Embedded Systems Energy Characterization Using Non-intrusive Measurements

Abstract

This paper presents a non intrusive methodology for building embedded systems energy consumption models. The method is based on measurements on real hardware in order to get a quantitative approach that takes into account the full architecture. Based on these measurements, data are grouped into class of instructions and events. These classes can then be reused in software simulators and in cost functions for high-level source code transformation in optimizing compilers. The computed power model is much simpler than previous ones while being accurate at the platform level. The methodology is illustrated using experimental results obtained on an ARM Integrator platform for which an accurate and complete system energy model is build.

1 Introduction

With present day technology, it is possible to build very small platforms with enormous processing power. However, physical laws dictate that high processing power is linked to

high energy consumption. Since these platforms are mostly used in hand held appliances, and since battery capacity does not increases at the same pace, their designers are faced with the problem of minimizing power requirements under performance constraints.

The first approach is the devising of low-energy technologies, but this is outside the scope of this paper. The second approach is to make the best possible use of the available energy e.g. by adjusting the processing power to the instantaneous needs of the application, or by shutting down unused parts of the device. These tasks can be delegated to the hardware; however it is well known that the hardware only source of knowledge is the past of the application; it is only software that can anticipate future needs. Energy can also be minimized as a side effect of performance optimization. For instance, replacing a conventional Fourier transform by an FFT greatly improves the energy budget; the same can be said of locality optimization, which aims at replacing costly main memory accesses by low-power cache accesses.

The ultimate judge in the matter of energy consumption is measurement of the finished product. However, designers, compilers and operating systems need handier methods for assessing the qualities of their designs and directing possible improvements. Hence the need for simple analytical models, which must be expressed in term of software visible events like instructions, cache hits and misses, peripheral activity and the like. There are several ways of constructing such models. One possibility is electrical simulation of the design; this method is too time-consuming for use on systems of realistic size. Another method is to interpolate/extrapolate from measurements on a prototype. This is the method we have applied in this work.

The paper is organized as follows. After reviewing state of the art techniques, we present in Sect. 3 our experimental setup and our measurement methodology. Sect. 4 presents our results for an ARM development platform, evaluates their accuracy and describes the model which represents them. Sect. 5 validates this model on a more significant piece of code, the thread switching service of a simple operating system. We then conclude and discuss future work.

2 Related Work

Previous works on energy consumption modeling can be characterized using two main criteria: their level of abstraction and their building method. For the first criterion, we can group the models in three main categories, which are, by increasing level of abstraction, transistor/gate level models, architectural level models and finally instruction level models. There are three methods for building consumption models. The first method is analytical construction, the second one is simulation based, and the third is based on physical measurements.

These two criteria of classification are not fully orthogonal, since all combination are not possible or pertinent. Indeed, analytically built models targets low level units due to the complexity of the building process. As far as simulation based model are concerned, they are rarely used in the highest level of granularity since they are highly time consuming at building time. Finally we do not find lower level model based on measurements since it is complicated to extract low level informations from measures which are representing the whole system consumption.

The basic model Before giving examples of models by architectural layer class, we present here the basic power model of VLSI (Very Large Scale Integration) circuits. This model is one of the lowest level power model, since it models the consumption of the most elementary gate, the inverter (2 transistors). It is generalized to all gates.

The consumption of a gate can by divided into two main parts: static and dynamic power. The first part, static power, does not depend on the gate input changes and hence gate activity. However, the second part, dynamic power, is correlated to these input signal changes and can be subdivided into short-cut power and output capacitance load power.

In present day technologies, static and short-cut power dissipations are negligible against the amount of power that the third represents. The power consumption of the inverter is then simplified to the output capacitance load described by equation (1). This formula express the fact that the energy stored in the output capacitance is shorted to ground when the input switches from 1 to 0.

$$P = \frac{1}{2} C V_{dd}^2 \tag{1}$$

where C is the output capacitance of the gate. This model is generalized to full blocks or chips by the extension given in equation (2).

$$P = \frac{1}{2} C_m V_{dd}^2 f \alpha \tag{2}$$

where C_m is the total output capacitance of the system, f is the operating clock frequency and α is the proportion of gates switching from 0 to 1 in a clock cycle. The parameters α and C_m are difficult to estimate but can be obtained by detailed simulation.

This latter model is widely used, even in other models, where it gives the shape of the

power consumption of a system or block, but it is not adapted to the specificity of the logic contained in the block.

