Energy Consumption Estimation for Embedded Applications

Momcilo V. Krunic[1], Miroslav V. Popovic[2], Vlado M. Krunic[3], Nenad B. Cetic[1]
[1]*RT-RK, Institute for Computer Based Systems LLC,*
Narodnog Fronta 23a, 21000 Novi Sad, Serbia
[2]*Department of Computer Engineering and Communications, Faculty of Technical Sciences,*
University of Novi Sad,
Trg Dositeja Obradovica 6, 21000 Novi Sad, Serbia
[3]*Department of Mathematics and Informatics, Faculty of Natural Sciences and Mathematics,*
University of Banja Luka
momcilo.krunic@rt-rk.com

***Abstract*—Energy consumption, indeed, represents one of the essential properties of embedded applications, especially for those devices whose autonomy depends on battery life. The lack of accurate and suitable methodology for energy consumption estimation for embedded applications based on ultra-low power heterogeneous multicore DSP platforms inspired a solution that will be presented in this paper. The solution has been developed as a plugin for the Eclipse based MIDE (Multicore Integrated Development Environment), in order to facilitate production of energy efficient firmware solutions. Evaluation of energy loss has been calculated using instruction-level power analysis, virtual platform, debug information, and diverse input loads. The primary goal was to obtain a precise model of energy consumption that will establish a direct link between program solutions and the amount of energy required for their execution, whilst processing different input loads. Estimation has been validated against empirical data, measured on a real DSP platform. Results show that very high accuracy has been reached.**

***Index Terms*— Embedded software; energy consumption; performance evaluation; software metrics.**

I. INTRODUCTION

Energy consumption has always been promoted as one of the most important aspects of engineering in general, since it has immense influence on designing process. Accurate estimation and evaluation of energy consumption, therefore, could facilitate development and production of energy efficient solutions. The scope of this paper is focused on the calculus for power analysis and energy cost of software solutions whilst running on embedded DSP platforms. Increased ongoing expansion of embedded devices implies the necessity of a research in this area.

Basically, there are two different approaches to energy consumption estimation that can be applied to embedded devices [1]: Physical measurements on real hardware, and simulation based modelling. It was established in [2], [3] that the first approach, which includes measurements of the current drawn by the processor, gives the most accurate information about the power cost. Taking that into consideration, as well as the lack of a hardware simulation model, such as SPICE or similar, for the target DSP platform, has encouraged us to choose the first strategy, and to conduct measurements on our own.

There are five distinct levels of power management that can be applied to computer systems [4]: application level, compiler level, operation system level, architecture level, and circuit level. The case study presented in this paper deals with power management mostly on application level [5]. The main goal was to provide an efficient and accurate tool for power analysis that will guide a firmware developer through the application development process in order to enhance software energy efficiency. The solution presented in this paper extends significantly the simple model presented in [6]. This paper proposes a new instruction-level, cycle accurate, energy consumption estimation model that was applied and tested on a multicore, ultra-low power, heterogeneous DSP platform. Besides instruction energy costs, the model also takes into account energy costs related to DSP platform peripherals. The estimation model is universal and applicable to any DSP platform; only target specific measurements should be performed using the methodology described in this paper.

The entire solution has been developed as a plugin for the Eclipse RCP [7] based Multicore IDE presented in [6]; therefore, it is easily transferrable to any other RCP based IDE, thus contributing to the universality of the solution.

Sections below are organized in the following order. Section II depicts related papers. Section III presents mathematical model of energy consumption estimation. Section IV provides insight into the proposed measurement methodology. Section V contains a detailed description of experiments, and the results, that have been conducted as a part of validation and verification process. Section VI concludes the paper.

II. RELATED WORK

As diverse as embedded platforms and the appliance domains are, there is a wide variety of solutions dealing with

Manuscript received 11 November, 2015; accepted 14 January, 2016.
This research was partially funded by the Ministry of Education and Sciences of the Republic of Serbia under project No. TR-32031. This research was performed in cooperation with RT-RK Institute for Computer Based Systems.

energy consumption estimation, but an absence of universal solutions that can be applied to the entire set. The model proposed by this solution aims to make a contribution towards that direction.

