0,r0,r0
add r0,r0,r0
Register numbers
All registers have same fixed value
and r1,r0,r8
and r2,r3,r7
and r9,r11,r2
and r3,r9,r10
Data values
All registers have random but known values
and r2,r8,r7
and r3,r5,r9
and r9,r11,r2
and r10,r2,r1
Analysis Steps

Figure 5: Stepwise derivation of characterizing parameters.

Table 1: Base Costs of Different Instructions

Instruction	Pipeline Stage		
	IF	ID	EX
`add`	300.77	317.03	328.00
`sub`	298.14	200.34	189.36
`and`	300.10	272.60	330.83
`eor`	298.53	419.70	332.07
$H_s^{\mathrm{op}/*}$	3.31	11.71	0.30

(units: pJ, $R^2 = 0.97$)

investigation. In this case, the coefficient of determination was $R^2 = 0.98$, which means that the resource FA was capable of accounting for approximately 98 % of the energy variation due the changes in the fetch address. The results indicate that the energy consumption is proportional to the number of bit switches in the fetch address, and negatively proportional to the number of 1's in the fetch address. With this information available, we can apply program optimization towards low energy such as code replacement, where frequently executed instructions are placed in program addresses with small Hamming distances and a large number of 1's.

To derive the base costs of various instructions at each of the three pipeline stages, we executed another set of test programs. The test program consists of an instruction mix of different instructions, with the same operand specifiers and the same data values for all the instructions. Note that we cannot execute the instruction mix without varying the Hamming distance and the weight of the opcode encoding. Therefore, we derive the characterizing parameters for resource opcode (denoted by op) at the same time. On the other hand, to eliminate the effect of inevitable changes in the instruction fetch address as we execute the test program, we calculated the energy variation caused by changes in the fetch address using the parameter values derived in the previous analysis step, and subtracted it from the measured energy values. Table 1 presents the results from our regression analysis for four sample instructions[2]. The results indicate that the energy consumed in the IF stage is not noticeably different from one instruction to another, as shown in the second column of the table. However, the energy consumed in the stages ID and EX vary substantially depending on the instructions executed. For example, the `sub` instruction appears to consume much less

[2]Due to the space limitation, we only present the results for four instructions, although we derived the parameters for all the data-processing instructions in the ARM instruction set [9].

Table 2: Parameters for Register Numbers

Resource	Variable Type	Pipeline Stage IF	ID	EX
Rd	H	0.86	7.64	0.84
	W	0.64	2.77	0.50
Rn	H	1.34	3.92	1.02
	W	1.55	3.40	1.14
Rm	H	0.87	6.51	1.44
	W	2.20	3.47	−0.47

(units: pJ, $R^2 = 0.92$)

energy in the ID and EX stages than the other instructions, while the `eor` instruction consumes much more in the ID stage. The results also signifies that the energy consumption is proportional to the Hamming distance of the opcode, as shown in the last row of Table 1, while the weight of the opcode turned out to have little or no impact on the energy consumption.

This large difference of the base costs of different instructions and the tendency of increase in the energy consumption with the increase in the Hamming distance of the opcode indicate that the energy consumption of software can be reduced by applying instruction selection and/or instruction scheduling techniques tailored for low energy [22, 25]. Specifically, the energy consumption can be significantly reduced by judiciously selecting the instructions with low base costs when more than one choices are possible for the same execution semantics. Moreover, careful scheduling of instructions can save a substantial portion of energy consumed in the processor core, by reducing the number of bit switches in the opcodes of adjacent instructions. For our target processor, for instance, reducing one opcode bit flip on average will save approximately 15.32 pJ of energy per instruction (the sum of savings in the three pipeline stages), which amounts to approximately 1.2 % of the total energy.

To derive the characterizing parameters for register numbers, we executed still another set of test programs where the same instructions are executed with changes in the register numbers (i.e., `Rd`, `Rn`, and `Rm`). Again, to eliminate the influence of instruction fetch addresses, we calculated the variation due to the changes in the fetch address and subtracted it from the measurement data. Table 2 summarizes the results from our regression analysis. We successfully derived the parameter values for the Hamming distance and the weight of each register number in each of the pipeline stages, which could be merged for all the instructions under consideration. The results indicate that the major portion of the energy variation caused by register numbers is in the ID stage, as can be induced from the large coefficient values in the fourth column of the table. Especially, the most significant factors are the Hamming distances of `Rd` and `Rm`, which have the largest coefficient values. This information can be used in program optimization techniques such as register assignment and/or register relabeling for low energy [2, 4, 13]. Specifically, if we reduce one bit flip in each of the three register numbers on average, the resulting energy savings will be approximately 24.43 pJ per instruction, which corresponds to 1.9 % of the total energy.