Transistor/Gate-level models One of the most accurate methods to estimate power consumption before a circuit is realized is doubtlessly transistor/gate level simulation. In fact most of synthesis tools provide power consumption prediction, as for example PowerMill [7] from Synopsys and QuickPower [6] from Mentor. These tools are low level (circuit or HDL) simulators. Other simulators operate at circuit level, such as Spice-based simulators (Star-Sim [9] for example). This kind of simulation gives accurate fine-grained results, but are very time consuming. In fact the simulation time limits the number of events simulated.

A first improvement for this situation is gate level simulation. The elementary unit is not the transistor anymore but the gate (an assembly of transistor). Mynoch [14] for example, runs 450 times faster than Spice based simulation.

The models (and simulators based on these models) presented here requires detailed information on the hardware modeled, HDL source or equivalent information. At software development phase, it is almost impossible to get these information from the manufacturers. Besides this kind of models do not meet our needs for simplicity. Moreover measurement based model does not exist and are probably impossible to build at such a low level of granularity.

Architectural-level models The main specificity of architectural level models is that they divide the targeted system into functional units, and that each unit can have its own energy model. Kim *et al.* [8] even make their architectural model recursive, which means

that each unit can be divided in turn in sub-units.

The ability of using different models for blocks allows Chen *et al.*, in [5], to use *bit-dependent* or *bit-independent* models for blocks depending if the block's consumption varies when input is changing. In Wattch [2] the different models are chosen according to the structure of the units.

Parameters of these models are mainly based on behavioral (cache misses, ...) and architectural (cache geometry, ...) informations ([8]). The main advantage of these models is that they are flexible, their main aim is the reuse of part of the model between different target system. Indeed, there is no need to recreate an entirely new model for a new architecture, but to add, remove or modify existing functional blocks.

Some of the models presented in this family are easily adaptable to our objectives of modelling the full platform, since they can be augmented to take into account the full system (CPU plus peripherals, memories, ...). Unfortunately most of the models of this category are not measurement based.

Instruction-level models Instruction level models are centered on CPU consumption. The main task in these models is to list all instructions consumptions ([18], [10], [15]). A first enhancement of this approach is given by Tiwari *et al.* in [18], it consists in characterizing the inter-instructions power consumption, which represents the logic switching between two different instructions. Others works also take into account the logic switching due to data parameters [16]. Measurement-based methods are widely used at this level of abstraction, some of the building methods proposed here have the characteristics we want for our methodology: simplicity and minimum architectural information.

The solutions proposed for measurement setups have a wide range of complexity. It ranges from a simple ammeter in [17] to complex cycle accurate setups in [4], via current mirrors in [12]. The most relevant proposition in our case is the one in [15]. The data acquisition solution use a digital oscilloscope connected to a resistor placed in series with the power supply. For this kind of measurement, they use a high performance oscilloscope (LeCroy LC534) which has a high sample rate. This setup is augmented with a trigger signal, which gives the beginning and the end of the measurement period.

To conclude, simulation and analytical model building methods are generally oriented for early stage of VLSI design, before hardware production. Conversely measurement based method needs less information on the underlying architecture. These last points drive us to propose a measurement based methodology, since at software development phase, all information needed for simulation based method will not necessarily be available.

Finally, all instruction level models proposed before are centered on a CPU, and do not take into account peripherals. The whole system is then not taken into account with these models. Some exceptions are present in the architectural level models. For example, Li *et al.* [11] propose a model in which CPU, memory and busses are different units. Our model will be closer to system-level modeling than the instruction-level ones.

3 Architectural Level Energy Consumption Model

3.1 Measurement setup

The choice of measurement point is very important. In fact, this choice will have an influence on many other choices in the following steps. The most important thing is that it is tightly coupled with the informations we can/want to extract from the measures.

The point here is that we want our model to be built and used by people who do not have necessarily the skills to build a complex electronic measurement setup. To meet this constraint we made the decision to use measures collected at the power supply input of the system. This is the best way to make non-intrusive simple measures.

This point of measure is where the battery is connected, hence all measured values will represent exactly what is consumed on it. By this we mean the consumption of the main chips (System-On-Chip, ...) and their integration components (capacitors, ...).

3.2 Model Parameters Selection

From section 2, we can draw the conclusion that the resulting model should be an extension of an architectural level model. In that condition, the model parameter selection is composed of two main steps made by the model builder, *i.e.* the software developer.

