Embedded Systems Energy Characterization Using Non-intrusive Measurements

ABSTRACT

We propose in this paper a complete system energy model based on non-intrusive measurements. This model aims at being integrated in fast simulation tools to give energy consumptions during software design of embedded systems. Estimations takes into account the whole system consumption, peripherals included. Experiments on a complex ARM9 platform show that our model estimates are in error by less than 10 percent, which is precise enough for application design.

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 as clock frequency, 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, software 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 in section 2 we present in section 3 a methodology to build complete platform energy consumption model oriented for software development. Section 4 presents the resulting model for an ARM9 development platform. This section also validates our model on more significant pieces of code, multimedia applications, thanks to its implementation in a fast and cycle accurate simulation tool. We then conclude and discuss future work.

2. RELATED WORKS

Many works focus on energy characterization of VLSI circuits. We can organize them using two main criteria: their level of hardware abstraction and the calibration 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. Among these models there are usually 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 and all combinations are not possible or pertinent. Indeed, analytically built models target low level units due to the complexity of the building process. As far as simulation based models are concerned, they are seldom used at the highest level of granularity since they have a huge building time. Finally we do not find low level model based on measurements since a VLSI implementation does not allow easy access to individual gates or transistors.

In transistor (gate) level models, all transistor (gate) state changes are computed to give an energy consumption approximation for a VLSI component. This method is highly accurate, but a complete description of the component is needed. The simulation of all transistors (gates) is necessary to estimate the consumption, which represents a huge running time. The models built at this level of abstraction

are generally reserved to hardware designers.

At the upper level of abstraction, architectural or RTL level, the system is divided in functional units. Each unit can be represented by a specific model (*e.g.* bit-dependent or bit-independent models for Chen *et al.* [2]). The model is then adapted to the functional block internal structure. To be more accurate some works, like Kim *et al.* [4], subdivide the block into sub-blocks to apply different models on each sub-block. This family of models allows to extend model to a complete platform, but the models proposed so far will not be able to execute a full software system.

The highest level is instruction/system level of abstraction. At this level, models are based on events such as instructions execution ([12, 6, 8]). Tiwari *et al.* in [12] propose to characterize the inter-instructions energy consumption, which represents the logic switching between two different instructions. Others works also take into account the logic switching due to data parameters [10]. The system considered in this kind of models is generally composed of CPU, bus and memory.

Only few works focus on modeling a complete platform. Among them, EMSIM [11] is a simulator based on Simunic *et al.* [9] model for the StrongARM SA110 energy characterization. This simulator poorly characterizes the peripherals.

The closest work to ours, AEON's model [5], is a complete platform energy consumption model based on measurement. The targeted platform is a micro-controller based sensor network node.

As far as calibration methods are concerned, analytical models are generaly based on manufacturers data, *e.g.* in Simunic *et al.* [9] the model is built thanks to electrical datasheet informations. Simulation based calibration needs a full knowledge of the underlying level architecture, which means that it needs a description of low level hardware (VHDL, or Verilog descriptions). Measurement based method only needs few informations on the hardware. On top of that, works like [12, 3] prove that it is possible to extract internal unit consumption from system measurements.

In this paper we propose a methodology for complete platform energy consumption model construction based on simple and non-intrusive measurements. The model thus built is placed at a level of abstraction close to the system level presented before, but is extended to the complete platform by coupling it with architectural level principles presented by Kim *et al.* in [4]. We also take into account peripherals energy models and dynamic frequency and voltage scaling into account.

3. MODEL CONSTRUCTION BASICS

We present in this section our methodology to build complete platform models. We first give more details on the structure and the parameters of the resulting model. We then give guidelines to calibrate this model through micro-benchmarks and simple and non-intrusive measurements on the hardware target. Section 4 will present the target dependent model parameters through a case study on an ARM9 based platform.

3.1 Model structure and parameters

Our choice among all the modeling method which have been presented in Sect. 2 is to build an architecture level model, in which the platform is divided in functional blocks, as in figure 1.