Similarly, we executed a set of test programs with random data values to derive the coefficients for data values used in the computation. Table 3 shows the results from our regression analysis for four sample instructions[3], where `src1` and `src2` denote the first and the second source operands, respectively, while `dest` denotes the destination operand. The results show that the resources defined in terms of data values

[3]Again, we list the results for only four sample instructions, due to the space limitation.

Table 3: Parameters for Data Values

Instruction	Resource	Variable Type	Pipeline Stage ID	Pipeline Stage EX
`add`	`src1`	W	3.93	6.61
	`src2`	W	0.18	7.32
	`dest`	W	–	3.63
`sub`	`src1`	W	8.81	5.95
	`src2`	W	9.58	8.36
	`dest`	W	–	4.88
`and`	`src1`	W	8.68	8.66
	`src2`	W	0.41	9.39
	`dest`	W	–	-2.30
`eor`	`src1`	W	2.75	7.09
	`src2`	W	-1.30	7.58
	`dest`	W	–	2.82

(units: pJ, $R^2 = 0.97$)

have significance only in the weights, not in the Hamming distances. This phenomenon is presumably due to the *precharge-and-evaluation* scheme used in the dynamic CMOS implementation of the target processor datapath, as previously explained in Section 3.1. Note that the weight of the destination operand in the ID stage is not included in the table because the computation result will not be generated until the EX stage. Also note that, unlike the results from the previous analysis steps, different instructions have different parameter values for the model variables defined in terms of data values. This is because different instructions behave differently with respect to data values, according to the operations specified by the instructions. We conjecture that this difference in energy behavior is due to the different ways of utilizing the arithmetic and logical functional units by different instructions.

Likewise, we performed regression analysis on the impact of the immediate operand, which turned out to have only significance in the Hamming distances in the IF and ID stages. The coefficient values derived are $H_{\mathrm{IF}}^{\mathrm{imm}/*} = 4.06$ (pJ), and $H_{\mathrm{ID}}^{\mathrm{imm}/*} = 8.98$ (pJ), respectively, with the R^2 value of 0.98.

5.2 Software Energy Estimation

To show the accuracy of our energy consumption model in estimating the energy consumed by an instruction sequence, we implemented an energy consumption analyzer. The analyzer takes an assembly source program (or a binary program) as its input, and estimates the energy consumption of the given instruction sequence at each clock cycle. The analyzer first extracts the values of the model variables by examining the instruction sequence.[4] Then it computes the energy consumption at each clock cycle using the model equation and the values of the characterizing parameters previously determined by the regression analysis, as described in Section 5.1.

For the experiments, a sample program was generated that contains a random mixture of ARM data-processing instructions, with the operand specifiers (i.e., the register numbers and the immediates) are chosen at random. Also, the data values are randomly generated and prepared by the sample program prior to

[4]In deriving the values of the model variables defined in terms of data values, we assumed that the initial values in the registers are known a priori.

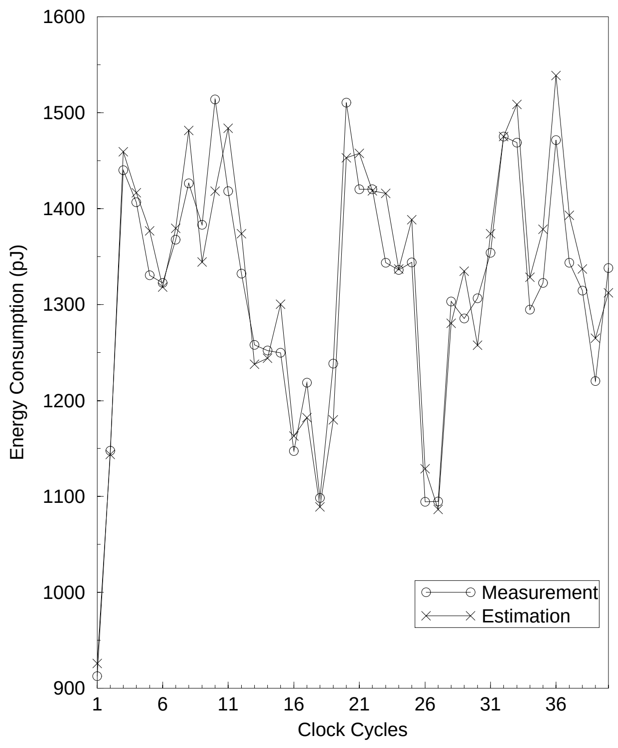


Figure 6: Comparison between estimated and measured energy.

the execution of the instruction sequence under investigation. Figure 6 compares the estimated energy with the actual energy consumption measured on the hardware, for the same sample instruction sequence that executes for 40 clock cycles.