The first step is the components identification. By architectural exploration, it should be possible to build an exhaustive list. Main components are generally memories, peripherals, interconnects and CPU.

The second step consists in defining all possible parameters for these components. Due

to the limited literature available, the developers would not necessarily know the behavior of intra-blocks logic. The parameters for the blocks are then limited to behavioral parameters (cache misses, ...) and their hardware configuration such as operating mode.

Finally since the main interest of software developers is the software application, we can assume that the model will be used in a cycle accurate simulation framework. The resulting model may work as follows. A base operating energy consumption will be added for each time slot (probably CPU cycle) and penalties will be accounted on top of this base cost for special behavioral events (cache misses, bus accesses, ...) or configuration modification (peripheral state modification). This type of energy accounting model is more adapted to the data we are gathering, since no details are available on the distribution of the consumption when measures are made, more particularly on base consumption.

On top of the different architectural units of the overall system our model should also take into account special features of the hardware target which are changing the energy consumption. Indeed, frequency scaling and dynamic voltage scaling should be taken as parameters. This kind of parameters could influence the length and the energy cost of the events which were previously selected as parameters.

3.3 Benchmark Structure

The next step in the model construction is the parameters cost measurement. As our parameters would probably range from instructions to operating system services, it is attended that the attainable time accuracy of the setup will fall below the necessary time resolution.

To solve this problem, we built micro-benchmark for each of the event selected as a

possible model parameter. The benchmark is built as a repetition of the event in a loop. In the case where the system has caches, the loop body size is chosen by minimizing the influence of cache misses for cache loading of the loop and loop overhead.

The benchmark changes the state of the trigger signal before entering in the loop and after exiting it. These actions allows us to measure the consumption of an exact number of repetition of the targeted event.

The last characteristic of these benchmarks is that they are built over a lightweight operation system (OS), Mutek [13]. Only the hardware initialization part of the OS is used, OS initialization is replaced by the benchmark body. The use of this lightweight OS allows us to have a full control on what is running on the system during the measures.

4 Experimentation on an ARM Integrator Platform

This methodology was applied to an ARM Integrator platform. This platform is a development board based on an ARM922T. The architectural exploration reveals that it has a two levels bus architecture and three distinct levels of memory. The third level is main memory, the second level is scratch pad memory and finally the first level is cache memory. All peripherals are accessible through the two levels of bus. The measurement setup used for these experiments is close to the one depicted in [15]. We used a digitalizing oscilloscope, the shunt resistor is replaced by a current probe, and we also used a voltage probe.

The power signal measured at the power supply input of the board is varying with a frequency of about 500 kHz. This frequency is the operating frequency of the onboard voltage stabilizers. This confirms the fact that we could not have enough accuracy to directly

measure instructions events.

The selected parameters for our model are the CPU instructions, the bus accesses, the scratch pad memory accesses, the memory accesses and the peripheral operating modes.

4.1 Benchmarks

Here is a short list of benchmarks that were used, and their target event:

• *insn-cmp:* This benchmark is comparing different instructions executed in the CPU (add, mul, mov, ...). It is only executing the instruction in a loop.

• *I-and-D-cache:* With this benchmark we can get the load/store instruction cost. When the accessed information and the instructions are in cache.

• *AHB-reg-write:* This one gives the bus access (level 1 and 2) overhead. We write in a peripheral register a value that has no effect on the peripheral. The register must be accessible through the first level of bus.

• *I-cache-D-mem:* To get data memory access overhead with this benchmark, we deactivate the D-cache. All `load` or `store` instructions access the main memory.

• *I-cache-D-spm:* The aim of this benchmark is to get the scratch pad memory overhead. The D-cache is deactivated. Every memory access is then made on the SP-SRAM.

• *timer-test:* As an example of peripherals energy characterization, this benchmark allows us to get running/stopped timer consumption. It is subdivided into two benchmarks, one in which the timer is stopped and the second in which the timer is running. The structure of the loop is the same as the `insn-cmp` benchmark with a nop instruction, since it is the instruction generating the less activity.

• *loop-calibration:* Finally, this benchmark is the one which gives loop skeleton overhead. By running an empty loop benchmark, we can estimate the loop overhead.