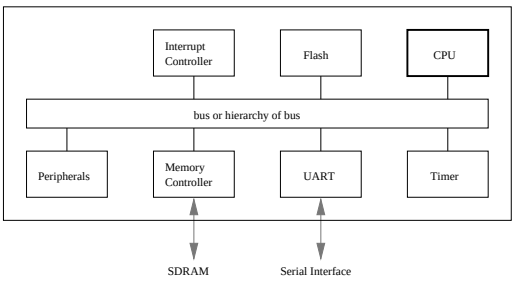

Figure 1: model block examples

The energy consumption of an application E_{app} is obtained be adding the all blocks consumptions E_{bl}:

$$E_{\text{app}} = \sum_{blocks} E_{\text{bl}} \tag{1}$$

Each block can have its own energy consumption model. To have a platform model better suited for software development, we apply the instruction level model solution for CPU modeling. The CPU energy consumption E_{CPU} is thus model as described in the next equation.

$$E_{\text{CPU}} = E_{\text{insn}} + E_{\text{cache}} + E_{\text{MMU}} \tag{2}$$

The energy consumption is the sum of the energy consumed by instruction execution, plus cache and MMU overheads consumptions, and consumption of the other blocks of the platform.

$$E_{\text{app}} = E_{\text{CPU}} + \sum_{blocks} E_{\text{bl}} \tag{3}$$

This model aims at being integrated in a cycle accurate simulation tool of the complete platform. The most interesting way of writing the model for this kind of purpose is to define a per time slot energy consumption. The chosen time slot is the CPU instruction execution. There are two reason for choosing this time reference. The first is that it is the finest time reference since CPU have generally the highest clock frequency in embedded systems. Secondly, interrupt requests, the only mean for the hardware peripherals to interact with the software, are managed at the end (or beginning) of the instruction execution. From a software point of view, there is no need to use a finer time reference to report hardware events more precisely.

The model can be rewritten in a form where the consumption of CPU and other blocks are reported for the currently executed instruction. All E_* will be kept for overall application consumptions, for the sake of notation simplicity instruction reported consumptions will be noted as $\mathcal{E}_*$. This new model formula is expressed in the following equation:

$$\mathcal{E}_{\text{slot}} = \mathcal{E}_{\text{CPU}} + \sum_{blocks} \mathcal{E}_{\text{bl}} \tag{4}$$

The last peculiarity in this model is the measurement based data collection. As we only get global measures for the platform consumption, we can foresee that the base consumptions of every blocks will not be easily distinguishable. We mean here that once the embedded system is put in its

laziest state, idle state for example with all possible units powered off, the resulting consumption is considered as a base consumption regrouping the base consumption of every powered peripherals. Obviously, a part of this consumption is static power dissipation. We will call this term $\mathcal{E}_{\text{base}}$, it is important to note that this consumption is reported to the current executed instrustion on the CPU. It can be expressed as in equation (6), as it is dependent on the instruction length l_{insn}. Equation (4) becomes equation (5).

$$\mathcal{E}_{slot} = \mathcal{E}_{\text{base}} + \mathcal{E}_{\text{CPU}} + \sum \mathcal{E}_{\text{bl}} \quad (5)$$

$$\mathcal{E}_{\text{base}} = l_{insn} \times \mathcal{E}_{\text{c\_base}} \quad (6)$$

The CPU and other blocks consumption are then expressed as overhead against the idle state.

As described in equation (2), CPU energy consumption is given by the executed instruction energy cost. This model can be simplified by regrouping instructions in classes as proposed in [6].

As far as other blocks are concerned, we can expand them as bus, memories and other peripherals. This is interesting since bus and memories will be subject to events generated by the processor, such as memory writes. The peripherals will be then modeled by state machines giving the energy consumption of the peripheral during the time slot.

The last step in model construction consists in defining all possible parameters for these components. Due to the limited information available, the developers would not necessarily know the behavior of intra-blocks logic. The parameters for the CPU are already selected, since it is modeled thanks to instructions consumptions. The same can be done for cache, MMU and even co-processors consumptions. The parameters for other blocks are limited to behavioral parameters (UART sending a byte) and their states such as operating mode (running, stopped).

Each energy cost in this model is function of the running frequency and power supply voltage to allow dynamic and frequency scaling capabilities of the platform to be modeled. An example of this is presented in the next section.