The results indicate that the proposed approach enables an accurate energy estimation, which is verified by the fact that the graph for the estimated energy closely resembles that of the measured energy. The coefficient of determination (R^2) value in this case was 0.99, which indicates that 99 % of the variation in the measured energy value was captured by our energy model equation. The total of the measured energy consumption for the 40 clock cycles was 5.2532 nJ, while that of the estimated energy was 5.2993 nJ. The error in this total energy was less than 1 %. We also calculated the error ratio in each cycle given by

$$r_e = \frac{|\text{estimation} - \text{measurement}|}{\text{measurement}} , \tag{9}$$

whose average value was 0.0251, which means that the error of the energy estimation by our model equation was on average 2.5 %. Besides, the maximum of r_e was 0.0633, which means that the error was at most 6.3 %. The error comes from a number of sources, including the following.

- The energy consumption may not be perfectly linear to the Hamming distances and the weights of the binary representation of the resources defined.
- The choice of the model variables might have missed one or more factors that have significant impacts on energy consumption.
- The quantization error inherent in the measurement hardware used in the model construction may cause inaccuracies in deriving the values of the characterizing parameters.

6 Conclusions and Future Work

We have proposed a technique for deriving an accurate energy consumption model for an embedded microprocessor by abstracting the physical energy behavior of the processor hardware at a higher level. The proposed approach combines an empirical method and a statistical analysis technique called regression analysis. In the empirical method, we base the model construction upon the actual measurement data from hardware, without describing the implementation of the target processor. For this purpose, we designed and implemented a cycle-accurate high-precision measurement hardware that provides accurate energy consumption data at the clock-cycle level. In the statistical analysis, we use linear regression modeling to derive a model equation that explains the complex interactions among different sources of energy variation. The technique described in this paper enables a simple but yet accurate estimation of energy consumption of software, based on a linear model equation. Moreover, we can identify the contributing factors that affect the software energy consumption, and further detect the significance of each factor. Therefore, the availability of such an energy consumption model will provide an important basis for various energy reduction techniques for embedded software, such as program optimization techniques for low energy. Another advantage of the proposed approach is that the model derivation technique can be applied to a number of different processors, since it is not dependent on a specific processor implementation.

We applied our proposed method to derive an energy model for the ARM7TDMI processor core, to prove the validity of the approach. This case study shows that an accurate energy model can be derived from instruction-level characterization of software, using the technique proposed in this paper. The results from our experiments indicate that our energy model can accurately estimate the energy consumption of a given instruction sequence. Statistical measures show that approximately 99 % of the energy variation can be explained by the derived model equation for the experimented case. The maximum error ratio of the estimation was 6.3 %, while the average was approximately 2.5 %.

We are currently investigating the tradeoff between the estimation accuracy and the model accountability by modeling resources at different levels of abstraction. For example, we can build a model based on the architecture-level resources, where the model variables correspond to the binary states of internal functional units such as ALUs, pipeline latches, address registers, etc. In this case, we modify a functional simulator for the target processor to extract the values of the model variables, by calculating the binary states of the functional units at each clock cycle. The advantage of such architecture-level modeling is that the model equation can be closer to the actual energy behavior of the target processor, since it incorporates more detailed information about the internal structure of the processor. In addition to this enhanced accuracy, architecture-level resource modeling will enable capturing of more complex situations such as pipeline stalls, execution of multi-cycle operations, and memory loads/stores, which cannot be expressed by modeling instruction-level resources alone. However, using such architecture-level resource models, we surrender to some extent the ability to relate the energy consumption to the properties that can be exploited by high-level optimization techniques such as program transformation for low energy. Therefore, we aim to pinpoint the level of abstraction appropriate for providing meaningful information to program optimizer and enabling accurate energy estimation of instruction sequences at the same time. That is, we hope to determine where to put the boundary between instruction-level modeling and architecture simulation for deriving an energy model, by covering a broad range of analysis levels in a single framework for energy model derivation.

References

[1] Advanced RISC Machines Ltd. *ARM7TDMI Data Sheet*, August 1995.

[2] R. Anand, M. Jacome, and G. De Veciana. Heuristic tradeoffs between latency and energy consumption in register assignment. In *Proceedings of the International Conference on Hardware Software Codesign*, pages 115–119, May 2000.

[3] C. Brandolese, W. Fornaciari, F. Salice, and D. Sciuto. An instruction-level functionality-based energy estimation model for 32-bits microprocessors. In *Proceedings of the Annual ACM IEEE Design Automation Conference*, pages 346–351, June 2000.