On top of these various benchmarks, which correspond to most of the parameters events, the benchmarks are all configurable to allow us to estimate the remaining parameters of the model. All benchmarks can be run at different frequencies. As we mentioned before, frequency and voltage scaling are potential parameters of the model, and our tested platform only allows us to modify frequency. However, we can extrapolate our results to a platform with Dynamic Voltage Scaling (DVS) by assuming that the supply voltage can be scaled down in proportion to the clock frequency, which is a reasonable approximation if far away from the threshold voltage.

4.2 Results

bench name	length	energy (nJ)	error (pJ)
loop-calibration	4	61.026	48.594
insn-cmp_nop	1	15.005	6.7025
AHB1-reg-write	7	105.93	159.22
AHB2-reg-write	12	180.37	232.35
I-and-D-cache_ldr	1	16.853	9.9561
I-cache-D-mem_ldr	40	682.69	251.21
I-cache-D-spm_ldr	8	117.62	63.055

Table 1: Results of benchmarks

The results presented in table 1 summarizes the full measurements available in [1]. The measures allow us to conclude that we have four classes of instructions. The first contains the intra-CPU instruction. This category is represented by `insn-cmp` benchmark in table 1. Class 2 represents load/store instructions from the cache (`I-and-D-cache_ldr`). In the class 3 we put all load/store accessing the busses (`AHB1-reg-write`, `AHB2-reg-write`). The

`I-cache-D-spm` can also be considered as a bus access, since scratch pad consumption seems to be negligible against the bus consumption. Finally, class 4 instructions are memory accesses (`I-cache-D-mem`).

4.3 Frequency Scaling

As we stated before, the Integrator/CM has no dynamic voltage scaling (DVS) capabilities, hence when we reduce the frequency we cannot decrease energy consumption.

When repeating five benchmarks at different frequencies, we obtain the curves in figure 1. This figure represents the per event energy values for the five benchmarks as a function of the clock divisor, $r_f = \frac{f_{ref}}{f}$ where f_{ref} is the nominal frequency (198 MHz in our case).

We must underline that all five benchmarks generate activity in the modified clock domain, but not on the remaining part of the platform. As we can see from this figure, the five experiments gives linear results against the frequency ratio. Hence, the event energy cost can be modeled as:

$$E_{evt} = E_{base}^{cycle} \times l_{evt} \times r_f + E_{evt}^{eff} \tag{3}$$

where E_{base}^{cycle} is the base energy consumed by the remaining part of the platform in one full speed CPU cycle. The CPU cycle is the unit of time since it is the shortest event which can be measured and also which can happen on the platform. The E_{evt}^{eff} represents the unvarying consumption of the event, the effective energy consumption. This part can be model as follows:

$$E_{evt}^{eff} = \frac{1}{2} C V^2 \alpha l_{evt} \tag{4}$$

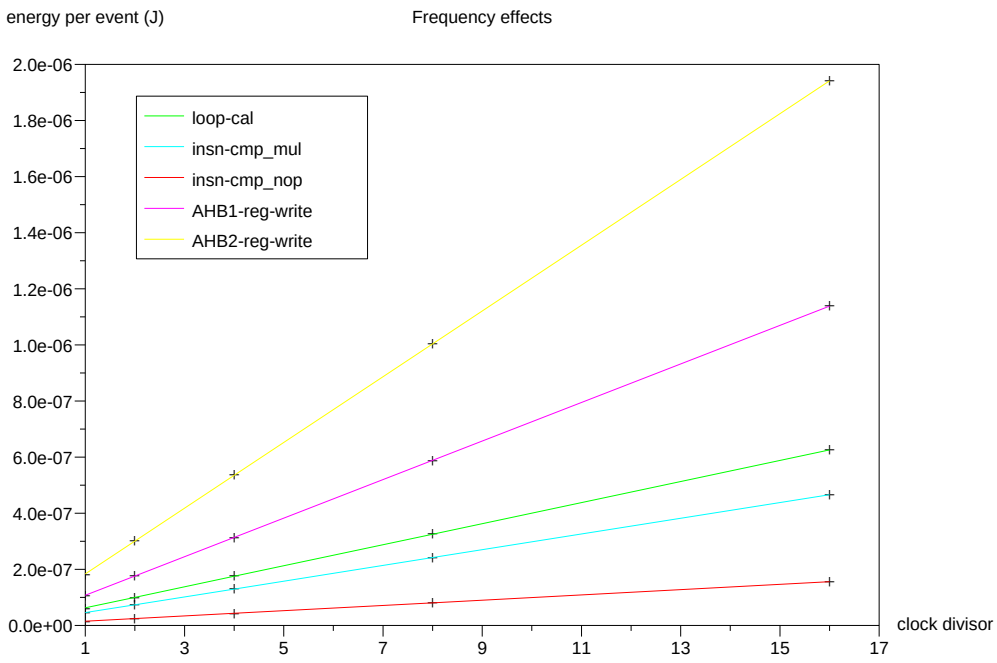

Figure 1: Multiple frequencies experiments: This figure shows that the energy per event increases linearly against frequency ratio.