The models proposed in [1], [8] require hardware simulation models that are, in most cases, unavailable (like for the target platform presented in this paper), and less accurate than the physically measured data.

Methodologies presented by [2], [9], [10] take into account only the energy that is consumed by the processor while executing instructions, as well as inter-instructions effects, but discussion about other energy dependent modules, such as peripherals, is omitted.

In [3], the spotlight was placed on the power estimator that could be used for making architectural choices in the design process, so the intent was to achieve power management on the architectural level, unlike the solution described in this paper, which provides a framework for power analysis on application level.

Experimental results obtained in [11]–[13] have shown that current consumption of the entire instruction set, for selected target platforms, is quite uniform, so energy consumption models were adjusted according to that observation. However, instruction power profiling performed on the DSP platform presented in this paper has shown that current consumption varies from 120 µA for the *NOP* instruction, up to 387.5 µA for the *LOAD X[addr], X0* instruction, which lead us to the conclusion that models represented in [11]–[13] are not applicable to this platform.

Estimation model presented by [14] excluded detailed inter-instruction cost estimations and, instead of that, used a Hamming distance and weight of the instructions, since the results have shown that inter-instruction energy cost measured on that target platform is around 5 %. We have used a different approach, because, according to our measurements, inter-instruction effect has a much higher influence on the consumption for this target platform, in some cases over 40 % of the instruction's base energy cost [2].

None of the abovementioned models include energy consumption estimation of a multicore system, whereas the estimation model presented here does. Multicore energy consumption estimation model provided by [15] calculates consumption based on cores' frequency and utilization, unlike the solution presented here, which evaluates each core average power based on instruction execution at the current cycle and the cycle before that one. Also, it is worth noticing that this estimation model considers heterogeneous multicore platform, unlike the model presented in [15].

Since this paper represents an enhanced and extended version of the solution presented in [6], many significant improvements have been made on the existing platform. In the previous version of energy consumption estimation, only core activity and average DSP consumption per core were used as parameters in power analysis, whereas the enhanced version, presented in this paper, calculates power at instruction level, whilst achieving finer granulation and significantly higher accuracy.

III. ENERGY CONSUMPTION ESTIMATION MODEL

The embedded platform, for which this model has been derived, was developed to enhance performance in a hearing aid. Nevertheless, the model is universal and applicable to any embedded platform. The DSP platform that was used for this research contains five heterogeneous cores: two DSPs used mostly for numerical accelerations and three general purpose DSP cores. One general purpose DSP is a micro-controller which synchronizes and controls the whole system. All cores were developed with emphasis on an ultra-low power design. Bearing in mind that one of the most important characteristics of hearing aids, and all other embedded devices that depend on batteries, is autonomy, it could be concluded that any energy savings will enhance product competitiveness. Therefore, besides the energy savings that were achieved in hardware design, it should also be considered what energy savings could be accomplished in a software solution.

Besides DSPs, the target platform also contains several other peripherals, such as: analog, system, input/output, local processor unit, utility, and wireless. In order to obtain an accurate energy consumption model, besides the instruction-level energy profiling, it is necessary to empirically measure the average energy consumption of all counted peripherals, and to incorporate those values into the estimation model, unlike the estimation models presented in [2], [9], [10], [14].

The main issue that inspired the research was: is it possible to conduct software energy efficiency profiling depending on various input signals, and to provide an answer to the question how much time will pass until the hearing aid battery is discharged when processing a specific input signal.

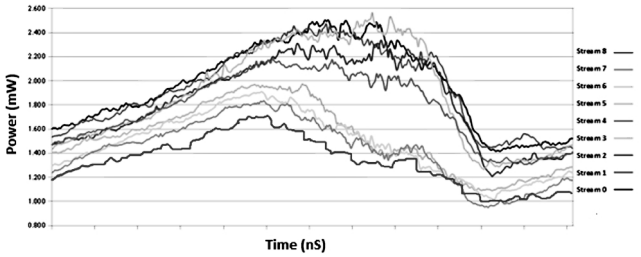

Fig. 1. Trend of energy consumption at various input loads.

Figure 1 depicts software energy consumption profiling at various input loads. The abscissa represents discrete time in resolution of cycles and instructions being executed within it. Execution flow, as well as cores and peripherals activity, have been obtained using virtual platform, profiler tool [16], and debug information.