3.2 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 simple non-intrusive measures accounting the consumption of the main chips and their integration components (*e.g.* capacitors).

3.3 Benchmark Structure

The next step is the parameter cost measurement. As our parameters range from instructions to operating system services, it is expected 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 events selected as a possible model parameters. 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 compulsory cache misses 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 [7]. 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 full control on what is running on the system during the measurement.

4. MODEL CONTRUCTION CASE STUDY

In this section we propose an example of our methodology application. This methodology was applied on a ARM based development board. This platform uses an ARM922T, more precisely on an Altera Excalibur EPXA10, which is a FPGA integrating an ARM922T and usual embedded systems peripherals (*e.g.* UART, Timers) on the same chip. Our hardware architecture exploration reveals that the platform has three distinct levels of memory, a cache, a scratchpad and main memory. All peripherals are accessible through two levels of AMBA bus. We will give details about the energy consumption model construction for this platform, then we will check the accuracy of the built model.

4.1 Methodology application

The complete platform modeling method presented in section 3.1 was applied on our ARM Integrator CM922T-XA10 platform. We will give more details about the model construction steps, and the resulting model.

Measurement setup

The measurement setup used for these experiments is close to the one depicted in [8]. We used a digitalizing oscilloscope, the shunt resistor is replaced by a current probe, and we also used a voltage probe.

The voltage at the power supply input of the board is oscillating with a frequency of about 500 kHz, the operating frequency of the onboard voltage stabilizers. This confirms the fact that the attainable time resolution should not be accurate enough to directly measure instructions events.

Calibration benchmarks

We built benchmarks to calibrate our model, more precisely our block models. The hardware exploration gives us the main blocks to be modeled, namely the CPU, the different bus levels, the memory levels, and the other peripherals such as UART, interrupt controller or timers.

For example, the selected parameters for our CPU model are the CPU instructions, or possibly class of instructions, plus the caches and MMU activities. We thus built benchmarks to evaluate the cost of possible parameters, in order to select only relevant ones. Here are examples of benchmarks that were used, and their target event:

- *loop-calibration:* This benchmark is the one which gives loop skeleton overhead. By running an empty loop benchmark, we can estimate the loop overhead.
- *insn-XXX:* Comparition of the instructions execution costs in the CPU (add, mul, mov, ...). The target instruction is executed many times inside a loop.

• *XXX-access:* Calibration of costs of each bus level (AHB1/2) and memory level (cache, scratchpad or main memory), depending on the address accessed.
• *timer-test:* Example of peripherals energy characterization, this benchmark allows us to measure the timer power 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.

Calibration results

The benchmarks are run on the target platform and their energy consumptions are measured thanks to the measurement setup described earlier in this section. The results of these measures are listed in table 1. This table only reports a few examples, the full results are available in [1].

bench name	length	energy (nJ)	error (pJ)
loop-calibration	4	69.084	5.1777
insn-nop	1	16.747	1.2884
AHB1-access	6	101.33	7.7132
AHB2-access	18	300	22.998
Dcache-access	1	17.146	1.3007
mem-access	40	775.44	54.551
spm-access	8	131.72	10.168
timer-test_on(nop)	1	16.754	1.2857

Table 1: Benchmarks results for simple operation energy calibration.

These results represents for each example benchmark the length of the calibrated event in CPU clock cycles (second column), the per-event raw energy cost measured on the complete platform (third column) and finally the measurement error (fourth column). The energy costs reported here give the consumption of the complete platform during the execution of one event. These raw costs need to be refined to give the final parameters costs.

For example, the scratchpad memory access benchmark result (`spm-access`) gives the energy consumption of the CPU executing a load instruction, the bus conveying the load request and the load result and finally the scratchpad memory. The refinement of this results gives the following cost. The bus access cost includes the register accesses in the targeted peripherals, since it is impossible to dissociate their cost. By removing the consumption of the CPU (one laod and seven nop) and the consumption of the bus, we finally obtain the scratchpad memory access cost. The conclusion is that the scratchpad memory do not consume more energy than a register access via the bus. The `timer-test` gives us the timer running state overhead. From the table results, we can draw the conclusion that this overhead can be neglected.