This formula is derived from the equation (2). We can see here that it is not dependent on the frequency. l_{evt} is the event length in terms of full speed CPU cycles. This term explains the slope difference between the five lines, since each benchmark executes an event of different length.

Linear regressions on the results presented in figure 1 gives the following results:

Benchmark name	E_{base}^{cycle} (nJ)	E_{evt}^{eff} (nJ)	error (pJ)
insn-cmp_mul	9.34	17.62	340.8
loop-calibration	9.38	25.96	992.5
insn-cmp_nop	9.37	5.73	97.49

Table 2: Linear regression results

Equation (3) gives a good explanation for the experiments on clock frequency variation. These results gives us an estimation of what we can consider as base energy, which is not changing against software execution. The value obtained is about 9.37 nJ $\pm 1.4\,10^{-2}$.

4.4 Model

4.4.1 Basic Model

As we presented in section 3.2, the energy model can be viewed as the sum of the instruction costs and peripherals state costs. We cannot extract exact energy consumption of peripherals from global measures. In that case we can manage the peripherals consumption as a state machine that contains extra cost relative to a base cost. This base energy cost is part of the instruction energy cost.

The energy consumption model is then given by equation (5).

$$E_{app} = \sum E_{evt} + \sum E_{peri}^{over} \tag{5}$$

E_{app} is the application energy consumption, E_{evt} is the sum of the events costs and E_{peri}^{over} represents the peripheral state energy overhead relative to the state chosen as basis.

The E_{evt} values are given by equation (3). As we can realize from this formula, one part is time dependent, the first, whereas the second is not.

A first step is then to express the total time of application. This quantity is given by equation (6).

$$t = \sum c_i \times \frac{r_f^i}{f_i} \tag{6}$$

where c_i represents the durations of events of class i expressed in CPU cycles, r_f^i is the frequency ratio of class i event domain frequency presented before, f_i is the nominal frequency of domain i. Class 1 is containing CPU instructions, class 2 contains loads and stores from the cache, class 3 are loads and stores accessing to the bus and finally class 4 contains memory accesses.

As far as energy is concerned, we can extend the first member of equation (5) as follows.

$$\sum E_{evt} = t \times E_{base}^{cycle} + \sum E_i^{eff} \times c_i \tag{7}$$

where E_i represents the energy for each classes of event.

The complete model is depicted in equation (8).

$$E_{app} = \sum E_i^{eff} \times c_i + t \times E_{base}^{cycle} + \sum E_{peri}^{over} \tag{8}$$

4.4.2 DVS Extrapolation

In this section we present an hypothetical extension of the previous model for a DVS enabled platform. The effective energy of different classes events presented before as E_i^{eff} are approximated by the basis dynamic power model (equation (4)).

As we saw before the frequency influence on these part is null. This is the reason why frequency scaling has no effect on energy consumption in our experiments. But if we introduce the fact that V_{dd} can be adjusted this not true any more. We take the assumption that if we divide the frequency by r_f we can divide V_{dd} by the same amount. This is true when we are not too close to the transistor threshold voltage. In that situation the E_{evt}^{eff} is modified and expressed in equation (9).

$$E_{evt}^{eff} = P_{evt} t_{evt} = \frac{1}{2} C \frac{V_{dd}^2}{r_f^2} \alpha f_{ref} t_{evt}^{ref} \tag{9}$$

The benefit is then of $\frac{1}{r_f^2}$. In our particular case, we can apply this on the three first class of events since they are all three is the same clock domain. The fourth is not in the same clock domain, hence the V_{dd} can not be modified in the same way.

It is clear that voltage and frequency scaling would have been of great interest for the elements of our platform outside the CPU clock domain.

5 Model Validation

At this point we showed that it is possible to build a model of energy consumption by using simple non-intrusive measurements. The last step is to validate this model. This

validation is made by testing the accuracy of an application consumption estimation against real consumption. Estimations are obtained as presented in section 4.4, by estimating the model parameters and computing the application consumption. The effective consumption is obtained by the same method than the one used for model building, by using the same measurement setup and benchmark construction.