The ordinate shows average power consumption, which is calculated based on empirical data. With this kind of representation, critical points, power consumption peaks, are easy to catch, as well as parts of the source code that consume the greatest amount of energy.

For example, it was measured that average power consumption during FFT processing was P_s = 3.4 mW, at DSP platform voltage U_p = 1.25 V, and battery capacity K_b = 310 mAh. Then, evaluation of battery life time – T_b can be calculated with the following equations:

$$E_b = U_p \times K_b = 387.5 \text{ [mWh]}, \quad (1)$$

$$T_b = E_b / P_s = 113.97 \text{ [h]}. \quad (2)$$

This implies that the battery will run out of charge after 113.97 hours of continuous FFT processing.

Further on, a mathematical model, which has been developed in order to facilitate this kind of analysis, is presented.

It is obvious from (2) that the only parameter that should be analysed furthermore is average power consumption P_s. This parameter can be calculated as an arithmetic mean of cycle's average power consumptions P_c

$$P_s = \frac{1}{n}\sum_{k=0}^{n-1} P_{c(k)}. \quad (3)$$

In (3), n represents the number of executed cycles, and $P_{c(k)}$ denotes the average power consumption of the *k-th* cycle. The average power consumption of a cycle remains to be defined

$$P_c = P_{dc} + \sum_{l=0}^{p-1} P_{f(l)} + \sum_{m=0}^{N-1} (P_{b(m)} + P_{o(m)}), \quad (4)$$

where P_{dc} is power consumption of the static component, and it is not dependent on the contents of a program memory, but represents overall power consumption of a DSP platform, whilst all cores and peripherals are in the idle state. $P_{f(l)}$ denotes average power consumption of the *l-th* peripheral. $P_{b(m)}$ stands for base power cost [2], [17] of the *m-th* core instruction. $P_{o(m)}$ denotes power consumption overhead of the *m-th* core instruction, which is generated by the inter-instruction effect [2], [18], [14] caused by a circuit state overhead [19]. Inter-instruction effect appears only when two different adjacent instructions are executed successively; therefore it is worth noticing that $P_{o(m)}$ element participates in the calculus only in the case when instruction from previous cycle differs from the current one. Stalls and cache misses that were included into the energy consumption model presented in [19] are omitted from this model, since the DSP platform presented here does not support cache and stalls by hardware design. Parameter p from the first sum denotes the number of active peripherals. N from the second sum represents the number of active cores. It should be noticed that power consumption of cores is considered as the sum of power consumptions of each individual core, and the effect presented in [20], power consumption overhead due to the core's shared resources, was not included in this model, since hardware design does not support such mechanism.

By combining (3) and (4), the following derivation is performed:

$$P_s = \frac{1}{n}\sum_{k=0}^{n-1}\left(P_{dc} + \sum_{l=0}^{p_k-1} P_{f(l)} + \sum_{m=0}^{N_k-1} (P_{b(m)} + P_{o(m)})\right), \quad (5)$$

$$P_s = P_{dc} + \frac{1}{n}\sum_{k=0}^{n-1}\left(\sum_{l=0}^{p_k-1} P_{f(l)} + \sum_{m=0}^{N_k-1} (P_{b(m)} + P_{o(m)})\right), \quad (6)$$

where p_k and N_k denote the number of active peripherals and the number of active cores, respectively, at the *k-th* cycle.

IV. Proposed Measurement Methodology

From (6), we identified four different aspects of power consumption that should be measured: DSP platform static component – P_{dc}, power consumption of peripherals – P_f, instruction's base power cost – P_b, and instruction's overhead power cost – P_o. All measurements were performed using Fluke True-RMS Industrial Logging Multimeter.

A. DSP Platform Static Component

Static component has been obtained by measuring the instantaneous current, drawn by the DSP platform, whilst DSP cores and peripherals run in idle state

$$P_{dc} = I_m \times U_p = 716\mu\text{A} \times 1.25\text{V} = 0.895\,[\text{mW}], \quad (7)$$

where I_m denotes measured current, and U_p is DSP platform voltage.