Other model simplifications are possible in the case of this platform. For example, the CPU cache models are simplified by taking into account only memory access bursts in case of misses since the overhead can be neglected. The MMU has the same kind of simplification, since the TLB (Translation Look aside Buffer) misses generate memory accesses, and the table walk represents only a negligible amount of energy.

Model

The basic model presented in section 3.1 can be rewritten, by using models simplifications obtained by calibration.

We found that most instructions have the same energy consumptions as long as they stay inside the CPU. Currently only ARM32 instruction set is modeled. Thumbs (16bit) instruction set could be modeled using them same benchmark methodology in no time. In our setup, it is not possible to isolate the instruction cache consumption, which is lumped with the instruction consumption. Cache misses will then be modeled as simple memory accesses.

We finally have a model where CPU instructions are regrouped in two classes, the logical and integer intra-CPU instructions, and the load and store instructions. A memory load access is modeled as a load instruction, plus a bus overhead, plus a memory overhead. Last but not least, the peripherals energy consumption are taken into account thanks to state machines that give their consumption during the instructions execution.

This model is finally resumed by Equation 7.

$$\begin{aligned} \mathcal{E}_{\text{slot}} &= \mathcal{E}_{\text{base}} \\ &+ \mathcal{E}_{\text{insn}} + \mathcal{E}_{\text{bus\_access}} + \mathcal{E}_{\text{mem}} \\ &+ \sum_{periph} \mathcal{E}_{\text{periph\_state}} \end{aligned} \quad (7)$$

where $\mathcal{E}_{\text{slot}}$ is the energy consumption of the instruction execution time slot, $\mathcal{E}_{\text{insn}}$ is the cost of instruction given by its class cost, $\mathcal{E}_{\text{bus\_access}}$ is the bus overhead cost for load or store instructions, $\mathcal{E}_{\text{mem}}$ is the overhead for memory accesses. The last term represents the sum of the energy overhead of peripherals state. These cost are all overhead costs, since the full consumption of a peripheral cost, for example, is given by its base energy cost comprised in $\mathcal{E}_{\text{base}}$ and the overhead.

Frequency Scaling

The model presented before is valid for full speed software execution. However, the Integrator CM922T-XA10 has frequency scaling capabilities but 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 2. This figure represents the per event energy values for the five benchmarks as a function of the clock divisor, $r = \frac{f^{\text{ref}}}{f}$ where f_{ref} is the nominal frequency (198 MHz in our case).

These curves show that energy per event increases when frequency is decreased, and this may seem counter-intuitive. To understand these results observe first that a given event, *e.g.* the execution of some specific instruction, entails an almost constant number of bit flips, and that each flip uses a fixed amount of energy. Hence, to a first approximation, and in the absence of voltage scaling, the energy for a given event should be a constant. However, in our platform, frequency scaling acts only on the processor and Excalibur embedded peripherals; the consumption of other peripherals, external memories and FPGA is not affected. Hence, the addition of a parasitic term which is roughly proportional to the duration of the event or inversely proportional to frequency. This is clearly the case for the curves of Fig. 2.

We must underline that all five benchmarks generate ac-

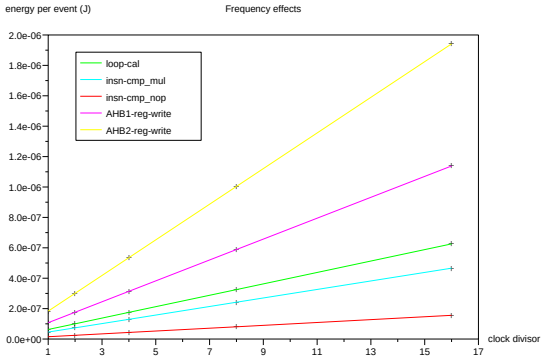

Figure 2: Multiple frequencies experiments: This figure shows that the energy per event increases linearly with the clock period.

tivity in the modified clock domain, but not on the remaining part of the platform. On top of that we kept all peripherals in the modified clock domain in an idle state. Hence, the event energy cost namely $\mathcal{E}_{\text{evt}}$, which can be an instruction execution or a bus access for examples. In this consumption we can identified two types of consumption. The first is the energy due to modified clock domain $\mathcal{E}_{\text{mc}}$, which is constant (CPU + bus + some peripherals). The second is the one due to the remaining part of the platform $\mathcal{E}_{\text{rp\_base}}$. Their relation in the total consumption of event is given by relation:

$$\mathcal{E}_{\text{evt}} = \mathcal{E}_{\text{rp\_base}} \times l_{\text{insn}} \times r + \mathcal{E}_{\text{mc}} \quad (8)$$

The first term is dependent on the frequency ratio r and the instruction length l_{insn}, whereas the second is not.