A first approximation of the model accuracy was obtained by reproducing this procedure by hand on an operating system service, the commutation routine. To build a benchmark whose structure is similar to the one created before, we make the commutation procedure save and restore the same context. The overall structure of the benchmark is the same as in the previous experiments.

The estimation of the energy consumption is 814.10 nJ. By physical measurement we obtained a cost of 772.02 nJ per commutation, which means that our estimation has an error smaller than 6%. This results gives a first approximation of the model accuracy. Further validation tests will be made on more complex applications by deriving event counts from a functional simulator. However, this first test allows us to confirm that the model is valid and gives accurate results.

6 Conclusion

In this paper we have explained how an accurate energy consumption model for a full embedded system can be built from external measurements and micro-benchmarks. Our methodology necessitates a prototype platform of comparable technology. Quantitative energy data are gathered at the battery output, and are translated into per instruction energy figures by

data analysis.

The resulting model is thus driven by the embedded software activity. It is for instance possible to augment a software simulator with an energy estimator, or to use the analytical model to arbitrate between possible implementation at the compiler or library level. The resulting model is simple enough to be used efficiently in a simulator. Consumption data clearly identify power hungry operations, thus offering guidelines for design tradeoffs.

Our immediate aim is to integrates our model in a simulator such as SimpleScalar [3]. Other work will provide extensions to this methodology in order to support advanced energy consumption optimization techniques such as frequency and voltage scaling which may be found on prototype platforms.

References

[1] Anonymous. Reference removed for the sake of anonymity. Technical report.

[2] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *27th International Symposium on Computer Architecture ISCA*, 2000.

[3] Doug Burger and Todd M. Austin. The SimpleScalar tool set, version 2.0. *ACM SIGARCH Computer Architecture News*, 25(3):13–25, June 1997.

[4] Naehyuck Chang and Kwanho Kim. Real-time per-cycle energy consumption measurement of digital systems. *IEE Electronics Letters*, 36(13):1169–1171, 2000.

[5] Rita Yu Chen, Mary Jane Irwin, and Raminder S. Bajwa. Architecture-level power estimation and design experiments. In *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, volume 6, pages 50–66, January 2001.

[6] Mentor Graphics Corporation, 1999.

[7] Synopsys Corporation. Powermill data sheet, 1999.

[8] Nam Sung Kim, Todd Austin, Trevor Mudge, and Dirk Grunwald. *Power Aware Computing*, chapter Challenges for Architectural Level Power Modeling. Kluwer Academic, 2001.

[9] Ram K. Krishnamurthy. *Mixed Swing Techniques for Low Energy/Operation Datapath Circuits*. PhD thesis, Carnegie Mellon University, 1997.

[10] Mike Tien-Chien Lee, Masahiro Fujita, Vivek Tiwari, and Sharad Malik. Power analysis and minimization techniques for embedded dsp software. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 1997.

[11] Yanbing Li and Jög Henkel. A framework for estimating and minimizing energy dissipation of embedded HW/SW systems. In *35th Conference on Design Automation Conference (DAC'98)*, pages 188–193, 1998.

[12] S. Nikolaidis and T. Laopoulos. Instruction-level power consumption estimation embedded processors low-power applications. In *Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, International Workshop on*, pages 139–142, 2001.

[13] Frédéric Pétrot and Pascal Gomez. Lightweight Implementation of the POSIX Threads API for an On-Chip MIPS Multiprocessor with VCI Interconnect. In *DATE 03 Embedded Software Forum*, pages 51–56, 2003.

[14] D. J. Pursley. *A gate level simulator for power consumption analysis.* PhD thesis, Carnegie Mellon University, May 1996.

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

[16] S. Steinke, M. Knauer, L. Wehmeyer, and P. Marwedel. An accurate and fine grain instruction-level energy model supporting software optimizations. In *International Workshop on Power And Timing Modeling, Optimization and Simulation (PATMOS)*, 2001.

[17] V. Tiwari, S. Malik, and A. Wolfe. Power analysis of embedded software: a first step towards software power minimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 1994.

[18] V. Tiwari, S. Malik, A. Wolfe, and M. Lee. Instruction level power analysis and optimization of software. *Journal of VLSI Signal Processing*, 1996.