B. Peripheral's Power Cost

Average power consumption of a peripheral is measured in the similar manner as the static component, but the crucial difference is configuration of the DSP platform, as well as the test image that should be created for each individual peripheral. Then, the current drawn by the peripheral (I_f) can be measured as the difference between overall current consumption (I_m) and the sum of the static component (I_{dc}) and the current drawn by the core (I_{core}) before the peripheral has received the clock:

$$I_f = I_m - (I_{dc} + I_{core}), \quad (8)$$

$$P_f = I_f \times U_p, \quad (9)$$

where P_f denotes peripheral's average power consumption.

C. Base Power Cost

Instructions base costs has been determined using the well-established methodologies, described in [2], [21]. The methodology is intuitive. The base current, drawn by the instruction execution, could be measured when target instruction is executed simultaneously. That could be achieved by executing target instruction in an infinite loop. In order to minimize the influence of a jump instruction from the loop, it is recommended to put a number of instances of target instruction in the loop.

Figure 2 represents the source code that has been used for measurement of the instruction's *SUB x1, B0, B0* base cost. This method differs from the one presented in [17], where the loop contains only one instance of the target instruction and the number of instances of reference instruction (NOP)

$$P_b = (I_m - I_{dc}) \times U_p = \\ = (837\mu\text{A} - 710\mu\text{A}) \times 1.25\text{V} = 0.159\,[\text{mW}]. \quad (10)$$

Equation (10) represents evaluation of the base cost. The difference between the measured current and the static component represents the base current. Values in (10) were obtained after launching the source code from Fig. 2. Since the processor used in this research is a three-stage pipelined

DSP, then the base cost obtained in (10) is related to all three stages of execution: fetch, decode, and execute.

```
inline assembly void _SUB_x1_b0_b0 (void)
property(loop_free)
clobbers()
{
    asm_begin
    asm_text
    .lrepeat 1000
     SUB x1, b0, b0
    asm_end
}

int main(void)
{

    while(1)
    {
        _SUB_x1_b0_b0();
    }

    return(0);  //don't return
}
```

Fig. 2. The source code for measurement of the base cost of instruction SUB x1, B0, B0.

TABLE I. BASE COSTS OF INSTRUCTIONS

Instruction	I_{dc} [µA]	I_m [µA]	I_b [µA]	P_b [µW]
LOAD x[256], x0	710	1003	293	366.25
ADD y0, 1, y0	710	844	134	167.5
SUB x1, b0, b0	710	837	127	158.75
ABS b0, b0	710	836	126	157.5
MOVE x1, t1	710	838	128	160
LSHL x1, x0, x1 ## NOP	710	911	201	251.25
MPY y0, y0, a0 ## NOP	710	913.5	203.5	254.375
MSUB x1, y1, a0 ## NOP	710	913	203	253.75
STORE x0, x[0x2000]	709	911	202	252.5
STORE x0, x[256]	709	1014	305	381.25
NOP	710	830	120	150

Table I contains base costs of eleven different instructions, measured using the described methodology. The quantum of energy (*Eb*) that is consumed during an instruction execution can be calculated using the following equation

$$E_b = P_b \times N \times T_c = \\ = 0.159\text{mW} \times 1 \times 1/10^6\text{s} = 0.159\,[\text{nWs}], \quad (11)$$

where N denotes the number of cycles required for instruction execution, and T_c represents cycle time (f = 10 MHz). So, it can be noticed that instruction *SUB x1, B0, B0* consumes a minimum of 0.159 nWs every time it is executed.

D. Overhead Power Cost

Overhead cost occurs as a consequence of execution of two different adjacent instructions. This is caused by switching activity in the circuit, which is also known as a circuit state effect [1], [2], [18], [19], [21]. There are several approaches in modelling an inter-instructions energy effect, and some of those are explained in [1] in detail. Methodology for measurement of instructions overhead costs proposed in [2], [18], [19], [21] requires measurements between all individual instructions. Taking into account that target DSP, considered here, contains a set of approximately one hundred instructions, leads us to the conclusion that a huge amount of tests and measurements (12) needs to be conducted to assure overarching information about inter-instruction costs

$$K = \binom{n}{r} = \frac{n!}{r!(n-r)} = \frac{100!}{2! \times 98!} = 4950. \quad (12)$$

In (12), n denotes the size of instruction's set, and r is the number of selections.

```
inline assembly void _SUB_x1_b0_b0__NOP (void)
property(loop_free)
clobbers()
{
    asm_begin
    asm_text
    .lrepeat 1000
     SUB x1, b0, b0
     NOP
    asm_end
}

int main(void)
{

    while(1)
    {
        _SUB_x1_b0_b0__NOP();
    }

    return(0);  //don't return
}
```

Fig. 3. Source code for measurement of overhead cost of instruction SUB x1, B0, B0.