Linear regressions on the results presented in figure 2 gives the following results:

Benchmark name	$\mathcal{E}_{\text{rp\_base}}$ (nJ)	$\mathcal{E}_{\text{mc}}$ (nJ)	error (pJ)
insn-mul	10.91	26.37	572.36
insn-	10.52	19.22	258.90
insn-nop	10.54	6.35	105.61
access-AHB1	11.06	36.72	1085.37
access-AHB2	11.06	106.32	3431.46

Table 2: Linear regression results

As shown in this table, equation (8) 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 last two figures are the real consumption of the events in the modified clock domain, in these cases instruction executions, and the regression error. The value for the base energy can be approximated by the mean of obtained values 10.82 nJ (with a standard deviation of $\pm 2.610^{-2}$) per CPU clock cycle. The conclusion of this experiment is that frequency scaling and even DVS capabilities could help to define energy consumption repartition with a finer grain.

DVS Extrapolation

In this section we present an hypothetical extension of the previous model for a DVS enabled platform. The energy of different events presented before as $\mathcal{E}_{\text{evt}}$ are approximated by the basis dynamic power model (equation (8)).

As we saw before the frequency influence on $\mathcal{E}_{\text{mc}}$ 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 is not true any more. We take the assumption that if we divide the frequency by r we can divide V_{dd} by an $r_{\text{v}} = \frac{V_{dd}^{ref}}{V_{dd}}$ amount depending on r. For example, Siminuc *et al.* prove experimentally that the relation between the voltage and frequency of their StrongARM SA1100 can be approximated by: $\frac{1}{r_{\text{v}}} = 0.66\frac{1}{r} + 0.33$. In that situation the $\mathcal{E}_{\text{evt}}$ is modified and expressed like this for instruction execution for example:

$$\mathcal{E}_{\text{insn}} = \frac{1}{2} C \frac{V_{\text{dd}}^2}{r_{\text{v}}^2} \alpha l_{\text{insn}} \quad (9)$$

The benefit is then of $\frac{1}{r_{\text{v}}^2}$. In case we approximate the relation between voltage and frequency in our platform, by the one given earlier, we would have a benefit of $(0.66\frac{1}{r} + 0.33)^2$. For a clock ratio of 2, half the speed, the energy benefit would be of 56 % less consumption. This relation can be applied on the modified clock domain, and thus a NOP instruction would have cost 2.76 nJ instead of 6.35. The base energy would not be affected.

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.

4.2 Model validation

To check the accuracy of the model thus built for the ARM Integrator CM922T-XA10, we describe here our accuracy tests experiments. The model were implemented in a simulator, and its results were compared to physical measurements.

Simulator integration

Our model is implemented in a simulation tool suite. This simulation tools are composed of two simulator.

The first is a complete platform functional simulator in charge of generating a cycle-accurate execution trace of the software. This trace reports all executed instructions, and all peripherals activities (state changes). This first step allow software developers to functionally debug their applications and supply them the material to make the second step simulation. To fulfill this step task, we implemented the behaviour of the Integrator platform in the open source simulator `skyeye`. We also upgraded it to the cycle accurate trace generation.

The second step is energy simulation tool proper. This simulator implements the model presented in the previous section. Its main task is to compute model parameters from the cycle-accurate execution trace. It accumulates all computed energies, and reports them in an energy profile file. The format of this file is an emerging file format, which can be visualized thanks to the open source project `KCacheGrind`. This simulation step allows to get the overall consumption of the software 'run', figures we will use in the next step of this validation.