[4] J.-M. Chang and M. Pedram. Register allocation and binding for low power. In *Proceedings of the Annual ACM IEEE Design Automation Conference*, pages 29–35, June 1995.

[5] N. Chang and K. Kim. Real-time per-cycle energy consumption measurement of digital systems. *IEE Electronic Letters*, 36(13):1169–1170, June 2000.

[6] N. Chang, K.-H. Kim, and H. G. Lee. Cycle-accurate energy consumption measurement and analysis: Case study of ARM7TDMI. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 185–190, July 2000.

[7] S. Chatterjee, A. S. Hadi, and B. Price. *Regression Analysis by Example, Third Edition*. John Wiley & Sons, Inc., 2000. ISBN 0-471-31946-5.

[8] R. Y. Chen, M. J. Irwin, and R. S. Bajwa. An architectural level power estimator. In *Proceedings of the Power-Driven Microarchitecture Workshop*, 1998.

[9] S. Furber. *ARM System Architecture*. Addison-Wesley, 1996. ISBN 0-201-40352-8.

[10] C. Gebotys, R. Gebotys, and S. Wiratunga. Power minimization derived from architectural-usage of VLIW processors. In *Proceedings of the Annual ACM IEEE Design Automation Conference*, pages 308–311, June 2000.

[11] C. H. Gebotys and R. J. Gebotys. An empirical comparison of algorithmic, instruction, and architectural power prediction models for high performance embedded DSP processors. In *Proceedings of the International Symposium on Low Power Electronics and Design*, 1998.

[12] M. Kandemir, N. Vijaykrishnan, M. J. Irwin, and W. Ye. Influence of compiler optimizations on system power. In *Proceedings of the Annual ACM IEEE Design Automation Conference*, pages 304–307, June 2000.

[13] M. Kandemir, N. Vijaykrishnan, M. J. Irwin, W. Ye, and I. Demirkiran. Register relabeling: A post-compilation technique for energy reduction. In *Proceedings of the Workshop on Compilers and Operating Systems for Low Power*, October 2000.

[14] B. Klass, D. E. Thomas, H. Schmit, and D. F. Nagle. Modeling inter-instruction energy effects in a digital signal processor. In *Proceedings of the Power-Driven Microarchitecture Workshop*, June 1998.

[15] M. T.-C. Lee, V. Tiwari, S. Malik, and M. Fujita. Power analysis and minimization techniques for embedded DSP software. *IEEE Transactions on VLSI Systems*, pages 1–14, March 1997.

[16] S. Lee, A. Ermedahl, S. L. Min, and N. Chang. An accurate instruction-level energy consumption model for embedded RISC processors. In *Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems*, pages 1–10, June 2001.

[17] H. Mehta, R. M. Owens, and M. J. Irwin. Instruction level power profiling. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, 1996.

[18] H. Mehta, R. M. Owens, M. J. Irwin, R. Chen, and D. Ghosh. Techniques for low energy software. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 72–75, August 1997.

[19] J. T. Russell and M. F. Jacome. Software power estimation and optimization for high performance, 32-bit embedded processors. In *Proceedings of the International Conference on Computer Design*, October 1998.

[20] M. Sami, D. Sciuto, C. Silvano, and V. Zaccaria. Instruction-level power estimation for embedded VLIW cores. In *Proceedings of the International Conference on Hardware Software Codesign*, pages 34–38, May 2000.

[21] S. Segars. ARM7TDMI power consumption. *IEEE Micro*, 17(4):12–19, July/August 1997.

[22] C.-L. Su, C.-Y. Tsui, and A. M. Despain. Low power architecture design and compilation techniques for high-performance processors. In *Proceedings of the IEEE COMPCON*, pages 489–498, 1994.

[23] V. Tiwari, S. Malik, and A. Wolfe. Power analysis of embedded software: A first step towards software power minimization. *IEEE Transactions on VLSI Systems*, 2(4):437–445, December 1994.

[24] V. Tiwari, S. Malik, A. Wolfe, and M. T.-C. Lee. Instruction level power analysis and optimization of software. *Journal of VLSI Signal Processing*, 13(2/3):223–238, August 1996.

[25] M. C. Toburen and T. M. Conte. Instruction scheduling for low power dissipation in high performance microprocessors. In *Proceedings of the Power-Driven Microarchitecture Workshop*, June 1998.

[26] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. The design and use of SimplePower: A cycle-accurate energy estimation tool. In *Proceedings of the Annual ACM IEEE Design Automation Conference*, pages 340–345, June 2000.