This observation affected the research to find appropriate approximation that will reduce the amount of tests required for measurements of overhead costs. The main idea was to find one instruction that will serve as a reference point for measurements. The candidate that stood out immediately was a NOP instruction. The new paradigm has been adopted: when the NOP instruction is executed with the target instruction, the whole inter-instruction cost could be assigned to the target instruction. The following derivation of (6) explains this approximation:

$$P_s = P_{dc} + \frac{1}{n}\sum_{k=0}^{n-1}\left(\sum_{l=0}^{p_k-1} P_{f(l)} + \sum_{m=0}^{N_k-1}\left(P_{b(m)} + P_{o(m)}\right)\right), \quad (13)$$

$$P_s = P_{dc} + \frac{1}{n}\sum_{k=0}^{n-1}\left(P_{b(k)} + P_{o(k)}\right), \quad (14)$$

$$P_s = P_{dc} + \left(P_{b(nop)} + P_{o(nop)}\right)/2 + \\ + \left(P_{b(inst)} + P_{o(inst)}\right)/2,\ P_{o(nop)} = P_{o(inst)}, \quad (15)$$

$$P_s = P_{dc} + \left(P_{b(nop)} + P_{b(inst)} + 2 \times P_{o(inst)}\right)/2, \quad (16)$$

$$P_{o(inst)} = P_s - (P_{dc} + \left(P_{b(nop)} + P_{b(inst)}\right)/2), \quad (17)$$

$$U_p \times I_{o(inst)} = \\ U_p \times I_s - (U_p \times I_{dc} + U_p \times \left(I_{b(nop)} + I_{b(inst)}\right)/2), \quad (18)$$

$$I_{o(inst)} = I_s - (I_{dc} + \left(I_{b(nop)} + I_{b(inst)}\right)/2), \quad (19)$$

where $I_{o(inst)}$ denotes the current consumed by the circuit state effect that can be assigned to a target instruction, I_s is an overall current measured during the experiment, I_{dc} current of the static component, $I_{b(nop)}$ represents the base cost of NOP
instruction, and $I_{b(inst)}$ denotes the base cost of the target instruction.

Figure 3 represents the source code that has been used for

measurement of the instruction's (*SUB x1, B0, B0*) overhead cost. Derivation presented in (13) and (14) has been made based on the facts that only one core was launching the source code from Figure 3 and peripherals were in idle state during the experiment. Approximation based on the adopted paradigm is introduced in (15).

TABLE II. OVERHEAD COSTS OF INSTRUCTIONS.

Instruction	$\Delta I=I_s-I_{dc}$ [μA]	$I_{o(inst)}$ [μA]	Overhead %
LOAD X[256], X0	301	94.5	32.25 %
ADD y0, 1, y0	175	48	35.82 %
SUB x1, b0, b0	145	21.5	16.93 %
ABS b0, b0	141	18	14.29 %
MOVE x1, t1	141	17	13.28 %
LSHL X1, X0, X1 ## NOP	252	91.5	45.52 %
MPY X0, Y0, A0 ## NOP	229	67.25	33.05 %
MSUB X1, Y1, A0 ## NOP	236	74.5	36.70 %
STORE x0, x[0x2000]	258	97	48.02 %
STORE X0, X[256]	306	93.5	30.66 %
GOTO	271	115	95.83 %

Figure 4 depicts the trend of current consumption with and without overhead cost. The trend is obtained from Table I (lower line), base cost, and Table II (upper line), overhead cost.