		Measured values		Simulated values		Error	
Bench-name	code lines	cycles	energy (J)	cycles	energy (J)	cycles (%)	energy (%)
jpeg	25819	6916836	1.142440e-01	6607531	1.037940e-01	- 4.4	- 9.1
jpeg2k	4686	7492173	1.268535e-01	7663016	1.200488e-01	+ 2.2	- 5.3
mpeg2	24657	13990961	2.335522e-01	14387358	2.208065e-01	+ 2.8	- 5.4

Table 3: Simulators results: the results obtained for execution time and energy consumption by real hardware measurement are shown in second and third columns, the simulation ones in fourth and fifth columns. The last two columns give the error percentile of the simulation.

Validation methodology

To check the accuracy of the resulting model, we propose to compare the consumption estimation of the model, thus implemented in our tool to physical measurement on the real platform.

The test application chosen for this model validation are widely spread multimedia applications : JPEG, JPEG2000 and MPEG2. The implementations of these three applications are Linux standard libraries. Hence they use operating system services and standard libc functions. All experiments could have been made with Linux (or even uClinux), since the simulation tools are complete enough to run these operating systems. For limited measurement duration reasons, we decided to replace these heavy OS by the lightweight one, Mutek [7]. Linux hardware layer abstraction makes interrupt request managment too long to allow a reasonable sized image to be decoded in our measure time window.

The three applications are executed in the simulation tools to get model estimations of their executions. As far as the measurement setup is concerned, we kept the same setup as the one used for model calibration, presented in section 4.1.

Accuracy

Results of model estimations and physical measurements are presented in table 3. The second column gives an idea of the application code complexity, by giving the total number of source code lines. These figures do not integrate the operating system source code.

The third and fourth columns reports the physical measurement results, in terms of execution duration in CPU clock cycles and in terms of energy consumption in Joules. Fifth and sixth columns gives the same kind of informations concerning the simulation results. Finally, the last two columns gives the percentile error of simulation errors of the simulation results against the physical measurement on the target hardware platform.

These results show that a 10% error rate can be achieve by our simple complete platform energy model. This estimations are obtained in roughly less than a minute (25s for the first simulation plus 20s for the second). We think that the error rate of 10% is largely acceptable in regard of the simulation time.

5. 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 requires 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 functional simulator with an energy estimator. Consumption data clearly identify power hungry operations, thus offering guidelines for software design tradeoffs. The model built on an ARM based development board using this methodology achieved an error rate of less than 10 %, which is accptable compared to its simplicity of implementation and its fast running time.

6. REFERENCES

[1] Anonymous. Research report.

[2] R. Y. Chen, M. J. Irwin, and R. S. Bajwa. Architecture-level power estimation and design experiments. In *ACM TODAES*, volume 6, pages 50–66, January 2001.

[3] G. Contreras, M. Martonosi, J. Peng, R. Ju, and G.-Y. Lueh. XTREM: a power simulator for the Intel XScale core. In *LCTES '04*, pages 115–125, 2004.

[4] N. S. Kim, T. Austin, T.r Mudge, and D. Grunwald. *Power Aware Computing*, chapter Challenges for Architectural Level Power Modeling. Kluwer Academic, 2001.

[5] O. Landsiedel, K. Wehrle, and S. Götz. AEON: Accurate Prediction of Power Consumption in Sensor Nodes. In *SECON*, Santa Clara, October 2004.

[6] M. T.-C. Lee, M. Fujita, V. Tiwari, and S. Malik. Power analysis and minimization techniques for embedded dsp software. *IEEE Transactions on VLSI Systems*, 1997.

[7] F. Pétrot and P. 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.

[8] 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.

[9] T. Simunic, L. Benini, and G. De Micheli. Cycle-accurate simulation of energy consumption in embedded systems. In *36th Design Automation Conference*, pages 867–872, May 1999.

[10] S. Steinke, M. Knauer, L. Wehmeyer, and P. Marwedel. An accurate and fine grain instruction-level energy model supporting software optimizations. In *PATMOS*, 2001.

[11] T. K. Tan, A. Raghunathan, and N. K. Jha. EMSIM: An Energy Simulation Framework for an Embedded Operating System. In *ISCAS 2002*, May 2002.

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