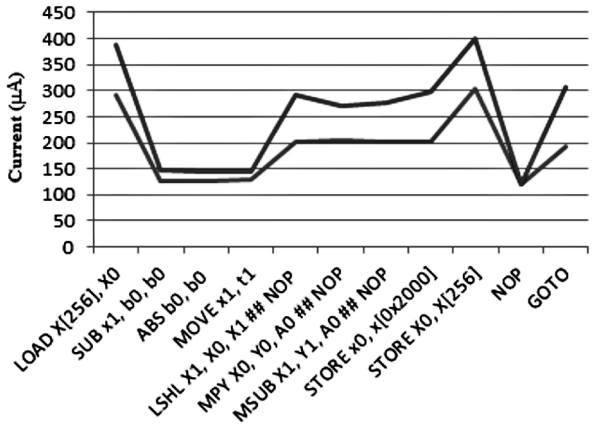

Fig. 4. Trend of current consumption, with and without overhead cost.

The described approximation decreases the required set of measurements from $O(N^2)$ space to $O(N)$ space, where N denotes the size of instructions set, which implies great savings in time and resources.

V. EXPERIMENTAL RESULTS AND VALIDATION

Validation of the proposed model has been performed using two benchmarks, as in [14]. In Benchmark 1, the current drawn by the processor was measured while executing separate blocks of instructions from Table I., where each individual block contains only instructions of the same type, thousand instances, in order to cancel inter-instruction effect. Also, peripherals were in idle state. Bearing that in mind, the fact that only one core was running during the experiment, and the value calculated in (7), (6) derives into

$$P_s = P_{dc} + \frac{1}{n}\sum_{k=0}^{n-1}\left(\sum_{l=0}^{p_k-1} P_{f(l)} + \sum_{m=0}^{N_k-1}\left(P_{b(m)} + P_{o(m)}\right)\right) = \\ = P_{dc} + \frac{1}{10}\sum_{k=0}^{9} P_{b(k)} = 1127.8\ [\mu\text{W}], \quad (20)$$

During the experiment, the measured current was I_m = 902 μA, which implies

$$P_m = I_{\text{m}} \times U_p = 902\mu\text{A} \times 1.25\text{V} = 1127.5\ [\mu\text{W}]. \quad (21)$$

From (20) and (21) accuracy ensues

$$A_c = \left(1 - \frac{abs\left(P_s - P_m\right)}{P_{ms}}\right) \times 100 = 99.97\ \%. \quad (22)$$

Benchmark 2 has been performed on an interleaved set of instructions from Table III.

In this experiment, the same as in the first one, peripherals were in idle state and only one core was active, but since instructions were interleaved, inter-instructions influence was not ignored, so (6) that derives into

$$P_s = P_{dc} + \frac{1}{n}\sum_{k=0}^{n-1}\left(\sum_{l=0}^{p_k-1} P_{f(l)} + \sum_{m=0}^{N_k-1}\left(P_{b(m)} + P_{o(m)}\right)\right) = \\ = P_{dc} + \frac{1}{11}\sum_{k=0}^{10}\left(P_{b(k)} + P_{o(k)}\right) = 1212.1\ [\mu\text{W}]. \quad (23)$$

The current that was measured during the experiment was I_m = 970 μA, which implies

$$P_m = I_{\text{m}} \times U_p = 970\mu\text{A} \times 1.25\text{V} = 1212.5\ [\mu\text{W}], \quad (24)$$

From the formula used in (22), and the results obtained in (23) and (24), it can be calculated that the accuracy for the interleaved set of instructions is 99.96 %.

Bearing in mind the accuracy that has been achieved, it seems that selection of the NOP instruction was a good choice at this point of the research. If estimation accuracy decreases when measurements extend to the entire instruction set, then some alternatives should be considered as reference points. These results confirm the proposed estimation model and measurement methodology, but it should be noticed that new benchmarks should be developed that will aim to challenge the proposed model against diverse applications.

TABLE III. BASE COSTS + OVERHEADS OF INSTRUCTIONS

Instruction	Base cost + Overhead [μA]	Base cost + Overhead (Pb+Po) [μW]
LOAD x[256], x0	387.5	484.375
SUB x1, b0, b0	148.5	185.625
ABS b0, b0	144	180
MOVE x1, t1	145	181.25
LSHL x1, x0, x1 ## NOP	292.5	365.625
MPY x0, y0, a0 ## NOP	270.75	338.4375
MSUB x1, x1, a0 ## NOP	277.5	346.875
STORE x0, x[0x2000]	299	373.75
STORE x0, x[256]	398.5	498.125
NOP	120	150
GOTO	307	383.75

The results certainly seem promising and they establish a solid ground for further research.

VI. CONCLUSIONS

Estimation of energy consumption may have a significant

impact on the final version of the software solution, as it would provide an accessible tool for power analysis of the source code. This observation will be further discussed in impending research, when measurements extend to entire instructions set. At this point, we are not sure about overall impact on firmware development, since this is in early stages of the research, but that is certainly one of our goals. The connection between energy consumption and the source code has been accomplished at the instructional level, so each cycle would be tagged with the amount of energy that it consumes and instructions executed within it. This has been achieved using the proposed mathematical model (4), measurement methodology and the profiler tool [16]. A vivid representation of the energy consumed by the source code (Fig. 1.) may provide a straightforward input for reorganisation of the source code at the critical points, power consumption peaks, in order to achieve energy savings. Also, instructions' power profiles will provide an insight into the energy costs, which may influence instructions selection during application development, since the proposed measurement methodology and estimation model provide a high level of accuracy (over 99 %).

In Benchmark 1, measurements were conducted whilst executing separate blocks of instructions, and it was measured that 1127.5 μW was consumed during the experiment. Also, power consumption estimation was calculated using Table I and mathematical model (20), and it amounts to 1127.8 μW (20). That implies that estimation accuracy in Benchmark 1 was 99.97 % (22). In Benchmark 2, measurements were performed during execution of interleaved set of instructions, and it was measured that this setup consumes 1212.5 μW. Using the estimation model (23) and Table III, power consumption estimation value was calculated to be 1212.1 μW. These numbers lead us to the conclusion that the achieved estimation accuracy in Benchmark 2 was 99.96 %. It should be emphasized that the achieved accuracy was calculated for 10 % of the entire instruction set. Future work could be focused on expansion of measurements to the entire instruction set. Afterwards, estimation should be challenged against various applications in order to obtain more reliable data about accuracy. Also, application level optimization, presented in here, could be extended to a compiler level power management. The obtained empirical data could be used as an input during compiler's instructions selection and scheduling, similarly to what was proposed in [21], [22].

The proposed model, as well as the measurement methodology, are rather general and it seems that they could be applied to other embedded processors.

REFERENCES

[1] B. Klass, D. E. Thomas, H. Schmit, D. F. Nagle, "Modelling interinstruction energy effects in a digital signal processor", in *Proc. Digital Signal Processor, Power-Driven Microarch. Workshop in Conjunction with Int. Symp. Comput. Arch*, Barcelona, Spain, 1998.

[2] V. Tiwari, S. Malik, A. Wolfe, "Power analysis of embedded software: A first step towards software power minimization", *IEEE Trans. VLSI Systems*, vol. 2, no. 4, p. 437–445, 1994. [Online]. Available: http://dx.doi.org/10.1109/92.335012

[3] N. Sung, T. Austin, T. Mudge, D. Grunwald, "Challenges for architectural level power modelling", *Power aware computing*, pp. 317–338, 2002.

[4] R. Graybill, R. Melhem, *Power Aware Computing*. New York: Springer US, 2002. [Online]. Available: http://dx.doi.org/10.1007/978-1-4757-6217-4

[5] J. A. Barnett, "Application-level power awareness", in *Power Aware Computing*, pp. 227–242, 2002. [Online]. Available: http://dx.doi.org/10.1007/978-1-4757-6217-4_12

[6] M. Krunic, I. Povazan, M. Popovic, J. Kovacevic, "Data flow CAD tool for firmware development and power consumption estimation in multi-core hearing aids", in *IEEE Int. Conf. Consumer Electronics (ICCE)*, 2016, Las Vegas, NV., 2016, to be published.

[7] L. Vogel, *Eclipse Rich Client Platform (vogella series)*. Hamburg, Germany: Lars Vogel, 2015.

[8] G. Callou, P. Maciel, E. Tavares, E. Andrade, B. Nogueira, C. Araujo P. Cunha, "Energy consumption and execution time estimation of embedded system applications", *Microprocessors Microsyst*., vol. 35, no. 4, p. 426–440, 2011. [Online]. Available: http://dx.doi.org/10.1016/j.micpro.2010.08.006

[9] N. Kavvadias, P. Neofotistos, S. Nikolaidis, K. Kosmatopoulos, T. Laopoulos, "Measurements analysis of the software-related power consumption of microprocessors", *IEEE Trans. Instrumentation and Measurement*, vol. 53, no. 4, pp. 1106–1112, 2004. [Online]. Available: http://dx.doi.org/10.1109/TIM.2004.830784

[10] P. V. Joshi, K. S. Gurumurthy, "Software power analysis for embedded DSP software", in *Proc. of the Intl. Conf. on Advances in Computing and Information Technology, (ACIT 2014)*, Bangkok, Thailand, 2014.

[11] V. Konstantakos, A. Chatzigeorgiou, S. Nikolaidis, T. Laopoulos, "Energy consumption estimation in embedded systems", in *Instrumentation and Measurement Technology Conf., (IMTC 2006)*, Sorrento, Italy, 2006. [Online]. Available: http://dx.doi.org/10.1109/imtc.2006.328405

[12] V. Konstantakos, A. Chatzigeorgiou, S. Nikolaidis, T. Laopoulos, "Energy consumption estimation in embedded systems", *IEEE Trans. Instrum. Meas.*, vol. 57, no. 4, p. 797–804, 2008. [Online]. Available: http://dx.doi.org/10.1109/TIM.2007.913724

[13] A. Sinha, A. Chandrakasan, "Software energy profiling", in *Power Aware Computing*, New York, Springer US, 2002, pp. 339–359. [Online]. Available: http://dx.doi.org/10.1007/978-1-4757-6217-4_17

[14] M. Bazzaz, M. Salehi, A. Ejlali, "An accurate instruction-level energy estimation model and tool for embedded systems", *IEEE Trans. On Instrumentation and Measurement*, vol. 62, no. 7, pp. 1927–1934, 2013. [Online]. Available: http://dx.doi.org/10.1109/TIM.2013.2248288

[15] M. Kim, J. Kong, S. W. Chung, "An online power estimation technique for multi-core smartphones with advanced display components", in *IEEE Int. Conf. on Consumer Electronics (ICCE)*, Las Vegas, NV, 2012.

[16] I. Povazan, M. Krunic, M. Popovic, "A profiling tool for heterogeneous multi-core systems", in *ECBS-EERC*, Brno, Czech Republic, 2015. [Online]. Available: http://dx.doi.org/10.1109/ecbs-eerc.2015.31

[17] S. Nikolaidis, N. Kavvadias, P. Neofotistos, "Base instruction cost measurements", in *Instruction level power measurements and analysis*, Thessaloniki, Energy-Aware SYstem-on-chip design of the HIPERLAN/2 standard, 2002, pp. 14–15.

[18] S. Nikolaidis, N. Kavvadias, P. Neofotistos, "Inter-instruction effect cost measurements", in *Instruction level power measurements and analysis*, Thessaloniki, Energy-Aware SYstem-on-chip design of the HIPERLAN/2 standard, 2002, pp. 15–16.

[19] K. Roy, M. C. Johnson, "Software design for low power", in *Low Power Design in Deep Submicron Electronics*, New York, Springer US, 1997, pp. 433–459. [Online]. Available: http://dx.doi.org/10.1007/978-1-4615-5685-5_15

[20] R. Basmadjian, H. de Meer, "Evaluating and modeling power consumption of multi-core processors", in *Third Int. Conf., Future Energy Systems: Where Energy, Computing and Communication Meet (e-Energy)*, 2012 Madrid, 2012. [Online]. Available: http://dx.doi.org/10.1145/2208828.2208840

[21] M.-C. Lee, V. Tiwari, S. Malik, M. Fujita, "Power analysis and low-power scheduling techniques for embedded DSP software", in *Proc. Eighth Int. Symposium on System Synthesis*, Cannes, 1995. [Online]. Available: http://dx.doi.org/10.1109/ISSS.1995.520621

[22] M. Kandemir, N. Vijaykrishnan, M. J. Irwin, "Compiler optimizations for low power systems", in *Power aware computing*, 2002, pp. 191–210.